Métaheuristiques : Stratégies pour l'optimisation de la production de biens et de services.

Marc Sevaux

Marc Sevaux

Métaheuristiques

Stratégies pour l'optimisation de la production de biens et de services

UVHC

Numéro d'ordre : 04/03

HABILITATION À DIRIGER DES RECHERCHES

Préparée au Laboratoire d'Automatique, de Mécanique d'informatique
Industrielles et Humaines du CNRS (UMR CNRS 8530)
dans l'équipe Systèmes de Production

Marc Sevaux

Métaheuristiques
Stratégies pour l'optimisation de la production de biens et de services

Soutenue le 1er Juillet 2004 devant le jury composé de :

Président	Prof. Christian Prins
Rapporteurs	Prof. Michel Gourgand
	Prof. Jin-Kao Hao
	Prof. Eric Taillard
Examinateurs	Prof. Stéphane Dauzère-Pérès
	Prof. Bernard Grabot
Directeur	Prof. Christian Tahon

Table des matières

Remerciements

La rédaction de ce manuscrit est l'aboutissement de plusieurs années de travail. Ce travail lui-même n'aurait pu être mené sans l'aide de plusieurs personnes auxquelles je souhaite exprimer ma gratitude.

Avant tout, je remercie les rapporteurs de cette dissertation, Michel Gourgand, Jin-Kao Hao et Eric Taillard, qui ont sûrement passé de longues heures à relire et corriger ce document. Que les autres membres du jury soient aussi remerciés.

Mes remerciements vont à la direction du LAMIH pour son accueil, et aussi à l'ensemble de l'équipe "Systèmes de Production" et sa bonne ambiance, ainsi qu'à Christian Tahon, son responsable, pour m'avoir fait confiance et donné une grande liberté d'action ces dernières années.

Je tiens à exprimer ma gratitude tout particulièrement à Christian Prins, qui a été mon professeur il y a de nombreuses années et qui a éveillé en moi le goût et la passion de la recherche. Je lui suis aussi reconnaissant pour les nombreuses discussions productives, pour cette collaboration que nous avons entamée depuis la fin de ma thèse et enfin, pour avoir transformé au fil des années notre partenariat en amitié sincère.

Que Stéphane Dauzère-Pérès, mon directeur de thèse soit aussi remercié chaleureusement ; sous sa direction de 1996 à 1998, j'ai pu comprendre et apprendre le métier de chercheur et une certaine éthique de la recherche. Son amitié a heureusement dépassé cette période.

Je remercie aussi les professeurs qui m'ont enseignés la recherche opérationnelle avec passion, Eric Pinson, Philippe Chrétienne, Jacques Carlier et Claude Berge ainsi que les personnes qui ont su m'aider quand j'en avais besoin, Philippe Thomin, Alain Gibaud, tout particlièrement Xavier Gandibleux et bien d'autres.

Je garde une place de choix pour Kenneth Sörensen, pour tout ce que nous avons créé et partagé ensemble, pour notre amitié ; le groupe EU/ME n'étant que la partie émergée de l'iceberg...

Enfin je tiens à terminer cette préface en remerciant de tout cœur mon épouse Sandrine et mes enfants, Victoria, François et Jean pour leur patience pendant mes nombreuses absences et pour leur amour sans limite.

Marc Sevaux
Valenciennes, Mai 2004

Première partie

Curriculum Vitæ

1 Informations générales

1.1 Etat Civil

NOM, Prénom	SEVAUX, Marc
Date et lieu de naissance	Né le 25 mai 1969 à Brest (29)
Nationalité	Française
Etat civil	Marié, trois enfants
Service militaire	effectué en 1995-1996, en tant qu'officier du corps technique et administratif du service de santé des armées.

1.2 Fonction actuelle

Poste	Maître de conférences (classe normale) en 61^{e} section du CNU
Adresse professionnelle	Université de Valenciennes et du Hainaut-Cambrésis UMR CNRS 8530, LAMIH / SP Le Mont Houy, Bâtiment Jonas 2 F-59313 Valenciennes Cedex 9 France
Téléphone	03 27 51 13 26
Fax	03 27 51 13 10
Email	`Marc.Sevaux@univ-valenciennes.fr`
Url	`http ://www.univ-valenciennes.fr/sp/sevaux/`
Faculté de rattachement	Institut universitaire de technologie Département Organisation et Génie de la Production (OGP) délocalisé à Cambrai
Laboratoire de rattachement	LAMIH/SP : Laboratoire d'Automatique, de Mécanique, d'Informatique industrielles et Humaines – UMR CNRS 8530 – Equipe Systèmes de Production

1.3 Prime d'encadrement doctorale et de recherche

Titulaire de la prime d'encadrement doctorale et de recherche (PEDR) depuis 2003.

1.4 Parcours et formation

Depuis Sept. 99 Maître de conférences en 61^{e} section à l'université de Valenciennes et du Hainaut-Cambrésis. Titulaire d'un poste à l'Institut Universitaire de Technologie, au département Organisation et Génie de la Production, délocalisé à Cambrai.

Déc. 98 – Août 99 Ingénieur de recherche (contrat à durée déterminée de 9 mois) au département Automatique et Productique de l'Ecole des Mines de Nantes. Chargé de la rédaction d'un cahier des charges pour la création d'une plateforme logistique (voir section 6.2).

1996 – 1998 Université Pierre et Marie Curie, Paris
Doctorat de l'Université Pierre et Marie Curie (Paris VI).

Spécialité	**Informatique et Recherche Opérationnelle**.
Laboratoire d'accueil	Ecole des Mines de Nantes, département Automatique et Productique.
Soutenance	le 11 Décembre 1998 à l'université Pierre et Marie Curie (Paris VI).
Sujet	Etude de deux problèmes d'optimisation en planification et ordonnancement.
Président du Jury	P. Chrétienne (Professeur, Université P. et M. Curie, Paris).
Rapporteurs	Y. Crama (Professeur, Université de Liège, Belgique), J.B. Lasserre (Directeur de Recherche, LAAS/CNRS, Toulouse)
Examinateurs	M.-C. Portmann (Professeur, Ecole des Mines de Nancy), C. Prins, (Maître Assistant, HDR, Ecole des Mines de Nantes), S. Dauzère-Pérès (Maître Assistant, HDR, Ecole des Mines de Nantes, *Directeur de thèse*).

Etude de deux problèmes d'optimisation en planification et ordonnancement (problème de planification de la production en temps continu et problème général d'ordonnancement à une machine). Résolution par l'utilisation de techniques de recherche opérationnelle. Validation par le développement de logiciels prototypes.

1994 – 1995 Université Pierre et Marie Curie, Paris
Diplôme d'Etudes Approfondies de l'Université Pierre et Marie Curie (Paris VI).

Spécialité	Informatique et Recherche Opérationnelle
Mention	Bien
Responsable	Ph. Chrétienne (Professeur, Université Pierre et Marie Curie, Paris)
Sujet de DEA	Les problèmes d'ordonnancement avec délais de communication
Encadrement	C. Picouleau (Maître de Conférences, CNAM, Paris).

1992 – 1994 Institut de Mathématiques Appliquées (IMA), Angers

Diplômes obtenus	**DEUG, Licence, Maîtrise**
Spécialité	Mathématiques Appliquées et Sciences Sociales

2 Enseignement

2.1 Enseignements à l'UVHC

Récapitulatif des heures enseignées à l'UVHC

Le tableau ci-dessous résume les heures d'enseignement depuis le recrutement à l'université de Valenciennes et du Hainaut-Cambrésis. Le descriptif des matières enseignées est présenté après le tableau. Dans ce tableau, sont reprises les heures de cours magistraux (CM), de travaux dirigés (TD) et de travaux pratiques (TP), ainsi que l'équivalence en heures TD (EqTD). L'IUT permet aussi de comptabiliser certaines heures pour des tâches administratives (Responsabilité des projets, Relations internationales), ainsi que des heures pour l'encadrement des stages et des projets.

Année 2003-2004	Niveau	CM	TD	TP	EqTD
Recherche Opérationnelle	IUT 2 et APPC	20h	2×40h	–	110h
Informatique	IUT 1	15h	2×20h		62.5h
Responsabilité de relations Internationales				10h	6.67h
Heures de stages et projets entrant dans le décompte du service					13h
Total année 2003-2004					**192.17h**

Année 2002-2003	Niveau	CM	TD	TP	EqTD
Recherche Opérationnelle	IUT 2 et APPC	20h	2×40h	–	110h
Informatique	IUT 1	15h	2×14h	12h	58.5h
Mathématiques de la décision	ISTV Master Info. 1	12h	–	–	18h
Responsabilité des projets				15h	10h
Heures de stages et projets entrant dans le décompte du service					29h
Total année 2002-2003					**225.5h**

Année 2001-2002	Niveau	CM	TD	TP	EqTD
Recherche Opérationnelle	IUT 2 et APPC	20h	2×40h	–	110h
Informatique	IUT 1	15h	1×24h	20h	59.83h
Programmation linéaire	EIGIP 2	–	1×18h	–	18h
Responsabilité des projets				15h	10h
Heures de stages et projets entrant dans le décompte du service					19.5h
Total année 2001-2002					**217.33h**

Année 2000-2001	Niveau	CM	TD	TP	EqTD
Recherche Opérationnelle	IUT 2 et APPC	20h	3×40h	–	150h
Informatique	IUT 1	–	1×25h	–	25h
Programmation linéaire	EIGIP 2	–	1×18h	–	18h
Heures de stages et projets entrant dans le décompte du service					27h
Total année 2000-2001					**220h**

Année 1999-2000	Niveau	CM	TD	TP	EqTD
Qualité	IUT 1	30h	1×30h	–	75h
Recherche Opérationnelle	IUT 2	–	1×40h	–	40h
Informatique	IUT 1	–	1×25h	2×25h	58.3h
Heures de stages et projets entrant dans le décompte du service					19.5h
Total année 1999-2000					**192.83h**

Descriptif des interventions à l'UVHC

La ligne "# heures" correspond aux heures affectées à ce module pour les étudiants. Pour les heures d'intervention me concernant, se reporter au tableau.

M17 – Algèbre et recherche opérationnelle

# heures	CM 20h, TD 40h.
Public	IUT OGP 2 et APPC
Lieu	IUT OGP Cambrai
Année(s)	depuis 1999
Responsabilité	responsable du module depuis 2000

- *Objectifs du cours :* Donner à l'étudiant les principaux outils de la recherche opérationnelle pour lui permettre d'optimiser les fonctions de production et de logistique principales en entreprise. Acquérir les connais sances de base en algèbre pour une poursuite d'étude éventuelle.
- *Contenu du cours :* Algèbre linéaire (bases de l'algèbre, inversion de matrices) ; introduction à la recherche opérationnelle (historique, principales définitions et termes de la recherche opérationnelle) ; théorie des graphes ; algorithmique des graphes (plus courts chemins, flot maximum, problèmes de transport) ; notions de programmation linéaire (modélisa tion mathématique, propriétés et théorie de la programmation linéaire) ; résolution graphique et algébrique ; applications industrielles de la recherche opérationnelle (gestion de production, ordonnancement, MRP, logistique) ; cas particuliers de la programmation linéaire (matrice de contraintes totalement unimodulaires) ; langages de modélisation (apprentissage par l'exemple, principaux modèles rencontrés en industrie de production, résolution de cas industriels avec Xpress-IVE de Dash Associates) ; mise en place d'un challenge depuis 2001 pour les élèves avec points de bonification pour les meilleurs résultats obtenus.
- *Notation :* une note de TD reflétant le travail pratique en continu, un devoir maison, deux devoirs surveillés d'une heure, un devoir final de deux heures.

M9 – Informatique

# heures	CM 15h, TD 24h, TP 20h.
Public	IUT OGP 1
Lieu	IUT OGP Cambrai
Année(s)	depuis 1999
Responsabilité	responsable du module depuis 2001

- *Objectifs du cours :* Donner aux étudiants les moyens d'utiliser l'informatique de manière avancée. Enseigner les bases de la programmation structurée.
- *Contenu du cours :* Introduction à l'informatique et à la bureautique (traitement de texte, tableur) ; introduction à la programmation structurée

(arbres programmatiques, langage structuré de 4^{e} génération, concepts de base, expressions, instructions, instructions conditionnelles, boucles, fonctions, procédures) ; construction de macros évoluées en VBA ; conception objet ; algorithmique générale ; mise en place de projets informatiques (depuis 2001) pour la construction d'application VBA complètes et utiles à un technicien OGP (un projet guidé, un projet libre) ; liaison entre les programmes et les feuilles de calcul Excel (récupération de données, écriture dans des feuilles de calcul, manipulation des objets Excel en VBA).

- *Notation :* deux devoirs surveillés d'une heure, un interrogation individuelle sur ordinateur, deux notes de projets.

MD – Mathématiques de la décision

# heures	CM 12h, TD 18h, TP 6h.
Public	Master Informatique 1
Lieu	ISTV Valenciennes
Année(s)	2002-2003
Responsabilité	responsable du module

- *Objectifs du cours :* Enseigner à l'étudiant les outils mathématiques nécessaires à la résolution de problèmes dans le cadre de l'aide à la décision.
- *Contenu du cours :* Introduction générale à la programmation mathématique (résolution linéaire, résolution entière, branch and bound en programmation entière, dualité) ; algorithmes de plus courts chemins ; algorithmes de flots ; algorithmes de résolution des problèmes de transports ; modélisation mathématique ; mise en œuvre de la résolution pratique en utilisant les logiciels professionnels (CPlex, MPL).
- *Notation :* deux devoirs surveillés, une note de TP.

F63 – Programmation linéaire

# heures	CM 12h, TD 18h
Public	EIGIP 2
Lieu	Ecole d'ingénieurs de Valenciennes
Année(s)	2000-2002
Responsabilité	intervenant de TD

- *Objectifs du cours :* Enseigner les bases de la programmation linéaire et de la modélisation mathématique pour la résolution de problèmes d'optimisation industrielle.
- *Contenu du cours :* Modélisation mathématique (techniques de modélisation classiques, principaux modèles, modélisation entière ou binaire) ; résolution graphique ; résolution algébrique (algorithme du simplexe, dualité) ; analyse de sensibilité.
- *Notation :* un examen final.

M11 – Qualité

# heures	CM 30h, TD 30h
Public	IUT OGP 1
Lieu	IUT OGP Cambrai
Année(s)	1999-2000
Responsabilité	responsable du module

- *Objectifs du cours :* Donner aux étudiants les bases de la qualité en entreprise et les outils nécessaires à son application industrielle.
- *Contenu du cours :* Introduction à la qualité (historique, définitions) ; rappels statistiques (moyenne, écart-type, variance, histogramme) ; démarche qualité en entreprise ; normes ISO ; outils mathématiques de la qualité.
- *Notation :* deux devoirs d'une heure, un devoir final de deux heures, une note de TD.

2.2 Enseignements avant l'intégration à l'UVHC

La formation pédagogique acquise depuis plusieurs années n'a pas débuté à l'université de Valenciennes, mais remonte aux années 1992-1993. Depuis, un certain nombre de matières ont été enseignées pour des public divers. La ligne "# heures" correspond ici aux heures que j'ai effectué.

Programmation linéaire

# heures	TD 35h, TP 52.5h
Public	Elèves ingénieurs (Bac+3 et Bac +4)
Lieu	Ecole des Mines de Nantes
Année(s)	1997-1999
Responsabilité	chargé de travaux dirigés
Contenu	Méthode du simplexe ; application à des problèmes du monde industriel ; utilisation de logiciels commerciaux.

Gestion de la production

# heures	TD 27.5h
Public	Elèves ingénieurs (Bac +4)
Lieu	Ecole des Mines de Nantes
Année(s)	1997-1999
Responsabilité	chargé de travaux dirigés
Contenu	Gestion des stocks ; MRP ; planification des capacités.

Mathématiques

# heures	120h
Public	CAP coiffure 1 et 2
Lieu	Institut technique des études et des carrières à Angers
Année(s)	1993-1994
Responsabilité	enseignant
Contenu	Notions élémentaires de mathématiques ; fractions ; pourcentages ; bases de la comptabilité.

Physique

# heures	60h
Public	CAP coiffure 1 et 2
Lieu	Institut technique des études et des carrières à Angers
Année(s)	1993-1994
Responsabilité	enseignant
Contenu	Notions de base en électricité ; dangers du courant électrique ; applications pratiques en salon de coiffure.

Chimie

# heures	60h
Public	CAP coiffure 1 et 2
Lieu	Institut technique des études et des carrières à Angers
Année(s)	1993-1994
Responsabilité	enseignant
Contenu	Principes d'une réaction chimique ; application au traitement des cheveux.

Informatique

# heures	TD 60h
Public	DEUG 1
Lieu	Institut de mathématiques appliquées à Angers
Année(s)	1992-1993
Responsabilité	chargé de travaux dirigés
Contenu	Initiation à la programmation ; algorithmique.

2.3 Encadrements pédagogiques

Encadrements à l'UVHC

2003-2004

IUT-OGP 1 1 Projet : Développement des relations avec les anciens étudiants de l'IUT-OGP, constitution d'un annuaire internet (8 étudiants).

IUT-OGP 2 1 Projet : Responsable du lancement du jeu d'entreprise FirStrat - Stra&Logic (40 étudiants).

IUT-OGP 2 2 Stages : Sujets en cours.

2002-2003

IUT-OGP 1 1 Projet : Développement des relations avec les anciens étudiants de l'IUT-OGP (8 étudiants).

IUT-OGP 2 2 Projets : Entreprise Blankaert Moto, assistance au choix d'une moto (3 étudiants), UVHC, gestion des déchets (2 étudiants).

IUT-OGP 2 2 Stages : 1. et 2. Campack SA, organisation d'une ligne de production de bouteilles de lait pendant sa mise en exploitation.

2001-2002

IUT-OGP 1 1 Projet : Réalisation d'un annuaire des anciens élèves de l'IUT OGP de Cambrai (6 étudiants).

IUT-OGP 2 1 Projet : Entreprise Former - Réorganisation d'un atelier de frappe à froid (6 étudiants).

IUT-OGP 2 3 Stages : 1. Babyliss SA, gestion des stocks pour le centre logistique, 2. Ygnis industrie, gestion de production, 3. SA Textiles Miersmann et Fils, optimisation de production.

2000-2001

IUT-OGP APPC 2 Projets : 1. Société Amival, mise en place d'une ligne de production alimentaire (4 étudiants), 2. Société Carolus Acier, aménagement d'un atelier de fabrication (4 étudiants).

IUT-OGP 2 2 Stages : 1. Entreprise Bohain Textile, gestion de production, 2. Société Sobotex, rédaction d'un cahier des charges pour un logiciel d'ordonnancement.

1999-2000

IUT-OGP 2 3 Stages : 1. Usine d'embouteillage de Saint-Amand-les-eaux, normes qualité, 2. Vieux-Condé estampages, gestion et suivi de production, 3. Mallez imprimerie, suivi de production.

IUT-OGP 1 1 Projet : mise en place d'une démarche qualité au sein du département OGP pour le suivi de la scolarité d'un élève (7 étudiants).

Encadrements avant l'intégration à l'UVHC

- Encadrement de mini-projets (découverte d'un outil spécifique de RO), élèves ingénieurs de dernière année de l'Ecole des Mines de Nantes (1997-1999).
- Encadrement de stages à l'étranger des élèves de troisième année (niveau maîtrise) de l'Ecole des Mines de Nantes (1996-1999).
- Encadrement de projets transversaux en informatique sur le thème de la planification et de l'ordonnancement d'ateliers (1998-1999).

2.4 Administration de l'enseignement

Elu au conseil restreint de l'IUT – depuis 2003

Le conseil restreint de l'IUT supervise l'attribution des postes d'enseignants-chercheurs, d'enseignants et d'attachés temporaires d'enseignement et de recherche au sein de l'IUT tout entier. Cette attribution se fait sur proposition des différentes commissions de spécialistes et mon intervention concerne les métiers du secteur secondaire.

Responsable des relations internationales – depuis 2003

La direction de l'IUT a décidé de motiver les départements pour relancer les relations internationales. Trois axes sont privilégiés : 1) les échanges d'étudiants, 2) les échanges d'enseignants et 3) le développement d'axes de recherche en collaboration avec l'étranger. La mission du responsable des relations internationales est de promouvoir ces échanges et d'assister les différents intervenants du département dans leurs démarches vers l'étranger en accord avec le directeur de département et la direction de l'IUT.

Responsable de l'organisation des projets – 2001-2003

Axés sur la communication en première année à l'IUT, ces projets ont pour but de faire travailler un groupe d'élèves autour d'un thème particulier et par la suite de présenter le résultat de leurs travaux devant leurs collègues et professeurs. Ils acquièrent ainsi une capacité à organiser le travail d'un groupe et présenter un travail. Une seconde partie du projet consiste à proposer un produit innovant et son processus de fabrication pour participer au concours national des OGP. Les meilleures propositions seront étendues dans le cadre du projet de 2^{e} année.
Les étudiants de deuxième année vont en début d'année réaliser un projet en groupe au sein d'une entreprise. Ils seront mis face à des industriels à qui ils doivent rendre des comptes. Ces projets portent principalement sur la rédaction de procédures qualité et sur la réorganisation générale d'ateliers, le développement de modules informatiques, l'étude de logiciels de GPAO pour une implantation sur site, etc. Depuis 2004, les projets en entreprise sont progressivement remplacés par la participation au concours national OGP (création d'un produit innovant et de son processus de production) et par la semaine du jeu d'entreprise.

Organisation des visites en entreprise – 1999-2001

Permettre aux élèves de connaître différents types d'entreprises et d'environnements de travail est un des buts de ces visites. Un questionnaire étudié en fonction de chaque entreprise nous permet de voir comment les élèves appréhendent leur futur cadre de travail industriel et éventuellement d'adapter certaines parties de la formation. Il s'agit d'environ 4 à 5 visites par an.

Recrutement, admission, jury

Participation aux jurys d'admission en DEA AISIH (2001-2003).
Participation au recrutement des élèves EIGIP (2000-2002).
Participation aux jurys d'attribution du DUT (depuis 2000).
Participation aux jurys de passage en IUT 2 (depuis 1999).

3 Supervision de travaux d'étudiants 2e et 3e cycle

3.1 Thèses de doctorat

Co-direction de Thèse européenne (2003-2006)

Frédéric BEUGNIES a débuté une thèse (bourse ministère) en septembre 2003 sur l'*Optimisation multiobjectif de problèmes de routage dans les réseaux informatiques*. Encadrement : Dir. de thèse X. Gandibleux (UVHC), Co-encadrant M. Sevaux (UVHC). Autres encadrants : S. RANDRIAMASY (Alcatel).

Frédéric BEUGNIES poursuit son travail de DEA sur l'étude de deux problèmes d'optimisation dans le cas de l'exploitation de réseaux informatiques. Ces travaux entrent dans le cadre d'une collaboration entre le LAMIH et ALCATEL.
Les questions posées relèvent de deux problèmes d'optimisation dans les graphes. Dans un premier temps, il s'agit d'étudier des algorithmes dans un cadre temps-réel et dans une seconde partie de faire de la gestion prévisionnelle de trafic. Bien que ces deux approches portent sur des problèmes de caractéristiques distinctes (la nature, la dimension, les contraintes, etc), elles se rejoignent dans le sens où l'optimisation finale ne devra pas se faire sur un seul objectif, mais sur plusieurs à la fois.

Co-direction de Thèse (2003-2006)

Karim BOUAMRANE a débuté en Mars 2003 une thèse sur le thème *Système d'aide à la régulation d'un réseau de transport bi-modal*. Encadrement : Dir. de thèse : Ch. Tahon (UVHC). Co-encadrant : M. Sevaux (UVHC).

Karim BOUAMRANE s'intègre dans le cadre d'un projet coopératif SART (Système d'Aide à la Régulation de Transport, anciennement projet COMPACT, voir section 6). Dans ce projet, un certain nombre de modules permettant la régulation du transport sont déjà opérationnels et ont été développés par les différents partenaires du projet. Le travail dévolu lors de cette thèse concerne la proposition et la mise en place opérationnelle du système d'aide proprement dit par le biais d'un moniteur d'activation. Un simulateur est intégré à l'application pour simuler le comportement du réseau de transport avant l'implantation du SART.

Co-direction de Thèse (2000-2004)

Yann LE QUÉRÉ a débuté en mai 2000 une thèse (en contrat Cifre avec la SNCF) sur le thème de *la planification réactive des tâches de maintenance pour les TGV*. Directeur de Thèse : Ch. Tahon. Co-encadrants : M. Sevaux (UVHC) et D. Trentesaux (UVHC). Superviseurs industriels : G. Martin (SNCF) et E. Blervacque (SNCF). Soutenance : 30 juin 2004. Jury : B. Grabot, M. Gourgand (Rapporteurs),

M.-J. Huguet, A. Thomas (Examinateurs).

Face à l'accroissement du trafic ferroviaire tant au niveau du fret que des passagers, il est important de diminuer les temps d'immobilisation lors des tâches de maintenance des TGV. Pourtant les tâches de maintenance sont complexes et souvent incertaines. L'étude s'intéresse alors à proposer une méthode consistant à mesurer l'impact d'une structure de décision et des activités décisionnelles sur un ordonnancement réactif.
Cette thèse a donné lieu à la publication d'un article en revue [9], une communication au GRP [45], une conférence internationale avec actes [20], une présentation aux journées doctorales d'automatique [16] et à la soumission d'un autre article pour publication [33]. Un contrat d'accompagnement de la thèse sur trois ans a été signé avec la SNCF pour un montant de *160 kF*.

3.2 Mémoires de DEA

DEA AISIH (2003-2004)

Changhong BIAN a réalisé un stage de DEA au sein de notre équipe. Le sujet était l'étude d'un *problème d'ordonnancement juste-à-temps sur une machine*.
Les outils développés pour cette étude se basent sur des heuristiques pour la création de solutions initiales dans des algorithmes génétiques qui sont implantés par la suite. Les algorithmes génétiques s'appuient sur un codage binaire particulièrement adapté à ce problème.

DEA AISIH (2003-2004)

Yan QIU a réalisé un stage de DEA au sein de notre équipe. Le sujet était l'*Application d'un algorithme génétique pour un problème de CAO* (ce sujet a été co-encadré avec Yves Mineur).
Dans ce travail, il s'agissait de mettre en œuvre un algorithme génétique permettant de résoudre un problème d'optimisation continu rencontré dans le cadre de la modélisation de formes géométriques en CAO. Un algorithme génétique déjà opérationnel a été fourni comme base d'étude et le travail a consisté à proposer des améliorations importantes.

DEA AISIH (2002-2003)

Johann SAINT MARS, étudiant en DEA a réalisé un stage au sein de notre équipe. Le sujet était l'étude d'un *problème d'ordonnancement à une machine avec minimisation du retard pondéré*.
Dans ce travail, une approche par programmation linéaire a été privilégiée. En faisant appel aux outils commerciaux et aux modèles les plus récents pour ce problème, l'étudiant a évalué les résultats trouvés et les a comparé aux meilleurs résultats des heuristiques connues.

DEA AISIH (2000-2001)

Christophe TILLEUL, étudiant en DEA a réalisé un stage au sein de notre équipe. Le sujet était l'étude d'un *problème d'ordonnancement bicritère à une machine* (ce sujet a été co-encadré avec X. Gandibleux).
Les problèmes d'ordonnancement sont le plus souvent très délicats à résoudre lorsqu'ils sont $\mathcal{NP}$-complets. Dans le cas bi-critère, la difficulté augmente encore. L'objectif de ce travail a été de résoudre efficacement le problème d'ordonnancement à une machine avec dates de disponibilité pour les critères de minimisation du nombre de tâches en retard et le retard total de ces tâches.

3.3 Projets de DESS

DESS ICHM (2003-2004)

Chris BAUCHOT et Sébastien LEGENDRE ont réalisé au sein de notre équipe un projet de DESS ICHM. Le sujet est la *proposition d'une interface permettant la manipulation et l'édition de graphes* (ce sujet a été co-encadré avec P. Thomin).
Suite à la publication du livre "Algorithmes de graphes" [1], la construction d'une interface graphique pour manipuler les graphes et exécuter les différents algorithmes développés dans le livre est devenue indispensable. Une nouvelle version de ce livre sera proposée en C++ prochainement. L'interface a été développée en QT et est toujours opérationnelle.

DESS CCI (2002-2003)

Shiva ROUHOLAMINI et Rachid HARMAOUI, étudiants en DESS "Connaissances Complémentaires en Informatique" ont réalisé un projet au sein de notre équipe. Le sujet traité était l'étude d'un *problème d'ordonnancement juste à temps*.
Dans ce stage, une approche par heuristiques constitue le cœur du travail demandé. Les étudiants ont pu développer un certain nombres d'heuristiques et les résultats ont été comparés aux heuristiques classiques connues pour ce problème.

DESS ICHM (1999-2000)

Laurent HULIN a effectué son projet dans notre équipe. Le sujet est la *Proposition d'une interface permettant la manipulation de références bibliographiques* (ce sujet a été co-encadré avec P. Thomin).
Il avait pour but de développer une interface qui permette d'effectuer des recherches de références bibliographiques à partir de fichiers au format BibTEX. L'outil a été développé en php.

3.4 Projets IUP GEII

IUP GEII (2003-2004)

Yohan COLIN et Nicolas KOWALSKI ont effectué un projet dans notre équipe. Le sujet est l'*Optimisation de tournées de véhicules*.
Le but du projet est d'utiliser les outils de la suite "Optimisation" d'ILOG pour résoudre le problème de tournées de véhicules dans sa version académique. Une comparaison a été menée entre les résultats du logiciel commercial et les meilleures solutions connues de la littérature.

IUP GEII (2001-2002)

Sébastien LECHARDEUR et Johan SAINT MARS ont effectué un projet dans notre équipe. Le sujet est *Ordonnancement à une machine : une approche génétique* (ce sujet a été co-encadré avec P. Thomin).
Le critère à optimiser est la retard pondéré total dans le cas où les tâches sont soumises à des dates de disponibilité. L'issue de ce projet a permis de présenter une communication à IFORS'2002 [42].

IUP GEII (2001-2002)

Mathieu LEGEZYNSKI et Stéphane DEBACKER effectuent un projet dans notre équipe. Le sujet est *Ordonnancement à une machine : une approche par recherche dispersée* (ce sujet a été co-encadré avec P. Thomin).
Les deux binômes travaillent en parallèle sur un même problème d'ordonnancement mais avec deux approches différentes. L'approche par recherche dispersée est plus difficile à mettre en œuvre mais a donné de meilleurs résultats.

4 Administration et animation de la recherche

4.1 Animation de la recherche

Sur le plan international

Création et pilotage du groupe EU/ME (EURO Working Group – European chapter on Metaheuristics). Groupe européen de recherche sur le thème des métaheuristiques (créé à la suite d'une école d'hiver, avec K. Sörensen – Université d'Anvers, Belgique et C. Wynants – Université libre de Bruxelles, Belgique). Ce groupe compte aujourd'hui 750 membres de 65 nationalités différentes, avec un potentiel encore élevé. Le but de ce groupe, créé en mars 2000, est de donner l'opportunité aux chercheurs et industriels de se rencontrer et partager leur expérience de l'utilisation et du développement des métaheuristiques. D'autres renseignements sur le groupe se trouvent à l'adresse suivante :
`http://webhost.ua.ac.be/eume/`

Parmi les tâches effectuées dans ce cadre, on retrouve

- l'administration du groupe proprement dite (gestion des membres, organisation d'assemblées générales, etc),
- la co-gestion du site web du groupe (mailing list, forum de discussion, informations sur les conférences, etc),
- les relations avec EURO, l'organisme de tutelle (participation à des réunions générales, rédaction de rapports d'activités),
- la gestion du budget,
- l'organisation d'une manifestation internationale annuelle (depuis 2001).

Au niveau du laboratoire

Création et animation (de 2000 à 2002) des séminaires du LAMIH permettant à toutes les équipes de proposer des présentations d'intervenants locaux ou extérieurs. Ces séminaires ont lieu sur un rythme régulier (2 fois par mois environ). Ce travail a été réalisé en collaboration avec plusieurs membres d'autres équipes du LAMIH. Depuis 2002, la direction du laboratoire a confié cette tâche à son secrétariat.

Au niveau de l'équipe

Chargé de la collecte des informations (publications et activités de recherche et d'encadrement) pour la mise à jour du rapport d'activité.

Mise en place et gestion des rapports de recherche internes à l'équipe. Ces rapports permettent de diffuser l'information et prendre date pour une publication qui est soumise à une revue mais pas encore acceptée.

4.2 Organisation de manifestations à Valenciennes

Congrès International francophone PENTOM
Dates : 27-29 mars 2003
Organisateur : LAMIH/SP
Président du CO : O. Sénéchal
Thème : Performance et nouvelles technologies en maintenance
Participants : 115

Congrès International IFIP FEATS
Dates : 12-14 juin 2001
Organisateur : LAMIH/SP
Président du CO : D. Deneux
Thème : Modélisation avancée par *features* dans la conception produit
Participants : 80

Réunion du groupe Bermudes
Dates : 2 février 2001
Organisateurs : X. Gandibleux et M. Sevaux
Thème : Ordonnancement dans les ateliers flexibles
Participants : 40

Journées GRP
Dates : 27-28 octobre 2000
Organisateur : LAMIH/SP
Président du CO : D. Trentesaux
Thème : Productique
Participants : 80

4.3 Organisation de manifestations en dehors de Valenciennes

3e Joint meeting EU/ME
Dates : 18-19 décembre 2003
Lieu : Université d'Anvers (Belgique)
Organisateurs : EU/ME et l'université d'Anvers
Comité d'organisation : W. Dullaert, M. Sevaux, K. Sörensen, J. Springael
Thème : Applications réelles des métaheuristiques
Participants : 60
Site web : `http ://webhost.ua.ac.be/eume/workshops/reallife.html`

2e Joint meeting EU/ME
Dates : 4-5 novembre 2002
Lieu : Paris, Carré des sciences
Organisateurs : EU/ME et le groupe français PM2O (programmation mathématique multi-objectif)
Comité d'organisation : X. Gandibleux, M. Sevaux, K. Sörensen, V. T'Kindt
Thème : Les métaheuristiques multi-objectifs

Participants : 60
Site web : `http ://webhost.ua.ac.be/eume/workshops/momh.html`

1er Joint meeting EU/ME
Dates : 28 novembre 2001
Lieu : City university, London (UK)
Organisateurs : EU/ME et UK Local search group
Comité d'organisation : C. Glass, C. Potts, M. Sevaux, K. Sörensen
Thème : Métaheuristiques
Participants : 30
Financement : obtenu pour 3 jeunes chercheurs auprès d'EURO

4.4 Organisation / président de sessions

Organisation d'un cluster de sessions
Conférence : INFORMS Denver Colorado 2004
Dates : 24-27 octobre 2004
Thème principal : métaheuristiques
Sessions : Tabu and scatter search, Genetic algorithms, Hybrid search.

Organisation de session invitée
Conférence : MIC'2001, Porto
Dates : 16-20 juillet 2001
Thème : ordonnancement et métaheuristiques.

Président de session
2004 : EURO XX (semi-plenary session)
2003 : MIC 2003, EMO 2003
2002 : PMS 2002, ROADEF 2002
2001 : MIC 2001, MOSIM 2001, ORBEL 2001.

5 Visibilité, rayonnement et autres activités

5.1 Collaborations

Université polytechnique de Hong-Kong

Correspondant : C. Oğuz
Thème : Approches hybrides pour le flow-shop multi-stages
Visites : M. Sevaux à Hong-Kong (11-21 mai 2004 et 4-16 mars 2003). C. Oğuz à Valenciennes (22-29 mai 2004 et 12-27 juillet 2003)
Description : Cette collaboration a lieu dans le cadre d'un programme d'action intégré (voir section 6.3). C. Oguz et moi-même travaillons sur des problèmes d'ordonnancement de type flow-shop avec de nouvelles approches intégrant les métaheuristiques et les techniques de type propagation de contraintes.

Université d'Anvers (Belgique)

Correspondant : K. Sörensen
Thème : Métaheuristiques ; Robustesse
Visites : En moyenne trois par an depuis 2000 ; accueil de K. Sörensen en moyenne trois fois par an depuis 2001.
Description : Développement de nouvelles métaheuristiques hybrides [31] et de méthodes pour l'ordonnancement robuste [18]. Rédaction de deux articles en commun [7, 31]. Gestion du groupe EU/ME et mise en place d'un nouveau projet [37]. Présentation lors d'un séminaire [56].

Université de Clermont-ferrand

Correspondants : G. Fleury et P. Lacomme
Thème : Ordonnancement stochastique
Description : Présentation en commun d'un article sur le problème de l'ordonnancement stochastique [14].

Université de Technologie de Troyes et de Clermont-ferrand

Correspondants : C. Prins (UTT) et P. Lacomme (UBP)
Thème : Tournées sur arcs ; algorithmes de graphes
Visites : deux en 2002 et deux en 2003
Description : Rédaction en commun d'un article sur le problème des tournées sur arcs multiobjectifs [17]. Rédaction d'un livre [1].

Ecole des Mines de Nantes

Correspondant : S. Dauzère-Pérès
Thème : Ordonnancement à machines parallèles et à une machine
Visites : M. Sevaux à Nantes (4-9 février 2002 et 17-21 avril 2001)
Description : Développement de nouveaux algorithmes pour la résolution d'un problème à machines parallèles [19]. Poursuite des travaux engagés précédemment et rédaction de plusieurs articles en commun [5, 8, 10, 35].

Ecole des Mines de Nantes et Univ. de Technologie de Troyes
Correspondants : C. Guéret (EMN) et C. Prins (UTT)
Thème : Programmation linéaire
Description : Collaboration et co-écriture du livre *Programmation Linéaire* et de sa traduction [2, 3].

Massachusetts Institute of Technology
Correspondant : S.B. Gershwin
Thème : Planification de la production en temps continu ; détermination de politiques de contrôle
Visites : M. Sevaux au MIT (15 avril-17 mai 1997)
Description : Rédaction d'un article en commun [10].

Institut de Math. Appliquées et Univ. de Technologie de Compiègne
Correspondants : P. Baptiste (UTC), L. Péridy (IMA), E. Pinson (IMA) et D. Rivreau (IMA)
Thème : Ordonnancement à une machine
Description : Collaboration dans le cadre de l'étude des problèmes d'ordonnancement à une machine. Partage d'informations et de jeux de données. Partage des résultats. Présentation conjointe des résultats lors d'une journée du Gotha [59].

5.2 Appartenance à des sociétés et des groupes de recherche

Sociétés nationales et internationales

SOGESCI-B.V.W.B
Description : société belge de Recherche Opérationnelle
Adhésion : depuis 2001
Activités : participation aux manifestations nationales [46].

EU/ME
Description : EURO working group, European chapter on metaheuristics
Adhésion : depuis 2000
Activités : administration du groupe (voir section 4.1), présentation lors des manifestations [37, 38, 40].

ROADéF
Description : société française de Recherche Opérationnelle et d'Aide à la Décision
Adhésion : depuis 1998
Activités : participation et présentations [39, 43, 47, 48] lors des manifestations nationales et régionales, participation aux assemblées générales, action auprès de la société pour les écoles d'été et d'hiver d'EURO.

INFORMS

Description : Institute For Operations Research and the Management Science
Adhésion : depuis 1996
Activités : Organisation d'un cluster au congrès INFORMS 2004. Présentation des travaux au congrès INFORMS 1998 [49].

Groupes Nationaux

GDR MACS/STP

Description : Groupe de recherche du CNRS en Modélisation Analyse et Conduite des Systèmes, pôle Sciences et Techniques de la Production
Adhésion : depuis 2003 (création du GDR en 2003)
Activités : Participation aux manifestations nationales. Présentation au thème ORT [53].

Club EEA

Description : Club des enseignants et des chercheurs en électronique, électrotechnique et automatique.
Adhésion : depuis 2003

GRP

Description : Groupe de Recherche en Productique
Adhésion : de 2000 à 2003
Activités : Présentation de travaux aux manifestations nationales [41, 44, 45], participation à l'organisation d'une manifestation à Valenciennes.

PM2O

Description : Groupe de travail de la ROADEF en Programmation Mathématique Multi-Objectif
Adhésion : depuis 2000
Activités : participation aux réunions du groupe, organisation d'une manifestation conjointe avec EU/ME. Présentations [40, 54].

Bermudes

Description : Groupe de recherche en ordonnancement d'ateliers flexibles
Adhésion : depuis 1999
Activités : participation et présentation [63] aux manifestations nationales, organisation d'une réunion Bermudes à Valenciennes (avec X. Gandibleux).

GOThA

Description : Groupe de recherche en Ordonnancement Théorique et Appliqué
Adhésion : depuis 1996
Activités : participation et présentation [59, 65, 66] à de nombreuses réunions du groupe (en particulier dans le thème flexibilité), rédaction d'un chapitre de livre [32].

Groupes Régionaux

GRAISyHM

Description : Groupe de Recherche en Automatique, Informatique des Systèmes Homme-Machine, thème commande et pilotage
Adhésion : depuis 2000 Activités : participation et présentation [62] aux réunions du groupe.

5.3 Fonction d'édition

Editeur associé de IJCI

Depuis 2004, membre du comité éditorial de la revue *International Journal of Computational Intelligence*. Le but de cette revue internationale est de développer des systèmes qui reproduisent l'analyse, la résolution des problèmes et les facultés d'apprentissage du cerveau. Ces systèmes apportent les avantages de la connaissance et de l'intelligence pour la résolution des problèmes complexes. Les domaines couverts par le journal incluent : les outils intelligents génériques (techniques et algorithmes), des applications (utilisant des techniques intelligentes) et des technologies intelligentes naissantes.

Editeur associé de IJSP

Depuis 2004, membre du comité éditorial de la revue *International Journal of Signal Processing*. Le but de la revue IJSP est de prendre en compte tous les aspects de la théorie et de la pratique du traitement du signal. La politique de la revue est une diffusion rapide des connaissances et des expériences auprès des ingénieurs et des scientifiques travaillant dans la recherche, le développement ou l'application pratique du traitement du signal.

Numéro Spécial EJOR

Suite au 3^{e} Joint EU/ME workshop en collaboration avec l'université d'Anvers, un numéro spécial de la revue *European Journal of Operational Research* est en préparation. Ce numéro spécial porte sur le thème de l'application des métaheuristiques. Date limite de soumission : 7 avril 2004, fin prévisionnelle du processus de relecture : octobre 2004.

Edition d'un livre

Suite au 2^{e} Joint EU/ME workshop "MOMH", il a été décidé de publier dans un livre, une sélection d'articles soumis à un processus de relecture de type revue internationale. Cet ouvrage est publié comme "Lecture Notes in Economics and Mathematical Systems" [4]. Taux de sélection : 36%

5.4 Evaluation de la recherche

Rapporteur de revues internationales

Régulier pour les revues et chapitres de livres
- European Journal of Operational Research – EJOR
- Kluwer (chapitres de livre)
- Journal of Heuristics
- Annals of Operations Research
- IEEE System Man and Cybernetics – IEEE-SMC
- Engineering Applications of Artificial Intelligence – EAAI

Occasionnel pour les revues et chapitres de livres
- Quaterly Journal of the Belgian, French and Italian Operations Research Societies – 4OR
- International Journal of Modelling and Simulation
- IEEE Transactions on Robotics and Automation – IEEE TRA
- Information Systems and Operational Research – INFOR
- Robotics and Computer Integrated Manufacturing
- IIE Transactions
- Decision Support Systems
- Rairo - Operations Research
- Journal Européen des Systèmes Automatisés – JESA
- Hermes (chapitre de livre)

Comités scientifiques

MIC'2005 Metaheuristics International Conference
Lieu : Vienne, Autriche
Dates : Eté 2005
Charge : Appel à communications, relecture d'articles soumis.

PENTOM'2005 PErformance et Nouvelles TechnOlogies en Maintenance
Lieu : Rabat, Maroc
Dates : Pâques 2005
Charge : Appel à communications, relecture d'articles soumis.

FRANCORO 2004 Conférence Francophone Intl. de Recherche Opérationnelle
Lieu : Fribourg, Suisse
Dates : 18-21 août 2004
Charge : Appel à communications, relecture d'articles soumis.

EURO'2004 EURO Conference
Lieu : Rhodes Island, Grèce
Dates : 4-7 juillet 2004
Charge : Appel à communications, organisation des "medals and awards", recherche de conférenciers invités.

ISS'2004 International Symposium on Scheduling
Lieu : Yumebutai, Japan
Dates : 24-26 Mai 2004
Charge : Appel à communications, relecture d'articles soumis.

MIC'2003 Metaheuristics International Conference
Lieu : Kyoto, Japon
Dates : 25-28 août 2003
Charge : Appel à communications, relecture d'articles soumis.

ESI XXI EURO Summer Institute
Lieu : Neringa, Lithuania
Date : 25 juillet-7 août 2003
Charge : Choix de candidat français.

PENTOM'2003 PErformance et Nouvelles TechnOlogies en Maintenance
Lieu : Valenciennes
Dates : 27-29 mars 2003
Charge : Relecture d'articles soumis.

MIC'2001 Metaheuristics International Conference
Lieu : Porto, Portugal
Dates : 16-20 juillet 2001
Charge : Appel à communications, relecture d'articles soumis.

Jury de thèses

Comité d'accompagnement de la thèse de N. Souaï (Faculté polytechnique de Mons, Belgique), *Problèmes de "crew scheduling" pour les campagnes aériennes*, soutenance prévue en 2006.

Examinateur de la thèse de K. Sörensen (Université d'Anvers, Belgique), *Robustness in optimization*, soutenue le 13 juin 2003.

Examinateur de la thèse de W. Ramdane-Chrérif (Université de technologie de Troyes), *Problèmes de tournées de véhicules sur arcs* soutenue le 12 décembre 2002.

Commission de spécialistes

Membre élu titulaire de la commission de spécialistes en 61^{e} section depuis 2004.

Membre élu suppléant de la commission de spécialistes en 61^{e} section en 2003.

6 Contrats industriels, projets de recherche et financements

6.1 Contrats industriels

Contrat Alcatel - Anvers/Marcoussis

Contrat international – 12 mois – 2003-2004
Co-responsable scientifique, correspondant pour le LAMIH dans le cadre de cette collaboration : M. Sevaux (LAMIH/SP) – 4 participants.

Une clause de confidentialité a été signée pour ce projet. Les activités menées lors de cette collaboration et les montants importants engagés ne pourront être dévoilés que lorsque le contrat sera terminé et la clause de confidentialité levée. Par contre, au fur et à mesure de l'avancement certaines parties de l'activité pourront être rendues publiques avec l'accord d'Alcatel. Elles seront alors mentionnées dans ce paragraphe et dès que possible brevetées et publiées.

Thèse Cifre SNCF

Contrat régional – 24.2k€ (160kF) – 36 mois – 2000/2003
Responsable scientifique : C. Tahon (LAMIH/SP) – 4 participants.

Convention CIFRE avec l'EIMM (Etablissement Industriel de Maintenance du MAtériel) d'Hellemmes. Deux personnes de la SNCF sont impliquées dans le suivi du projet. Une description plus détaillée du sujet de thèse se trouve dans la section 3.1.

Centre Hospitalier de Valenciennes

Contrat local – 48k€ – 36 mois – 2000/2003
Responsable scientifique : C. Tahon (LAMIH/SP) – 4 participants.

Ce contrat entre notre équipe et le Centre Hospitalier de Valenciennes avait pour objectif de réaliser une étude complémentaire du schéma directeur pour les flux logistiques en vue de la construction d'un nouveau bâtiment. En fait, avant même de prendre en compte la réalisation du nouveau bâtiment, nous avons dû travailler sur des modifications internes structurelles (construction d'une nouvelle blanchisserie, déplacement des cuisines). Ce travail a débuté au début de l'année 2000 et pour une durée de 3 ans. Pour cette étude, la simulation a été l'outil de travail privilégié.

6.2 Projets de recherche

Projet Coopératif – SART (Phase 2, COMPACT)

Contrat régional – 139k€ (Part labo 42.7k€) – 18 mois – 2003/2004
Responsable scientifique : S. Hayat (INRETS) – 30 participants.

Dans cette seconde phase, le problème de transport multi-modal est directement abordé et des propositions de modélisation et de résolution sont faîtes pour proposer un système d'aide à la régulation du trafic d'un réseau de transport multimodal (bimodal tramway/bus). Valenciennes et la Semurval, société exploitant les transports en commun de Valenciennes sont les principaux acteurs de ce projet. La construction future du tramway fait que la réalisation de ce projet devient indispensable pour nos partenaires. C'est dans ce cadre que s'inscrit en partie la thèse en co-tutelle de Karim Bouamrane (voir section 3.1). Présentation lors d'un séminaire [62].

Projet Coopératif – Phase 1, MOST/TACT

Contrat régional – 150k€ (Part labo 27.4k€) – 18 mois – 2001/2002
Responsable scientifique : C. Tahon (LAMIH/SP) – 30 participants.

Méthodologies pour l'Optimisation dans les Systèmes de Transport et de Télécommunications / Technologies Avancées dans le domaine de la Communication et des Transports Terrestres. Dans le cadre du CPER 2000-2006, le GRRT Nord-Pas de Calais a obtenu le financement pour le projet coopératif. Il s'agit dans un premier volet de ce projet de développer des outils en commun pour la résolution de problèmes liés aux transports mais pas exclusivement. Présentation lors d'un séminaire [62].

Projet S.M.I.L.E.

Projet interne EMN – 2 mois – 1999
Responsable scientifique : P. Dejax (EMN/AUTO) – 2 participants.

Rédaction du cahier des charges du projet de développement d'une plate forme d'essais d'un Système de Modélisation et d'Intégration de la Logistique d'Entreprise. Ce projet vise à permettre aux entreprises le suivi de la chaîne logistique avec un module spécifique pour le suivi en temps réel des camions de collecte ou de livraison ou l'adaptation aux perturbations du réseau routier est instantanée.

6.3 Financements obtenus

Financements obtenus pour déplacements

Dans le cadre du support d'activités scientifiques, le ministère des affaires étrangères peut venir en aide aux laboratoires et aux chercheurs pour financer des déplacements :

Aout 2003 Financement individuel (850€) pour la conférence MIC'2003 au Japon. Présentation d'un article [15].

Juillet 1999 Financement d'équipe (530€ – P. Castagliola, C. Prins et M. Sevaux) pour la conférence IEPM'99 en Ecosse. Présentation d'un article [25].

Programme d'action intégré - Hong-Kong

Projet international d'échange bilatéral – 8k€ – 24 mois – 2003-2004
Responsable scientifique : M. Sevaux (LAMIH/SP) – 3 participants.

Ce programme d'échange avec Hong-Kong, développé de manière individuelle au laboratoire est le début d'une collaboration avec Ceyda Oğuz. Le thème retenu est le développement de méthodes de résolution pour des problèmes d'ordonnancement avec l'intégration des techniques de propagation de contraintes dans les métaheuristiques ou dans la programmation linéaire. Ce programme est financé par l'Egide sous la forme d'un programme d'action intégré Procore. Visites : M. Sevaux à Hong-Kong (11-21 mai 2004 et 4-16 mars 2003). C. Oğuz à Valenciennes (22-29 mai 2004 et 12-27 juillet 2003)

Projet national prospectif – Jemstic

Contrat national – 13k€ – 12 mois – 2002
Responsable scientifique : D. Trentesaux (LAMIH/SP) – 6 participants.

Conception de systèmes coopératifs pour l'aide au pilotage distribué du cycle de vie des systèmes de production de biens et de services. Il s'agit de savoir ici, si l'action de coopération est bénéfique, toujours, parfois, ou peu souvent pour l'ensemble des partenaires et aussi de mesurer l'impact de la coopération sur les modes de fonctionnement de chacun des acteurs.

7 Thématiques de recherche

Les différents thèmes de recherche décrits ci-dessous sont le fruit de plusieurs périodes de travail. On notera les périodes *DEA* (1994-1995), *thèse* (1996-1998), *après-thèse* (1998-1999) et *MDC* (depuis 1999). Chaque thème abordé reprend dans un cartouche la période de travail dans laquelle les recherches ont été effectuées et les résultats obtenus en terme de publication et de visibilité du thème (chaque publication n'est citée que dans un seul cadre, même si certains sujets se recoupent). La figure 1 reprend les thématiques de recherche menées depuis la thèse. Dans cette même figure, on retrouve aussi les outils nécessaires à la résolution des problèmes.

Introduction

Le contexte industriel de plus en plus compétitif oblige les entreprises à sans cesse améliorer leur productivité tant au niveau des biens que des services. Dans ce cadre, on retrouve classiquement les différents niveaux de décision *stratégique*, *tactique* et *opérationnel*. Pour la production de biens, on s'intéresse à deux aspects correspondants à deux niveaux de décision : la planification de production (niveau tactique) et l'ordonnancement (niveau opérationnel). Pour les services, on s'intéresse plus spécifiquement aux tournées de véhicules.

Le problème de planification est abordé dans un premier temps sous un angle nouveau, mettant en avant la notion d'horizon continu (par opposition au classique horizon discrétisé). Basée sur cette modélisation, la détermination de politiques de contrôles est abordée avant de terminer par un domaine en plein essort, la planification réactive.

Les problèmes d'ordonnancement, plus académiques dans leur présentation, concernent l'objectif particulier de la minimisation du nombre de jobs en retard. Ce type d'objectif est résolu pour les problèmes à une machine et à machines parallèles avec des dates de disponibilité. Le nombre de jobs en retard peut être pondéré ou non. Par la suite, on s'intéresse au problème à une machine, sans doute le plus difficile à résoudre (toute relaxation donne aussi un problème $\mathcal{NP}$-difficile au sens fort). Il s'agit du problème de la minimisation du retard pondéré total avec dates de disponibilité sur une machine unique. Enfin, sont abordés des problèmes de robustesse indispensables pour les industriels soucieux d'appliquer ces méthodes dans le cas opérationnel.

Les tournées de véhicules attirent l'attention des chercheurs depuis de nombreuses années et pourtant il reste encore des possibilités pour améliorer les méthodes existantes et en proposer de nouvelles encore plus performantes. Les tournées de véhicules sur arcs représentent un intérêt particulier car on retrouve ces problèmes dans la réalité avec par exemple le salage des routes en cas de verglas, l'entretien des bordures des routes, la distribution de courrier et la collecte des ordures ménagères par exemple. On s'intéresse alors à l'optimisation bi-objectif de ce problème. Le cas mono-objectif est aussi résolu de manière très efficace

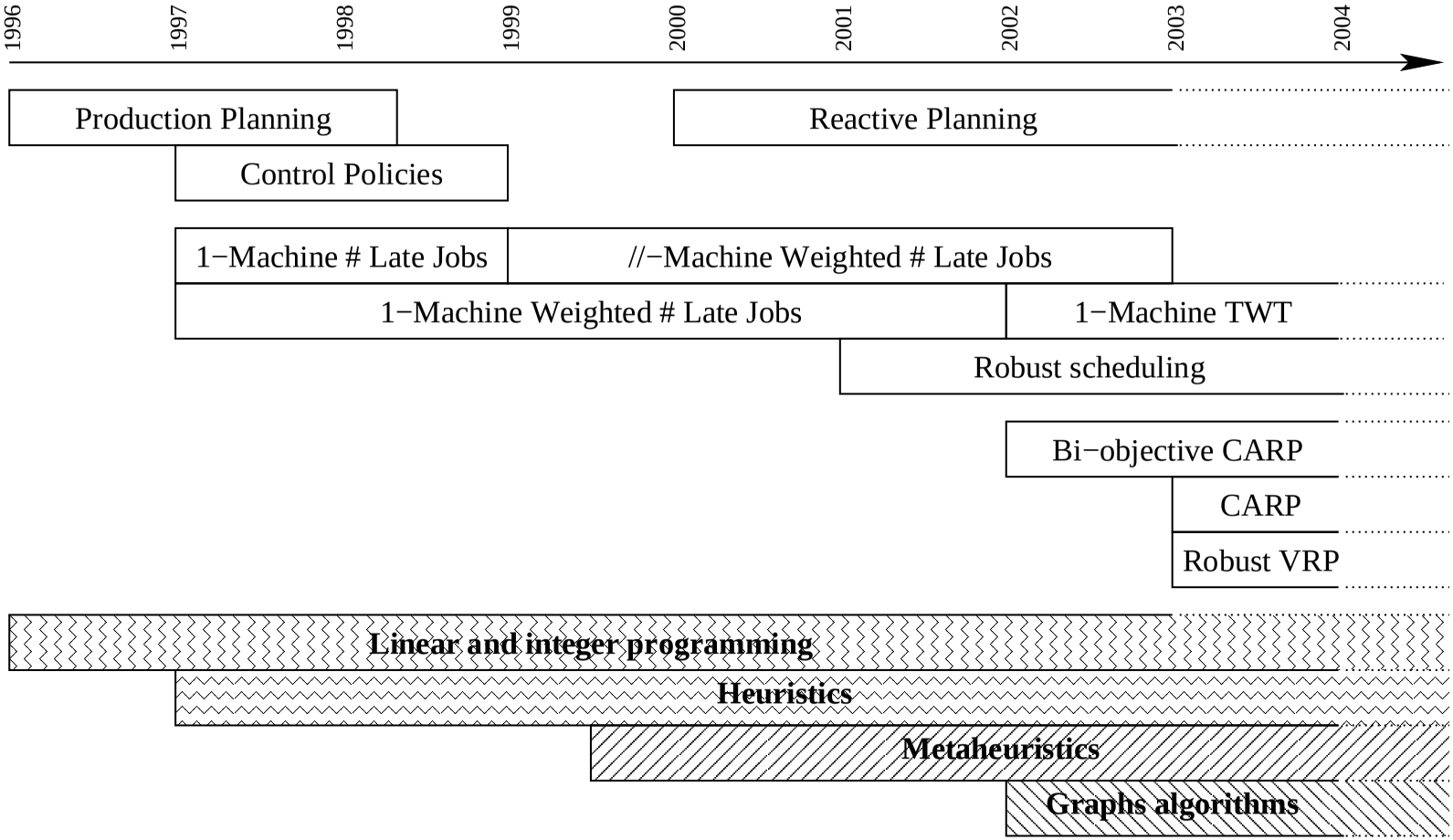


FIG. 1 – Problèmes et outils abordés

donnant aujourd'hui sur des jeux de tests de la littérature les meilleurs résultats. Quand le problème de tournées de véhicules est à résoudre non plus sur les arcs, mais sur les nœuds du réseau, on s'intéresse alors à produire des solutions robustes.

7.1 Planification de la production

Planification de la production en temps continu

Période : 1996–1998.
Résultats : un article publié en revue [11],
une conférence avec actes [28],
une conférence sans actes [51] et
deux séminaires [64, 65].

La planification de production se résout habituellement en discrétisant un horizon de planification. Pourtant dans le cas de la production "gros volumes", il est souvent préférable de conserver l'horizon intact et de considérer un problème de *planification de la production en temps continu* [11]. Dans ce cas, la demande (habituellement exprimée en quantités) est remplacée par des taux de demande et la production est alors exprimée elle aussi sous forme de taux [67]. Les taux de demande sont supposés connus et constants par morceaux. L'horizon de planification est séparé en périodes où la demande est constante, et les périodes peuvent avoir des longueurs différentes. La capacité de production étant parfois insuffisante la rupture est autorisée. Ainsi, dans d'autres périodes, il est parfois nécessaire de produire plus que les demandes courantes. Il faut donc déterminer combien produire en plus, et pendant combien de temps.

Pour réduire encore les coûts, on autorise les taux de production à changer à d'autres instants que ceux où la demande change. Ces dates, ainsi que les fins de périodes, seront appelées temps de changement [51]. Ainsi, il faut déterminer à la fois les taux de production et les temps de changement. Parce que ce double problème est trop difficile à résoudre simultanément, une procédure itérative à deux étapes est proposée.

Dans un premier temps, avec des temps de changement fixés, les taux de production optimaux sont déterminés à l'aide d'un modèle de programmation linéaire. Dans une seconde étape, avec des taux de production fixes, les temps de changement redondants sont éliminés, et de nouveaux instants sont ajoutés à l'aide de règles. La procédure itère entre les deux étapes jusqu'à ce qu'il n'y ait plus d'amélioration significative (voir [11, 64, 65, 67]).

Une extension permettant de traiter la demande discrète dans le cas où l'on conserve le modèle en temps continu a aussi été développée [27, 67].

Détermination de politiques de contrôle

Période : 1997–1999.
Résultats : une conférence avec actes [25].

Dans le cas particulier où la demande est constante au cours du temps (sur un intervalle de temps réduit par exemple), il peut être intéressant de déterminer, partant d'un état des stocks quelconque, comment se ramener à un état stationnaire, *i.e.*, déterminer une **politique de contrôle**. De tels résultats sont particulièrement utiles pour un responsable de production dans le cadre de l'**aide à la décision**. Pour le cas simple, réduit à une seule machine, une preuve d'optimalité de la politique développée est donnée [25]. Dès que plusieurs machines sont mises en jeu, des conditions particulières sont indispensables pour que cette politique reste optimale. Les précédents travaux ont permis de trouver des contre-exemples dans le cas général [67]. L'utilisation des travaux de la planification en temps continu a permis de valider les preuves et les contre-exemples.

Planification réactive

Période : 2000–2003.
Résultats : un article publié en revue [9],
deux conférences avec actes [16, 20] et
deux conférences sans actes [43, 45].

Dans le cadre de la thèse Cifre de Yann Le Quéré au sein de l'établissement industriel de la maintenance du matériel ferroviaire de la SNCF, l'étude s'intéresse à proposer une méthode consistant à mesurer l'impact d'une structure de décision et des activités décisionnelles sur un ordonnancement réactif. Un modèle permettant d'intégrer les aléas possibles sur les tâches est développé. Il permet de prendre en compte non-seulement ces aléas [20, 43, 45], mais aussi les délais de communication entre les différents centres de décision [9]. Une approche par réseaux de pétri [16] est associée à des techniques de propagation de contraintes.

7.2 Ordonnancement

Minimisation du nombre de jobs en retard sur une machine

Période : 1997–1999.
Résultats : un article accepté [5],
deux conférences avec actes [27, 29],
deux conférences sans actes [48, 49] et
un séminaire [66].

Sur l'ensemble des problèmes d'ordonnancement, les problèmes à une machine ont été les premiers étudiés, pour leur apparente simplicité. Malheureusement, la plupart des problèmes généraux d'ordonnancement à une machine

sont $\mathcal{NP}$-complets. Tout au long de cette étude, c'est le problème général d'ordonnancement à une machine dans lequel l'objectif est de minimiser le nombre de tâches en retard ou le nombre de tâches pondérées en retard, qui sera étudié [67]. Pour ce problème, l'objectif était de proposer une méthode de résolution exacte. A travers une nouvelle notion de séquence maître [67] des formulations mathématiques ont été proposées [27] et ont permis de résoudre efficacement le problème en dérivant des bornes inférieures par relaxation Lagrangienne [49]. Un nouveau cas particulier polynomial [27] a été détecté et peut être résolu optimalement en $\mathcal{O}(n \log n)$. De ces résultats, une méthode arborescente efficace a permis d'obtenir les meilleurs résultats au moment de leur publication [5, 29, 48].

Minimisation du nombre pondéré de jobs en retard sur une machine

Période : 1997–2002.
Résultats : deux articles publiés en revue [8, 10],
participation à une école d'hiver [24],
deux conférences avec actes [23, 26],
deux conférences sans actes [47, 50]
trois séminaires [59, 60, 63] et
un article en re-écriture [35].

L'étude du problème précédent apporte de nombreux éléments de réponse pour le problème de la minimisation du nombre pondéré de tâches en retard pour l'ordonnancement à une machine. Pour la plupart, les méthodes utilisées dans le cas non-pondéré ont été étendues au cas pondéré [10, 26, 35, 50, 59]. L'ajout de poids à chacun des jobs permet même d'améliorer certaines techniques et notamment la résolution par relaxation Lagrangienne [10, 60, 63]. Les algorithmes génétiques [8, 23, 24, 26] ont aussi permis à résoudre un certain nombre d'instances qui résistaient aux autres approches.

Minimisation du retard pondéré total sur une machine

Période : 2002–2003.
Résultats : une conférence sans actes [42].

En continuant à étudier les problèmes d'ordonnancement à une machine, on s'intéresse désormais à la minimisation du retard pondéré total. Dans le cas général (avec des dates de disponibilité des tâches différentes), le problème est extrêmement complexe, puisque toutes les réductions simples conduisent à des sous problèmes qui sont toujours $\mathcal{NP}$-difficiles au sens fort. Une approche par métaheuristiques donne des résultats parfois compétitifs par rapport aux meilleurs résultats connus [42].

Minimisation du nombre pondéré de jobs en retard sur machines parallèles

Période : 1999–2003.
Résultats : trois conférences avec actes [19, 21, 22],
une conférence sans actes [46],
deux séminaires [61, 62] et
un article en re-écriture [34].

Le critère de la minimisation du nombre pondéré de jobs en retard est maintenant étudié sur un environnement à machines parallèles. Ce problème devient alors beaucoup plus compliqué que celui à une machine. Les réductions à des sous problèmes voisins restent difficiles. On ne peut donc pas calculer facilement des bornes sur le problème. Plusieurs solutions approchées sont données par des heuristiques et des métaheuristiques [46, 61, 62]. Une approche basée sur la méthode taboue [22] dans laquelle plusieurs raffinements sont proposés donne des résultats très compétitifs. Une exploration plus large des métaheuristiques et des heuristiques de ce problème [21] permet de donner de très bons résultats, jusqu'à 100 jobs et 6 machines [34]. Une relaxation Lagrangienne (celle du problème à une machine) est étendue au cas machines parallèles et semble particulièrement bien s'adapter [19] pour les problèmes de taille raisonnable.

Construction d'ordonnancements robustes

Période : 2001–2003.
Résultats : un article accepté en revue [7],
deux conférences avec actes [14, 18],
une conférence sans actes [39] et
un séminaire [57] et
un article soumis [33].

Dans l'industrie automobile, la plupart des sous-traitants se sont installés à proximité des usines d'assemblage. Ils délivrent les pièces prêtes à monter plusieurs fois par jour selon les directives de l'usine d'assemblage (basées sur la *car sequence*). Malheureusement, il arrive que les sous-traitants ne puissent livrer à temps. On se propose alors de calculer un ordonnancement des pièces à assembler sachant que les dates de début au plus tôt sont sujettes à modification [18]. L'objectif ici est un indicateur de performance, la minimisation du nombre pondéré de tâches en retard. Le problème étudié est $1|r_j|\sum w_j U_j$ ou les r_j peuvent varier. Une approche par algorithme génétique [7] est proposée. A chaque itération du GA, on évalue une série de fonctions dont les r_j ont été modifiés aléatoirement. Le résultat est un ordonnancement plus robuste qui peut absorber des variations des r_j. Ce type d'algorithmes génétiques modifiés permet de proposer des solutions plus générales pour des problèmes où l'approche statistique n'est pas envisageable [33, 57]. Durant la thèse de Yann Le Quéré, une problématique d'ordonnancement à contraintes de ressource ou les durées des tâches sont variables a pu être résolue [14, 39].

Ordonnancement avec délais de communication

Période : 1994–1995.
Résultats : un rapport de DEA [68].

Le problème étudié est habituellement dénoté UET-UCT. Le but du stage de DEA [68] était de donner des éléments de réponse (en terme d'optimisation) pour le problème considéré. Dans un premier temps, le cas général a été présenté avec une modélisation par programmation linéaire en nombres entiers. Des bornes inférieures et supérieures ont été établies. Deux cas particuliers sont aussi présentés : - le cas réduit à deux processeurs et - le cas où le graphe des précédences est une arborescence. Par la suite, plusieurs algorithmes ont été établis et implémentés en langage ADA pour résoudre le problème par heuristiques.

7.3 Tournées de véhicules

Optimisation multi-objectif en tournées de véhicules sur arcs

Période : 2001–2003.
Résultats : un article accepté en revue [6],
une conférence avec actes [17],
une conférence sans actes [40] et
un séminaire [54].

La collecte des déchets en milieu urbain est une des nombreuses applications du *capacitated arc routing problem*. Une approche multi-objectif est proposée pour à la fois réduire la durée totale des tournées mais aussi équilibrer l'ensemble des tournées [40, 54]. Ces travaux permettent de proposer des résultats sur les instances classiques [17] et ont prouvé leur intérêt [6]. Les problèmes multi-objectifs sont bien souvent difficiles à résoudre et les métaheuristiques peuvent apporter une réponse à ces problèmes [4].

Métaheuristiques avancées pour les tournées de véhicules sur arcs

Période : 2003–2004.
Résultats : une conférence avec actes [13] et
deux séminaires [52, 53].

Le problème de tournées de véhicules sur arcs dans sa forme académique est déjà largement étudié [52]. Pourtant les derniers développements des algorithmes à base de population ont permis de trouver encore des améliorations [53]. Les derniers résultats [13] sont aujourd'hui les plus performants. Ils sont même meilleurs que les résultats précédemment publiés sur un ensemble de données tests de la littérature à la fois en terme d'écart à la meilleure solution et en temps de calcul. Dans ce cas précis, sur la base d'un algorithme génétique, une technique de gestion fine de la population a été rajoutée dans l'algorithme et permet de gérer une population plus petite et de meilleure qualité.

Robustesse en tournées de véhicules

Période : 2001–2003.
Résultats : une conférence avec actes [12].

Les problèmes de tournées de véhicules, largement étudiés dans la communauté scientifique, sont maintenant biens résolus par de nombreuses techniques. Pourtant, toutes ces méthodes se basent sur une hypothèse très forte : la connaissance *à priori* exacte des données sur lesquelles s'appuie la problématique. Toutes ces méthodes peuvent alors voir leurs performances s'effondrer si les données varient, ne serait-ce qu'infimement. Dans cette étude, on s'intéresse à résoudre le problème permettant de proposer des solutions qui resteraient satisfaisantes pour le décideur même après variation. Encore une fois, ce sont les métaheuristiques qui permettent de résoudre le problème en y incluant un cycle de simulation [12].

7.4 Autres approches ou problématiques

Optimisation en conception CAO

Période : 2003–2004.
Résultats : une conférence sans actes [38] et
un article soumis [30].

Encore une fois, les métaheuristiques peuvent être d'un grand secours, même dans le domaine de l'optimisation continue. Dans le cas de la conception CAO en automobile, il est indispensable de pouvoir reconstituer des courbes à partir de séries de points mesurés sur un prototype. En définissant une fonction objectif particulière et adaptée au problème considéré, on cherche à optimiser un certain nombre de paramètres constituant les courbes modélisant la forme du prototype. Le nombre de paramètres est important et nécessite un réglage minutieux difficile à trouver manuellement. Un algorithme génétique "continu" permet de faire une recherche efficace de ces paramètres et de trouver des configurations inattendues et pertinentes. Ces résultats ont été présentés lors d'une conférence [38] et ont donné lieu à la soumission d'un article en revue [30].

Optimisation multiobjectif

Période : 2003–2004.
Résultats : un livre édité [4].

La programmation mathématique multiobjectif est un vaste domaine couvrant entre autre l'optimisation d'un problème donné en tenant compte de plusieurs critères à la fois. Il ne s'agit pas de classer les critères entre eux, mais bien au contraire de proposer des solutions de compromis lorsque les critères sont antagonistes.

Avec l'aide du groupe français PM20, EU/ME a organisé en France deux jours sur l'optimisation multiobjectif par métaheuristiques. Cette manifestation a réuni

60 chercheurs du monde entier. Lors de celle-ci, quatre sessions plénières ont permis aux jeunes chercheurs de trouver ou retrouver un certain nombre d'informations indispensables pour leurs recherches. Une sélection d'articles soumis à un processus de relecture de type revue internationale a été publié dans un "Lecture Notes in Economics and Mathematical Systems" [4].

Métaheuristiques

Période : 1999–2004.
Résultats : une conférence avec actes [15],
deux conférences sans actes [41, 44] et
trois séminaires [55, 56, 58],
un article soumis [31] et
un chapitre en préparation [32].

Ce thème transversal est étudié à travers des problèmes d'optimisation et en particulier des problèmes d'ordonnancement [41, 44]. La présentation de synthèses importantes lors de séminaires invités [55, 56, 58] permet de mettre en évidence l'information sur ces méthodes aujourd'hui incontournables pour la résolution des problèmes d'optimisation. En s'appuyant sur l'expérience acquise par l'étude de nombreux problèmes d'optimisation, on est à même aujourd'hui de pouvoir proposer de nouvelles méthodes d'optimisation qui permettent de palier à certaines des lacunes des métaheuristiques "classiques". en particulier, la gestion de la population est un point critique et ce dernier est partiellement résolu dans une nouvelle méthode hybride appelée GA|PM [31]. De plus, pour des problèmes de robustesse ou de flexibilité, les métaheuristiques offrent une possibilité rapide et simple de résolution des problèmes. Un chapitre dans un ouvrage collectif est en préparation sur ce thème [32].

Un livre de référence sur les métaheuristiques, cette fois-ci en Anglais, est en préparation. Cet ouvrage va regrouper à la fois les développements théoriques nécessaires à la compréhension des métaheuristiques et des programmes permettant au lecteur de comprendre et d'appliquer rapidement les méthodes sur ses propres problèmes. Les étudiants de troisième cycle et post-doctorants sont le public de ce livre qui sera co-écrit avec C. Prins et K. Sörensen.

Programmation linéaire

Période : 1996–2003.
Résultats : deux livres [2, 3].

La programmation linéaire est une branche de l'optimisation permettant de résoudre de nombreux problèmes économiques et industriels. L'apparition de logiciels puissants permet, aujourd'hui, de mettre cet outil à la disposition d'un large public.

Ce livre, le premier du genre en français [3], met l'accent sur la démarche de modélisation, appliquée ensuite à la résolution de 65 problèmes concrets regroupés par discipline en onze chapitres. À côté de problèmes industriels classiques (comme le transport ou l'ordonnancement), vous y trouverez des champs d'application moins connus ou plus récents, comme l'économie, la gestion du personnel ou le secteur public.

Chaque chapitre est consacré à un domaine d'applications, et comporte six problèmes concrets et suffisamment compliqués pour ne pas être traitables à la main. À partir du descriptif de chaque cas, vous apprendrez, pas à pas, à construire un modèle de programmation linéaire que vous pourrez résoudre et modifier grâce au puissant logiciel Xpress fourni sur le CD-Rom d'accompagnement. Des compléments à la fin de chaque chapitre et une bibliographie très complète permettront aux curieux d'en savoir plus. Loin d'être un recueil d'exercices corrigés, ce livre est en fait auto-contenu grâce à deux chapitres de cours sur les bases théoriques de la programmation linéaire et à un chapitre sur les logiciels du marché et les ressources web. Une version en Anglais de ce livre existe [2].

Algorithmes de graphes

Période : 2003–2004.
Résultats : un livre [1].

Les graphes et leurs algorithmes sont des outils mathématiques utilisés pour modéliser et résoudre des problèmes complexes dans des domaines aussi variés que l'optimisation (production industrielle, aide à la décision), la conception de réseaux (électriques, routiers, télécoms) ou la modélisation de systèmes évolutifs (économie, automatique). Cet ouvrage va permettre à tous de mettre en œuvre des algorithmes plus ou moins complexes pour résoudre de tels problèmes. Ce livre est une version complètement refondue de l'ouvrage "Algorithmes de graphes avec programmes en Pascal", de Christian Prins, publié aux éditions Eyrolles en 1994, épuisé et non-réédité depuis plusieurs années.

L'objet de ce livre est de rendre ces techniques fondées sur la théorie des graphes accessibles à des non-mathématiciens et de montrer comment les mettre en œuvre dans des cas concrets. Une première partie introduit les notions d'optimisation combinatoire et de complexité des algorithmes, et donne un large panorama des méthodes existantes, des plus classiques aux plus récentes (recuit simulé, tabou). La seconde partie traite des différents problèmes de graphes : chemins optimaux, flots, tournées, coloration, etc. Les algorithmes, soigneusement justifiés, sont accompagnés de programmes en pseudo-code et en langage Delphi (Pascal objet), ainsi que d'exemples d'applications commentées [1].

Un nouveau livre sur ce même thème est en préparation mais cette fois-ci en utilisant le langage C++ qui offre une plus grande souplesse dans la manipulation des structures de données. Cet ouvrage sera écrit avec un nouvel auteur, spécialiste en C++, Philippe Thomin.

Ontologies pour la recherche opérationnelle

Période : 2003–2004.
Résultats : une conférence sans actes [37].

Définie comme "la spéculation de l'être en tant qu'être", l'ontologie peut être plus simplement décrite comme une représentation formelle de parties importantes de notre monde. Avant tout considéré comme un concept philosophique, les ontologies sont utilisées de manière croissante comme des outils pour partager le savoir et l'information.

L'intérêt croissant pour le langage XML (eXtensible Markup Language) pour décrire l'information d'un très grand nombre de domaines variés (mathématique, chimie, physique, informatique, etc.) n'a fait que développer l'intérêt pour les ontologies. En recherche opérationnelle, de nombreux jeux de données (ou instances) de problèmes standards sont couramment utilisés pour tester la performance des algorithmes d'optimisation créés par les chercheurs. Une des plus grandes collection de ces problèmes est la OR-Library. Mais cette collection n'est pas unique. Ces instances sont des sources d'informations très fortement structurées qui sont très souvent échangées entre les chercheurs. A notre avis, l'échange de ces données pourrait être grandement facilité en appliquant certains des résultats de la littérature sur les ontologies.

Le but de ce projet est de développer et d'implémenter une nouvelle version de la OR-Library qui aurait les caractéristiques suivantes :

- Les jeux de données sont en format XML. Pour chaque problème, un schéma XML du jeu de données correspondant est fourni. Les instances peuvent être facilement validées et on y accède en utilisant des interpréteurs XML. Ces interpréteurs XML sont disponibles partout et peuvent facilement être inclus dans les programmes des algorithmes.
- Les instances peuvent être stockées dans la bibliothèque de façon décentralisée. Chaque chercheur qui crée un nouveau jeu de données pour un problème spécifique peut charger/stocker ces données. Les jeux de données ainsi chargés peuvent être validés en les comparant au schéma XML du problème.
- De nouveaux problèmes peuvent être proposés en soumettant un fichier de schéma XML correspondant au nouveau problème. Pour une nouvelle soumission, une demande de commentaires (RFC : request for comments) est envoyé à la communauté. Quand le nouveau problème est accepté par la communauté des chercheurs, le schéma XML et les données peuvent être ajoutées à la bibliothèque.
- Les solutions et les valeurs des fonctions objectifs pour chaque instance sont inclus dans la bibliothèque. Saisir ces données se fait aussi de manière décentralisée. Avec peu d'efforts, les chercheurs peuvent partager les résultats des développements récents avec l'ensemble de la communauté. Les comparaisons entre les différentes techniques de résolution est ainsi facilité.

En atteignant nos objectifs, il sera facile de convaincre la communauté de ve-

nir ″alimenter″ notre bibliothèque et d'éviter l'éparpillement des collections de données comme on a pu le constater ces cinq dernières années. Une présentation de ce nouveau domaine de recherche a été proposée lors du 3^{e} EU/ME joint workshop, [37].

8 Liste des publications

**Attention les auteurs sont toujours classés par ordre alphabétique sauf si l'implication d'un ou plusieurs d'entre eux est plus forte.*

Livres (auteur)

[1] P. Lacomme, C. Prins, and M. Sevaux. *Algorithmes de graphes*. Eyrolles, 2003. ISBN 2-212-11385-4, 425 pages (in French).

[2] C. Guéret, C. Prins, and M. Sevaux. *Applications of optimisation with Xpress-MP*. Dash optimization, 2002. ISBN 0-9543503-0-8, Translated by S. Heipke.

[3] C. Guéret, C. Prins, and M. Sevaux. *Programmation linéaire*. Eyrolles, 2000. ISBN 2-212-09202-4, 365 pages (in French).

Livres (éditeur)

[4] X. Gandibleux, M. Sevaux, K. Sörensen, and V. T'Kindt, editors. *Metaheuristics for multi-objective optimisation*, volume 535 of *LNEMS*. Springer, 2004. ISBN 3-540-20637-X, 230 pages.

Revues Internationales

[5] S. Dauzère-Pérès and M. Sevaux. An exact method to minimize the number of tardy jobs in single machine scheduling. *Journal of Scheduling*, 2004. In Press.

[6] P. Lacomme, C. Prins, and M. Sevaux. A genetic algorithm for a bi-objective capacitated arc routing problem. *Computers and Operations Research*, 2004. In Press.

[7] M. Sevaux and K. Sörensen. A genetic algorithm for robust schedules in a just-in-time environment. *4OR – Quaterly journal of the Belgian, French and Italian Operations Research Societies*, 2(2) :129–147, 2004.

[8] M. Sevaux and S. Dauzère-Pérès. Genetic algorithms to minimize the weighted number of late jobs on a single machine. *European Journal of Operational Research*, 151(2) :296– 306, 2003.

[9] Y. Le Quéré, M. Sevaux, C. Tahon, and D. Trentesaux. Reactive scheduling of complex system maintenance in a cooperative environment with communication times. *IEEE-SMC*, 33(2) :225–234, 2003.

[10] S. Dauzère-Pérès and M. Sevaux. Using lagrangean relaxation to minimize the weighted number of late jobs. *Naval Research Logistics*, 50(3) :273–288, 2003.

[11] S. Dauzère-Pérès, S.B. Gershwin, and M. Sevaux. Models and solving procedures for continuous-time production planning. *IIE Transactions*, 32(2) :93–103, 2000.

Conférences avec actes

[12] K. Sörensen and M. Sevaux. Robust and flexible vehicle routing in practical situations. In *Proceedings of 5th triennial symposium on transportation analysis, TRISTAN V*, Le Gosier, Guadeloupe, France, 13-18 June 2004. To appear.

[13] C. Prins, M. Sevaux, and K. Sörensen. A genetic algorithm with population management (GA|PM) for the carp. In *Proceedings of 5th trienal symposium on transportation analysis, TRISTAN V*, Le Gosier, Guadeloupe, France, 13-18 June 2004. To appear.

[14] G. Fleury, Lacomme P, and M. Sevaux. Stochastic maintenance scheduling problem. In *Proceedings of ninth international workshop on project management and scheduling, PMS 2004*, pages 405–409, Nancy, France, 26-28 April 2004.

[15] K. Sörensen and M. Sevaux. GA|PM : genetic algorithms with population management for permutation problems. In *Proceedings of 5th Metaheuristics International Conference, MIC 2003*, pages 69/1–7, Kyoto, Japan, 25-28 August 2003.

[16] Y. Le Quéré, M. Sevaux, C. Tahon, and D. Trentesaux. Modèle de coopération d'un processus de ré-ordonnancement distribué. In *Proceedings of the Automatic control doctoral days (Journées doctorales d'automatique)*, pages 401–406, LAMIH, Valenciennes, France, 25-27 June 2003. ISBN 2-905725-53-2.

[17] P. Lacomme, C. Prins, and M. Sevaux. Mutliple objective capacitated arc routing problem. In *Proceedings of 2nd International Conference on Evolutionary Multi-Criterion Optimization, EMO'2003*, pages 550–564, Faro, Portugal, 8-11 April 2003. LNCS 2632.

[18] M. Sevaux and K. Sörensen. Genetic algorithm for robust schedules. In *Proceedings of 8th International Workshop on Project Management and Scheduling, PMS'2002*, Valencia, Spain, April, 3-5 2002.

[19] S. Dauzère-Pérès and M. Sevaux. Lagrangean relaxation for minimizing the weithed number of late jobs on parallel machines. In *Proceedings of 8th International Workshop on Project Management and Scheduling, PMS'2002*, Valencia, Spain, April, 3-5 2002.

[20] Y. Le Quéré, M. Sevaux, D. Trentesaux, and C. Tahon. Planification réactive des opérations de maintien et d'actualisation réglementaire et technologique des systèmes complexes. In *Proceedings of the International Conference on Computer aided Maintenance*, Rabat, Morocco, November, 7-8 2001.

[21] M. Sevaux and P. Thomin. Heuristics and metaheuristics for parallel machine scheduling : a computational evaluation. In *Proceedings of 4th Metaheuristics International Conference, MIC 2001*, pages 411–415, Porto, Portugal, July, 16-20 2001.

[22] M. Sevaux and P. Thomin. Recherche taboue améliorée pour l'ordonnancement sur machines parallèles. In *Actes de la 3ième conférence Francophone de MOdélisation et de SIMulation, MOSIM'01*, pages 829–833, Troyes, France, April, 25-27 2001.

[23] M. Sevaux and S. Dauzère-Pérès. A genetic algorithm to minimize the weighted number of late jobs on a single machine. In *Proceedings of 7th International Workshop on Project Management and Scheduling, PMS'2000*, pages 242–244, Osnabrük, Germany, April, 17-19 2000.

[24] M. Sevaux and S. Dauzère-Pérès. Building a genetic algorithm for a single machine scheduling problem. In *Proceedings of the 18th EURO Winter Institute, ESWI XVIII*, Lac Noir, Switzerland, March, 4-18 2000.

[25] S. Dauzère-Pérès and M. Sevaux. Using lagrangean relaxation to minimize the (weighted) number of late jobs. In *National contribution for the 15th triennal conference, IFORS'99*, Beijin, P.R. of China, August, 16-20 1999.

[26] S. Dauzère-Pérès and M. Sevaux. Control policies for some continuous-time production scheduling problems. In *Proceedings of the International Conference on Industrial Engineering and Production Management, IEPM'99*.

[27] S. Dauzère-Pérès and M. Sevaux. Handling discrete demand in continuous-time production planning. In *Proceedings of the 9th symposium of the International Federation of Automatic Control on Information Control in Manufacturing, INCOM'98*, pages 467–472.

[28] S. Dauzère-Pérès and M. Sevaux. Various mathematical programming formulations for a general one machine sequencing problem. In *Proceedings of the 4ème Journées Nationales sur la Résolution Pratique des Problèmes NP-Complets, JNPC'98*, pages 63–68.

[29] S. Dauzère-Pérès and M. Sevaux. A branch and bound method to minimize the number of late jobs on a single machine. In *Proceedings of the 6th International Workshop on Project Management and Scheduling, PMS'98*, pages 193–196.

Articles en cours de révision, soumis ou en re- écriture

[30] M. Sevaux and Y. Mineur. Continuous optimisation by GA for a curve fitting problem in autmobile industry. Technical report, University of Valenciennes, CNRS UMR 8530, LAMIH/SP, 2004. Submitted.

[31] K. Sörensen and M. Sevaux. GA|PM : genetic algorithms with population management. Technical report, University of Valenciennes, CNRS UMR 8530, LAMIH/SP, 2004. Submitted.

[32] M. Sevaux, K. Sörensen, and Y. Le Quéré. *Flexibilité et robustesse en ordonnancement*, chapitre Métaheuristiques pour la planification et l'ordonnancement robuste, pages 91–110. Hermes, 2004. Under revision.

[33] M. Sevaux and Y. Le Quéré. Solving a robust maintenance scheduling problem at the F rench railway company. Technical Report LAMIH/SP-2003-3, University of Valenciennes, CNRS UMR 8530, LAMIH/SP, April 2003. Under revision.

[34] M. Sevaux and P. Thomin. Heuristics and metaheursitics for a parallel machine scheduling problem : a computational evaluation. Technical Report LAMIH/SP-2001-2, University of Valenciennes, CNRS UMR 8530, LAMIH/SP, November 2001. Under revision.

[35] S. Dauzère-Pérès and M. Sevaux. An efficient formulation for minimizing the number of late jobs in single-machine scheduling. Technical Report 98/9/AUTO, Ecole des Mines de Nantes, July 1998. Under revision.

Conférences sans actes

[36] C. Prins, M. Sevaux, and K. Sörensen. Genetic algorithms with population management : an application to arc routing. In *Optimization*, Lisboa, Portugal, 27-27 July 2004.

[37] K. Ven, K. Sörensen, M. Sevaux and J. Vereist. Lib-OR – Library of OR data sets. In *3rd Joint EU/ME workshop with the university of Antwerp on Real-life application of metaheuritics*, Antwerp, Belgium, Dec. 18-19 2003.

[38] Y. Mineur and M. Sevaux. Curve fitting for styling application by genetic algorithm. In *3rd Joint EU/ME workshop with the university of Antwerp on Real-life application of metaheuritics*, Antwerp, Belgium, Dec. 18-19 2003.

[39] Y. Le Quéré and M. Sevaux. Approche robuste pour un problème à contraintes de ressources. In *5ième conférence nationale de la société française de recherche opérationnelle, ROADEF'2003*, Feb. 26-28 2003.

[40] P. Lacomme, C. Prins, and M. Sevaux. Multiobjective capacitated arc routing problem. In *2nd Joint EU/ME workshop with the french PM2O group (MOMH Workshop) on multi-objective metaheuristics*, Paris, France, Nov. 4-5 2002.

[41] M. Sevaux. Les méthodes de recherche à population. In *Groupe de Recherche en Productique - Organisation et Gestion de Production*, Oct. 24-25 2002.

[42] M. Sevaux and P. Thomin. Scatter search and ga : a one machine scheduling problem comparison. In *The sixteenth triennial conference of international federation of operational research societies, IFORS'2002*, Edinburgh, UK, Jul. 6-13 2002.

[43] Y. Le Quéré, M. Sevaux, D. Trentesaux, and C. Tahon. Résolution d'un problème industriel de maintenance des TGV à la SNCF. In *4ième conférence nationale de la société française de recherche opérationnelle, ROADEF'2002*, Paris, France, Feb. 20-22 2002.

[44] M. Sevaux. Les méthodes de recherche à voisinage. In *Groupe de Recherche en Productique - Organisation et Gestion de Production*, Nov. 8-9 2001.

[45] Y. Le Quéré, D. Trentesaux, M. Sevaux, and C. Tahon. Gestion réactive des opérations de maintien et d'actualisation réglementaire et technologiques des systèmes complexes. In *Groupe de Recherche en Productique – Organisation et Gesion de Production*, Montpellier, France, Mar. 29-30 2001.

[46] M. Sevaux and P. Thomin. Efficient heuristic and tabu search for parallel machine scheduling. In *ORBEL Conference*, Antwerp, Belgium, Jan. 29-30 2001.

[47] M. Sevaux and S. Dauzère-Pérès. Un algorithme génétique pour minimiser le nombre pondéré de jobs en retard sur une machine. In *3ième conférence nationale de la société française de recherche opérationnelle, ROADEF'2000*, Nantes, France, Jan. 26-28 2000.

[48] S. Dauzère-Pérès and M. Sevaux. Une méthode arborescente améliorée pour la minimisation du nombre de jobs en retard. In *2ième conférence nationale de la société française de recherche opérationnelle, ROADEF'2000*, Autrans, France, Jan. 13-15 2000.

[49] S. Dauzère-Pérès and M. Sevaux. On minimizing late jobs in single machine scheduling. In *INFORMS/Cors'98*, Montreal, Canada, Apr. 26-29 1998.

[50] S. Dauzère-Pérès and M. Sevaux. Using lagrangean relaxation to minimize the weighted number of late jobs in one machine sequencing with release dates. In *Symposium on Combinatorial Optimization, CO'98*, Brussels, Belgium, Apr. 15-17 1998.

[51] S. Dauzère-Pérès, S.B. Gershwin, and M. Sevaux. Modeling and solving procedures of continuous-time production planning problems. In *8th Annual Meeting of the Production and Operations Management Society, POM-97*, Miami Beach, Florida, USA, Apr. 12-15 1997.

Séminaires

[52] M. Sevaux. A genetic algorithm with population management GA|PM for the CARP. Invited seminar, The Hong-Kong Polytechnic University, Department of Logistics, 14 May 2004.

[53] M. Sevaux. Production de biens et services : application en ordonnancement et transport. Séminaire invité, GDR MACS/STP groupe ORT (in French), 13 February 2004.

[54] M. Sevaux. Advances in mutliple objective capacitated arc routing problem. Invited seminar, University of Valenciennes, French research group on multiple objective mathematical programming, 15 May 2003.

[55] M. Sevaux. Population metaheuristics. Invited seminar, The Hong-Kong Polytechnic University, Department of Management, 14 March 2003.

[56] M. Sevaux. Metaheuristics : a quick overview. Invited seminar, University of Antwerp, Faculty of Applied Economic Sciences, 7 February 2003.

[57] M. Sevaux. Métaheuristiques pour la résolution robuste de probl èmes d'optimisation combinatoire. Séminaire invité, Mathématiques Appliquées de Bordeaux (in French), 14 November 2002.

[58] M. Sevaux. Les méthodes de recherche à voisinage. Séminaire invité, Faculté Polytechnique de Mons, groupe Image (in French), 6 March 2002.

[59] M. Sevaux and D. Rivreau. Single machine scheduling : minimizing the [weighted] number of late jobs, a review. Séminaire invité, Groupe de Recherche en Ordonnancement Théorique et Appliqué, GOThA (in French), 28 September 2001.

[60] M. Sevaux. Single machine scheduling : Minimizing the [weighted] number of late jobs. Invited seminar, Université Libre de Bruxelles - Service des Mathématiques de la Gestion, 10 May 2001.

[61] M. Sevaux and Ph. Thomin. Parallel machine scheduling : a metaheuristic computational evaluation. Séminaire invité, Institut de Recherche en Communication et Cybernétique de Nantes, IRCCyN (in French), 19 April 2001.

[62] M. Sevaux and P. Thomin. Parallel machine scheduling : a (meta)heuristic computational evaluation. Séminaire invité, Groupe MOST (in French), 5 April 2001.

[63] M. Sevaux. Ordonnancement à une machine par relaxation lagrangienne. Séminaire invité, Groupe de Recherche Bermudes (in French), 11 June 1999.

[64] S. Dauzère-Pérès and M. Sevaux. Modeling and solving procedures of continuous-time production planning problems. Invited seminar, Massachusetts Institute of Technology, Cambridge, MA, USA, 23 April 1997.

[65] S. Dauzère-Pérès and M. Sevaux. Modèles et procédures de résolution pour la planification en temps continu. Séminaire invité, Groupe de Recherche en Ordonnancement Théorique et Appliqué, GOThA (in French), 19 December 1997.

[66] S. Dauzère-Pérès and M. Sevaux. Minimisation du nombre de jobs dans le problème d'ordonnancement à une machine. Séminaire invité, Groupe de Recherche en Ordonnancement Théorique et Appliqué, GOThA (in French), 19 December 1997.

Thèse de doctorat

[67] M. Sevaux. *Etude de deux problèmes d'optimisation en planification et ordonnancement (On Two Optimization Problems in Production Planning and Scheduling)*. Thèse de doctorat, Université Pierre et Marie Curie (Jussieu, Paris VI), 1998. Ecole des Mines de Nantes.

Rapport de DEA

[68] M. Sevaux. Ordonnancement avec délais de communication. Rapport de DEA, Université Pierre et Marie Curie (Jussieu, Paris VI), 1995.

Deuxième partie

Synthèse scientifique

1 Introduction générale

Présenter les différents problèmes abordés aux cours des années précédentes n'aurait qu'un intérêt limité pour une telle synthèse. Dans ce document, nous avons volontairement choisi de présenter les métaheuristiques utilisés pour la résolution de nombreux problèmes d'optimisation. Ces outils seront mis en avant et rassemblés par catégories. Mais avant tout, cette synthèse a pour modeste ambition de proposer un panorama général des plus importantes métaheuristiques d'aujourd'hui qui permettent de résoudre un problème d'optimisation combinatoire. La conclusion proposera ce que pourraient être, selon nous, les caractéristiques d'une bonne métaheuristique.

1.1 Pourquoi les métaheuristiques ?

Depuis toujours, les chercheurs ont tenté de résoudre les problèmes $\mathcal{NP}$-difficiles le plus efficacement possible. Pendant longtemps, la recherche s'est orientée vers la proposition d'algorithmes exacts pour des cas particuliers polynomiaux. Ensuite, l'apparition des heuristiques a permis de trouver des solutions en général de bonne qualité pour résoudre les problèmes. En même temps, les méthodes de type "séparation et évaluation" ont aidé à résoudre des problèmes de manière optimale, mais souvent pour des instances de petite taille.

Lorsque les premières métaheuristiques apparaissent, beaucoup de chercheurs se sont lancés dans l'utilisation de ces méthodes. Cela a conduit à une avancée importante pour la résolution pratique de nombreux problèmes. Cela a aussi créé un engouement pour le développement même de ces méthodes. Il existe des équipes entières qui ne travaillent qu'au développement des métaheuristiques.

Il faut aussi reconnaître que c'est un formidable outil pour la résolution efficace des problèmes posés.

1.2 Intensification et diversification

Toutes les métaheuristiques s'appuient sur un équilibre entre l'intensification de la recherche et la diversification de celle-ci. D'un côté, l'intensification permet de rechercher des solutions de plus grande qualité en s'appuyant sur les solutions déjà trouvées et de l'autre, la diversification met en place des stratégies qui permettent d'explorer un plus grand espace de solutions et d'échapper à des minima locaux.

Ne pas préserver cet équilibre conduit à une convergence trop rapide vers des minima locaux (manque de diversification) ou à une exploration trop longue (manque d'intensification). Cette vision est bien sûr simpliste mais malheureusement on l'observe trop souvent encore aujourd'hui. Il existe de nombreux articles

soumis à publication qui se trouvent rejetés par les arbitres pour ces raisons.

Dans les sections qui vont suivre, nous avons essayé, de présenter chaque métaheuristique de façon uniforme. D'abord, nous rappelons, quand c'est possible, les papiers où la méthode a été introduite la première fois. Une description textuelle de l'algorithme est proposé avec ses particularités. Parfois, il existe des articles plus récents que les articles d'origine qui présentent soit une vue synthétique soit un tutorial détaillé et que nous citons. Ensuite, nous donnons une présentation algorithmique de chaque métaheuristique, avant de montrer quels sont les facteurs d'intensification et de diversification de chacune de ces méthodes.

1.3 Techniques de résolution pratique

Nous nous plaçons ici dans le contexte particulier d'un chercheur faisant face à un problème d'optimisation pour lequel les méthodes exactes ont atteint leurs limites. Nous introduisons alors un ensemble d'outils qui permettront de trouver des solutions et de les améliorer. Pour chacune des sections ci-dessous, un certain nombre de méthodes sont présentées. Il est impensable de déconnecter les méthodes des applications pour lesquelles elles ont été dessinées. Chacune des sections propose donc une partie "Applications" mettant en œuvre les méthodes proposées sur des problèmes que nous avons rencontrés pendant nos recherches. Dans cette partie application, nous présenterons à la fois quelques références importantes de la littérature et nos propres travaux.

Face à un problème d'optimisation, l'étape la plus importante est sans doute l'étape de la modélisation (c'est à dire passer d'un problème réel à une formulation mathématique de ce problème). C'est aussi l'étape la plus difficile à enseigner car la plupart du temps, c'est l'expérience qui permet de définir un bon modèle. Dans notre cas, une fois ce modèle établit (et la preuve que le problème appartient à la classe des problèmes $\mathcal{NP}$-difficiles obtenue), nous essayons de le transformer en un problème de programmation mathématique que nous allons tenter de résoudre en utilisant un logiciel commercial (pour des exemples de telles modélisations et résolutions, voir Guéret et al. [53, 54]). Utiliser la programmation mathématique permet aussi d'obtenir des bornes sur le problème et sans doute de mieux appréhender sa difficulté propre.

Une fois obtenues des bornes et éventuellement quelques solutions par heuristique , on peut commencer à mettre en œuvre des métaheuristiques. Pour améliorer des solutions initiales, on peut utiliser des techniques de recherche locale. Les méthodes de recherche locale, des plus simples aux plus compliquées sont un atout majeur et certaines d'entre elles sont décrites dans la section 2. On y trouvera les méthodes de descente, de recuit simulé, de recherche tabou, de recherche à voisinage variable, de GRASP, d'*iterated local search*, de *guided local search*.

Dans la section 3, les méthodes à base de population de solutions les plus simples sont détaillées. Elles présentent un avantage indéniable, mettre en place un parallélisme intrinsèque et explorer un espace de solutions très vaste. Les algorithmes génétiques et les colonies de fourmis sont présentés dans cette section. Pour diverses raisons, nous n'avons pas eu l'occasion de tester les colonies de fourmis, seule une application en ordonnancement utilisant des algorithmes génétiques sera détaillée.

Ces méthodes à population, dans leur version de base, ont souvent du mal à trouver des solutions compétitives avec d'autres techniques. La section 4 présente des améliorations indispensables à la réussite de telles approches. En ajoutant des techniques de recherche locale aux algorithmes génétiques, on obtient les algorithmes mémétiques. A côté, le scatter search, technique innovante, sera présenté et testé. Le dernier point concerne un nouvel algorithme (GA|PM) présentant des caractéristiques très intéressantes pour l'optimisation.

La section 5 présente des compléments intéressants. Il s'agit soit de travaux supplémentaires que nous avons mené sur les métaheuristiques, soit des utilisations des métaheuristiques pour d'autres propos. Des techniques de réglages automatiques de paramètres sont proposées, et des résolutions en robustesse, optimisation multiobjectif et optimisation continue sont présentées.

Dans la conclusion (section 6), nous mettons en avant, selon nous, les bonnes caractéristiques que devrait posséder une métaheuristique aujourd'hui et surtout quelques mises en garde importantes pour la conception de ces méthodes.

La section 7 expose les champs d'application qui nous intéressent particulièrement aujourd'hui et que nous souhaitons traiter dans un futur proche. En parallèle à ces applications, il nous apparaît indispensable de pouvoir proposer des thèmes de recherche sur le plus long terme. D'un côté, une collaboration industrielle pourrait déboucher rapidement sur une thèse dans le domaine de l'optimisation du trafic dans les réseaux de télécommunications. De plus, nous souhaiterions voir avancer un travail de grande envergure sur la proposition d'une vue unifiée des métaheuristiques à population.

2 Méthodes de recherche locale

Les méthodes de recherche locale ou métaheuristiques à base de voisinages s'appuient toutes sur un même principe. A partir d'une solution unique x_0, considérée comme point de départ (et calculée par exemple par une heuristique constructive), la recherche consiste à passer d'une solution à une solution voisine par déplacements successifs. L'ensemble des solutions que l'on peut atteindre à partir d'une solution x est appelé *voisinage* $\mathcal{N}(x)$ de cette solution. Déterminer une solution voisine de x dépend bien entendu du problème traité.

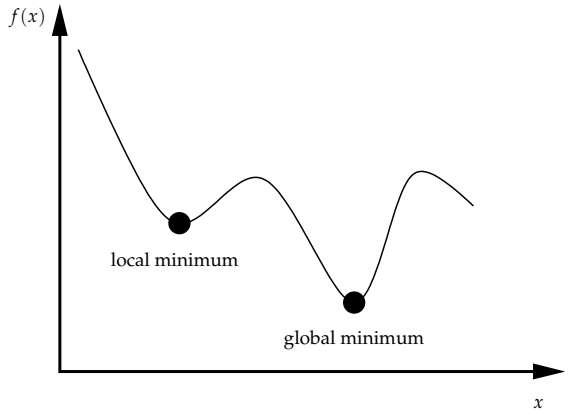


FIG. 2 – Minimum local et global

De manière générale, les opérateurs de recherche locale s'arrêtent quand une solution localement optimale est trouvée, c'est à dire quand il n'existe pas de meilleure solution dans le voisinage. Mais accepter uniquement ce type de solution n'est bien sûr pas satisfaisant. C'est le cas des méthodes de descente présentées ci-dessous (voir section 2.1)

Dans un cadre plus général, il serait alors intéressant de pouvoir s'échapper de ces minima locaux (voir Fig. 2). Il faut alors permettre à l'opérateur de recherche locale de faire des mouvements pour lesquels la nouvelle solution retenue sera de qualité moindre que la précédente. C'est le cas immédiat des méthodes de recuit simulé (section 2.2), de la recherche tabou (section 2.3) et de l'*iterated local search* (section 2.6). Dans le cas de la *VNS*, de *GRASP* et du *guided local search*, les techniques pour échapper aux minima locaux sont différentes et seront expliquées dans les sections 2.4, 2.5 et 2.7. Les méthodes de recuit simulé et tabou sont plus anciennes et sans doute plus populaires. L'attention qui leur sera portée sera alors un peu plus grande.

Certaines des méthodes détaillées ci-dessous ont fait l'objet de présentation en conférences ou séminaires. Citons par exemple [98, 100, 101] qui présentent à la fois des méthodes de descente, de recherche tabou et de recuit simulé. Bien sûr, ces méthodes sont rarement présentées sans contexte applicatif. La section 2.8 présentera pour un problème d'ordonnancement à machines parallèles l'application de quelques unes de ces méthodes. Pour une explication générale sur ces méthodes de recherche, on peut se référer à l'ouvrage ancien mais assez complet de Papadimitriou et Steiglitz [88].

2.1 Méthodes de descente

A partir d'une solution trouvée par heuristique par exemple, on peut très facilement implémenter des méthodes de descente. Ces méthodes s'articulent toutes

autour d'un principe simple. Partir d'une solution existante, chercher une solution dans le voisinage et accepter cette solution si elle améliore la solution courante.

L'algorithme 2.1 présente le squelette d'une méthode de descente simple. A partir d'une solution initiale x, on choisit une solution x' dans le voisinage $\mathcal{N}(x)$ de x. Si cette solution est meilleure que x, ($f(x') < f(x)$) alors on accepte cette solution comme nouvelle solution x et on recommence le processus jusqu'à ce qu'il n'y ait plus aucune solution améliorante dans le voisinage de x.

Algorithme 2.1 Simple descent

1: *initialise* : find an initial solution x
2: **repeat**
3: *neighbourhood search* : find a solution $x' \in \mathcal{N}(x)$
4: **if** $f(x') < f(x)$ **then**
5: $x' \leftarrow x$
6: **end if**
7: **until** $f(y) \geq f(x)$, $\forall y \in \mathcal{N}(x)$

Une version plus "agressive" de la méthode de descente est la méthode de plus grande descente. Au lieu de choisir une solution x' dans le voisinage de x, on choisit toujours la meilleure solution x' du voisinage de x. L'algorithme 2.2 donne une description de cette méthode.

Algorithme 2.2 Deepest descent

1: *initialise* : find an initial solution x
2: **repeat**
3: *neighbourhood search* : find a solution $x' \in \mathcal{N}(x)$ / $f(x') \leq f(x'')$, $\forall x'' \in \mathcal{N}(x)$
4: **if** $f(x') < f(x)$ **then**
5: $x' \leftarrow x$
6: **end if**
7: **until** $f(x') \geq f(x)$, $\forall x' \in \mathcal{N}(x)$

Ces deux méthodes sont évidemment sujettes à de nombreuses critiques. Elles se basent toutes les deux sur une amélioration progressive de la solution et donc resteront bloquées dans un minimum local dès qu'elles en rencontreront un. Il existe de manière évidente une absence de diversification. L'équilibre souhaité entre intensification et diversification n'existe donc plus et l'utilisateur de ces deux méthodes doit en être conscient.

Un moyen très simple de diversifier la recherche peut consister à re-exécuter un des algorithmes en prenant un autre point de départ. Comme l'exécution de ces méthodes est souvent très rapide, on peut alors inclure cette répétition au

sein d'une boucle générale. On obtient alors un algorithme de type "Multi-start descent" décrit par l'algorithme 2.3.

Algorithme 2.3 Multistart descent

1: *initialise* : find an initial solution x, $k \leftarrow 1$, $f(B) \leftarrow +\infty$
2: **repeat**
3: *Starting point* : choose an initial solution x_0 at random
4: $x \leftarrow$ result of Simple Descent or Deepest Descent
5: **if** $f(x) < f(B)$ **then**
6: $B \leftarrow x$
7: **end if**
8: $k \leftarrow k + 1$
9: **until** stopping criterion satisfied

Dans la section 2.8, nous verrons que cette dernière méthode peut dans certains cas être particulièrement efficace. De manière évidente, la diversification est totalement absente des algorithmes 2.1 et 2.2. En ne conservant que l'aspect intensification, la convergence est souvent trop rapide (plus encore avec la version *Deepest descent*) et on se trouve très rapidement bloqué dans un optimum local. Les résultats communément admis indiquent que ces techniques conduisent en général à des solutions en moyenne à 20% de l'optimum. Dans le cas de l'algorithme 2.3, la diversification est simplement insérée par le choix aléatoire d'une solution de départ. Cette simple technique peut s'avérer payante dans certains cas (voir section 2.8).

2.2 Recuit simulé

La méthode du recuit simulé a été introduite en 1983 par Kirkpatrick et al. [64]. Cette méthode originale est basée sur les travaux bien antérieurs de Metropolis et al. [81]. Cette méthode que l'on pourrait considérer comme la première métaheuristique "grand public" a reçu l'attention de nombreux travaux et principalement de nombreuses applications.

Le principe de fonctionnement s'inspire d'un processus d'amélioration de la qualité d'un métal solide par recherche d'un état d'énergie minimum correspondant à une structure stable de ce métal. L'état optimal correspondrait à une structure moléculaire régulière parfaite. En partant d'une température élevée où le métal serait liquide, on refroidit le métal progressivement en tentant de trouver le meilleur équilibre thermodynamique. Chaque niveau de température est maintenu jusqu'à obtention d'un équilibre. Dans ces phases de température constante, on peut passer par des états intermédiaires du métal non satisfaisants, mais conduisant à la longue à des états meilleurs.

L'analogie avec une méthode d'optimisation est trouvée en associant une so-

lution à un état du métal, son équilibre thermodynamique est la valeur de la fonction objectif de cette solution. Passer d'un état du métal à un autre correspond à passer d'une solution à une solution voisine.

Pour passer à une solution voisine, il faut respecter l'une des deux conditions :
- soit le mouvement améliore la qualité de la solution précédente, *i.e.* en minimisation la variation de coût est négative ($\Delta C < 0$),
- soit le mouvement détériore la qualité de la solution précédente et la probabilité p d'accepter un tel mouvement est inférieure à une valeur dépendant de la température courante t ($p < e^{-\Delta C/t}$).

Le schéma de refroidissement de la température est une des parties les plus difficiles à régler dans ce cas. Ces schémas sont cruciaux pour l'obtention d'une implémentation efficace. Sans être exhaustif, on rencontre habituellement trois grandes classes de schémas : la réduction par paliers, la réduction continue, Lundy et Mees [78] (où la température est réduite à chaque itération) et la réduction non-monotone, Connolly [15] (où des augmentations de température sont possibles). Pour d'autres schémas de refroidissement de température et une large discussion, on peut se reporter à plusieurs articles, dont les travaux de Aarts et Korst [1], de Collins et al. [14] et de Van Laarhoven et Aarts [125] par exemple. La convergence du recuit simulé vers un optimum global, en choisissant le schéma adapté, a pu être prouvée (voir [1, 55, 125]).

La popularité du recuit simulé a été incontestable pendant des années. D'abord cette méthode est facile à implémenter et elle a permis de résoudre de nombreux problèmes $\mathcal{NP}$-difficiles [9, 126]. Parmi les bibliographies et articles de synthèse intéressants, on peut citer Koulamas et al. [65] et Collins et al. [14]. Un excellent tutorial récent pourra guider les chercheurs dans leurs premiers pas avec cette méthode [90]. En attendant, l'algorithme 2.4 présente les principales caractéristiques d'un recuit simulé.

Algorithme 2.4 Basic simulated annealing

1: *initialise* : find an initial solution x, fix an annealing schedule $\mathcal{T}$, set initial temperature t
2: **repeat**
3: *neighbourhood search* : find a solution $x' \in \mathcal{N}(x)$
4: determine $\Delta C = f(x') - f(x)$
5: draw $p \sim \mathcal{U}(0,1)$
6: **if** $\Delta C < 0$ or $e^{-\Delta C/t} > p$ **then**
7: $x' \leftarrow x$
8: **end if**
9: update temperature t according to $\mathcal{T}$
10: **until** stopping criterion satisfied

Dans cet algorithme, l'équilibre entre intensification et diversification est respecté. L'intensification peut se faire par l'intermédiaire des étapes des lignes 3, 6

et 9 de l'algorithme 2.4. A la ligne 3, on effectue une recherche des voisins et ceci peut être fait par une recherche locale. De plus, à la ligne 6, on accepte tous les mouvements strictement améliorants comme dans une méthode de descente. Le schéma de refroidissement de la température réduit lui-aussi la probabilité d'accepter des mouvements non-améliorants et intensifie la recherche. Par ailleurs, la diversification est obtenue en autorisant des mouvements non-améliorants sous condition de probabilité (ligne 6) ou en choisissant un schéma $\mathcal{T}$ (ligne 9) permettant l'augmentation de température, donc en augmentant la probabilité d'acceptation de solutions plus diverses.

2.3 Recherche tabou

Dans un article présenté par Glover [37] en 1986, on voit apparaître pour la première fois, à la fois le terme *tabu search* et *métaheuristique*. A la même époque, Hansen [56] présente une méthode similaire, mais dont le nom n'a pas marqué autant que *tabou*. En fait, les prémices de la méthode ont été présentés initialement à la fin des années 1970 par Glover [36]. Pourtant ce sont les deux articles de référence de Glover [38, 39] qui vont contribuer de manière importante à la popularité de cette méthode. Pour certains chercheurs, elle apparaît même plus satisfaisante sur le plan scientifique que le recuit simulé, car la partie "aléatoire" de la méthode a disparu.

Contrairement au recuit simulé qui ne génère qu'une seule solution x' "aléatoirement" dans le voisinage $\mathcal{N}(x)$ de la solution courante x, la méthode tabou, dans sa forme la plus simple, examine le voisinage $\mathcal{N}(x)$ de la solution courante x. La nouvelle solution x' est la meilleure solution de ce voisinage (dont l'évaluation est parfois moins bonne que x elle-même). Pour éviter de cycler, une liste *tabou* (qui a donné le nom à la méthode) est tenue à jour et interdit de revenir à des solutions déjà explorées. Dans une version plus avancée de la méthode tabou, on peut voir dans cette recherche une modification temporaire de la structure de voisinage de la solution x permettant de quitter des optima locaux. Le voisinage $\mathcal{N}^*(x)$ intégrant ces modifications de structure est régit par l'utilisation de structures de mémoire spécifiques. Il s'agit de mémoire à court terme ou de mémoire à long terme.

La mémoire à court terme correspond à la mise en place d'une liste tabou. La liste contient les quelques dernières solutions qui ont été récemment visitées. Le nouveau voisinage $\mathcal{N}^*(x)$ exclut donc toutes les solutions de la liste tabou. Lorsque la structure de donnée correspondant aux solutions est trop complexe ou occupe une grande place mémoire, il est courant de ne garder dans la liste tabou que des informations soit sur les caractéristiques des solutions, soit sur les mouvements. Ce type de mémoire à court terme est aussi appelé *recency-based memory*. En conservant des caractéristiques des solutions ou des mouvements, il est possible alors qu'une solution de bien meilleure qualité ait un statut tabou.

Accepter tout de même cette solution revient à outrepasser son statut tabou, c'est l'application du *critère d'aspiration*. Si le voisinage d'une solution est très grand, évaluer toutes les solutions de ce voisinage peut-être impossible. Il convient alors de mettre en place des stratégies permettant sa réduction. Les solutions les plus courantes proposent des listes de solutions candidates qui pourraient conduire à des solutions de bonne qualité (*candidate list strategy*).

La mémoire à long terme permet d'une part d'éviter de rester dans une seule région de l'espace de recherche et d'autre part d'étendre la recherche vers des zones plus intéressantes. Par exemple, la *frequency-based memory* ou mémoire à base de fréquence attribue des pénalités à des caractéristiques des solutions plusieurs fois visitées au cours de la recherche. Cette technique simple permet de diversifier la recherche facilement. Par ailleurs, les mouvements ayant conduit a des bonnes solutions peuvent être aussi encouragés. On peut par exemple garder en mémoire une liste de solutions *élites* que l'on utilisera comme nouveau point de départ quand la recherche deviendra improductive pendant plusieurs itérations consécutives.

Il existe aujourd'hui un très grand nombre de références sur cette méthode. Un des résultats les plus attendus et pourtant ayant un impact limité en pratique est la preuve de convergence de la méthode proposée par Glover et Hanafi [44]. Parmi les articles introductifs intéressants, on peut citer celui de De Werra et Hertz [20] et de Glover [40]. Une autre référence particulièrement adaptée se doit de figurer ici, Hertz et al. [59]. Plus récemment le livre très complet de Glover et Laguna [45] donne le détail de toutes les techniques citées précédemment et bien d'autres.

Algorithme 2.5 Basic tabu search

1: *initialise* : find an initial solution x
2: **repeat**
3: *neighbourhood search* : find a solution $x' \in \mathcal{N}^*(x)$
4: *update memory* : tabu list, frequency-based memory, aspiration level, ...
5: *move* $x \leftarrow x'$
6: **until** stopping criterion satisfied

L'algorithme 2.5 présente une version très simplifiée d'un algorithme de recherche tabou. L'avantage de cette présentation, c'est qu'elle n'exclut aucune des extensions possibles mais évidemment avec un peu moins de clarté. L'alternance entre intensification et diversification est bien respectée. L'intensification réside dans la technique de base qui est l'acceptation d'une solution améliorante et bien sûr par le critère d'aspiration. Dans la mémoire à long terme, la conservation de solutions élites est aussi un facteur d'intensification. Pour l'aspect diversification, le rôle le plus important est joué par la liste tabou elle-même, mais aussi par certains points de la mémoire à long terme (la *frequency-based memory* par exemple).

2.4 Recherche à voisinages variables

La recherche à voisinages variables est une méthode récente et pourtant très simple, basée sur la performance des méthodes de descente. Introduite par Mladenović et Hansen [85], la méthode propose simplement d'utiliser plusieurs voisinages successifs quand on se trouve bloqué dans un minimum local.

Avant tout, il est nécessaire de définir un ensemble de k_{max} voisinages, dénotés par $\mathcal{N}_{k=1...k_{max}}$ (et de préférence tels que $\mathcal{N}_k \subset \mathcal{N}_{k+1}$). On choisit une solution de départ x par heuristique. Ensuite, à partir d'une solution initiale x' choisie dans le premier voisinage $\mathcal{N}(x)$ de x, on applique une méthode de descente (ou une autre méthode de recherche locale) jusqu'à arriver dans un minimum local (ou que la recherche locale s'arrête). Si la solution trouvée x'' est meilleure que x alors on recentre la recherche en repartant du premier voisinage, sinon on passe au voisinage suivant (qui a priori est plus grand). La recherche s'arrête quand tous les voisinages ne sont plus capables d'améliorer la solution.

Le point crucial dans une VNS, c'est bien évidement la constitution des voisinages de plus en plus grands et inclus les uns dans les autres. Une bonne structure de voisinage –ceci étant vrai pour toute recherche locale– conduit généralement à de bons résultats ou au moins à une recherche efficace.

Cette méthode récente intéresse de plus en plus de chercheurs et est promise à un bel avenir. Le nombre de papiers utilisant cette méthode est en très forte augmentation dans les congrès. La méthode est décrite en détail par Hansen et Mladenović [57] et un squelette général de cette méthode est proposé dans l'algorithme 2.6.

Algorithme 2.6 Basic variable neighbourhood search

1: *initialise* : find an initial solution x, k $\leftarrow 1$
2: **repeat**
3: *shake* : generate a point x' at random from the neighbourhood $\mathcal{N}_k(x)$
4: *local search* : apply a local search procedure starting from the solution x' to find a solution x''
5: **if** x'' is better than x **then**
6: $x \leftarrow x''$ and $k \leftarrow 1$ (centre the search around x'' and search again with a small neighbourhood)
7: **else**
8: $k \leftarrow k + 1$ (enlarge the neighbourhood)
9: **end if**
10: **until** $k = k_{max}$

Dans l'algorithme 2.6, la diversification est gérée par deux techniques. La première consiste à choisir dans le voisinage courant une solution aléatoirement (étape *shake*) et la seconde c'est le changement de voisinage lui-même qui agrandit

l'espace de recherche autorisé et donc agrandit le voisinage exploré. L'intensification de la recherche est effectuée par l'appel à une procédure de recherche locale. Le plus simple est d'implémenter une méthode de descente (intensification uniquement), mais parfois on peut souhaiter appliquer à cet endroit une recherche plus évoluée comme une recherche tabou qui contient elle aussi des techniques de diversification (voir par exemple Gagne et al. [33]).

2.5 GRASP

Introduite en 1989 par Feo et Resende [26] et présentée dans sa forme plus définitive en 1995 [27], la méthode *Greedy Randomized Adaptative Search* (GRASP) combine une heuristique gloutonne et une recherche aléatoire. A chaque itération, on construit une solution comme dans une heuristique gloutonne (en se servant d'une liste d'attributs comme liste de priorité). Cette solution est améliorée par l'intermédiaire d'une méthode de descente. En se basant sur la qualité générale de la solution ainsi obtenue, on met à jour l'ordre de la liste des attributs et le processus est itéré jusqu'à satisfaction d'un critère d'arrêt. Un des avantages de cette méthode est la simplicité avec laquelle on peut comprendre le processus d'optimisation. La mise en œuvre elle aussi n'est pas trop compliquée. Des implémentations réussies et un tutorial récent présentant cette méthode sont présentées par Festa [28].

Algorithme 2.7 Basic GRASP

1: *construct* an ordered list of solution attributes S
2: **repeat**
3: **repeat**
4: take $s \in S$ in a greedy way
5: $x = x \cup s$
6: **until** solution x complete
7: *local search* : find a locally optimal solution x' starting from x
8: update the benefits of the solution attributes in S
9: **until** stopping criterion satisfied

Dans l'algorithme 2.7, on retrouve les deux phases indispensables à la méthode au sein de la boucle *repeat* (lignes 2 à 9). La construction d'une solution par l'algorithme glouton se fait par la boucle interne *repeat* (lignes 3 à 6). L'étape de mise à jour de la liste des attributs n'est pas anodine pour la méthode et se trouve être un des points clés pour la réussite de cette dernière. L'intensification est obtenue par l'application de la recherche locale au sein de la boucle principale. Mais la mise à jour de la liste des attributs est aussi un facteur important permettant de contrôler la convergence de la méthode et donc de son intensification. Mettre en évidence l'aspect diversification de cette méthode n'est pas simple, pourtant c'est le rôle de l'algorithme glouton. De par sa nature, cet algorithme glouton peut

commencer à construire une bonne solution, mais en effectuant un choix aveugle à chaque étape de la construction d'une solution, il peut s'éloigner rapidement d'une bonne solution très proche. C'est donc un facteur de diversification dans la méthode GRASP.

2.6 Iterated local search

La méthode *iterated local search* est une variante très simple des méthodes de descente qui pallient au problème de l'arrêt de ces dernières dans des optima locaux. Donner la paternité de cette méthode à un ou plusieurs auteurs serait abusif. On peut tout de même citer dans l'ordre chronologique quelques unes des références présentant des méthodes très proches : Baxter [5], Baum [4], Martin et al. [80]. Par contre, le nom de la méthode pourrait revenir sans conteste à Lourenço et al. [76] (voir aussi Lourenço et al. [77]).

Dans cette méthode, on génère une solution initiale qui servira de point de départ. Ensuite, on va répéter deux phases : une phase de perturbation aléatoire dans laquelle la solution courante va être perturbée (parfois en tenant compte d'un historique maintenu à jour) et une seconde phase de recherche locale (ou tout simplement de méthode de descente) qui va améliorer cette solution jusqu'à buter sur un optimum local. Dans cette boucle, on est aussi à même d'accepter ou non la nouvelle solution selon que l'on souhaite donner un caractère plus ou moins agressif à la méthode.

Pour une présentation plus pragmatique de la méthode nous orientons le lecteur vers den Besten et al. [19]. L'implémentation détaillée de la méthode pour résoudre un problème d'ordonnancement à une machine y est présentée.

Algorithme 2.8 Basic iterated local search

1: *initialise* : find an initial solution x
2: **repeat**
3: *random perturbation* : find a solution x' "close" to x
4: *local search* : starting from x', find a solution x'' locally optimal
5: **if** x'' is accepted (e.g. better than x) **then**
6: $x \leftarrow x''$
7: **end if**
8: **until** stopping criterion satisfied

L'algorithme 2.8 présente un squelette général de la méthode *iterated local search*. Sans ambiguïté, la diversification est obtenue par la perturbation aléatoire. Même si cette dernière peut dans certaines implémentations tenir compte d'un historique des perturbations, il est important de conserver ici le caractère aléatoire. Encore une fois, c'est une recherche locale qui intensifie la recherche. La phase d'acceptation de la solution au sein de la boucle générale est importante. On peut

par exemple accepter une solution si elle améliore strictement la solution courante. Mais on peut aussi accepter seulement si elle améliore la meilleure solution trouvée, ou pourquoi pas, si elle ne dégrade pas la meilleure solution trouvée de plus d'un certain pourcentage. Comme on le voit, on peut adapter cette méthode facilement et ajouter des facteurs d'intensification ou de diversification à cet endroit.

2.7 Guided local search

Présentée pour la première fois dans un rapport de recherche de Voudouris et Tsang [128], la *guided local search* (ou recherche locale guidée) est une variante assez élaborée d'une méthode de descente classique.

La méthode de base est simple. Elle consiste à modifier la fonction à optimiser en ajoutant des pénalités. La recherche locale est appliquée alors sur cette fonction modifiée. La solution trouvée (qui se trouve être un optimum local) sert à calculer les nouvelles pénalités. Pour cela, on calcule l'utilité de chacun des attributs de la solution et on augmente les pénalités associées aux attributs de valeur maximale. Ces étapes successives sont répétées jusqu'à ce qu'un critère d'arrêt soit validé.

L'article initial de Voudouris et Tsang [129], soumis en octobre 1995 n'a été publié qu'en 1999 et présente de nombreuses améliorations de l'algorithme 2.9. Pour une présentation très approfondie, la thèse de Voudouris [127] est le document le plus complet.

Algorithme 2.9 Basic guided local search

1: *initialise* : find an initial solution x, set penalties to 0
2: **repeat**
3: *cost function* : augment the original cost function of x with penalties
4: *local search* : apply a local search procedure to find a solution x' locally optimal (based on the augmented cost function)
5: *update penalties* : by computing utility expressions
6: $x \leftarrow x'$
7: **until** stopping criterion satisfied

Dans l'algorithme 2.9, nous avons volontairement simplifié la présentation pour ne pas surcharger la lecture et la compréhension. L'intensification de la recherche est fait directement par l'appel à la recherche locale au sein de la boucle principale. Les expressions d'utilités des différents attributs d'une solution servent, pour celles qui sont à leur maximum, à pénaliser la fonction de coût. Ce qui veut dire que l'on va forcer la recherche à s'effectuer dans d'autres directions que celles qui semblaient prometteuses lors des précédentes recherches. Cette mise à jour des fonctions de pénalité est donc un facteur de diversification de la recherche.

2.8 Applications

Applications de la littérature

Comme il a été mentionné précédemment, il existe assez peu de références qui ne présentent que des méthodes de descente. Pourtant, certains ouvrages plus complet ou dépassant ce cadre sont intéressants (voir par exemple Blum et Roli [8] ou Johnson et al. [63]). Il existe énormément d'applications utilisant le recuit simulé. On peut citer des articles importants de Van Laarhoven et Aarts [125] ou de Vidal [126] et la bibliographie commentée de Collins et al. [14] qui référence un très grand nombre d'applications. Pour la recherche tabou, on ne compte plus les articles se référant à cette méthode. On peut citer un numéro spécial de Glover et al. [47] présentant plusieurs articles sur le sujet. La méthode GRASP est de plus en plus populaire. Il existe un site web `http://www.graspheuristic.org` qui est maintenu à jour et référence plusieurs centaines d'articles. Régulièrement une bibliographie commentée de GRASP est publiée (voir Festa et Resende [29]). La méthode *Iterated local search*, un peu ignorée au début, sans doute pour son apparente simplicité, commence à prendre place au sein de toutes ces méthodes (voir Paquete et Stützle [89]). Du côté du *Guided local search*, c'est le même constat. Les applications se multiplient et la méthode prouve son efficacité dans plusieurs domaines (voir Beullens et al. [7] ou Voudouris et Tsang [129]).

Ordonnancement sur machines parallèles

De notre côté, nous avons abordé plusieurs de ces méthodes pour un problème d'ordonnancement sur machines parallèles. Un ensemble de n jobs ($j = 1 \dots n$) doit être séquencé sur m machines parallèles. Chaque job doit être exécuté sur une des machines et ne peut démarrer avant sa date de début au plus tôt, r_j (release date). Le temps d'exécution de ce job est de p_j unités de temps (processing time). Une date échue, d_j (due date) et un poids, w_j (weight) sont donnés pour chaque job. Si le job se termine après sa date échue, on dit qu'il est en retard, sinon le job est terminé à temps. Le critère retenu est la minimisation de la somme des poids des jobs en retard $\min \sum w_j U_j$. Ce problème est noté $Pm|r_j| \sum w_j U_j$ dans la littérature [51]. Même réduit à deux machines, ce problème est $\mathcal{NP}$-difficile [35]. Ce problème et les résultats présentés plus loin sont extraits des différentes présentations du problème (voir par exemple [113–118]).

Le tableau 1 est directement extrait de [115]. Nous comparons ici les méthodes en utilisant toujours le même voisinage. Dans ce cas-ci, il s'agit d'échanger un job actuellement *en retard* par un ou plusieurs jobs *en avance* (voir Sevaux et Thomin [115]). Sans entrer dans des détails, les lignes notées EMP, BWS, BNO, BRS représentent différents points de départ des heuristiques (sans objet pour la méthode multi-start).

Sans tirer des conclusions générales, on constate dans le tableau 1 que le point de départ a peu d'impact sur la qualité d'une solution obtenue par méthode de

TAB. 1 – Résultats extraits de Sevaux et Thomin [115].

Initial Heuristics	Optimum hits (%)	First pos. (%)	Gap to opt. (%)	Avg. CPU time (s)	Max. CPU time (s)
Deepest descent heuristics					
EMP	45	33	21.5	0.02	0.12
BWS	49	35	19.6	$< .01$	$< .01$
BNO	44	31	20.3	$< .01$	$< .01$
BRS	44	31	20.3	$< .01$	$< .01$
Multistart deepest descent heuristic					
MD	78	63	3.9	4.58	41.1
Simulated annealing					
EMP	70	55	6.72	9.10	60.3
BWS	70	54	6.59	7.68	53.2
BNO	71	56	6.03	9.01	64.9
BRS	70	55	5.98	8.87	63.3
Tabu search					
EMP	89	79	1.94	10.63	101.9
BWS	88	76	2.00	9.42	112.0
BNO	88	80	1.89	10.17	93.7
BRS	88	79	1.95	10.16	107.5

descente et que les solutions se trouvent en moyenne à 20% de l'optimum. En utilisant le même voisinage, et pour une configuration particulière des paramètres de recuit simulé et recherche tabou, les écarts à l'optimum sont respectivement de 6% et 2% environ. Par contre, ce qui est très surprenant, c'est que la méthode *multi-start* obtient de très bons résultats avec un écart de 4%. Faut-il pour autant renoncer à implémenter un recuit simulé ? Certainement pas. Comme il a été signalé, le réglage des paramètres est fixe et en faisant varier ces derniers on devrait pouvoir atteindre des résultats meilleurs. Ce qui est important à retirer de cette expérience, c'est que même si l'implémentation des méthodes de descente donne des résultats médiocres, en passant à une simple méthode *multi-start* on obtient des solutions très satisfaisantes avec des temps de calcul raisonnables et surtout un temps d'implémentation minimum (le plus dur étant déjà fait dans l'écriture du voisinage lui-même). Donc, passer ensuite à une méthode plus complexe comme le recuit simulé ou la recherche tabou est un pas supplémentaire à franchir pour peu qu'on en ait le temps.

3 Métaheuristiques à base de population

Les méthodes de recherche à population, comme leur nom l'indique, travaillent sur une population de solutions et non pas sur une solution unique. On peut

trouver d'autres noms génériques pour ces méthodes, le plus en vogue étant sans doute *evolutionary algorithms* (voir Michalewicz [82]). Le principe général de toutes ces méthodes consiste à combiner des solutions entre elles pour en former de nouvelles en essayant d'hériter des "bonnes" caractéristiques des solutions parents. Un tel processus est répété jusqu'à ce qu'un critère d'arrêt soit satisfait (nombre de générations maximum, nombre de générations sans améliorations, temps maximum, borne atteinte, etc). Parmi ces algorithmes à population, on retrouve deux grandes classes qui sont les algorithmes génétiques et les colonies de fourmis. Les algorithmes génétiques ont beaucoup fait parler d'eux et depuis longtemps. Les colonies de fourmis sont des techniques plus récentes, mais l'auteur a su faire parler de la méthode et intéresser non- seulement les chercheurs, mais aussi les grandes instances décisionnaires comme la commission européenne par exemple. Il existe bien d'autres techniques que nous aurions pu décrire ici comme le *Genetic programming* de Koza [67] par exemple. Nous avertissons aussi ici le lecteur, que les prises de position sur différents points sont celles de l'auteur de cette synthèse et n'engage que lui. Elles sont pour la plupart discutables...

3.1 Algorithmes génétiques

Proposé dans les années 1975 par Holland [61], les algorithmes génétiques doivent leur popularité à Goldberg [49]. Avant la parution de son livre qui est une des références les plus citées dans le domaine de l'informatique, on a pu voir un certain nombre d'autres présentations, citons Goldberg [48], Holland [62], Schwefel [97]. Le sujet connaît une très grande popularité. Il existe aujourd'hui plusieurs milliers de références sur le sujet et le nombre de conférences dédiées au domaine (que ce soit sur les techniques elles-mêmes ou sur les applications) ne fait qu'augmenter.

De manière générale, les algorithmes génétiques utilisent un même principe. Une population d'individus (correspondants à des solutions) évoluent en même temps comme dans l'évolution naturelle en biologie. Pour chacun des individus, on mesure sa faculté d'adaptation au milieu extérieur par le *fitness*. Les algorithmes génétiques s'appuient alors sur trois fonctionnalités :

la sélection qui permet de favoriser les individus qui ont un meilleur *fitness* (pour nous le fitness sera la plus souvent la valeur de la fonction objectif de la solution associée à l'individu).

le croisement qui combine deux solutions parents pour former un ou deux enfants (*offspring*) en essayant de conserver les "bonnes" caractéristiques des solutions parents.

la mutation qui permet d'ajouter de la diversité à la population en mutant certaines caractéristiques (*gènes*) d'une solution.

La représentation des solutions (le codage) est un point critique de la réussite d'un algorithme génétique. Il faut bien sûr qu'il s'adapte le mieux possible au problème et à l'évaluation d'une solution. Le codage phénotypique ou codage direct correspond en général à une représentation de la solution très proche de la réalité. L'évaluation d'une solution représentée ainsi est en général immédiate. On peut aussi utiliser un codage indirect (codage génotypique) qui est souvent plus éloigné de la réalité et qui nécessite un algorithme de décodage pour reconstituer une solution valide. Pour ce dernier codage, il existe aujourd'hui beaucoup de travaux et notamment des opérateurs de croisement et de mutation en quantité.

En effectuant une recherche sur internet, on trouve plusieurs milliers de pages web répondant à la requête *genetic algorithms*. Donner une bonne référence est indispensable. Nous renvoyons donc le lecteur vers deux livres importants, Haupt et Haupt [58], Mitchel [84] et vers un bon article introductif qui a le mérite d'être en français, Fleurent et Ferland [30] et qui explique avec suffisamment de détails une implémentation parmi d'autres. Comme pour d'autres algorithmes, la preuve de convergence est apportée par Cerf [12]. Même si l'intérêt pratique de cette preuve est limité, il est toujours plus satisfaisant de savoir que la méthode converge sous certaines conditions.

Nous présentons deux versions possibles d'algorithme génétique. L'algorithme 3.1 propose une version qualifiée en Anglais par *population replacement* (ou algorithme générationnel) tandis que l'algorithme 3.2 est une version incrémentale. La différence entre les deux réside dans la gestion de la population.

Dans le premier cas (algorithme 3.1), on créé une seconde population en parallèle à la première, puis on insère des individus obtenus par croisement et mutation des individus de la population précédente. Dès qu'une nouvelle population est formée, elle remplace la population précédente et le processus recommence. Un des points importants à prendre en compte est la perte possible du meilleur individu ou même de la solution optimale lors du remplacement de la population. Il faudra donc prévoir un mécanisme permettant d'éviter cela. Dans cet algorithme, la sélection est faite ici en effectuant un tirage aléatoire biaisé par la valeur du fitness des individus (par exemple par la méthode de ranking de Reeves [95]). Comme dans les systèmes d'évolution naturels, le croisement et la mutation se font sous conditions de probabilité.

Dans la seconde version de l'algorithme génétique (algorithme 3.2), on créé un nouvel individu par croisement et mutation et il remplace un individu existant dans la population courante. Ma préférence va à la seconde méthode qui permet (potentiellement) d'exploiter immédiatement tout nouvel individu créé à l'itération précédente. Dans cet algorithme, nous choisissons la méthode de tournoi binaire (ou *roulette wheel*) pour la sélection. Le croisement est fait à chaque itération et la mutation sous conditions (par exemple avec une faible probabilité mais en respectant un critère d'aspiration). Pour le choix de l'individu à rem-

Algorithme 3.1 A simple genetic algorithm (population replacement)

initialise : generate an initial population P of solutions with size $|P| = n$
repeat
  $P' \leftarrow \emptyset$
  repeat
    selection : choose 2 solutions x and x' from P with probability proportional to their fitness
    crossover : combine parent solutions x and x' to form child solutions y and y' with high probability
    mutate y and y' with small probability
    add y and y' to P'
  until $|P'| = n$
  $P \leftarrow P'$
until stopping criterion satisfied

placer, on peut comme dans Reeves [95] choisir un individu parmi ceux dont le fitness dépasse la valeur médiane (plus mauvaise moitié des individus) ou alors utiliser la technique de tournoi binaire (inversée). On peut bien sûr modifier ce squelette facilement avec toutes les adaptations proposées dans la littérature.

Algorithme 3.2 A simple genetic algorithm (incremental replacement)

initialise : generate an initial population P of solutions with size $|P| = n$
repeat
  selection : choose 2 solutions x and x' with the roulette wheel technique
  crossover : combine parent solutions x and x' to form a child solution y
  mutate y under conditions
  choose an individual y' to be replaced in the population
  replace y' by y in the population
until stopping criterion satisfied

Pour l'un et l'autre des algorithmes, l'intensification est amenée à la fois par le processus de sélection et par l'opérateur de croisement. Comme la sélection est faite soit en biaisant un processus aléatoire, soit par tournoi binaire (favorisant dans les deux cas les meilleurs individus), on va forcer la population à converger. C'est donc un facteur d'intensification. Les opérateurs de croisement sont dessinés pour faire profiter aux enfants des "bonnes" caractéristiques des parents. Même si ce n'est pas toujours évident de construire de tels opérateurs, ces derniers devraient constituer un facteur d'intensification. Nous verrons dans le cas des algorithmes mémétiques (section 4.1), que le croisement peut être aussi vu comme un facteur de diversification dans certains cas. Mais dans les algorithmes génétiques initiaux, le facteur de diversification principal est la mutation qui modifie directement les caractéristiques d'une solution permettant d'explorer

de nouveaux espaces de solutions.

3.2 Algorithmes de colonies de fourmis

Les algorithmes à base de colonies de fourmis ont été introduits par Dorigo et al. [24] et dans la thèse de [21]. Une des applications principales de la méthode originale était le problème du voyageur de commerce et depuis elle a considérablement évolué.

Cette nouvelle métaheuristique imite le comportement de fourmis cherchant de la nourriture. A chaque fois qu'une fourmi se déplace, elle laisse sur la trace de son passage une odeur (la phéromone). Comme la fourmi est rarement une exploratrice isolée, avec plusieurs de ses congénères, elle explore une région en quête de nourriture. Face à un obstacle, le groupe des fourmis explore les deux côtés de l'obstacle et se retrouvent, puis elles reviennent au nid avec de la nourriture. Les autres fourmis qui veulent obtenir de la nourriture elles aussi vont emprunter le même chemin. Si celui-ci se sépare face à l'obstacle, les fourmis vont alors emprunter préférablement le chemin sur lequel la phéromone sera la plus forte. Mais la phéromone étant une odeur elle s'évapore. Si peu de fourmis empruntent une trace, il est possible que ce chemin ne soit plus valable au bout d'un moment, il en est de même si des fourmis exploratrices empruntent un chemin plus long (pour le contournement de l'obstacle par exemple). Par contre, si le chemin est fortement emprunté, chaque nouvelle fourmi qui passe redépose un peu de phéromone et renforce ainsi la trace, donnant alors à ce chemin une plus grande probabilité d'être emprunté.

On peut alors comprendre l'application directe du problème du voyageur de commerce. Chaque fourmi constitue un tour en empruntant des arêtes du graphes, puis au bout d'un moment les arêtes qui conduisent à une bonne solution deviendront les plus empruntées jusqu'à obtention d'une très bonne solution. Même si le principe est simple, on peut voir apparaître des difficultés comme, la quantité de phéromone à déposer, le coefficient d'évaporation, la définition du choix biaisé, etc. La métaheuristique telle qu'elle est décrite schématiquement ci- dessous est détaillée par Dorigo et al. [23] et Dorigo et Di Caro [22]. Bien que cette méthode soit plutôt récente, elle a suscité de nombreux articles en "tournées de véhicules", peut-être par son exotisme ou parce qu'il est plus facile de publier avec une nouvelle méthode moins connue.

Si on regarde en détail l'algorithme 3.3, on pourrait le comparer à l'algorithme GRASP (voir section 2.5) dans lequel on ferait entrer en jeu plusieurs solutions simultanément. Chaque fourmi est associée à un algorithme glouton biaisé où les phéromones sont associées aux attributs de GRASP. Pour parfaire la comparaison, il manque la recherche locale de GRASP qui n'est pas présente ici, mais que l'on commence à voir apparaître dans des exposés en conférence.

Algorithme 3.3 Basic ant colony optimisation

1: *initialise* : create an initial population of ants
2: **repeat**
3: **for** each ant **do**
4: *construct a solution* based on the construction procedure, biased by the pheromone trails
5: *update the pheromone trails* based on the quality of the solutions found
6: **end for**
7: **until** stopping criterion satisfied

Les facteurs d'intensification et de diversification sont plus difficiles à cerner. Néanmoins, le choix d'un nouvel attribut se fait aléatoirement (même s'il est biaisé) c'est donc un facteur de diversification. L'intensification se retrouve dans les paramètres de dépôt et d'évaporation qui concentrent plus ou moins les teneurs en phéromone des attributs de la solution.

3.3 Applications

Pour des présentations générales de méthodes à population qui incluent des applications (principalement en ordonnancement), nous renvoyons le lecteur vers certains de nos travaux et présentations [99, 102].

Applications de la littérature

Que dire des applications des algorithmes à population. Elles sont de plus en plus nombreuses, et de plus en plus évoluées. Du côté des techniques de colonies de fourmis, le sujet est encore un peu jeune pour proposer plusieurs articles de synthèse sur différents travaux. On peut tout de même citer les articles de Dorigo et al. [23] et Dorigo et Di Caro [22]. Pour les algorithmes génétiques, on peut citer en premier lieu une bibliographie proposée par Alander [2] qui recense des implémentations. Ce rapport comporte quelques 850 références d'implémentations des algorithmes génétiques. Le plus impressionnant, c'est que le nombre d'articles sur les algorithmes génétiques est d'environ 1000 par an depuis 1995 avec plus d'une centaine d'articles sur des implémentations chaque année. La différence entre ces deux chiffres montre qu'il y a encore un potentiel très intéressant pour de nouvelles applications (voir aussi De Jong [18]). Un site web particulièrement intéressant présente des articles et implémentations `http://www-illigal.ge.uiuc.edu`.

En dehors des ouvrages généraux comme [49, 94] et de tous les autres, on peut mettre en avant une très bonne introduction aux problèmes résolus par les algorithmes génétiques présentée par Reeves [96]. Cet article référence des applications dans tous les domaines de la recherche opérationnelle et en particulier de

l'optimisation combinatoire.

Ordonnancement à une machine

De notre côté, nous avons mené depuis 1999 un certain nombre de travaux sur les algorithmes génétiques. Par exemple, dans Sevaux et Dauzère-Pérès [107], nous proposons une étude assez détaillée d'un algorithme génétique classique pour lequel nous avons essayé d'appliquer différentes variantes. Le problème concerné est la minimisation du nombre de jobs en retard sur une machine. Un ensemble de n jobs ($j = 1 \dots n$) doit être séquencé sur une machine unique où la préemption n'est pas autorisée. Chaque job ne peut démarrer avant sa date de début au plus tôt, r_j (release date). Le temps d'exécution de ce job est de p_j unités de temps (processing time). Une date échue, d_j (due date) et un poids, w_j (weight) sont donnés pour chaque job. Si le job se termine après sa date échue, on dit qu'il est en retard, sinon le job est terminé à temps. Le critère retenu est la minimisation de la somme des poids des jobs en retard min $\sum w_j U_j$. Ce problème est noté $1|r_j|\sum w_j U_j$ dans la littérature [51] et $\mathcal{NP}$-difficile au sens fort [75].

Dans cette étude, pour un même codage indirect, nous étudions l'impact de différents moteurs de décodage de la solution, de plusieurs opérateurs de croisement et de mutation. Ces résultats sont comparés avec des solutions obtenues par la relaxation Lagrangienne de Dauzère-Pérès et Sevaux [17]. Cette étude est le fruit de plusieurs expériences (voir [103, 104, 106]).

TAB. 2 – Comparaison pour le moteur "List", extrait de [105]

	Best Indiv.		Average Indiv.		Cpu Time	Av. Nb. of
	Mean	St-Dev	Mean	St-Dev	(in sec.)	Iterations
OX + Swap	4.52	5.53	4.84	5.83	11.30	39309.04
+ BigSwap	4.46	5.46	4.80	5.78	12.79	43156.70
+ Shift	4.53	5.51	4.86	5.81	11.05	37542.16
LOX + Swap	4.53	5.57	4.66	5.68	6.64	23407.71
+ BigSwap	3.83	4.90	4.01	5.05	15.40	55058.94
+ Shift	4.23	5.29	4.39	5.43	10.76	37850.51
PMX + Swap	4.31	5.41	4.48	5.56	7.37	25536.58
+ BigSwap	3.99	5.09	4.22	5.29	10.68	37171.44
+ Shift	4.00	5.09	4.31	5.36	9.95	34677.41
X1 + Swap	4.81	5.76	4.91	5.85	1.99	14335.95
+ BigSwap	1.01	1.09	1.44	1.58	10.66	78381.29
+ Shift	4.64	5.58	4.73	5.67	2.42	18533.48

La table 2 (resp. 3 et 4) présente les résultats des différentes combinaisons d'opérateurs de croisement et de mutation obtenus pour un moteur de décodage d'un individu de type "List" (resp. de type "Sans délais", et "FIFO"). Les quatre opérateurs de croisement sont OX, LOX, PMX et X1 et les trois opérateurs de

TAB. 3 – Comparaison pour le moteur “ND”, extrait de [105]

	Best Indiv.		Average Indiv.		Cpu Time	Av. Nb. of
	Mean	St-Dev	Mean	St-Dev	(in sec.)	Iterations
OX + Swap	1.58	1.13	2.11	1.55	44.90	26464.21
+ BigSwap	1.48	1.03	2.05	1.49	51.10	29443.37
+ Shift	1.53	1.07	2.08	1.51	46.64	26837.89
LOX + Swap	1.45	1.04	1.72	1.22	37.56	20468.35
+ BigSwap	1.07	0.82	1.46	1.08	57.18	29790.79
+ Shift	1.18	0.87	1.54	1.10	50.12	26661.19
PMX + Swap	1.21	0.93	1.54	1.16	41.34	22046.54
+ BigSwap	0.89	0.72	1.33	1.00	61.65	31927.69
+ Shift	0.88	0.73	1.45	1.12	58.99	32960.50
X1 + Swap	2.11	1.57	2.30	1.72	11.28	14065.76
+ BigSwap	1.29	0.99	1.58	1.17	26.94	33426.13
+ Shift	1.63	1.15	1.84	1.30	20.28	23572.64

TAB. 4 – Comparaison pour le moteur “Fifo”, extrait de [105]

	Best Indiv.		Average Indiv.		Cpu Time	Av. Nb. of
	Mean	St-Dev	Mean	St-Dev	(in sec.)	Iterations
OX + Swap	1.02	1.01	1.52	1.38	42.61	31572.30
+ BigSwap	0.91	0.91	1.41	1.27	46.81	34969.10
+ Shift	0.98	0.95	1.49	1.32	42.02	31391.71
LOX + Swap	1.12	1.10	1.23	1.17	24.92	18061.56
+ BigSwap	0.72	0.75	0.88	0.86	38.62	28357.40
+ Shift	0.83	0.82	0.98	0.93	33.85	24807.48
PMX + Swap	0.93	0.97	1.06	1.05	27.91	20047.78
+ BigSwap	0.69	0.74	0.86	0.85	38.65	27862.21
+ Shift	0.69	0.74	0.95	0.95	35.90	27528.53
X1 + Swap	1.76	1.66	1.86	1.73	7.48	13791.81
+ BigSwap	0.74	0.75	0.84	0.83	17.68	32800.92
+ Shift	1.17	1.02	1.25	1.08	13.03	22752.71

mutation sont Swap (ou API, Adjacent Pairwise Interchange), BigSwap (ou GPI, General Pairwise Interchange) ou Shift (insertion d'un gène puis décalage) (pour plus de détails, voir [105]). Les mesures sont faites par rapport aux solutions optimales des problèmes. Les chiffres de ces tableaux sont les moyennes constatées sur 1000 instances. Les mesures sont effectuées en comparant les écarts (en moyenne et écart-type) du meilleur individu et de l'individu moyen (colonnes "Best Indiv." et "Average Indiv."). Les temps CPU et nombres moyens d'itérations sont donnés à titre indicatif.

De ces tables, on a pu mettre en évidence un avantage certain à un moteur de décodage, un opérateur de croisement et un opérateur de mutation. En regardant le critère de performance relative, le moteur de type "List" n'est pas satisfaisant quelque soient les opérateurs retenus. Le moteur ND de type "Sans délais" donne de bons résultats quand il est associé à l'opérateur de croisement PMX et pas avec l'opérateur de mutation "Swap". Le moteur de type "Fifo" donne de bons résultats avec les opérateurs de croisement PMX, LOX et X1 et les opérateurs de mutation "BigSwap" et "Shift". Pour la suite de l'étude nous avions retenu le moteur "Fifo", l'opérateur de croisement PMX et l'opérateur de mutation "Shift".

4 Métaheuristiques avancées

Pour aller plus loin dans la recherche de solutions, il faut avant tout pouvoir détecter de nouvelles solutions. Les algorithmes à base de population présentent un intérêt particulier : le parallélisme intrinsèque. En ajoutant de nouveaux composants à ces algorithmes, on peut alors construire des algorithmes "hybrides". L'une des faiblesses d'un algorithme génétique comme la vitesse de convergence trop lente peut, par exemple, être compensée par l'ajout d'une méthode de recherche locale. C'est le cas des algorithmes mémétiques de Moscato [86] présentés section 4.1. Il est clair que sans l'aide des recherches locales les méthodes à population n'arrivent pas à fournir des solutions très satisfaisantes. De même, sans la gestion efficace d'une population de solutions, il est difficile pour une recherche locale de parcourir efficacement l'espace des solutions souvent très vaste. Une autre des caractéristiques importantes est de pouvoir limiter la taille de la population. C'est le cas du "scatter search" de Glover [42] et du nouvel algorithme GA|PM initialement proposé par Sörensen [121]. Une taxonomie des méthodes dites hybrides est proposée dans Talbi [124].

4.1 Algorithmes mémétiques

Moscato [86] introduit en 89 pour la première fois les algorithmes mémétiques. On rencontre aussi le nom d'*algorithmes génétiques hybrides* (trop vague à notre goût) ou celui de *genetic local search* (à ne pas confondre avec *guided local search*).

Quelque soit le nom qu'on lui donne, l'idée principale de cette technique est de rendre plus agressif un algorithme génétique par l'ajout d'une recherche locale en plus de la mutation.

Une des observations générales provenant de l'implémentation d'un algorithme génétique basique est souvent la faible vitesse de convergence de l'algorithme. L'idée de Moscato est donc d'ajouter une recherche locale – qui peut être une méthode de descente ou une recherche locale plus évoluée (recuit simulé ou recherche tabou par exemple). Cette recherche locale sera appliquée à tout nouvel individu obtenu au cours de la recherche.

Il est évident que cette simple modification entraîne de profonds changements dans le comportement de l'algorithme lui-même. Après avoir créé un nouvel individu à partir de deux parents sélectionnés, on applique une recherche locale et sous certaines conditions on applique un opérateur de mutation à cet individu. Les conditions peuvent être une certaine probabilité. Il est aussi possible d'ajouter un critère d'aspiration ou d'autres techniques plus évoluées à cet endroit.

Pour une introduction aux algorithmes mémétiques, il est souvent plus facile de se reporter à la page web maintenue à jour par P. Moscato `http://www.densis.fee.unicamp.br/~moscato` qui présente les papiers de référence de cette méthode. Nous suggérons aussi au lecteur de consulter Moscato [87].

Algorithme 4.1 A simple memetic algorithm (incremental replacement)

1: initialise : generate an initial population P of solutions with size $|P| = n$
2: apply a local search procedure on each solution of P
3: **repeat**
4: *selection* : choose 2 solutions x and x' with the roulette wheel technique
5: *crossover* : combine parent solutions x and x' to form a child solution y
6: *local search* : apply a local search procedure on y under conditions
7: *mutation* : apply a mutation operator on y under conditions
8: choose an individual y' to be replaced in the population
9: replace y' by y in the population
10: **until** stopping criterion satisfied

Nous proposons une version incrémentale de l'algorithme mémétique (voir algorithme 4.1). La structure est très proche de l'algorithme 3.2. Dans cet algorithme on conserve la mutation à côté de la recherche locale, mais on pourrait effectuer l'un ou l'autre, ou les deux, systématiquement ou sous condition de probabilité. Les possibilités sont infinies. C'est ce qui complique sans doute la mise au point de ces méthodes.

L'intensification dans cet algorithme est produite de toute évidence par l'application nouvelle de la recherche locale. L'opérateur de mutation assure la diversification de la méthode. Par ailleurs, il existe des comportements pas toujours souhaitables. On constate par exemple que lorsque l'on choisit un codage indirect

de la solution, il n'est pas toujours facile de trouver des opérateurs de croisement efficaces qui permettent d'hériter des bonnes caractéristiques des parents. Une petite étude statistique menée sur la qualité des parents et de leurs enfants tend à prouver que la diversification pourrait être aussi amenée par l'opérateur de croisement lui-même [92]. Pourtant son but initial dans l'algorithme génétique est bien d'intensifier la recherche. Nous nous demandons alors, si la diversification n'est pas amenée simplement par l'ajout de nouvelles solutions de départ dans une recherche locale (un peu comme dans un *multi-start*, sauf que les solutions initiales ne sont pas aléatoires, mais résultant de l'opérateur de croisement).

4.2 Scatter search

L'origine de la méthode ou du moins la première publication que l'on trouve comportant ce nom est due à Glover [41]. Mais le travail conjoint de Fred Glover, Manuel Laguna et Raphael Martì a permis d'obtenir une méthode plus claire et plus sophistiquée, et qui garde un caractère générique indéniable [46].

Cette méthode fait évoluer une population de solutions et s'appuie sur le principe suivant. Une population de solutions (assez importante au départ) est générée (en essayant de proposer des solutions diverses les unes des autres) et chaque individu est amélioré par l'application d'une recherche locale. De cette population on extrait un *ensemble de référence* (*Reference set*, R) contenant les meilleures solutions de la population initiale. Ensuite ces solutions sont combinées entre elles (et avec les nouvelles solutions générées) puis améliorées jusqu'à ce qu'il n'y ait plus de nouvelles solutions générées par combinaison. Ensuite une moitié de la population est régénérée (remplacée par des solutions diverses) et le processus recommence jusqu'à satisfaction d'un critère d'arrêt.

Un des points importants, c'est la mesure de la diversité des solutions. Cette mesure doit être prise dans l'espace des solutions (et non dans l'espace des objectifs) et elle doit refléter la différence entre deux solutions. Ce type de mesure sera utilisé dans la métaheuristique GA|PM de la section 4.3. Pour plus de détails sur les mesures que l'on peut utiliser, voir la thèse de Sörensen [121] et Sörensen [120].

La particularité de cette méthode est l'acharnement à épuiser les ressources par combinaison. Ceci se traduit par une sorte d'exploration systématique de tout un voisinage possible et bien sûr par un temps d'exécution parfois prohibitif. Par la suite, on remplace une partie de la population et on recommence. Cette technique de *restart* est présente dans d'autres algorithmes hybrides (voir section 4.4). Nous renvoyons vers les publications suivantes pour plus de détails sur la méthode elle-même et sur des particularités d'implémentation [42, 43, 46].

L'algorithme 4.2 présente un squelette du scatter search. Volontairement nous avons utilisé une notation ensembliste plus claire à nos yeux. Dans cet algorithme,

Algorithme 4.2 Basic scatter search

```
1: initialise : generate an initial improved population
2: Select a diverse subset R (Reference set)
3: while stopping criterion is not satisfied do
4:    A ← R
5:    while A ≠ ∅ do
6:       combine solutions (B ← R × A)
7:       improve solutions of B
8:       update R (keep best solutions from R ∪ B)
9:       A ← B − R
10:   end while
11:   Remove half of the worst solutions in R
12:   Add new diverse solutions in R
13: end while
```

l'idée générale des auteurs est de garder à chaque itération (celles de la boucle externe) un ensemble de référence qui soit à la fois composé de bonnes solutions et de solutions diverses. Les facteurs d'intensification se retrouvent alors dans l'application systématique d'une méthode de recherche locale (lignes 1 et 7) ainsi que dans la combinaison répétée des solutions (boucle des lignes 5 à 10) et dans la mise à jour de R (ligne 8). Le choix des sous ensembles diverses (ligne 2) et le remplacement d'une partie de la population à chaque étape (ligne 12) sont les facteurs de diversification du scatter search.

4.3 GA|PM

En partant des observations faites sur la plupart des métaheuristiques précédentes, y compris des plus sophistiquées, plusieurs caractéristiques importantes sont apparues à Sörensen [121]. Nous le rejoignons sur la plupart des points. En premier lieu, une recherche locale est indispensable. De même, il est important de mesurer (comme dans le scatter search) la diversité des solutions. Par ailleurs, l'exploration systématique n'est pas toujours nécessaire. De ces constatations, Sörensen [121] a proposé cette nouvelle métaheuristique appelée GA|PM (genetic algorithm with population management – algorithme génétique avec gestion de la population).

Le fonctionnement est assez simple et est basé sur un algorithme génétique (ici dans sa version incrémentale). Nous supposons que nous savons comparer deux individus entre eux et mesurer leur dissemblance. Nous pouvons donc mesurer la similarité entre un individu et la population existante. Au départ, on génère une population initiale de petite taille et on choisit un paramètre Δ fixant le niveau de dissemblance des solutions entre elles. Ensuite, on procède comme dans un algorithme génétique, on choisit deux individus que l'on croise pour obtenir

deux enfants. Pour chacun on applique une recherche locale de façon à obtenir des optima locaux. S'ils ne répondent pas au critère de diversité, on applique un opérateur de mutation sur ces individus jusqu'à satisfaction de ce critère. Ensuite sous condition, on les insère dans la population à la place d'un autre individu. A chaque itération le paramètre Δ gérant la diversité est mis à jour (suivant plusieurs schémas – voir [121, 122]).

Ce qui fait sans doute que cette méthode donne de très bons résultats, c'est qu'elle combine les avantages du scatter search et d'un algorithme mémétique. D'un côté elle applique une recherche locale à toutes les nouvelles solutions et de l'autre, elle maintient une population de petite taille et diversifiée. A la différence du scatter search, la diversité est contrôlée précisément. L'autre avantage, c'est l'évolution du paramètre de diversité Δ (comme dans les schémas de refroidissement de température du recuit simulé) qui permet à tout moment d'augmenter ou de réduire la diversité des individus dans la population. Pour une introduction générale de cette méthode précisant des détails importants, consultez Sörensen et Sevaux [122].

Algorithme 4.3 GA|PM outline

```
 1: initialise population P
 2: set population diversity parameter Δ
 3: repeat
 4:    select : p1 and p2 from P
 5:    crossover : p1 ⊗ p2 → c1, c2
 6:    local search : on c1 and c2
 7:    for each child c do
 8:       while d_P(c) < Δ do
 9:          mutate c
10:       end while
11:       if c satisfies conditions for addition then
12:          remove solution : P ← P\b
13:          add solution : P ← P ∪ c
14:       end if
15:    end for
16:    update diversity parameter Δ
17: until stopping criterion satisfied
```

L'algorithme 4.3 présente un cadre général de l'algorithme GA|PM qui peut sans doute être amélioré pour chacun des problèmes que l'on souhaite traiter. Dans cet algorithme le facteur d'intensification principal est l'appel à la recherche locale. Concernant la diversification, c'est la boucle (lignes 8 à 10) imposant un certain niveau de diversité qui joue ce rôle. La manipulation du paramètre Δ peut être interprétée dans les deux sens. Si on diminue Δ, on autorise l'intégration de solutions moins diverses, donc on intensifie la recherche, dans le cas contraire, on

force les individus à être de plus en plus éloignés entre eux, on diversifie donc la recherche. Ce qu'il serait intéressant de mesurer ici, c'est le nombre d'itérations de la boucle des lignes 8 à 10 pour voir si la recherche locale concentre les individus dans une même région. Cela permettrait sans doute un ajustement plus précis de Δ au cours de la recherche.

4.4 Applications

Applications de la littérature

Les algorithmes mémétiques sont plus anciens et sans doute plus faciles à implémenter (il suffit de combiner le code d'un algorithme génétique et d'une recherche locale). On trouve naturellement plusieurs centaines d'applications. Pour des applications intéressantes, nous renvoyons le lecteur vers `http://www.densis.fee.unicamp.br/~moscato/`.

La *scatter search* plus récente commence à avoir les faveurs des chercheurs. Le retard pris dans l'utilisation de la méthode provient très certainement du fait que les papiers originaux étaient peu clairs sur certaines parties de l'algorithme lui-même. Il existe aujourd'hui un certain nombre d'articles acceptés ou de rapports de recherche qui présente des applications très intéressantes (voir `http://www-bus.colorado.edu/faculty/laguna/scattersearch.html`).

Ordonnancement à une machine

Au début de l'année 2001, nous avons commencé à travailler à la comparaison des algorithmes mémétiques et du scatter search. Peu après, les auteurs du scatter search ont proposé une série d'articles de même type (voir Campos et al. [10], Martí et al. [79]). De notre côté, nous avons travaillé sur un problème d'ordonnancement à une machine où l'objectif est la minimisation du retard total pondéré, Sevaux et Thomin [119]. Un ensemble de n jobs ($j = 1 \dots n$) doit être séquencé sur une machine unique où la préemption n'est pas autorisée. Chaque job ne peut démarrer avant sa date de début au plus tôt, r_j (release date). Le temps d'exécution de ce job est de p_j unités de temps (processing time). Une date échue, d_j (due date) et un poids, w_j (weight) sont donnés pour chaque job. Si le job se termine après sa date échue, on dit qu'il est en retard et l'on minimise la somme des retards pondérés. Ce problème est noté $1|r_j|\sum w_j T_j$ dans la littérature [51] et $\mathcal{NP}$-complet au sens fort. C'est sans doute l'un des problèmes à une machine les plus difficiles. La réduction au problème sans les poids, $1|r_j|\sum T_j$ est $\mathcal{NP}$-complet au sens fort [35]. Si on conserve les poids, mais que les jobs sont tous disponibles à l'instant 0, le problème $1||\sum w_j T_j$ est aussi $\mathcal{NP}$-complet au sens fort [74]. Il faut arriver au problème $1||\sum T_j$ pour rencontrer un problème $\mathcal{NP}$-complet au sens ordinaire, [25].

Dans notre étude, nous souhaitons comparer un algorithme mémétique (noté MA ici et GA dans l'article d'origine) et un scatter search (noté SS). Nous avons

utilisé deux jeux tests, ceux provenant de la OR-Library (ORLIB) et ceux que nous avons créé (ODD). Les instances ODD sont tirées de Sevaux et Sörensen [111]. Pour chacun des exemples, le critère d'arrêt des algorithmes est 10 minutes ou 10000 itérations sans amélioration du meilleur individu de la population. Tous les autres paramètres sont détaillés dans Sevaux et Thomin [119]. Les tables 5 et 6 présentent les résultats de cette étude. L'algorithme mémétique est toujours limité par le nombre d'itérations tandis que le scatter search atteint toujours la limite de temps. Les colonnes 2 et 3 des tables présentent donc soit le temps total utilisé (MA) soit le nombre d'itérations avant arrêt (SS). La colonne $MA = SS$ indique le nombre de fois où les deux méthodes trouvent le même résultat, et la colonne $MA > SS$ le nombre de fois où le scatter search obtient de meilleurs résultats (strictement) ; l'inverse ne se produit jamais.

TAB. 5 – Résultats sur ORLIB extraits de Sevaux et Thomin [119]

Instances	CPU MA	it. SS	$MA = SS$	$MA > SS$	Sol.
ORLIB_40	3.89s	4836	119	6	125
ORLIB_50	8.16s	2349	0	125	125
ORLIB_100∗	58.36s	2	42	62	54

∗Time limit = 120s

Dans la table 6 deux colonnes supplémentaires "MA best" et "SS best" indiquent à quel instant dans la recherche les meilleures solutions ont été trouvées.

TAB. 6 – Résultats sur ODD extraits de Sevaux et Thomin [119]

Inst.	CPU MA	it. SS	MA best	SS best	$MA = SS$	$MA > SS$
ODD_20∗	13.34s	8610	0.4s	0.0s	17	0
ODD_40	5.60s	1253	0.6s	3.1s	17	3
ODD_60	21.21s	291	4.6s	19.1s	15	5
ODD_80	62.68s	99	9.2s	248.9s	9	11
ODD_100	111.16s	43	33.6s	289.0s	9	10

∗SS cannot generate 10 diverse solutions for 3 instances

Ce qu'on peut conclure de ces résultats c'est la supériorité du scatter search, *avec des paramètres identiques* à l'algorithme génétique hybride. Il est impossible de généraliser sur un seul exemple et conclure qu'on ne devrait utiliser que le scatter search est faux. La difficulté d'implémentation est tout de même plus grande pour SS et les bons résultats sont obtenus avec un plus grand délai. Par contre on peut tout de même penser que mesurer la diversité de la population est un point important dans la résolution.

Lors de l'écriture de Sörensen et Sevaux [122], pour prouver le bon fonctionnement de GA|PM, nous avons effectué une comparaison entre le même algorithme mémétique (MA) et le GA|PM. Dans l'article d'origine l'algorithme mémétique est noté HGA. Les temps CPU ont été volontairement limités à une minute. Les

tables 7 et 8 présentent les résultats de cette étude.

Pour chaque groupe d'instances de la OR-Library (table 7), on retrouve le nombre de solutions optimales (ou meilleures solutions connues) retrouvées par chaque méthode ainsi que le nombre de fois où la méthode arrive en première position seule. L'écart moyen constaté entre la solution trouvée et la solution optimale (ou meilleure solution) ainsi que le nombre moyen et maximum d'itérations est reporté.

TAB. 7 – Résultats sur ORLIB extraits de Sörensen et Sevaux [122]

ORLib40 results					
Method used	Opt. sol.	First pos.	Avg. Gap (in %)	Iterations Avg.	Max
GAPM	125	4	0.000	5682	8961
MA	121	0	0.284	327763	470984
ORLib50 results					
Method used	Opt. sol.	First pos.	Av. Gap (in %)	Iterations Aver.	Max
GAPM	123	11	0.002	3334	5298
MA	113	0	0.595	196024	287802
ORLib100 results					
Method used	Best. sol.	First pos.	Av. Gap (in %)	Iterations Aver.	Max
GAPM	94	27	0.276	1030	1480
MA	85	16	2.110	34164	56521

Dans la table 8, on retrouve les mêmes informations présentées de manière différentes (à l'exception des optima qui ne sont pas connus ici, l'écart est donc mesuré par rapport à la meilleure solution connue).

TAB. 8 – Résultats sur ODD extraits de Sörensen et Sevaux [122]

	GAPM results			MA results		
Set of inst.	First pos.	Avg. gap (in %)	Avg. iter.	First pos.	Avg. gap (in %)	Avg. iter.
ODD20	0	0.000	32375	0	0.000	952757
ODD40	0	0.000	4142	0	0.000	258526
ODD60	6	0.088	1685	1	0.251	88106
ODD80	6	0.003	1153	1	0.083	39040
ODD100	8	0.064	844	4	0.118	19854
Global	20	0.057	8040	6	0.148	271657

Sur ce problème spécifique et avec les réglages que nous avons choisi (communs aux deux méthodes), GA|PM prouve sa supériorité par rapport à l'algorithme génétique hybride. Pour des commentaires détaillés, il faut consulter Sörensen et Sevaux [122].

Pour compléter cette étude, nous avons souhaité comparer le scatter search (SS) et GA|PM. Ces tests ont été faits spécifiquement pour cette synthèse et n'ont pas été exécutés sur le même ordinateur que les tests précédents. Bien que les conditions d'arrêt (60 secondes) et les autres paramètres soient identiques aux autres tests, les valeurs trouvées (notamment les optima retrouvés pour ORLIB) ne sont pas toujours les mêmes.

Dans la table 9, on constate que bien que le temps de résolution soit court, ce qui est en général défavorable au SS, il conserve un léger avantage. Le nombre de fois où SS trouve une meilleure solution (strictement) que GA|PM est supérieur pour les grandes instances (100 jobs) et les écarts mesurés à la meilleure solution sont aussi plus petits (0.011% pour SS contre 0.128% pour GA|PM). Pour le cas particulier du Scatter Search, nous avons ajouté le nombre moyen de crossover (colonne Xov.) qui donne une valeur du même ordre de grandeur que les itérations du GA|PM.

TAB. 9 – Résultats comparatif de SS et GA|PMs sur ODD

	GAPM results			SS results			
Set of inst.	First pos.	Avg. gap (in %)	Avg. iter.	First pos.	Avg. gap (in %)	Avg. iter.	Avg. Xov.
ODD20	0	0.000	33496	0	0.000	168	25509
ODD40	0	0.000	4142	0	0.000	31	5710
ODD60	0	0.057	1685	2	0.000	8	1859
ODD80	2	0.000	1153	0	0.013	3	1021
ODD100	3	0.128	844	7	0.010	1	938
Global	5	0.112	6070	9	0.011	31	5405

Dans la table 10, la tendance s'inverse et GA|PM prend légèrement le dessus sur SS. Le nombre de solutions optimales (ou meilleures solutions) retrouvé est toujours plus important pour GA|PM et en conséquence l'écart à la solution optimale (ou meilleure solution) est plus important pour SS.

On peut noter aussi que nous avons compté le nombre d'itérations de SS en comptant les itérations de la boucle extérieure. Pour corriger cette erreur, nous avons aussi ajouté le nombre de nouvelles solutions générées par croisement (colonne Xov.). Ce nombre est du même ordre de grandeur que le nombre d'itérations de GA|PM. En comparant les valeurs obtenues, on remarque que pour un même temps d'exécution, le nombre d'itérations de SS est souvent plus grand.

TAB. 10 – Résultats comparatif de SS et GA|PM sur ORLIB

ORLib40 results

Method used	Opt. sol.	First pos.	Avg. Gap (in %)	Iterations Avg.	Iterations Max	Xov. Avg.
GAPM	120	2	0.000	5772	8961	
SS	118	0	0.125	44	99	15703

ORLib50 results

Method used	Opt. sol.	First pos.	Av. Gap (in %)	Iterations Aver.	Iterations Max	Xov. Avg.
GAPM	120	4	0.002	3380	5298	
SS	117	1	0.011	20	36	9406

ORLib100 results

Method used	Best. sol.	First pos.	Av. Gap (in %)	Iterations Aver.	Iterations Max	Xov. Avg.
GAPM	94	44	0.276	1030	1480	
SS	73	11	2.008	2	3	1905

Tournées de véhicules sur arcs

Le même type de comparaison est conduit sur un problème différent. Il s'agit du *Capacitated Arc Routing Problem* (CARP). Dans ce problème, une flotte de véhicules identiques partent d'un dépôt, collectent des marchandises (le long des arcs d'un graphe définissant le réseau) et reviennent au dépôt lorsqu'ils sont remplis. L'objectif est de minimiser le trajet total parcouru par l'ensemble des véhicules. Pour une description complète du problème lire Lacomme et al. [69]. Les meilleurs résultats sont proposés par Lacomme et al. [68] et pour d'autres techniques, on peut se reporter à Greistorfer [52] qui proposent un "tabu scatter search" pour résoudre le problème.

Dans la table 11 on retrouve les résultats de cette étude sur les instances de Golden et al. [50]. La première colonne indique les algorithmes utilisés. SMA correspond à un algorithme mémétique simple. BMA est le meilleur algorithme résolvant ce problème aujourd'hui (voir Lacomme et al. [68]) et c'est un algorithme mémétique dans lequel une technique simple est utilisée pour disperser la population. L'indicateur "nr" est présent quand les restarts n'ont pas été utilisés (pas de redémarrage de l'algorithme avec une partie de la population renouvelée). Pour chaque algorithme, la déviation à la meilleure borne connue (donnée par Belenguer et Benavent [6]) est donnée en moyenne et au pire. La colonne "LB hits" donne le nombre de fois où la borne inférieure est atteinte et la colonne suivante le temps CPU en secondes. La colonne "Restarts" indique le nombre d'instances nécessitant des restarts et entre parenthèses le nombre d'instances pour qui les restarts ont conduit à l'optimum. La dernière colonne indique

le nombre moyen de croisement effectués.

TAB. 11 – Résultats du CARP extraits de Prins et al. [93]

Algorithm	Dev. LB	Worst	LB hits	Av. time	Restarts	Av. Xovers
SMA	0.65	4.07	15	0.37	0	2750.5
BMA-nr	0.33	2.23	18	0.95	0	3013.1
GAPM-nr	0.24	2.23	20	0.90	0	880.9
BMA	0.17	2.23	21	4.79	5(3)	9960.2
GA|PM	0.17	2.23	21	1.59	3(1)	1968.9

SMA est très rapide mais donne des résultats loin des meilleurs. Sans les restarts, GA|PM est bien meilleur que BMA en terme d'écart et en nombre de solutions optimales retrouvées. Si on autorise les restarts, le même nombre de solutions optimales est trouvé, mais GA|PM garde un avantage en temps de résolution. Le nombre d'instances pour lesquelles les restarts sont nécessaires est moins important. C'est donc la meilleure méthode résolvant ces instances aujourd'hui.

5 Compléments

Il existe bien d'autres méthodes qui auraient pu figurer dans ce document. Nous les avons volontairement ou non écartés. D'un côté, certaines de ces méthodes n'ont pas prouvé leur efficacité et d'autres peuvent se trouver être des cas particuliers des méthodes déjà présentées. Pour toutes les méthodes que nous avons oublié et pour les auteurs qui auraient souhaité avoir leur place dans ce document de synthèse, nous présentons nos plus plates excuses. Avant de conclure, nous pensons qu'il est important de mettre en avant des points relatifs aux métaheuristiques que nous avons abordés au cours des dernières années et qui sont pour certains un nouveau domaine d'investigation ou pour d'autres un champ d'application et de recherche à part entière.

Ci-dessous, nous soulevons un certain nombre de particularités et de questions qui nous paraissent intéressantes et auxquelles nous tenterons de répondre dans la conclusion de cette synthèse.

Codage d'une solution

L'implémentation de chacune des métaheuristiques des sections précédentes peut parfois poser des problèmes. En premier lieu, et quelque soit la méthode utilisée, le codage de la solution est le point crucial de la réussite de l'approche. Par un codage approprié, qu'il soit direct ou indirect, on manipule plus ou moins facilement les solutions de notre problème, on est plus ou moins rapidement capable

d'évaluer la qualité d'une solution et on peut extraire plus ou moins d'informations intéressantes sur l'espace des solutions.

Voisinage

En ce qui concerne les méthodes de recherche locale, la définition d'un bon voisinage est particulièrement sensible. Il est par exemple très important de savoir si le voisinage est connexe ou non. Et s'il ne l'est pas, comment être certain de ne pas passer à côté de la solution optimale ? La taille du voisinage en lui-même est un critère intéressant. Si le voisinage est petit, on va pouvoir l'explorer tout entier, mais on aura besoin d'énormément d'itérations pour explorer un espace des solutions le plus large possible. Si le voisinage est grand, le nombre d'itérations nécessaire sera sans doute plus petit, mais le temps nécessaire pour parcourir ce voisinage peut être exponentiel...

Opérateurs de croisement et de mutation

Pour les méthodes à population (à l'exception des colonies de fourmis), les opérateurs de croisement et de mutation sont toujours difficiles à choisir. Il existe heureusement dans beaucoup de domaines (comme par exemple en ordonnancement, Caux et al. [11], Portmann [91]) des études sur les différents opérateurs et leur performance en fonction du codage et du problème lui-même. Une caractéristique importante d'un opérateur de croisement est qu'il devrait "en principe" produire un enfant au moins aussi bon que les parents.

Mesure de la diversité

Une autre question importante à nos yeux et indispensable pour le scatter search et le GA|PM est la définition d'une mesure de distance entre individus. Celle-ci doit être capable de refléter une similitude ou une différence entre deux individus. On pourra alors en déduire la distance d'un individu à la population toute entière. En fonction du problème, du codage des individus et de la distance retenue, l'évaluation peut-être plus ou moins coûteuse en temps de calcul. Mais parfois il peut-être bon de passer du temps pour calculer cette distance si elle conduit au final à une meilleure diversité de la population. Certaines distances, principalement pour les codages de type permutation, sont décrites dans Campos et al. [10], Sörensen [120], Sörensen et Sevaux [122].

5.1 Réglages automatiques des paramètres

Passer de la théorie à la pratique est souvent difficile. L'implémentation de toutes ces métaheuristiques pose des problèmes insoupçonnables. Un des soucis, et sans doute pas le moindre, est la compétence informatique des chercheurs opérationnels. Savoir programmer dans un langage informatique de 4^{e} génération

(quel qu'il soit) est très important. Si on suppose que cette compétence est acquise, alors un autre problème va surgir : le réglage des paramètres.

Prenons quelques exemples. Pour le recuit simulé, il faut trouver la température de départ, et le schéma de variation de cette température au cours des itérations. Pour la recherche tabou, la longueur de la liste tabou est importante. Trop grande, celle-ci conduit à un blocage où dans le pire des cas tous les voisins d'une solution sont tabous ; trop petite, celle-ci ne peut éviter les cyclages et la méthode parcoure les mêmes solutions indéfiniment.

Nous proposons dans Sevaux et Thomin [115, 119] la mise en place de techniques de réglages automatiques de ces paramètres. Pour le recuit simulé, notre expérience est plutôt défavorable. Le but recherché était d'initialiser automatiquement la température de départ pour donner environ une chance sur deux à une solution dégradante d'être acceptée. Malheureusement, les liens avec le schéma de refroidissement de la température sont trop forts et ce réglage automatique dans certains cas nous a déçu. Pour la recherche tabou, nous sommes plus enthousiastes. Dans la méthode présentée par Sevaux et Thomin [115], une détection automatique de cyclage est effectuée en permanence et en cas de détection, la longueur de la liste est augmentée de la longueur du cycle. Ensuite la longueur est progressivement diminuée pour éviter de conserver des éléments non-nécessaires qui augmentent le temps de calcul de la vérification du statut tabou. Cette méthode s'apparente un peu à la recherche tabou réactive proposée par Battiti et Tecchiolli [3].

5.2 Robustesse

La recherche de solutions robustes à un problème d'optimisation semble être un challenge récent pour les chercheurs. Longtemps laissé aux statisticiens ou stochasticiens, ce domaine est en train de voir sa popularité augmenter dans la communauté de la recherche opérationnelle, Sevaux et Sörensen [110].

Quand le problème est déterministe c-à-d. que les données sont connues et fiables à 100%, la plupart du temps le problème sous-jacent est un problème d'optimisation $\mathcal{NP}$-difficile. Il n'existe alors pas d'algorithme efficace pour le résoudre optimalement. Les métaheuristiques deviennent un outil de résolution privilégié garantissant à la fois un temps de développement réduit et l'obtention d'une solution de bonne qualité. Même si les métaheuristiques et particulièrement les algorithmes génétiques sont appliqués souvent et avec succès pour traiter des problèmes, leur application en environnement incertain est très limitée. Nous affirmons que les métaheuristiques peuvent être très facilement adaptées aux nécessités d'un problème stochastique. L'utilisation de métaheuristiques pour ce type d'optimisation présente un certain nombre d'avantages (voir Sevaux et Sörensen [111], Sevaux et al. [112]). Le besoin pour l'optimisation robuste par métaheuristiques a été reconnu dans le livre majeur sur l'optimisation robuste

[66], quand les auteurs énoncent page 354 ¡¡ *We believe that considerable more effort should be spent on the systematic development of [...] metaheuristic frameworks, which with minimal adjustment effort can be applied to a large class of robust optimisation problem [...]* ¿¿.

On trouve dans la littérature un nombre impressionnant d'articles utilisant les métaheuristiques pour résoudre des problèmes d'ordonnancement. Parmi les techniques les plus prisées, on retrouve en tête les algorithmes génétiques et la recherche tabou. L'article de Reeves [96] présente une liste non-exhaustive d'applications des algorithmes génétiques dans la plupart des domaines de l'optimisation combinatoire. Pour la robustesse en ordonnancement, nous avons tenté d'apporter un certain nombre de réponses en utilisant le cadre général proposé par Sörensen [121] et que nous avons appliqué à un problème d'ordonnancement Sevaux et Le Quéré [108], Sevaux et al. [112]. En utilisant des particularités stochastiques de notre problème, on peut intégrer ces dernières à une métaheuristique. On obtient alors une métaheuristique "stochastique" hybride qui semble particulièrement bien s'adapter au problème Fleury et al. [32].

Dans la gamme des problèmes de tournées de véhicules, nous tentons d'appliquer encore une fois le cadre général de Sörensen [121]. Ce type d'approche semble être satisfaisant pour la résolution des différentes variantes stochastiques du problème VRP (VRP avec temps de parcours stochastiques). Cette fois-ci nous nous appuyons sur un algorithme de type GA|PM que nous avons adapté pour obtenir des solutions robustes, Sörensen et Sevaux [123]. Cette technique, même si elle apporte des solutions satisfaisantes, doit être couplée avec une évaluation stochastique assez précise de la robustesse d'une solution comme dans Fleury et al. [31].

5.3 Optimisation multiobjectif

Plus brièvement, nous souhaitons mettre en avant un domaine nouveau pour nous, mais dont la communauté, plutôt constituée d'informaticiens, s'est ouverte au monde de la recherche opérationnelle récemment. C'est sans doute grâce aux efforts permanents de plusieurs personnes, comme Xavier Gandibleux, qui n'ont pas hésité pas à s'investir énormément sur cette thématique fort intéressante (voir Gandibleux [34]).

La résolution de tournées de véhicules sur arcs est un problème dont les applications sont particulièrement importantes pour les industriels. La première application que nous souhaitons résoudre est la collecte de déchets en milieu urbain. Après plusieurs conversations avec des industriels, il est apparu que l'objectif classiquement traité (la minimisation de la durée totale des tournées) n'était pas toujours l'objectif prioritaire. Dans certains cas, obtenir à la fois des durées totales minimales et un équilibre entre les tournées s'avérait plus intéressant dans la réalité. Nous avons traité ce problème en appliquant une variante de NSGA-II

(un algorithme génétique dédié à la résolution de problèmes multiobjectifs) et les résultats obtenus sont très satisfaisants (voir Lacomme et al. [72, 73]).

5.4 Optimisation continue

La dernière application que nous souhaitons présenter sort totalement du cadre de l'optimisation combinatoire. La conception assistée par ordinateur en "design" automobile est partout.

Le procédé de création d'un véhicule prototype est très particulier. Dans un premier temps, un modèle est conçu sur ordinateur. A partir de ce modèle, le designer va fabriquer une maquette qu'il va re-modeler manuellement jusqu'à obtenir la forme désirée. Cette étape est indispensable car il est impossible au designer de se rendre compte du résultat sur un écran d'ordinateur. Une fois la maquette obtenue dans sa version définitive, par un procédé de *reverse engineering*, on mesure des quantités de points sur la maquette et on approxime ces points par de nouvelles courbes.

Trouver les paramètres de ces courbes est extrêmement difficile et les réglages manuels sont souvent impossibles à reproduire plusieurs fois à l'identique. Avec Y. Mineur, nous avons mis au point un algorithme génétique continu (basé sur les travaux de Chelouah et Siarry [13]) qui permet d'offrir une aide en CAO pour l'obtention de ces paramètres. L'ensemble des résultats est présenté dans Mineur et Sevaux [83] et dans Sevaux et Mineur [109].

6 Conclusions

Après ce tour d'horizon assez vaste sur les métaheuristiques et un regard sur les différentes applications que nous avons pu traiter, sommes-nous aujourd'hui capable de définir une métaheuristique universelle ? Probablement pas ! Serait-ce alors un constat d'échec ? Pas plus !

Ce que nous allons proposer c'est plutôt des indications sur les caractéristiques d'une bonne métaheuristique à notre sens. Voici donc des éléments importants à nos yeux :

Population Les méthodes à population ont un avantage certain, celui du parallélisme intrinsèque qu'elles induisent. Travailler sur un ensemble de solutions qui partagent des informations, c'est explorer un plus grand espace des solutions. De plus, l'échange d'informations entre les individus de différentes régions de l'espace des solutions est souvent bénéfique.

Codage Le codage retenu pour la représentation de la solution est un point clé comme on a pu le voir. Il peut être direct ou indirect (phénotypique ou génotypique comme diraient les promoteurs des algorithmes génétiques).

De notre point de vue et expérience, un codage indirect est souvent plus facile à manipuler (du point de vue de la structure informatique des données) mais nécessite un algorithme d'évaluation de la solution. C'est au concepteur de la méthode de trouver un juste équilibre entre les deux. D'expérience, un algorithme d'évaluation d'un individu de la population en $\mathcal{O}(n^2)$ reste raisonnable. A noter que l'on obtient de très bons résultats quand l'évaluation peut être optimale.

Coopération Les individus doivent être capables d'échanger des informations entre eux par croisement par exemple. Le croisement est souvent fait en prenant les individus deux à deux, mais comme le propose le scatter search, ce n'est pas une obligation et plusieurs individus peuvent coopérer pour fournir de nouvelles solutions. L'idéal serait de pouvoir récupérer systématiquement les bonnes caractéristiques des parents pour les transmettre aux enfants. Mais quelles sont ces bonnes caractéristiques, souvent on ne le sait pas. Un point que soulignent Hertz et Widmer [60] et sur lequel nous sommes d'accord, c'est que si l'on croise deux parents identiques, il semble raisonnable d'obtenir un enfant qui soit identique aux deux parents. Cette propriété n'est malheureusement pas toujours respectée.

Recherche locale Ajouter une recherche locale est aujourd'hui indispensable. Les méthodes qui n'en sont pas dotées ne sont pas compétitives en terme de qualité de solutions proposées. Par contre, si le codage retenu est indirect, la recherche locale doit se faire en tenant compte de la solution elle-même. Un bon exemple pourrait être celui de l'algorithme mémétique de Lacomme et al. [69]. Le codage retenu est une permutation, mais la recherche locale est effectuée en prenant des voisinages classiques en "tournées de véhicules". Un mauvais exemple est celui de l'algorithme génétique proposé dans [119] où le codage d'une solution est une permutation (comme souvent en ordonnancement) et la recherche locale se fait en utilisant le voisinage *general pairwise interchange* (GPI) qui consiste à échanger deux éléments de la permutation sans intégrer des propriétés importantes du problème. Cette erreur n'a pas été reproduite dans [115].

Diversité contrôlée Mesurer la diversité entre les solutions est particulièrement important. Quand cela est intégré à un mécanisme de contrôle de la diversité, la métaheuristique peut se concentrer sur les individus divers et ne pas effectuer d'exploration inutile. Le contrôle de la diversité est un facteur tout aussi important. Il permet en fonction de l'ensemble des individus de la population de plus ou moins diversifier et de plus ou moins intensifier la recherche.

Restarts Les observations des techniques de "restart" implémentées, et notamment celles que nous avons testé dans Prins et al. [93], montrent qu'il est parfois envisageable de re-générer une partie de la population. En effet, après plusieurs générations, une partie de la population tend à être très performante, tandis qu'une autre partie se retrouve sans potentiel intéressant

pour des croisements futurs. En remplaçant une partie de cette population (la moins bonne de préférence) on donne un sang neuf à la population et on peut ainsi parfois améliorer les meilleures solutions.

Aléatoire Les plus grandes découvertes du monde l'ont été pour certaines d'entre elles par le fait du hasard. Se priver de tout facteur aléatoire implique qu'il faut remplacer cette part de chance par une énumération complète, que ce soit au niveau de la sélection des individus ou par exemple des points de croisement d'un opérateur de croisement. Par certains côtés, c'est ce que fait le scatter search, mais au prix toujours d'un temps de calcul souvent important.

Si ces caractéristiques sont les bonnes, alors l'algorithme GA|PM y répond quasiment en totalité et c'est désormais le seul que nous devrions utiliser. Non, surtout pas ! Ce serait sans doute une grave erreur. Ce que nous n'avons pas pris en compte dans les caractéristiques ci-dessus, c'est 1) le problème lui-même et 2) le but de la résolution.

- Le problème en lui-même est un point fondamental dans la résolution. Il peut être plus ou moins simple à coder, à comprendre et à résoudre. Il peut exister des spécificités à ce problème, comme des cas particuliers polynomiaux sur lesquels s'appuyer. Par exemple, pour un problème d'ordonnancement juste-à-temps, si on connaît les jobs en avance et ceux en retard, sous certaines conditions on peut évaluer optimalement cette configuration. Du problème et de ses caractéristiques, on va pouvoir choisir le codage le plus approprié, l'évaluation la meilleure, etc.
- Le but de la résolution est primordial et souvent ignoré. Dans le contexte académique, on cherche à fournir le plus souvent la meilleure solution au prix d'une implémentation parfois difficile avec des temps de calculs souvent longs et le résultat est l'amélioration des meilleures solutions mais des fois de quelques pour cents uniquement. Si on s'adresse à un industriel, la notion de solution optimale lui échappe souvent ou lui est indifférente. Il a besoin de savoir si l'implémentation va être robuste, combien de temps on va passer dessus, bref, si l'investissement en vaut la peine. Est-ce nécessaire de trouver la solution optimale à un problème industriel auparavant résolu manuellement et dont une solution obtenue par simple heuristique améliore un rendement de plus de 50% ? Martin Grötschel lors du Euro Winter Institute en Mars 2000 rapportait que pour un problème de transport de personnes handicapés (dial-a-ride problem) il avait trouvé la solution optimale en quelques semaines de travail mais avait mis plus de cinq ans à la faire appliquer pour ne pas modifier en profondeur le travail des conducteurs du jour au lendemain.

De toute notre expérience, le seul impératif qui ressort c'est qu'il faut essayer. En conférence, dans les présentations sur les métaheuristiques, les questions simples comme "Mais pourquoi n'avez-vous pas utilisé la méthode X plutôt que la méthode Y ?" ont heureusement pratiquement disparu. Un seul mot d'ordre

à cela, il faut essayer. On ne peut assurément pas répondre à une question demandant le résultat d'une autre métaheuristique sans l'avoir implémenté et testé auparavant.

7 Perspectives de recherche

Un bon indicateur des perspectives de recherche est le nombre de publications en cours ou en préparation [16, 108, 109, 112, 115, 122]. Pour certaines d'entre elles, le sujet est bien abouti et sera sans doute terminé après la publication d'un article en revue. Pour d'autres, il y a encore beaucoup à faire.

Nous allons ici lister les thèmes en cours d'étude ou qui présentent un intérêt particulier et qui seront privilégiés dans l'avenir.

Résolution de problèmes multiobjectifs
: Ce nouveau type de problèmes, abordé sur le cas particulier du CARP [70, 72], a permis de montrer les lacunes des méthodes existantes. On peut par exemple voir que beaucoup d'algorithmes génétiques multiobjectifs ont des fonctionnements simplistes et que ce qui existe pour les problèmes monoobjectif pourrait être transposé ici. Un exemple simple, les conditions d'arrêt des algorithmes multiobjectifs ne sont pas encore ou sont peu traitées.

Développement de méthodes robustes
: Les problèmes rencontrés en industrie ont rarement des données fiables. Pourtant, toutes les méthodes d'optimisation travaillent sur des données fixées et connues à l'avance. Les premiers travaux [108, 110, 112] en ordonnancement sont prometteurs mais une meilleure intégration des données et particulièrement au niveau statistique permettrait de proposer des méthodes plus fiables, et moins consommatrices de temps de calcul. Ce sujet est déjà bien avancé Fleury et al. [32].

Problèmes dans les graphes
: Avec C. Prins et P. Lacomme, nous avons mis en projet de rénover les approches algorithmiques sur les graphes en proposant d'importantes bibliothèques "open source" orientées objet permettant d'utiliser ces algorithmes dans la résolution de problèmes plus complexes. Ce travail est partiellement accompli dans Lacomme et al. [71]. Une version C++ utilisant la STL, plus générique, est aussi prévue en collaboration avec P. Thomin. Elle fera l'objet d'un second tome de ce livre.

Programmation linéaire
: Ce thème de prédilection tient toujours une place importante. On a noté que dans certains cas, l'espace des solutions est trop grand pour être traité avec un logiciel commercial. Une technique de coupes ou de décomposition pourrait alors être envisagée. Mais pourquoi ne pas utiliser la propagation

de contraintes pour réduire cet espace de solutions ? De plus, l'intégration de métaheuristiques permettrait aussi de guider les algorithmes de programmation linéaire vers des solutions prometteuses et pourrait accélérer la recherche. La collaboration avec C. Oğuz sur les problèmes de type flow-shop hybrides nous permettra de mieux cerner ces possibilités prochainement.

Mais en plus des thèmes ci-dessus, il y a deux autres domaines qui retiennent plus particulièrement notre attention.

En premier lieu, la collaboration avec Alcatel que nous menons en ce moment (conjointement avec Xavier Gandibleux) nous a entraîné sur un terrain particulièrement intéressant. Le sujet est l'optimisation du trafic dans les réseaux de télécommunications. Une clause de confidentialité nous engage à ne pas dévoiler le cœur du travail que nous menons, mais c'est un domaine nouveau pour nous et particulièrement intéressant. Le contrat d'un an que nous avons signé avec notre partenaire industriel sera très probablement renouvelé et la charge de travail étant importante, il me parait indispensable d'engager une thèse sur ce domaine. Les moyens financiers obtenus lors de cette collaboration sont importants et pourraient permettre de proposer à un jeune chercheur brillant un contrat de thèse pour réaliser cette étude avec nous.

Alcatel est un des organismes à la pointe du progrès dans le domaine étudié, il entend le rester et sait qu'il a besoin d'étendre ses collaborations avec le monde académique. Le soucis de confidentialité d'une telle entreprise est légitime. On ne peut défendre des intérêts industriels forts sans se protéger de la concurrence. Pourtant, Alcatel connaît aussi les nécessités et les impératifs de publication de notre communauté. Ainsi, lors d'autres contrats, Xavier Gandibleux a pu après dépôt d'un brevet dont il est co-auteur avec Alcatel publier le résultat de ses travaux. Engager une thèse sur un tel sujet, même sensible, n'est donc pas incompatible avec les devoirs d'un encadrant vis à vis de son doctorant (en terme de communications et de publications).

Le dernier point qui nous anime et qui constituerait un prolongement naturel de cette synthèse concerne les métaheuristiques évidemment.

Dans la suite logique de cette composition, et en collaboration avec C. Prins et K. Sörensen, nous allons préparer un ouvrage recensant les principales métaheuristiques comme ici, mais en mettant l'accent sur l'aspect implémentation qui nous parait indispensable et indissociable des méthodes elles-mêmes. A l'issue de ce travail, certains problèmes ou particularités des métaheuristiques devraient nous apparaître plus clairement encore.

En faisant cet inventaire ici, nous avons pu mettre en avant ce que pourraient être les bonnes caractéristiques d'une métaheuristique. De notre point de vue, il reste encore beaucoup de chemin à parcourir avant d'être en mesure de proposer une métaheuristique universelle. Pourtant, c'est notre but aujourd'hui : propo-

ser une méthode suffisamment générique pour résoudre le plus grand nombre de problèmes et pas trop générique pour que les paramètres à régler ne soient pas trop nombreux avant que l'on puisse atteindre un degré d'efficacité suffisant. Un deuxième pas à franchir ensuite sera de proposer, en plus du cadre général d'une métaheuristique, un code source générique et en accès libre et sous licence publique, qui permettra à tous d'utiliser les métaheuristiques sans avoir à refaire des implémentations lourdes et qui consomment beaucoup de temps et d'énergie. Réaliser tout ceci ne pourra se faire sans l'aide de plusieurs participants et notamment des jeunes chercheurs dont la curiosité et l'esprit vif sont les meilleurs atouts pour une plein réussite.

Ce projet important que nous suggérons ne va pas balayer tous les bons résultats de ces dernières années sur le sujet, mais au contraire s'appuiera dessus pour essayer de proposer une vision unifiée des métaheuristiques. Si ce but, peut-être idyllique, n'est pas facile à atteindre, au moins aurons-nous apporté une brique supplémentaire à l'édifice...

Références

[1] E.H.L. Aarts et J. Korst. *Simulated Annealing and Boltzmann Machines*. John Wiley, Chichester, 1989.

[2] J.T. Alander. An indexed bibliography of genetic algorithm implementations. Technical Report 94-1-IMPLE, Dept. of Information technology and production economics, University of Vaasa, Finland, July, 23 1999. `ftp.uwasa.fi`.

[3] R. Battiti et G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6 :126–140, 1994.

[4] E.B. Baum. Iterated descent : A better algorithm for local search in combinatorial optimization problems. Technical report, Caltech, Pasadena, CA, 1986.

[5] J. Baxter. Local optima avoidance in depot location. *Journal of the Operational Research Society*, 32 :815–819, 1981.

[6] J.M. Belenguer et E. Benavent. A cutting plane algorithm for the Capacitated Arc Routing Problem. *Computers and Operations Research*, 30(5) :705–728, 2003.

[7] P. Beullens, L. Muyldermans, D. Cattrysse, et D. Van Oudheusden. A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research*, 147 :629–643, 2003.

[8] C. Blum et A. Roli. Metaheuristics in combinatorial optimization : Overview and conceptual comparison. *ACM Computing surveys*, 35(3) :268–308, 2003.

[9] E. Bonomi et J.L. Lutton. The N-city travelling salesman problem, statistical mechanics and the metropolis algorithm. *SIAM Review*, 25(4) :551–568, 1984.

[10] V. Campos, M. Laguna, et R. Martí. Context-independent scatter and tabu search for permutation problems, Unpublished manuscript, 2002. URL `http://leeds.colorado.edu/Faculty/Laguna/publications.htm`.

[11] C. Caux, H. Pierreval, et M.C. Portmann. Les algorithmes génétiques et leur application aux problèmes d'ordonnancement. In *Actes des Journées d'étude Ordonnancement et Entreprise : Applications Concrètes et Outils pour le Futur*, pages 5–45, Toulouse, 1994. CNRS.

[12] R. Cerf. *Une théorie asymptotique des algorithmes génétiques*. PhD thesis, Université de Montpellier II, France, 1994.

[13] R. Chelouah et P. Siarry. A continuous genetic agorithm designed for the global optimization of multimodal functions. *Journal of Heuristics*, 6 :191–213, 2000.

[14] N.E. Collins, R.W. Eglese, et B.L. Golden. Simulated annealing : An annotated bibliography. *American Journal of Mathematical and Management Sciences*, 8 :209–307, 1988.

[15] D. Connolly. An improved annealing scheme for the qap. *European Journal of Operational Research*, 46 :93–100, 1990.

[16] S. Dauzère-Pérès et M. Sevaux. An efficient formulation for minimizing the number of late jobs in single-machine scheduling. Technical Report 98/9/AUTO, Ecole des Mines de Nantes, July 1998. Submitted.

[17] S. Dauzère-Pérès et M. Sevaux. Using lagrangean relaxation to minimize the weighted number of late jobs. *Naval Research Logistics*, 50(3) :273–288, 2003.

[18] K.A. De Jong. Genetic algorithms : A 25 year perspective. In J.M. Murata, R.J. Marks II, et C.J. Robinson, editors, *Computational intelligence-imitating life*, pages 125–134. IEEE Press, New York, 1994.

[19] M.L. den Besten, T. Stützle, et M. Dorigo. Design of iterated local search algorithms : An example application to the single machine total weighted tardiness problem. In *Proceedings of EvoStim'01*, Lecture Notes in Computer Science, pages 441–452, Springer, Berlin, 2001.

[20] D. De Werra et A. Hertz. Tabu search techniques : A tutorial and an application to neural networks. *OR Spektrum*, 11 :131–141, 1989.

[21] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992.

[22] M. Dorigo et G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, et F. Glover, editors, *New Ideas in Optimization*, pages 11–32, London, 1999. McGraw Hill.

[23] M. Dorigo, G. Di Caro, et L.M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5 :137–172, 1999.

[24] M. Dorigo, V. Maniezzo, et A. Colorni. Positive feedback as a search strategy. Technical Report 91-016, Politecnico di Milano, Italy, 1991.

[25] J. Du et J.Y.T. Leung. Minimizing total tardiness on one machine is $\mathcal{NP}$-hard. *Mathematics of Operations Research*, 15(3) :483–495, 1990.

[26] T.A. Feo et M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8 :67–71, 1989.

[27] T.A. Feo et M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6 :109–133, 1995.

[28] P. Festa. Greedy randomized adaptative search procedures. *AIRO News*, 7 (4) :7–11, 2003.

[29] P. Festa et M. Resende. GRASP : An annotated bibliography. In C. Ribeiro et P. Hansen, editors, *Essays and surveys in metaheuristics*, pages 3–21. Kluwer Academic Publishers, Boston, 2002.

[30] C. Fleurent et J.A. Ferland. Algorithmes génétiques hybrides pour l'optimisation combinatoire. *RAIRO Recherche Opérationnelle*, 30(4) :373–398, 1996.

[31] G. Fleury, P. Lacomme, et C. Prins. Evolutionary algorithms for stochastic arc routing problems. In G. Raidl, S. Cagnoni, J. Branke, D.W. Corne, R. Drechsler, Y. Jin, C.G. Johnson, P. Machado, E. Marchiori, F. Rothlauf, G.D. Smith, et G. Squillero, editors, *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2004*, volume 3005, pages 501–512, Coimbra, Portugal, 5-7 April 2004. Springer-Verlag.

[32] G. Fleury, Lacomme P, et M. Sevaux. Stochastic maintenance scheduling problem. In *Proceedings of ninth international conference on project management and scheduling, PMS 2004*, pages 405–409, Nancy, France, 26-28 April 2004.

[33] C. Gagne, M. Gravel, et W.L. Price. A new hybrid tabu-vns metaheuristic for solving multiple objective scheduling problem. In *Proceedings of 5th Metaheuristic International Conference, MIC 2003*, pages 22.1–7, Kyoto, Japan, 25-28 August 2003.

[34] X. Gandibleux. Optimisation multiobjectif : problèmes combinatoire, résolutions exactes et approchées, applications. Habilitation à diriger les recherches, Université Valenciennes, France, 12 December 2003.

[35] M.R. Garey et D.S. Johnson. *Computers and Intractability : A Guide to Theory of NP-Completeness*. Freeman, San Franscisco, 1979.

[36] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8 :156–166, 1977.

[37] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13 :533–549, 1986.

[38] F. Glover. Tabu search – part I. *ORSA Journal on Computing*, 1 :190–206, 1989.

[39] F. Glover. Tabu search – part II. *ORSA Journal on Computing*, 2 :4–32, 1990.

[40] F. Glover. Tabu search : A tutorial. *Interface*, 20(1) :74–94, 1990. Special issue on the practice of mathematical programming.

[41] F. Glover. Genetic algorithms and scatter search - unsuspected potentials. *Statistics and Computing*, 4 :131–140, 1994.

[42] F. Glover. *A Template for Scatter Search and Path Relinking*, volume 1363 of *Lecture Notes in Computer Science*, pages 1–53. Springer, Springer, Berlin, 1997.

[43] F. Glover. A template for scatter search and path relinking. In J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, et D. Snyers, editors, *Artificial Evolution*, volume 1363 of *Lecture Notes in Computer Science*, pages 13–54, Springer, Berlin, 1998. Springer.

[44] F. Glover et S. Hanafi. Tabu search and finite convergence. *Discrete Applied Mathematics*, 119 :3–36, 2002.

[45] F. Glover et M. Laguna. *Tabu Search*. Kluwer, Boston, 1999.

[46] F. Glover, M. Laguna, et R. Martì. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39(3) :653–684, 2000.

[47] F. Glover, M. Laguna, E. Taillard, et D. De Werra, editors. *Tabu search*, volume 41. Annals of Operations Research, 1993.

[48] D. Goldberg. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, 1987.

[49] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, 1989.

[50] B.L. Golden, J.S. DeArmon, et E.K. Baker. Computational experiments with algorithms for a class of routing problems. *Computers and Operations Research*, 10(1) :47–59, 1983.

[51] R.L. Graham, E.L. Lawler, J.K. Lenstra, et A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling : A survey. *Annals of Discrete Mathematics*, 5 :287–326, 1979.

[52] P. Greistorfer. A tabu-scatter search metaheuristic for the arc routing problem. *Computers and Industrial Engineering*, 44(2) :249–266, 2003.

[53] C. Guéret, C. Prins, et M. Sevaux. *Programmation linéaire*. Eyrolles, 2000. ISBN 2-212-09202-4, 365 pages (in French).

[54] C. Guéret, C. Prins, et M. Sevaux. *Applications of optimisation with Xpress-MP*. Dash optimization, 2002. ISBN 0-9543503-0-8, Translated by S. Heipke.

[55] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13 :311–329, 1988.

[56] P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy, 1986.

[57] P. Hansen et N. Mladenović. An introduction to variable neighbourhood search. In S. Voß, S. Martello, I.H. Osman, et C. Roucairol, editors, *Meta-Heuristics : Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458, Kluwer, Boston, 1999.

[58] R.L. Haupt et S.E. Haupt. *Practical genetic algorithm*. John Wiley & Sons, New York, 1998.

[59] A. Hertz, E. Taillard, et D. de Werra. *Local search in combinatorial optimization*, chapter A tutorial on tabu search, pages 121–136. J. Wiley & Sons Ltd, New York, 1997.

[60] A. Hertz et M. Widmer. Guidelines for the use of meta-heuristics in combinatorial optimization. *European Journal of Operational Research*, 151 :247–252, 2003.

[61] J.H. Holland. Adaptation in natural and artificial systems. Technical report, University of Michigan, Ann Arbor, 1975.

[62] J.H. Holland, editor. *Adaptation in natural and artificial systems : An introductory analysis with applications to biology, control, and artificial intelligence*. MIT Press/Bradford books, Cambridge, MA, 1992. 2nd edition.

[63] D. Johnson, C.H. Papadimitriou, et M. Yannakakis. How easy is local search ? *Journal of conputer and system sciences*, 17 :79–100, 1988.

[64] S. Kirkpatrick, C.D. Gelatt, et M.P. Vecchi. Optimization by simulated annealing. *Science*, 220 :671–680, 1983.

[65] C. Koulamas, S.R. Antony, et R. Jaen. A survey of simulated annealing applications to operations research problems. *Omega*, 22 :41–56, 1994.

[66] P. Kouvelis et G. Yu. *Robust Discrete Optimisation and its Applications*, volume 14 of *Nonconvex Optimization and its Applications*. Kluwer Academic Publishers, Dordrecht, 1997.

[67] J.R. Koza. *Genetic Programming*. MIT Press, Cambridge, MA, 1992.

[68] P. Lacomme, C. Prins, et W. Ramdane-Chérif. Competitive memetic algorithms for arc routing problems. *Annals of Operations Research*, 2004. To appear.

[69] P. Lacomme, C. Prins, et W. Ramdane-Chérif. A genetic algorithm for the Capacitated Arc Routing Problem and its extensions. In E.J.W. Boers et al., editor, *Applications of evolutionnary computing–EvoWorkshops 2001*, Lecture Notes in Computer Science 2037, pages 473–483. Springer, 2001.

[70] P. Lacomme, C. Prins, et M. Sevaux. Multiobjective capacitated arc routing problem. In *2nd Joint EU/ME workshop with the french PM2O group (MOMH Workshop) on multi-objective metaheuristics*, Paris, France, 4-5 November 2002.

[71] P. Lacomme, C. Prins, et M. Sevaux. *Algorithmes de graphes*. Eyrolles, 2003. ISBN 2-212-11385-4, 425 pages (in French).

[72] P. Lacomme, C. Prins, et M. Sevaux. Mutliple objective capacitated arc routing problem. In *Proceedings of 2nd International Conference on Evolutionary Multi-Criterion Optimization, EMO'2003*, pages 550–564, Faro, Portugal, 8-11 April 2003. LNCS 2632.

[73] P. Lacomme, C. Prins, et M. Sevaux. A genetic algorithm for a bi-objective capacitated arc routing problem. *Computers and Operations Research*, 2004. In Press.

[74] E.L. Lawler. A psedopolynomial time algorithm for sequencing jobs to minimize total atrdiness. *Annas of discrete mathematics*, 1 :331–342, 1977.

[75] J.K. Lenstra, A.H.G. Rinnooy Kan, et P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1 :343–362, 1977.

[76] H.R. Lourenço, O. Martin, et T. Stützle. Iterated local search. Technical report, Technical University of Darmstadt, Germany, 2000. Preprint.

[77] H.R. Lourenço, O. Martin, et T. Stützle. Iterated local search. In F. Glover et G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, Nowell, MA, 2002.

[78] S. Lundy et A. Mees. Convergence of an annealing algorithm. *Mathematical programming*, 34 :111–124, 1986.

[79] R. Martí, M. Laguna, et V. Campos. Scatter search vs. genetic algorithms : An experimental evaluation with permutation problems. In C. Rego et B. Alidaee, editors, *Adaptive Memory and Evolution : Tabu Search and Scatter Search*, 2002. To appear.

[80] O. Martin, S.W. Otto, et E.W. Felten. Large-step Markov chains for the traveling salesman problem. *Complex Systems*, 5 :299–326, 1991.

[81] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, et E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21 :1087–1092, 1953.

[82] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin, 1999.

[83] Y. Mineur et M. Sevaux. Curve fitting for styling application by genetic algorithm. In *3rd Joint EU/ME workshop with the university of Antwerp on Real-life application of metaheuritics*, Antwerp, Belgium, 18-19 December 2003.

[84] M. Mitchel. *An introduction to genetic algorithms*. MIT Press, Cambridge, MA, 1998.

[85] N. Mladenović et P. Hansen. Variable neighbourhood decomposition search. *Computers and Operations Research*, 24 :1097–1100, 1997.

[86] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts : Towards memetic algorithms. Technical Report C3P 826, Caltech Concurrent Computation Program, 1989.

[87] P. Moscato. Memetic algorithms : A short introduction. In D. Corne qnd M. Dorigo et F. Glover, editors, *New ideas in optimization*, pages 219–234. McGraw-Hill, New York, 1999.

[88] C.H. Papadimitriou et K. Steiglitz. *Combinatorial optimization – algorithms and complexity*. Prentice-Hall, Upper Saddle River, New Jersey, 1982.

[89] L. Paquete et T. Stützle. An experimental investigation of iterated local search for coloring graphs. In S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, et G. Raidl, editors, *Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002 : EvoCOP, EvoIASP, EvoSTim*, volume 2279, pages 121–130, Kinsale, Ireland, 2002. Springer-Verlag.

[90] M. Pirlot et R.V. Vidal. Simulated annealing : A tutorial. *Control & Cybernetics*, 25 :9–31, 1996.

[91] M.-C. Portmann. Genetic algorithm and scheduling : a state of the art and some propositions. In *Proceedings of the workshop on production planning and control*, pages I–XIV, Mons, Belgium, 1996.

[92] M.-C. Portmann. The quality of offsprings after crossover. Private communication, 2000.

[93] C. Prins, M. Sevaux, et K. Sörensen. A genetic algorithm with population management (GA|PM) for the carp. In *Proceedings of 5th trienal symposium on transportation analysis, TRISTAN V*, Le Gosier, Guadeloupe, France, 13-18 June 2004. To appear.

[94] C.R. Reeves. *Modern heuristics techniques for combinatorial problems*. John Wiley & Sons, New York, 1993.

[95] C.R. Reeves. A genetic algorithm for flowshop sequencing. *Computers and Operations Research*, 22 :5–13, 1995.

[96] C.R. Reeves. Genetic algorithms for the operations researcher. *INFORMS Journal on Computing*, 9 :231–250, 1997.

[97] H.-P. Schwefel. Evolution strategies : A family of non-linear optimization techniques based on imitating some principles of organic evolution. *Annals of Operations Research*, 1 :165–167, 1984.

[98] M. Sevaux. Les méthodes de recherche à voisinage. In *Groupe de Recherche en Productique – Organisation et Gesion de Production*, Toulouse, France, 8-9 November 2001.

[99] M. Sevaux. Les méthodes de recherche à population. In *Groupe de Recherche en Productique - Organisation et Gestion de Production*, Tarbes, France, 24-25 October 2002.

[100] M. Sevaux. Les méthodes de recherche à voisinage. Séminaire invité, Faculté Polytechnique de Mons, groupe Image (in French), 6 March 2002.

[101] M. Sevaux. Metaheuristics : a quick overview. Invited seminar, University of Antwerp, Faculty of Applied Economic Sciences, 7 February 2003.

[102] M. Sevaux. Population metaheuristics. Invited seminar, The Hong-Kong Polytechnic University, Department of Management, 14 March 2003.

[103] M. Sevaux et S. Dauzère-Pérès. Building a genetic algorithm for a single machine scheduling problem. In *Proceedings of the 18th EURO Winter Institute, ESWI XVIII*, Lac Noir, Switzerland, 4-18 March 2000.

[104] M. Sevaux et S. Dauzère-Pérès. A genetic algorithm to minimize the weighted number of late jobs on a single machine. In *Proceedings of 7th International Workshop on Project Management and Scheduling, PMS'2000*, pages 242–244, Osnabrük, Germany, 17-19 April 2000.

[105] M. Sevaux et S. Dauzère-Pérès. Genetic algorithms to minimize the weighted number of late jobs on a single machine. Technical Report LAMIH/SP-2000-51, Laboratory for Automation Mechanical engineering Information sciences and Human-machine systems, 2000.

[106] M. Sevaux et S. Dauzère-Pérès. Un algorithme génétique pour minimiser le nombre pondéré de jobs en retard sur une machine. In *3ième conférence nationale de la société française de recherche opérationnelle, ROADEF'2000*, Nantes, France, 26-28 January 2000.

[107] M. Sevaux et S. Dauzère-Pérès. Genetic algorithms to minimize the weighted number of late jobs on a single machine. *European Journal of Operational Research*, 151(2) :296–306, 2003.

[108] M. Sevaux et Y. Le Quéré. Solving a robust maintenance scheduling problem at the French railway company. Technical Report LAMIH/SP-2003-3, University of Valenciennes, CNRS UMR 8530, LAMIH/SP, April 2003. Submitted.

[109] M. Sevaux et Y. Mineur. Continuous optimisation by GA for a curve fitting problem in autmobile industry. Technical report, University of Valenciennes, CNRS UMR 8530, LAMIH/SP, 2004. Submitted.

[110] M. Sevaux et K. Sörensen. Genetic algorithm for robust schedules. In *Proceedings of 8th International Workshop on Project Management and Scheduling, PMS'2002*, pages 330–333, Valencia, Spain, 3-5 April 2002. ISBN 84-921190-5-5.

[111] M. Sevaux et K. Sörensen. A genetic algorithm for robust schedules in a just-in-time environment with ready times and due dates. *4OR – Quaterly journal of the Belgian, French and Italian Operations Research Societies*, 2(2) : 129–147, 2004.

[112] M. Sevaux, K. Sörensen, et Y. Le Quéré. *Flexibilité et robustesse en ordonnancement*, chapter Métaheuristiques pour la planification et l'ordonnancement robuste, pages 91–110. Hermes, 2004. To appear.

[113] M. Sevaux et P. Thomin. Efficient heuristic and tabu search for parallel machine scheduling. In *ORBEL Conference*, Antwerp, Belgium, 29-30 January 2001.

[114] M. Sevaux et P. Thomin. Heuristics and metaheuristics for parallel machine scheduling : a computational evaluation. In *Proceedings of 4th Metaheuristics International Conference, MIC 2001*, pages 411–415, Porto, Portugal, 16-20 July 2001.

[115] M. Sevaux et P. Thomin. Heuristics and metaheursitics for a parallel machine scheduling problem : a computational evaluation. Technical Report LAMIH/SP-2001-2, University of Valenciennes, CNRS UMR 8530, LAMIH/SP, November 2001. Submitted.

[116] M. Sevaux et P. Thomin. Parallel machine scheduling : a metaheuristic computational evaluation. Séminaire invité, Institut de Recherche en Communication et Cybernétique de Nantes, IRCCyN (in French), 19 April 2001.

[117] M. Sevaux et P. Thomin. Parallel machine scheduling : a (meta)heuristic computational evaluation. Séminaire invité, Groupe MOST (in French), 5 April 2001.

[118] M. Sevaux et P. Thomin. Recherche taboue améliorée pour l'ordonnancement sur machines parallèles. In *Actes de la 3ième conférence internationale francophone de MOdélisation et de SIMulation, MOSIM'01*, pages 829–833, Troyes, France, 25-27 April 2001. ISBN 1-56555-212-1.

[119] M. Sevaux et P. Thomin. Scatter search and ga : a one machine scheduling problem comparison. In *The sixteenth triennial conference of international federation of operational research societies, IFORS'2002*, Edinburgh, UK, 6-13 July 2002.

[120] K. Sörensen. Distance measures based on the edit distance for permutation-type representations. In A. Barry, editor, *Proceedings of the Workshop on Analysis and Design of Representations and Operators (ADoRo), GECCO Conference*, pages 15–21, Chicago, 2003.

[121] K. Sörensen. *A framework for robust and flexible optimisation using metaheuristics with application in supply chain design*. PhD thesis, University of Antwerp, 2003.

[122] K. Sörensen et M. Sevaux. GA|PM : genetic algorithms with population management. Technical report, University of Valenciennes, CNRS UMR 8530, LAMIH/SP, 2004. Submitted.

[123] K. Sörensen et M. Sevaux. Robust and flexible vehicle routing in practical situations. In *Proceedings of 5th triennial symposium on transportation analysis, TRISTAN V*, Le Gosier, Guadeloupe, France, 13-18 June 2004. To appear.

[124] E.G. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8 : 541–564, 2002.

[125] P.J.M. Van Laarhoven et E.H.L. Aarts. *Simulated Annealing : Theory and Applications*. Kluwer Academic Publishers, Boston, 1987.

[126] R.V. Vidal, editor. *Applied simulated annealing*, volume 396 of *LNEMS*. Springer-Verlag, Berlin, 1993.

[127] C. Voudouris. *Guided local search for combinatorial optimization problems*. PhD thesis, Dept. of Computer Science, University of Essex, Colcherster, UK, 1997.

[128] C. Voudouris et E. Tsang. Guided local search. Technical Report TR CSM-247, University of Essex, UK, 1995.

[129] C. Voudouris et E. Tsang. Guided local search and its application to the travelling salesman problem. *European Journal of Operational Research*, 113 : 469–499, 1999.

Troisième partie

Sélection de publications

Description des publications

Pour conclure ce document, nous proposons ici quelques publications. A l'exception d'une seule [3], toutes les références sont publiées [1, 5] ou à paraître [2, 4]. Elles sont à notre sens représentatives de notre activité récente.

Concernant les métaheuristiques proprement dites, [1] présente l'étude de plusieurs opérateurs d'un algorithme génétique. Le problème traité est celui de l'ordonnancement à une machine. [2] propose un algorithme génétique adapté pour résoudre le même problème d'ordonnancement dans le cas où les données sont incertaines. L'utilisation des métaheursitiques pour résoudre un problème bi-critère de tournées de véhicules sur arcs est présenté dans [4].

Un travail de plus grande envergure présenté dans [3] propose la description d'une nouvelle métaheuristique qui gère de manière précise la population. Même si cet article n'est pas encore accepté, il mérite d'être présenté car il se trouve être une nouvelle source de travail et une nouvelle piste importante de recherche.

Enfin, nous souhaitions présenter un travail déconnecté des métaheuristiques [5] mais qui nous tient à cœur particulièrement. La résolution par relaxation Lagrangienne d'un problème d'ordonnancement à une machine. Ce article est là pour rappeler qu'il n'existe pas que les métaheuristiques et qu'il est important de pouvoir proposer des techniques de résolution alternatives (ou pourquoi pas des techniques à hybrider avec les métaheuristiques).

1 European Journal of Operational Research (2003)

[1] M. Sevaux and S. Dauzère-Pérès. Genetic algorithms to minimize the weighted number of late jobs on a single machine. *European Journal of Operational Research*, 151(2) :296–306, 2003.

Abstract : The general one-machine scheduling problem is strongly $\mathcal{NP}$-Hard when the objective is to minimize the weighted number of late jobs. Few methods exist to solve this problem. In an other paper, we developed a Lagrangean relaxation algorithm which gives good results on many instances. However, there is still room for improvement, and a metaheuristic might lead to better results. In this paper, we decided to use a Genetic Algorithm (GA). Although a GA is somewhat easy to implement, many variations exist, and we tested some of them to design the best GA for our problem. Three different engines to evaluate the fitness of a chromosome are considered, together with four types of crossover operators and three types of mutation operators. An improved GA is also proposed by applying local search on solutions determined

from the chromosome by the engine. Numerical experiments on different tests of instances are reported. They show that starting from an initial population already containing a good solution is very effective.

2 4OR – Quaterly journal of the Belgian, French and Italian Operations Research Societies (2004 à paraître)

[2] M. Sevaux and K. Sörensen. A genetic algorithm for robust schedules in a just-in-time environment with ready times and due dates. *4OR – Quaterly journal of the Belgian, French and Italian Operations Research Societies*, 2004. In Press.

Abstract : Computing a schedule for a given single machine problem is often difficult for irregular criteria, but when the data are uncertain, the problem is much more complicated. In this paper, we modify a genetic algorithm to compute robust schedules when release dates are subject to small variations. Two types of robustness are distinguished : quality robustness or robustness in the objective function space and solution robustness or robustness in the solution space. The modified genetic algorithm is applied to a just-in-time scheduling problem, a common problem in several industries.

3 Journal of Heuristics (2004 soumis)

[3] K. Sörensen and M. Sevaux. GA|PM : genetic algorithms with population management. Technical report, University of Valenciennes, CNRS UMR 8530, LAMIH/SP, 2004. Submitted.

Abstract : We present a new type of evolutionary algorithms called genetic algorithms with population management or GA|PM . The main distinguishing feature of GA|PM is the use of distance measures to control the diversity of a small population. Combined with a local search procedure to ensure the high quality of individual solutions in the population, population management strategies can be developed to avoid slow or premature convergence and achieve excellent performance. The new algorithm is tested on two problems : the multidimensional knapsack problem and the weighted tardiness single machine scheduling problem. On both problems, population management is shown to be able to improve the performance of an evolutionary algorithm.

4 Computers and Operations Research (2004 à paraître)

[4] P. Lacomme, C. Prins, and M. Sevaux. A genetic algorithm for a bi-objective capacitated arc routing problem. *Computers and Operations Research*, 2004. In Press.

Abstract : The Capacitated Arc Routing Problem (CARP) is a very hard vehicle routing problem for which the objective — in its classical form — is the minimisation of the total cost of the routes. In addition, one can seek to minimize also the cost of the longest trip. In this paper, a multi-objective genetic algorithm is presented for this more realistic CARP. Inspired by the second version of the Non-dominated Sorted Genetic Algorithm framework, the procedure is improved by using good constructive heuristics to seed the initial population and by including a local search procedure. The new framework and its different flavour is appraised on three sets of classical CARP instances comprising 81 files. Yet designed for a bi-objective problem, the best versions are competitive with state-of-the-art metaheuristics for the single objective CARP, both in terms of solution quality and computational efficiency : indeed, they retrieve a majority of proven optima and improve two best-known solutions.

5 Naval Research Logistics (2003)

[5] S. Dauzère-Pérès and M. Sevaux. Using lagrangean relaxation to minimize the weighted number of late jobs. *Naval Research Logistics*, 50(3) :273–288, 2003.

Abstract : This paper tackles the general single machine scheduling problem, where jobs have different release and due dates and the objective is to minimize the weighted number of late jobs. The notion of *master sequence* is first introduced, i.e., a sequence that contains at least an optimal sequence of jobs on time. This *master sequence* is used to derive an original mixed-integer linear programming. By relaxing some constraints, it is possible to propose a Lagrangean relaxation algorithm which gives both a lower and upper bound. Although the duality gap becomes larger with the number of jobs, it is possible to solve problems of more than 100 jobs, and some computational results are presented.

Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 151 (2003) 296–306
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/dsw

Genetic algorithms to minimize the weighted number of late jobs on a single machine

Marc Sevaux [a,*], Stéphane Dauzère-Pérès [b]

[a] Université de Valenciennes, Mont Houy, CNRS, UMR 8530, LAMIH/SP, F-59313 Valenciennes Cedex, France
[b] IRCCyN, Ecole des Mines de Nantes, CNRS, UMR 6597, La Chantrerie, BP 20722, F-44307 Nantes Cedex 3, France

Abstract

The general one-machine scheduling problem is strongly $\mathcal{NP}$-Hard when the objective is to minimize the weighted number of late jobs. Few methods exist to solve this problem. In an other paper, we developed a Lagrangean relaxation algorithm which gives good results on many instances. However, there is still room for improvement, and a meta-heuristic might lead to better results. In this paper, we decided to use a genetic algorithm (GA). Although a GA is somewhat easy to implement, many variations exist, and we tested some of them to design the best GA for our problem. Three different engines to evaluate the fitness of a chromosome are considered, together with four types of crossover operators and three types of mutation operators. An improved GA is also proposed by applying local search on solutions determined from the chromosome by the engine. Numerical experiments on different tests of instances are reported. They show that starting from an initial population already containing a good solution is very effective.

Keywords: Scheduling; Single machine; Weighted tardy jobs; GA; Hybrid GA

1. Introduction

The general one-machine sequencing problem can be described as follows: a set J of n jobs ($\{J_1,\ldots,J_n\}$) has to be scheduled on a single machine, and each job J_i has a release date r_i, a processing time p_i and a due date d_i. Moreover, the machine can only process one job at a time, and preemption is not allowed. A scheduled job completed before its due date is said to be *early* (or *on time*), and *late* otherwise. Finally, a weight w_i is associated to each job J_i, and the objective in this paper is to minimize the weighted number of late jobs (or equivalently to maximize the weighted number of early jobs).

Remark 1. A well-known and important remark is that there is always an optimal schedule in which late jobs are sequenced after all the early jobs. A partial sequence of early jobs describes completely a feasible solution.

This single machine scheduling problem, noted $1|r_j|\sum w_jU_j$ in the standard classification [10], is strongly $\mathcal{NP}$-Hard [14]. When release dates are all

* Corresponding author.
E-mail addresses: marc.sevaux@univ-valenciennes.fr (M. Sevaux), stephane.dauzere-peres@emn.fr (S. Dauzère-Pérès).

0377-2217/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0377-2217(02)00827-5

equal to zero ($1\|\sum w_jU_j$), the problem is still $\mathcal{NP}$-Hard. When all weights are equal ($1|r_j|\sum U_j$), the problem remains $\mathcal{NP}$-Hard, but can be solved by an $O(n\log n)$ algorithm if all release dates are equal ($1\|\sum U_j$) [15]. If release and due dates are similarly ordered ($r_i < r_j \Rightarrow d_i \leqslant d_j$, $\forall(J_i,J_j)$) Kise et al. [12] gave an $O(n^2)$ algorithm and Lawler [13] an $O(n\log n)$ algorithm for the same special case.

However, some exact approaches have recently been proposed for the problem $1|r_j|\sum U_j$ [2,6]. For $1\|\sum w_jU_j$, Lawler [13] showed that Moore's algorithm ([15]) could be applied when processing times and weights are agreeable ($p_i < p_j \Rightarrow w_i \geqslant w_j$, $\forall(J_i,J_j)$). Finally, branch-and-bound procedures have been developed to solve the case where all release dates are equal ($1\|\sum w_jU_j$) [21,17].

A new and efficient Lagrangean relaxation algorithm (LRA) has been presented by Péridy et al. [16] and their lower bound will be used in the computational experiments. In a recent work, Dauzère-Pérès and Sevaux [7] developed a Lagrangean relaxation heuristic to solve the problem in the general case. Because the Lagrangean relaxation heuristic gives at the same time a feasible solution (an upper bound) and a lower bound, a measure of the quality of a solution can be given for each instance. The same heuristic can be used to solve the problem when weights are equal and thus has been compared to optimal solutions given by a branch-and-bound algorithm [6]. Because we used generic parameters for the LRA (the parameters are the same for all the randomly generated instances), in some test problems, the gap between the solution and the lower bound is rather large and can be reduced.

This paper studies different classical genetic algorithm (GA) techniques, combines them and presents the first hybrid genetic algorithm (HGA) that solves efficiently the $\mathcal{NP}$-Hard problem $1|r_j|\sum w_jU_j$. An initial solution given by a LRA is used as a good starting point but, when the size of the instances increases, the HGA solution almost always improves the best solution found so far. This suggests that the HGA is very efficient for large size instances.

Section 2 introduces some general considerations about GA. Section 3 is devoted to finding a good combination for the engine, and the crossover and mutation operators. The genetic local search (GLS) is presented in Section 4 and numerical experiments are reported in Section 5.

2. Genetic algorithms

The basic concepts of GAs, introduced by Holland [11], have been successfully applied to solve many combinatorial optimization problems. We refer the reader to a recent feature article from Reeves [20] for an extensive description of GAs in operations research.

2.1. General considerations

A GA tries to mimic the genetic behavior of a species. The biggest difference with other metaheuristics (like Tabu search (TS) or simulated annealing (SA)) is that GA maintains a *population* of solutions rather than a unique current solution. Solutions are coded as finite-length strings called *chromosomes* and a measure of their adaptation (the *fitness*) is computed by an *engine*. Starting from an existing population, each iteration generates new chromosomes by applying operators (like *crossover* and *mutation*) to two "randomly" chosen parents. The main advantage of a GA is its intrinsic parallelism, which allows the exploration of a larger solution space than in TS or SA. With an appropriate design, GAs can give very good results.

2.2. Chromosome encoding

In [18], Portmann describes how GAs can be applied to scheduling problems, and some very interesting hints for building a specific GA can be found.

Often, when a GA is implemented to solve a one-machine scheduling problem, the representation of a chromosome is a permutation of the jobs "$J_1, J_2, \ldots, J_n$". This representation is easy to handle and has been adopted for our study.

2.3. Population management

Starting from an initial population of size m, two ways of managing the population are usually considered: either a new population of size m is generated and replaces the previous population, or only one offspring is generated and replaces one chromosome in the existing population. The former method is named *population replacement* and the latter *incremental replacement*.

In population replacement, it may happen that the best chromosome in the initial population, i.e., the best current solution for our problem, is no longer in the new population, and thus will not be used again by the crossover operator. This is the reason why most researchers prefer incremental replacement, also called *elitism* or *population overlaps* [8]. In the sequel, only incremental replacement will be considered.

2.4. Reproduction mechanism

At every iteration, the selection of the two parents is done according to a simple ranking mechanism [19] which gives more chances to chromosomes with high fitness values to be used for reproduction. The population is sorted in ascending order of fitness values (ascending order of the objective function), so the best individual is ranked first. The selection of the first parent is made using the probability distribution $2(m+1-k)/(m(m+1))$ where k is the kth chromosome in the ascending order of fitness values, and m the size of the population. Using this technique, the median chromosome $(m+1)/2$ has the probability $1/m$ of being selected, while the best individual has the probability $2/(m+1)$ (roughly twice the median). The second parent is randomly selected with an uniform distribution.

A crossover operator is applied on the two parents and a new offspring is generated. Normally, two offspring are created but only one is kept in the algorithm. To keep the size of the population constant, the new offspring should replace another individual in the existing population. In our procedure, an individual is randomly chosen after the median and replaced by the new offspring. Afterwards, the population is sorted again according to the fitness values.

3. Designing a genetic algorithm

3.1. Fitness

For a given chromosome (i.e., a permutation of the n jobs), a feasible solution, corresponding to the sum of the weights of the late jobs, has to be computed, i.e., a feasible schedule has to be determined. This is called the engine. Three engines, denoted by "list", "non-delay (ND)" and "Fifo" have been tested. Since a chromosome is a permutation of n jobs, an engine is nothing else than a list algorithm.

The list engine is very simple. Jobs are taken in the order of the chromosome and sequenced consecutively when on time. A job which is late if sequenced at the current position is noted late, and will be sequenced after the jobs on time (see Remark 1). Note that, if a job has been sequenced at time t and has left an idle period before t, the list engine will never schedule a job in this idle period. This engine builds semi-active schedules, i.e., in which jobs are sequenced as soon as possible without changing the order of the jobs on the machine. Baker [1] named this engine the semi-active schedule generator. The evaluation function runs in $O(n)$.

The ND engine builds non-delay schedules as the Baker's non-delay schedule generator [1]. The idea is to sequence a job each time it is possible with no delay. When more than one job can be sequenced at the same time, the order given by the chromosome determines the jobs that will be sequenced first. Jobs that cannot be scheduled on time are sequenced after all the early jobs. This engine might exclude the optimal solution. This engine is actually a modification of the Giffler algorithm, originally designed for problems without release dates and regular criteria (see [9]). The complexity of the ND engine is $O(n^2)$.

The Fifo engine builds active schedules and is equivalent to the strict order generator proposed first by Carlier [4]. A list L of idle periods where

jobs can be scheduled is maintained. At the beginning, the list L is the interval $[0, +\infty[$. Each job in the chromosome is sequenced at the earliest time, and the list L of idle periods is updated. Jobs that can no longer be sequenced are late and delayed after the jobs on time. Once a job is sequenced on time, its starting time is fixed and the job cannot be moved in the remainder of the algorithm, i.e., push and pull operations are not allowed with this engine. The complexity of the engine is $O(n^2)$.

3.2. Crossover

To determine the best GA for our problem, four types of crossover operators have been tested. Those operators are commonly used in GAs and particularly in scheduling problems: "OX", "LOX", "PMX" (two-point crossover operators) and "X1" (or "1X") (one-point crossover operator) (see [18]).

In the following, two parent chromosomes with seven jobs are used, and Table 1 illustrates the outcome of each two-point crossover operator. For the two-point crossover operators, the first point is between positions 2 and 3 and the second point between positions 5 and 6. These operators create two offspring but only one is kept.

For OX, LOX and PMX crossover operators, two crossover points are randomly chosen. In the OX and LOX operators, the partial sequence of the first parent between these two points are copied into the new offspring. In the OX operator, the second parent is scanned from the index of the second crossover point and filled up with the jobs that are not already in the offspring, starting after the second crossover point. The LOX operator fills up the offspring in the same manner but starts in the offspring from index 0.

Table 1
Results of the two-point crossover operators

Parent 1	1	2	3	4	5	6	7
Parent 2	4	3	7	6	2	5	1
Offspring OX	6	2	3	4	5	1	7
Offspring LOX	7	6	3	4	5	2	1
Offspring PMX	6	7	3	4	5	2	1

The PMX operator (partial matching crossover) creates a matching between the two parents. In the example, job J_7 is replaced by job J_3 or the opposite, J_4 by J_6 and J_5 by J_2. Parent 2 is scanned from the index 0 and, whenever possible, a job is replaced by its matching. In the offspring, the first job of parent 2, J_4, is replaced by J_6, job J_3 by J_7, J_7 by J_3, etc.

The X1 operator (one-point crossover) is a particular case of the LOX operator where the first crossover point is the index 0.

3.3. Mutation

The selection process combined with the crossover operators help the population to converge. However, after many iterations in the algorithm, the population could be composed of local optima and needs some diversity to reach new points in the solution space. This is the role of the mutation operators. In this paper, three types of mutation operators are evaluated. The "swap" operator permutes two contiguous elements. The "big swap" operator permutes two randomly chosen elements of the chromosome. The "shift" operator takes an element and inserts it after a partial sequence which is then left-shifted.

3.4. Other parameters

For our study, the number of individuals in the population is set to $m = 5 + n/5$, and the *mutation rate* to 0.25 (i.e., on average, only 1 out of 4 chromosomes is mutated). Two chromosomes are identical if the permutation of the n jobs is the same. Two completely different chromosomes can have two identical fitness values. This is why we decided to allow a maximum of three clones (three identical fitness values in the population).

Due to Remark 1, in any solution, late jobs are always scheduled after the jobs on time. It is possible to keep that "order" by reordering the chromosome after the evaluation. Early jobs are scheduled first and late jobs after in the order of the evaluated chromosome. This feature (called *reorder*) will be taken into account for the combination of parameters.

For the stopping conditions, the maximum number of iterations grows with the number of

jobs (size of the instance) ItMax $= 50 \times n \times m$, and the maximum number of iterations without improvement is ItMax/10.

3.5. Choosing a combination of parameters

To run one version of the GA, an engine, a crossover operator and a mutation operator have to be chosen. Optionally, we can reorder a chromosome after its evaluation according to two partial sequences of early and late jobs. Three engines, four crossover operators and three mutation operators and the reorder option are available. Combining these parameters leads to 72 differents possible test runs.

To choose the best combination of parameters, we have generated a set of 1000, 100-job problems whose optimal solutions were known. The deviation from the optimal solution has been measured for each instance and used to determine the best operators and parameters.

The best combination is obtained when the *reordering* of a chromosome occurs at the end of its evaluation, associated with the X1 crossover operator that keeps a partial sequence of early jobs. The best engine is the Fifo engine, and the mutation operator big swap gives the best results.

The "ND" engine builds non-delay schedules, and for an irregular criteria it can exclude the optimal solution. Nevertheless, on average, this engine gives good results. Since our objective is to reach the optimal solution and not give a good result, we will not keep it for the final version of the GA. The list engine is too simple and cannot improve the sequences of early jobs even if there is a large idle period at the beginning. The Fifo engine fills up these idle periods when it is possible, but can be improved again.

Of course, these crossover operators could be used in a GA with different probabilities. For example, the ND engine or the Fifo engine could be chosen with probability 1/2, improving on average the solution (ND engine) and not excluding the optimal solution (Fifo engine). This is a topic for future investigations.

When the reorder parameter is active, the early jobs are always sequenced first. The X1 crossover operator keeps a partial sequence starting from the beginning of the chromosome i.e., a partial sequence of early jobs. This is one of the reasons why this combination works well.

The big swap mutation operator could be replaced by the shift mutation operator since the results are quite equivalent. The swap mutation operator does not perturbate enough the chromosome to escape from local optima.

4. Genetic local search

GAs are general search techniques. Sometimes a chromosome may lead to a solution very close to an optimal solution but, due to the myopic search of the GA, will not find this optimal solution. An hybrid genetic algorithm (HGA) would help in this case, i.e., a technique that combines at the same time a GA and another improvement method inside the GA.

4.1. Local improvement

The GLS chosen here, is a simple method that improves each chromosome of the GA. The structure of the GLS developed for our study is the following:

Step 1 Initialization: create an initial population of size m.
Step 2 Improvement (1): apply a local search to improve each chromosome of the initial population (each chromosome is now a local optimum).
Step 3 Recombination: using genetic operators (crossover, mutation) create an offspring.
Step 4 Improvement (2): apply a local search to improve the offspring.
Step 5 Replacement: replace a randomly chosen individual in the population by the new offspring.
Step 6 Stop: if stopping conditions are not met go to *Step 3*.

Performing the local search in *Steps 2* and *4* requires the definition of a neighborhood structure, and a strategy to search this neighborhood.

4.2. Neighborhood

Since the objective is to minimize the weighted number of late jobs, each chromosome is composed of a list of early jobs and a list of tardy jobs after the execution of a chosen engine. A neighbor of a current solution is obtained by exchanging a job in the list of *tardy* jobs with a job in the list of early jobs.

4.3. Strategy

Many local search procedures can be used in *Steps 2* and *4* of the GLS. Some authors tried Tabu Search or SA and obtained good results. The problem is that we do not want to spend too much CPU time in completing the local search, in order to ensure that the GA can still perform each iteration quickly enough. This is why we chose a descent.

To improve the current solution, a greedy algorithm is used. A move is made if and only if it strictly improves the current value of the objective function. The algorithm to reach a local optimum is described below.

Repeat the following steps until every late job has been examined:

1. Choose a job J_t in the "initial tardy list".
2. Start from the first job J_e in the "early list".
3. If J_t can replace J_e and the total weight of the tardy jobs decreases, set J_t on time and add J_e to the "final tardy list" (J_e is definitively late and will not be chosen again in 1), go to 1.
4. Choose the next job J_e in the early list and go to 3. If no such job exists, add J_t to the final tardy list.

The complexity depends on the number of jobs in the "tardy list". If t denotes this number, the complexity is $O(t(n-t))$ and the maximum is reached for $t = n/2$. The overall complexity of the local search procedure is $O(n^2)$.

5. Computational results

In this section, two sets of instances are used to evaluate our metaheuristic. The first set of instances has been used in the Lagrangean relaxation algorithm proposed in [7]. The second one comes from [3]. On this last set of instances, another very efficient lower bound from Péridy et al. [16] is used to measure the quality of our results.

5.1. Instance generator

The first set of instances has been used in [5–7]. It has been extended for the weighted case as follows. For each job J_i, a processing time p_i is randomly generated in the interval $[1, 100]$ and a weight w_i is generated in the interval $[1, 10]$ or in $[1, 99]$. As in [5], two parameters K_1 and K_2 are used, and taken in the set {1, 5, 10, 20}. Because we want data to depend on the number of jobs n, the release date r_i is randomly generated in the interval $[0, K_1 n]$, and the due date in the interval $[r_i + p_i, r_i + p_i + K_2 n]$. The algorithm was tested for $n \in \{20, 40, 60, 80, 100, 120, 140\}$. For each combination of n, K_1, and K_2, 10 instances are generated, i.e., for each value of n, 160 instances are generated with a weight randomly choosen in $[1, 10]$ and 160 instances with a weight in $[1, 99]$. In [7], weights are only generated in the interval $[1, 10]$.

5.2. Numerical results of GA and GLS

In this section, the results of the Lagrangean relaxation algorithm (named LRA in the sequel) are recalled. The genetic algorithm and the genetic local search are denoted by GA and GLS, respectively. To evaluate the quality of a solution, the usual relative gap (rg) between an upper bound (UB) and a lower bound (LB) is computed using the following formula:

$$\mathrm{rg} = \frac{\mathrm{UB} - \mathrm{LB}}{\mathrm{UB}}.$$

5.2.1. Results of the Lagrangean relaxation algorithm

Table 2 gives the relative gap for the upper bound obtained with the LRA. The first column gives the number of jobs. The next three columns give the average value, the standard deviation and the maximum value of rg. The last two columns

Table 2
Relative gap for the LRA

No. of jobs	Relative gap (rg)			CPU time (seconds)	
	Mean	StDev	Max	Mean	Max
20	0.10	0.01	0.49	0.82	2.52
40	0.09	0.01	0.64	2.98	8.31
60	0.13	0.01	0.61	6.86	18.45
80	0.17	0.04	0.71	12.07	32.12
100	0.22	0.06	0.85	19.69	49.05
120	0.27	0.08	1.00	28.94	74.35
140	0.33	0.09	0.92	39.97	98.40

Table 3
Relative gap for the GA

No. of jobs	Relative gap (rg)			CPU time (seconds)	
	Mean	StDev	Max	Mean	Max
20	0.18	0.01	0.79	0.09	1.09
40	0.19	0.01	0.86	0.36	1.42
60	0.21	0.02	0.73	1.14	3.21
80	0.24	0.03	0.73	2.90	8.17
100	0.28	0.04	0.84	6.42	16.11
120	0.31	0.06	1.00	12.38	44.28
140	0.36	0.07	0.89	21.29	65.00

Table 4
Relative gap for the GLS

No. of jobs	Relative gap (rg)			CPU time (seconds)	
	Mean	StDev	Max	Mean	Max
20	0.14	0.01	0.63	0.11	1.17
40	0.15	0.01	0.78	0.88	3.10
60	0.16	0.02	0.61	3.92	13.48
80	0.19	0.03	0.71	12.07	51.34
100	0.21	0.04	0.77	73.15	280.80
120	0.26	0.06	1.00	63.48	238.82
140	0.30	0.07	0.87	122.56	528.25

give the CPU time in seconds before the LRA stops (average and maximum CPU time). It can be observed that the relative gap increases with the number of jobs. For $n = 140$, the relative gap is around 33%. This is one of the reasons that motivated the development of a metaheuristic.

5.2.2. Results of the genetic algorithm

The GA computes an upper bound for which the relative gap is on average bigger than the results of the LRA (see Table 3). The CPU times are smaller but not far from those of the LRA. Thus, the GA used alone is not more efficient than the LRA.

5.2.3. Results of the genetic local search

Because it requires more computation, the time to run the algorithm is bigger for the GLS than for the simple GA. But, for instances with more that 100 jobs, Table 4 shows that the solution is better than the upper bound given by the LRA. However, even if the solution is better, a relative gap of 21% is still large and can probably be improved again.

5.3. Impact of a good initial solution

It is known that a good initial solution for a genetic-based algorithm usually leads to better results. The upper bound of the LRA is used as a chromosome in the initial population of the GA and the GLS. These two algorithms will be denoted by GALag and GLSLag, respectively.

Table 5 gives the results for GALag and compares them to the simple LRA. The first column is the number of jobs. The next three column give the

Table 5
Improvement of the genetic algorithm (GALag)

No. of jobs	GALag relative gap			Improvement			CPU time (seconds)	
	Mean	StDev	Max	(%)	Mean (%)	Max (%)	Mean	Max
20	0.09	0.01	0.49	22.5	4.2	16.6	0.92	2.28
40	0.08	0.00	0.56	28.1	3.1	19.6	3.86	8.09
60	0.11	0.01	0.55	48.8	3.7	22.5	7.23	19.03
80	0.14	0.02	0.64	64.4	6.7	43.4	12.12	36.14
100	0.18	0.04	0.79	72.2	9.1	42.7	22.00	54.11
120	0.23	0.07	1.00	80.9	11.1	61.6	33.75	76.98
140	0.28	0.08	0.88	90.0	12.4	54.1	49.04	116.35

Table 6
Improvement of the genetic local search (GLSLag)

No. of jobs	GLSLag relative gap			Improvement			CPU time (seconds)	
	Mean	StDev	Max	(%)	Mean (%)	Max (%)	Mean	Max
20	0.09	0.01	0.49	18.8	4.5	16.6	0.86	2.57
40	0.08	0.00	0.64	24.4	2.7	10.7	3.26	8.61
60	0.11	0.01	0.54	46.3	3.8	20.2	8.21	20.80
80	0.14	0.02	0.64	63.8	6.9	39.3	16.30	50.45
100	0.18	0.04	0.78	74.4	9.2	42.6	30.12	80.65
120	0.23	0.07	1.00	82.5	11.3	54.2	51.45	174.88
140	0.27	0.08	0.87	90.6	13.4	52.9	87.18	237.69

Table 7
Comparison of the algorithms ($n = 10, \ldots, 50$)

Algorithm	LRA	GA	GLS	GALag	GLSLag
$n = 10$					
Dev. from LBLag (%)	10.3	29.4	32.6	10.3	10.3
Dev. from LBPer (%)	1.7	22.0	24.9	1.6	1.6
Optimum found (%)	91.1	52.2	53.3	93.3	93.3
# of time LRA is strictly improved (%)				2.2	2.2
Average CPU time (seconds)	0.32	0.02	0.02	0.33	0.33
Maximum CPU time (seconds)	0.61	0.09	0.05	0.63	0.63
$n = 20$					
Dev. from LBLag (%)	12.2	27.5	25.0	11.8	11.6
Dev. from LBPer (%)	4.2	20.7	18.0	3.7	3.6
Optimum found (%)	61.1	11.1	17.8	66.7	65.6
# of time LRA is strictly improved (%)				10.0	8.9
Average CPU time (seconds)	1.05	0.08	0.10	1.12	1.10
Maximum CPU time (seconds)	2.12	0.14	1.08	2.15	2.16
$n = 30$					
Dev. from LBLag (%)	12.8	26.6	22.4	12.3	12.4
Dev. from LBPer (%)	3.2	18.5	13.8	2.6	2.7
Optimum found (%)	46.7	5.6	12.2	54.4	54.4
# of time LRA is strictly improved (%)				15.6	13.3
Average CPU time (seconds)	1.84	0.17	0.27	1.92	1.93
Maximum CPU time (seconds)	4.66	0.36	0.78	4.71	4.83
$n = 40$					
Dev. from LBLag (%)	9.4	23.7	20.9	9.2	9.2
Dev. from LBPer (%)	2.1	17.3	13.4	1.8	1.8
Optimum found (%)	41.1	5.6	10.0	46.7	44.4
# of time LRA is strictly improved (%)				8.9	10.0
Average CPU time (seconds)	2.66	0.34	0.67	2.70	2.84
Maximum CPU time (seconds)	7.93	1.24	1.79	7.19	8.50
$n = 50$					
Dev. from LBLag (%)	11.1	25.6	19.5	10.7	10.6
Dev. from LBPer (%)	3.2	18.6	12.1	2.7	2.6
Optimum found (%)	35.6	2.2	4.4	40.0	37.8
# of time LRA is strictly improved (%)				21.1	16.7
Average CPU time (seconds)	3.07	0.66	1.38	3.37	3.47
Maximum CPU time (seconds)	7.64	1.27	3.80	7.75	8.16

relative gap (as in Tables 3 and 4). The three columns "improvement" give the number of time (in %) GALag improves the initial solution of LRA, the average and maximum value of the improvement $((UB_{LRA} - UB_{GALag})/UB_{LRA})$. The last two columns give the average and maximum CPU time in seconds (time for the LRA + GA).

Table 6 gives the same results for GLSLag and compares them to the ones of LRA. GLSLag gives the best results on these tests (the solution is the best almost 90% of the time) except for the CPU time (however still reasonable). The general behavior shows that if GLSLag is stopped too early, GALag may give better results because more iterations can be done in the same amount of time. However, even when small CPU times are allowed, GLSLag almost always dominates GALag. For a specific 100-job instance, GALag performs 14853 iterations in 11.59 seconds and the best solution is 2143. GLSLag performs only 3555 iterations in 18.14 seconds and the best solution is 2091. GALag obtains a better solution than GLSLag before a time limit of 2.56 seconds, but is surpassed by GLSLag afterwards. This is not as obvious in the

Table 8
Comparison of algorithms ($n = 60, \ldots, 100$)

Algorithm	LRA	GA	GLS	GALag	GLSLag
$n = 60$					
Dev. from LBLag (%)	10.8	23.4	21.7	10.2	10.2
Dev. from LBPer (%)	3.7	17.0	12.2	2.9	2.9
Optimum found (%)	24.4	4.4	7.8	32.2	36.7
# of time LRA is strictly improved (%)				26.7	27.7
Average CPU time (seconds)	4.00	1.19	2.46	4.35	4.80
Maximum CPU time (seconds)	15.38	2.71	8.22	11.62	16.97
$n = 70$					
Dev. from LBLag (%)	10.6	23.7	20.9	9.7	9.8
Dev. from LBPer (%)	3.8	17.6	12.3	2.9	3.0
Optimum found (%)	23.3	2.2	4.4	32.2	31.1
# of time LRA is strictly improved (%)				40.0	36.7
Average CPU time (seconds)	4.23	2.01	3.66	5.02	5.39
Maximum CPU time (seconds)	11.42	5.15	10.59	11.41	13.03
$n = 80$					
Dev. from LBLag (%)	10.3	23.6	22.4	9.6	9.6
Dev. from LBPer (%)	3.8	17.6	12.2	2.7	2.8
Optimum found (%)	25.6	1.1	2.2	26.7	28.9
# of time LRA is strictly improved (%)				32.2	28.9
Average CPU time (seconds)	4.84	3.22	5.87	5.98	6.66
Maximum CPU time (seconds)	14.48	6.99	17.11	16.40	19.86
$n = 90$					
Dev. from LBLag (%)	10.2	23.8	22.3	9.3	9.4
Dev. from LBPer (%)	3.5	17.8	12.0	2.6	2.6
Optimum found (%)	23.3	1.1	4.4	27.8	28.9
# of time LRA is strictly improved (%)				42.2	36.7
Average CPU time (seconds)	5.89	4.49	10.17	7.34	8.91
Maximum CPU time (seconds)	17.00	10.57	29.60	17.45	23.14
$n = 100$					
Dev. from LBLag (%)	10.9	23.9	18.0	10.0	10.1
Dev. from LBPer (%)	4.1	17.7	11.4	3.0	3.1
Optimum found (%)	20.0	1.1	2.2	24.4	23.3
# of time LRA is strictly improved (%)				47.8	38.9
Average CPU time (seconds)	6.05	7.94	13.64	8.66	10.27
Maximum CPU time (seconds)	15.83	17.07	32.41	19.08	23.79

results of the following section (see Tables 7 and 8) on a different set of instances.

Although GLSLag gives the best results, the relative gap for $n = 100$ is still large (18%). For $n = 140$, the gap is above 25%. This is probably due to the fact that the lower bounds obtained by Lagrangean relaxation are not good enough. Recall that general parameters are used for LRA for every instance.

5.4. Comparison with a better lower bound

Péridy et al. [16] developed a very efficient lower bound (denoted LBPer) also using Lagrangean relaxation, but very differently than in [7]. The authors sent us the results on a set of instances generated in a very different way than our set (see Baptiste et al. [3]), the main difference being that jobs are more constrained (smaller time windows) in these instances. We ran the different versions of the genetic-based algorithms and compared our results to the lower bound.

The size of these test problems ranges from $n = 10$ to 100. Table 7 gives the results for instances with sizes varying from $n = 10$ to 50, and Table 8 from $n = 60$ to 100. They show that the duality gap is much smaller with a better lower bound, and that our solutions are thus closer to optimality than what our set of instances indicated. However, the average deviation from the lower bound is close to 3% for problems of more than 50 jobs, which is still too large to conclude definitely on the quality of our solution procedures.

6. Conclusion

This paper presents a first metaheuristic for a $\mathcal{NP}$-Hard one-machine scheduling problem. A GA has been designed testing various engines and several crossover operators. Consistent improvements have been obtained on several tests of instances, although very good solutions were already provided by a LRA. However, on our test of instances, we do not know whether large gaps between upper and lower bounds are mostly due to the duality gap, or to the fact that better solutions can be determined, i.e., if our solutions are close to optimality or not. This question could be partially answered by using the lower bound proposed in [16], although duality gaps are still too large to conclude on the second set of instances.

Hence, our future research aims at designing an exact approach for this problem, which could be used to better evaluate the effectiveness of our metaheuristic.

Acknowledgements

The authors would like to thank C. Prins for the very useful hints and private conversations that motivated this paper. The authors are also grateful to A. Hertz and M. Laguna for their enlightenment, to P. Thomin for his help in the implementation of part of the code, and to two anonymous referees for their helpful comments.

References

[1] K.R. Baker, Introduction to Sequencing and Scheduling, John Wiley, New York, USA, 1974.
[2] P. Baptiste, C. Le Pape, L. Péridy, Global constraints for partial CSPs: A case study of resource and due-date constraints, 4th International Conference on Principles and Practices of Constraint Programming, Pisa, Italy, 1998.
[3] P. Baptiste, A. Jouglet, C. Le Pape, W. Nuijten, A constraint-based approach to minimize the weighted number of late jobs on parallel machines. Research report 2000/228, UMR, CNRS 6599, Heudiasyc, France, 2000.
[4] J. Carlier, Problèmes d'ordonnancement à contraintes de resources: Algorithmes et complexités, Thèse d'Etat, Université de Paris 6, 28 mai 1984.
[5] S. Dauzère-Pérès, Minimizing late jobs in the general one machine scheduling problem, European Journal of Operational Research 81 (1995) 134–142.
[6] S. Dauzère-Pérès, M. Sevaux, An exact method to minimize the number of tardy jobs in single machine scheduling, Research report 99/6/AUTO, Ecole des Mines de Nantes, France, 1999.
[7] S. Dauzère-Pérès, M. Sevaux, Using Lagrangean relaxation to minimize the (weighted) number of late jobs on a single machine, Research report 99/8/AUTO, Ecole des Mines de Nantes, France, 1999.
[8] L. Davis (Ed.), Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991.
[9] S. French, Sequencing and Scheduling: An Introduction to the Mathematics of the Job–Shop, Ellis-Horwood, 1990.

[10] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey, Annals of Discrete Mathematics 5 (1979) 287–326.
[11] J.H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, MI, 1975.
[12] H. Kise, T. Ibaraki, H. Mine, A solvable case of the one-machine scheduling problem with ready and due times, Operations Research 26 (1) (1978) 121–126.
[13] E.L. Lawler, Knapsack-like scheduling problems, the Moore–Hodgson algorithm and the 'tower of sets' property, Mathematical Computer Modelling 20 (2) (1994) 91–106.
[14] J.K. Lenstra, A.H.G. Rinnooy Kan, P. Brucker, Complexity of machine scheduling problems, Annals of Discrete Mathematics 1 (1977) 343–362.
[15] J.M. Moore, A *n* job, one machine sequencing algorithm for minimizing the number of late jobs, Management Science 15 (1) (1968) 102–109.
[16] L. Péridy, E. Pinson, D. Rivreau, Total cost single-machine scheduling problem with arbitrary release dates, Proceedings of the Workshop on Project Management and Scheduling, Osnabrück, Germany, April 17–19, 2000, pp. 233–235.
[17] C.N. Potts, L.N. Van Wassenhove, Algorithms for scheduling a single machine to minimize the weighted number of late jobs, Management Science 34 (7) (1988) 834–858.
[18] M.C. Portmann, Genetic algorithms and scheduling: A state of the art and some propositions, Proceedings of the Workshop on Production Planning and Control, Mons, Belgium, September 9–11, 1996, I–XIV.
[19] C.R. Reeves, A genetic algorithm for flowshop sequencing, Computers and Operations Research 22 (1) (1995) 5–13.
[20] C.R. Reeves, Genetic algorithms for the operations researcher, INFORMS Journal on Computing 9 (3) (1997) 231–250.
[21] F.J. Villarreal, R.L. Bulfin, Scheduling a single machine to minimize the weighted number of tardy jobs, IIE Transactions 15 (1983) 337–343.

4OR (2004) **Regular Paper**

DOI: 10.1007/s10288-003-0028-0

A genetic algorithm for robust schedules in a one-machine environment with ready times and due dates

Marc Sevaux[1] **and Kenneth Sörensen**[2]

[1] Université de Valenciennes, Le Mont Houy , 59313 Valenciennes, France
(e-mail: marc.sevaux@univ-valenciennes.fr)
[2] Universiteit Antwerpen, Middelheimlaan 1, 2020 Antwerp, Belgium
(e-mail: kenneth.sorensen@ua.ac.be)

Received: 19 December 2002 / Accepted: 2 July 2003

Abstract. Computing a schedule for a single machine problem is often difficult, but when the data are uncertain, the problem is much more complicated. In this paper, we modify a genetic algorithm to compute robust schedules when release dates are subject to small variations. Two types of robustness are distinguished: quality robustness or robustness in the objective function space and solution robustness or robustness in the solution space. We show that the modified genetic algorithm can find solutions that are robust with respect to both types of robustness. Moreover, the risk associated with a specific solution can be easily evaluated. The modified genetic algorithm is applied to a just-in-time scheduling problem, a common problem in many industries.

Key words: Quality robustness, solution robustness, single machine scheduling, weighted number of late jobs, genetic algorithm

Mathematics Subject Classification: 90B99

1 Introduction

Scheduling problems are generally computationally difficult problems and the problem described in this paper is no exception. The problem becomes even more difficult when some of the data are stochastic (Daveand Beck 2000; Herroelen and

Correspondence to: K. Sörensen

4OR Quarterly Journal of the Belgian, French and Italian Operations Research Societies

Leus 2002). Genetic algorithms – and metaheuristics in general – have been found to provide good approximations of the optimal solution with a relatively small computational effort (Portmann 1996). Although the literature on applications of metaheuristics (and genetic algorithms in particular) to scheduling problems is very large, the number of applications to problems with a stochastic component is rather small, see, e.g., (Kouvelis and Yu 1997).

One way to deal with stochastic problem data is to find solutions that are *robust*. We distinguish two types of robustness. *Quality robustness* is a property of a solution whose quality, measured by the objective function value, does not deviate much from optimality when small changes in the problem data occur.

The second type of robustness is *solution robustness* and can be described as robustness in the solution space. When changes in the problem data occur, the decision maker might be forced to re-optimise the problem. In this case, the quality of the solution is guaranteed by the optimisation procedure. In some situations however a solution is preferred that is "close" (in the solution space, not the objective function space) to the solution currently used. Many manufacturers operate with a production schedule that repeats itself on a regular basis (e.g., daily or weekly). When, e.g., a new job needs to be scheduled, the problem is re-optimised, but the new production schedule should be as similar as possible to the one currently used. This type of robustness stresses the importance of solution stability. The two types of robustness are not entirely equivalent in the sense that quality robustness is a property of a solution that is insensitive to changes in the problem data *before* these changes occur, whereas solution robustness refers to the stability of a solution *after* changes have occurred.

The object of the *single machine scheduling problem* considered in this paper is to determine the order in which a given set of *jobs* should be executed on a machine. Each job has a certain release date and cannot be scheduled before this date. Also, each job has a certain processing time and a due date, before which it should be finished. Jobs that are not finished before their due date are called *late*. The objective of the problem considered in this paper is to minimise the number of late jobs. A genetic algorithm for this problem was developed by Sevaux and Dauzère-Pérès (2002).

When a problem has stochastic parameters, the objective function value itself becomes stochastic. The objective of a stochastic problem has several dimensions. Some decision makers prefer a solution that has a high quality *on average*, others might prefer one that hedges against the worst possible situation. However, most decision makers like to minimise the risk of ending up with a bad solution, without necessarily choosing an extremely conservative solution. In this paper, we modify the GA proposed by Sevaux and Dauzère-Pérès (2002) to be able to deal with stochastic release dates and show that this GA is able to find solutions that are both quality robust and solution robust. The proposed approach can be labelled a *sampling* method, in that it uses sampling to estimate the robustness of a solution. We show that this approach is very flexible in that – although our GA is only used for

problems with stochastic release dates – other types of stochastic problem data can be easily incorporated. Simultaneously, the risk preference of the decision maker can be modelled and a solution can be found that minimises the estimated number of late jobs and the risk that this solution entails. Solution robustness is obtained by measuring the distance (in the solution space) to a given base-line solution. This distance measure is based on the *edit distance*. Minimising the distance to a baseline solution and simultaneously maximising the quality of a solution is obviously difficult. We show in this paper that both objectives can be achieved simultaneously by letting the GA search for high-quality solutions and measuring the distance of each solution encountered to the baseline solution. In a second stage, a solution can be chosen that provides a good trade-off.

This paper is structured as follows. After the literature review, the single machine scheduling problem is formulated. Section 4 introduces a genetic algorithm for the deterministic single machine scheduling problem. In Sect. 5, this GA is modified for the computation of schedules that are both quality robust and solution robust. Section 6 describes how the modified genetic algorithm can be used to find robust production sequences in a just-in-time environment. Section 7 reports on some computational results.

2 Literature review

Single machine scheduling problems have been successfully tackled using genetic algorithms. Genetic algorithms – pioneered by Holland (1975) and Goldberg (1989), as well as many others—use a Darwinian metaphor to find good solutions for difficult optimisation problems. A review of genetic algorithms can be found in Reeves (1993, 1997). For a review of applications of genetic algorithms to scheduling problems we refer to Portmann (1996).

This paper describes a simple GA, developed by Sevaux and Dauzère-Pérès (2002), that finds good solutions to the deterministic scheduling problem described above.

Although the literature on deterministic scheduling is several orders of magnitude larger, there is a significant body of work on scheduling under uncertainty. Two very recent literature surveys are Herroelen and Leus (2002) and Davenport and Beck (2000). Herroelen and Leus (2002) divide research on scheduling with uncertainty into six categories. In *reactive scheduling*, procedures are developed to re-optimise the schedule when disruptive events occur. *Stochastic scheduling* is the application of stochastic programming with recourse to scheduling problems. *GERT network scheduling* is used for problems with stochastic evolution structure and feedback. *Scheduling under fuzziness* recommends the use of fuzzy numbers for, e.g., project duration instead of stochastic variables. *Proactive (robust) scheduling* attempts to develop a schedule, the quality of which is relatively insensitive to a changing environment. Herroelen and Leus (2002) also discuss some recent advances in *sensitivity analysis* for project scheduling. The method developed in

this paper can be categorised under the heading *robust scheduling*. Davenport and Beck (2000) distinguish between *proactive* and *reactive* approaches. Proactive approaches include the insertion of *redundancy* (typically extra buffer time) into the schedule, the use of *probabilistic techniques* to determine robust schedules and the creation of *multiple schedules* to deal with contingencies.

Although genetic algorithms and meta-heuristics in general have been often and successfully applied to tackle deterministic scheduling problems, their application to scheduling under uncertainty is very limited. In this paper, we argue that meta-heuristics can very easily be adapted to the requirements of a stochastic problem formulation. Using meta-heuristics for this type of optimisation problem has several advantages that will be discussed later. That there is a need for robust meta-heuristic optimisation is recognised in the influential book "Robust discrete optimisation" (Kouvelis and Yu 1997), when the authors say on p. 354: "*We believe that considerable more effort should be spent in systematic development of [. . .] metaheuristic frameworks, which with minimal adjustment effort can be applied to a large class of robust optimisation problems [. . .]*". Previous work on robust genetic algorithms can be found in Tsutsui et al. (1996); Tsutsui and Ghosh (1997); Tsutsui and Jain (1998); Tsutsui (1999); Branke (1998, 2001), but the applications of this work are limited to the robust optimisation of continuous mathematical functions.

A genetic algorithm for robust job shop schedules is developed by Leon et al. (1994b). This algorithm uses an objective function that includes some measure of robustness. In their problem, the authors assume that machine operations can be disrupted and that the disrupted operations are restarted immediately after the end of the disruption period (a *right-shift reactive policy*). Robustness is defined as a function of the schedule delay, i.e., the difference between the actual makespan and the projected makespan. A number of different measures, specifically developed to estimate this measure of robustness is developed. Leon et al. (1993, 1994b) describe an approach in which recovering from disruptions is modelled as a game played against nature.

Hart and Ross (1999a, b) have recently developed *artificial immune systems* as a tool for scheduling under uncertainty. The author's goal is to develop off line a number of partial schedules that can be used as building blocks to quickly compose a good schedule when unexpected events occur. The underlying assumption is that such events (such as machine breakdowns) and the desired reaction of the scheduling system, are to some extend predictable. The partial schedules found by this GA therefore constitute a specific piece of domain knowledge, that can be used when disruptions occur.

Jensen (2001) introduces several novel ideas. In *neighbourhood-based robustness*, the basic idea of which is that if a small set of schedules close to the preschedule is known to be of good quality before an unexpected event occurs, then perhaps one of these schedules provide a good solution in case an unforeseen event occurs. *Co-evolution* is a term to describe genetic algorithms in which a population of robust solutions and a population of problem instances evolve simultaneously. At the

Table 1. Single machine scheduling problem notation

Description	Symbol	Remarks
Number of jobs	n	
Release date	r_j	
Processing time	p_j	
Due date	d_j	
Weight	w_j	
Starting time	t_j	$r_j \leq t_j$
Completion time	C_j	$C_j = t_j + p_j$
Lateness status	U_j	$U_j = 1$ iff $C_j > d_j$ and 0 otherwise

end, the first population contains the most robust solutions and the second one the most difficult problem instances. The first population might contain, e.g., a set of schedules and the second a set of machine breakdowns.

3 Single machine scheduling problem formulation

The objective of the scheduling problem in this paper is to schedule a given set of jobs on a single machine without preemption. Every job has a *release date* that represents the time at which production or assembly of the batch can begin at the earliest. The *processing times* of the jobs are known, as well as their *due dates*. Depending on its importance, a *weight* is attached to every job. A job is called *late* if its completion time is greater than its due date. The objective is to find a schedule of jobs that *minimises the total weighted number of late jobs*. Late jobs are arbitrarily moved to the end of the schedule, as the amount of lateness is considered unimportant. Table 1 clarifies the notation used.

The problem in this paper is denoted as $1|r_j|\sum w_j U_j$ in the standard classification. The problem is $\mathcal{NP}$-hard in the strong sense (Lenstra et al. 1977). For the static case, previous work on this problem can be found in Sevaux and Dauzère-Pérès (2002), Dauzère-Pérès (1995), Dauzère-Pérès and Sevaux (2003), Dauzère-Pérès and Sevaux (2001), Baptiste (1999) and Baptiste et al. (2003).

Although all parameters of the problem can be made stochastic, we only model the case of stochastic release dates. We will show however, that other stochastic parameters can easily be entered into the problem formulation and solved by the modified GA.

4 A genetic algorithm for a single machine scheduling problem

Algorithm 1 gives a schematic representation of the GA used in the rest of the paper.

In this GA, a solution is encoded as a permutation of the n jobs. An initial population is created by randomly generating permutations of jobs. The size of the population N is computed by the formula $N = 5+n/5$ where n is the number of jobs

Algorithm 1 Basic incremental genetic algorithm

```
1: Generate an initial population
2: while stopping conditions are not met do
3:    Select two individuals
4:    Crossover the two individuals
5:    Mutate offspring under probability
6:    Evaluate offspring
7:    Insert offspring under conditions
8:    Remove an individual under conditions
9: end while
10: Report results
```

of the selected instance. The main loop of the GA is stopped after a fixed number of iterations without improvement of the best solution or after a certain maximum number of iterations. The maximum number of iterations `ItMax` is computed by $\texttt{ItMax} = 50 * n * N$ and the maximum number of iterations without improvement is $\texttt{ItMaxwoImp} = \texttt{ItMax}/10$.

Two parent solutions are drawn from the population. The former is chosen according to a *ranking selection* method which give a higher probability to the best individuals (Reeves 1993) and the latter is randomly chosen. The population is sorted in ascending order of fitness values (ascending order of the objective function), so the best individual is ranked first. If N denotes the size of the population, the selection of the first parent is made using the probability distribution $2(N+1-k)/(N(N+1))$ where k is the k^{th} chromosome in the ascending order of fitness values. Using this technique, the median chromosome $(N+1)/2$ has the probability $1/N$ of being selected, while the best individual has the probability $2/(N+1)$ (approximatively twice the median). The second parent is randomly selected with an uniform distribution.

The crossover operator is a one-point crossover operator X1. A crossover point is randomly chosen. The first part of parent P1 is copied into the offspring. The rest of the chromosome is filled by reading the information of the second parent P2. The jobs that are not already in the offspring are copied, preserving the permutation property. See Fig. 1.

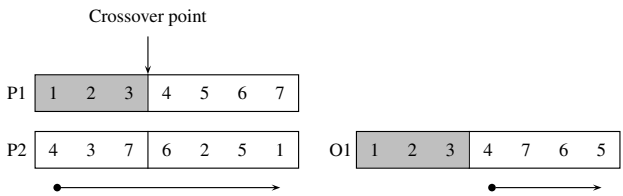


Fig. 1. Crossover operator X1

A *general pairwise interchange* mutation operator permutes two jobs randomly chosen in the chromosome. See Fig. 2. This mutation is done under a probability P_m. For the deterministic case, we chose $P_m = 0.25$.

Fig. 2. Mutation operator GPI

After mutation, the solution is evaluated. The fitness value of a solution is equal to the sum of the weights of the late jobs, i.e.,

$$f(x) = \sum_{j=1}^{n} w_j U_j, \tag{1}$$

where U_j is defined as in Table 1.

The genetic algorithm uses incremental replacement, i.e., the population size remains the same throughout the run. A new offspring is inserted if it improves at least the worst solution in the population. When a new solution is inserted, a solution randomly chosen from the worst half of the population, is discarded.

In Sevaux and Dauzère-Pérès (2002) different variations of this simple genetic algorithm, combined with a local search technique are compared to and found to be superior over other methods. The GA used in this paper does not attempt to locally improve solutions using a local search operator. Experimental results show that, when using the GA to find robust solutions, the use of a local search improvement operator tends to overemphasise solution quality. Finding robust solutions requires the exploration of a diverse set of solutions, covering a portion of the search space that is as large as possible. Adding a local search operator at this step will intensify the search and the solutions will have some jobs tightly sequenced so that any modification in the release date will cause an important modification of the optimisation function. For our study, the local search operator will not be used.

5 A genetic algorithm for robust schedules

In the following, we describe how the genetic algorithm for the deterministic scheduling problem can be modified so that it finds solutions that are quality robust and/or solution robust. As in Sörenesen (2001), this is done by replacing the fitness function of the GA by a so-called *robust evaluation function*. When we want to make the distinction, the fitness function for the deterministic problem is referred to as the *ordinary evaluation function*.

5.1 Quality robustness

Let x be a solution of the problem (a permutation of the jobs). The quality of x is computed by an evaluation function $f(x)$. When we want to indicate that f has parameters, we write $f(x, P)$, where P is the set of problem data. In our case, P represents the characteristics of the jobs (r_j, p_j, d_j, w_j). To allow the GA to find robust solutions, the evaluation function $f(x)$ is replaced by a *robust evaluation function* $f^*(x)$. The robust evaluation function for quality robust solutions adheres to the following principles (Sörensen 2001):

Principle 1 : Each solution is implemented on a modified set of characteristics $S_i(P)$. S is a *sampling function*, that takes a random sample from the stochastic elements of P. $S_i(P)$ is the i-th set of sampled parameters of P. We call the implementation of a solution on a modified set of characteristics a *derived solution*.

Principle 2 : Several evaluations of a solution x on a sample of P are combined into a new evaluation function. An evaluation of a derived solution is called a *derived evaluation*. This new function is the *robust evaluation function* $f^*(x)$.

A possible form of a robust evaluation function is a weighted average of m derived evaluations:

$$f^*(x) = \frac{1}{m} \sum_{i=1}^{m} c_i f(x, S_i(P)) \tag{2}$$

where c_i is a weight associated to this derived evaluation according to its importance and m is the number of derived solutions to evaluate.

A more conservative robust evaluation function examines the worst-case performance of a solution across all derived evaluations:

$$f^*(x) = \max_{i=1...m} f(x, S_i(P))$$

if f has to be minimised. In this paper, we only use the robust evaluation function of Eq. (2).

For the scheduling problem considered in this paper, we assume the release dates to be stochastic. The robust evaluation function evaluates each solution several times, each time on a new random instantiation of the release dates. The same permutation of the jobs is kept but the release dates are modified. Afterwards, the engine which computes the objective function value of the permutation is called and executed with this new instantiation of the problem. The derived evaluations are combined into a single robustness measure as indicated. This implies that a standard evaluation has to be performed several times for each solution, which increases the computational time. The advantage of this approach however is that the standard evaluation function does not have to be modified. The implementation of a robust evaluation function is therefore very easy and only requires a standard evaluation

function to be available and a metaheuristic already running for problems with static data. The only change made is to the evaluation step of Algorithm 1, rendering this way of finding robust solutions extremely easy to implement and very flexible.

5.2 Solution robustness

Solution robustness is a property of a solution that is similar to a given baseline solution x_0, i.e. for which the *distance* to the baseline solution (as measured by some distance function) is small. Of course, solution robustness cannot be used as the only objective, since solution quality or quality robustness should always be taken into account. The need for solution robustness therefore automatically transforms the problem into a multi-objective one and a solution should be found that simultaneously has a high quality (robustness) and a small distance to the baseline solution. In our framework, solution robustness is obtained by measuring the distance between the baseline solution and each solution generated by the GA. It is assumed that the GA visits a sufficiently diverse set of solutions, so that at least a fraction of them will be solution robust. A solution is then chosen using a multi-objective decision making process – taking into account the decision maker's preferences for solution robustness and quality (robustness).

Edit distance. First developed in the context of correcting binary codes transmitted across a faulty channel (Levenshtein 1966), the edit distance is a measure of the similarity between two strings, composed of characters of a finite alphabet Σ. An extra null-character Λ is also defined to signify the absence of a character. Three *elementary edit operations* are defined ($x, y \in \Sigma$): *insertion* ($\Lambda \rightarrow x$), *deletion* ($x \rightarrow \Lambda$) and *substitution* ($x \rightarrow y$). An edit transformation between strings s and t is a set of elementary edit transformations that transform s into t.

Simplifying, the edit distance $d(s, t)$ can be defined as the number of edit operations that is required to transform a *source* string s into a *target* string t[1] A straightforward dynamic programming algorithm to compute the edit distance is due to Wagner and Fischer (1974). The time complexity of this algorithm is $O(|s| \times |t|)$ where $|s|$ and $|t|$ are the lengths of strings s and t respectively. More efficient algorithms have been developed (e.g. by Ukkonen 1985), but these are beyond the scope of this paper. Under some conditions (Wagner and Fischer 1974), the edit distance is a *metric*, i.e. it satisfies the conditions that ($\forall s, t, u$): $d(s, t) \geq 0$, $d(s, t) = 0 \Leftrightarrow s = t$, $d(s, t) = d(t, s)$, and $d(s, t) \leq d(s, u) + d(u, t)$.

[1] In a more general formulation of the edit distance, a distance function $\gamma(x, y) \geq 0$ is defined that determines the non-negative real-valued cost of transforming character x into character y ($x, y \in \Sigma \cup \Lambda$). The cost of an edit transformation is defined as the sum of the costs of the elementary edit operations it contains. The edit distance $d(s, t)$ now is defined as the minimum-cost edit transformation that transforms s into t.

Distance between two schedules. If all jobs of the scheduling problem are represented by a different character, a schedule can be represented as a string of jobs. The distance between two solutions can be calculated by the edit distance and interpreted as the number of "changes" that have to be made to the first schedule to turn it in to the second one. A "change" can be either an insertion of a job into the schedule, a deletion of a job or the substitution of a job by another one.

A sensible distance measure should accurately reflect the "similarity" between two schedules. The meaning of this concept is highly dependent on the specific situation. Although the edit distance provides a relatively natural way of measuring the similarity between two schedules (the number of changes required to transform schedule s into schedule t), other distance measures could be developed, e.g., based on the actual starting times of the jobs.

5.3 Incorporating risk preference

The function $f^*(x)$ estimates the average performance or the worst case performance of a solution, given that some of the parameters of the problem are stochastic. Clearly, the worst case performance measure will lead to solutions that are more conservative. Solutions found using this form of the robust evaluation function will hedge only against the worst possible incidence, independent of the probability that this scenario will occur. This type of robust evaluation function can be used by extremely risk-averse decision makers.

A more subtle manner to incorporate the risk preference of the decision maker, is to include into the robust evaluation function an estimate of the probability that the quality of a solution will deviate from its expected value. A possible measure is the standard deviation of the quality of a given solution over all samples:

$$\sigma^*(x) = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}[f(x, S_i(P)) - f^*(x)]^2}.$$

The two measures can be integrated in a multi-objective decision making approach. A possible way is to find the solution that minimises $f^*(x) + \gamma\sigma^*(x)$, where γ is a parameter indicating the risk-averseness of the decision maker. A more advanced way is to retain all efficient solutions and choose one according to a multi-objective decision making method.

6 Application to just-in-time scheduling

6.1 Just-in-time scheduling

Many manufacturing companies have switched (at least partially) to a *just-in-time* policy over the last two decades. In this philosophy, suppliers are required to deliver

their raw materials or semi-finished products to the manufacturer's plant right on time for these parts or materials to be taken into production (Christopher 1998). The suppliers are required to locate a supply hub (e.g., a warehouse) close to the manufacturer's production facility and deliver parts or materials to the manufacturing plant several times a day, at the request of the manufacturer. Based on the *production sequence*, i.e., the order in which the different product batches are be produced in the manufacturing plant, release dates and amounts to deliver of the parts or materials are fixed by the schedulers in the manufacturing plant and communicated to the suppliers. The production sequence determines when the production of each batch begins, and consequently which batches will be finished on time and which not.

Just-in-time delivery promises several advantages, the most important ones being reduced costs and shorter lead times. A negative side-effect from the use of just-in-time however, is the fact that any delay in the delivery by the suppliers will postpone the production of a batch which will lead to longer and unpredictable lead times of the finished product and increase the number of late deliveries. Although manufacturing companies and their suppliers are working hard to reduce the delay of parts or materials, unexpected events can have a significant effect on them. This in turn will negatively influence the number of production batches that are finished on time.

While manufacturers are trying hard to reduce the number of late deliveries of parts and materials as well as the amount of time with which they are delayed, some factors remain beyond their control. Traffic jams, truck breakdowns, strikes etc., are factors that may delay delivery, but are very difficult to control. To deal with these types of unexpected events, a manufacturing company can attempt to design a *robust production sequence*. A quality robust production sequence can be defined as a production sequence, the quality of which is relatively insensitive to perturbations in the input data. The quality of a certain schedule can be measured by various quality measures, e.g. by counting the number of batches that are finished on time. A solution robust production sequence may be defined as a production sequence that is "close" to a base-line production sequence that is currently in use by the manufacturer. As already mentioned, changing the production sequence drastically when unexpected events occur may increase the quality but might cause a large disruption in the regular production routine. This in turn may lead to production errors, and other undesirable effects.

6.2 Modelling a just-in-time production system as a scheduling problem

A just-in-time production environment can be modelled as a scheduling problem, in which each of the production batches is represented by a job that has to be scheduled on a single machine. Solving the problem yields a production sequence

that minimises the weighted number of late batches. Of course, other objectives can be used, such as the weighted tardiness.

The following paragraphs describe the problem that a real French assembly factory in the car industry faces. For reasons of confidentiality, real data were not given, but enough information was provided to allow us to create a realistic simulation of reality.

In the assembly factory, we observe a huge number of elementary tasks that we aggregate in a reduced number of aggregated jobs. The number of jobs n to be processed each day varies between 20 and 80. The jobs are quite diverse and have widely spread-out processing times. Processing times for the different jobs are approximately uniformly distributed, with a high variability. Jobs are planned each day, but the schedule approximately repeats itself every week. Past observations show that a significant part of the jobs cannot start at their expected release date because of late deliveries of subcontractors. The percentage of jobs that is delayed is about 20%. Jobs almost never start early. Delays are approximately uniformly distributed between 0 and some parameter 20, where this number represents the number of time-periods added to the original release date.

Problem instance generator – deterministic problem. From these data, we have created a problem instance generator that generates instances according to the following rules. A single day of 80 five-minute periods is considered. The probability of release dates to arise in the morning is greater than in the afternoon (and the reverse for due dates). Hence release dates are generated according to a gamma law ($\Gamma(0.2, 4)$, which gives a mean of 20 and a standard deviation of 10) and due dates are generated according to the same law but considering the horizon from the end. If $r_i \geq d_i$ the associated job is discarded. To generate only feasible jobs, the condition $d_i - r_i \leq 1$ is checked (if not the job is discarded) and processing times are uniformly generated in the interval $[1, d_i - r_i]$. Weights are uniformly generated in the interval [1, 10]. In the sequel n will denote the number of jobs. Twenty different instances are generated for each value of n and n takes its value in the set {20, 40, 60, 80, 100}. This new set of instances will be called ODD for "One Day Data". Each file is numbered ODDn_i where n is the number of jobs and i the i^{th} instance for the specified number of jobs (ODD20_4 is the fourth instance with 20 jobs).

The rules of the problem instance generator are summarised in Table 2. $\mathcal{U}(l, u)$ indicates a uniform distribution between l and u. $\Gamma(a, b)$ indicates a gamma distribution.

Robust evaluation function. In the robust evaluation function, each solution is evaluated a fixed number of times on a modified instance of the problem data. At each evaluation, a number of jobs is randomly chosen and the release dates of these jobs are increased. The m evaluations are averaged to determine the value of the robust evaluation function. The parameters of the robust evaluation function are shown in Table 3.

Table 2. Deterministic problem instance generator rules

Parameter	Value
Problem size	(20,40,60,80,100)
Total time T	80
Release date r_j	$\Gamma(0.2, 4)$
Due date d_j	$T - \Gamma(0.2, 4)$
Processing time p_j	$\mathcal{U}(1, d_i - r_i)$
Weight w_j	$\mathcal{U}(1, 10)$

Table 3. Robust evaluation function parameters

Parameter	Value
Percentage of jobs with delayed release date	20%
Release date delay	$\mathcal{U}(0, 20)$
Number of evaluations per robust evaluation (m)	100

The number of evaluations m needs to be sufficiently high to avoid that a non-robust solution is accidentally chosen. Some experiments show that a value of $m = 100$ gives adequate results, while keeping the increase in computation time within acceptable limits. The robust evaluation function value is stored and used for population management purposes (see Sect. 4).

7 Numerical experiments

This section describes some experiments to find quality robust solutions. Experiments to increase solution robustness are reported in Sect. 8.2.

All the test code has been implemented in C using a gcc version 2.96 on a Linux RedHat 7.3 system for a Pentium III - 800Mhz PC machine.

To test the efficiency of the proposed method, a simulation procedure is used. The standard GA, using the ordinary evaluation function, is first run on the original data. The result obtained is called the *standard sequence*. The robust GA is also run with the systematic application (i.e., for evaluation of the solution and population management) of the robust evaluation function (Eq. 2). The result obtained is called the *robust sequence*. Once these two sequences are obtained, 1000 replications of the problem instance with randomly modified data according to the disturbances are evaluated through the sequences. Results are analysed below. In the sequel, the standard GA procedure will be denoted by the acronym SGA and the robust GA by RGA.

Table 4 gives the detailed results for the SGA for the subset of instances with 80 jobs. the column 'CPU' gives the CPU time of the SGA in seconds. The number of iterations is mentioned in the column '#It'. The next two columns respectively show the fitness value of the best individual and the average fitness value of the final population. The last two columns are the results of the simulation with the 1000 replications of the problem with modified instances. The sequence of the

Table 4. Results of the Standard GA for 80 jobs

Name	Cpu (s)	# It.	Fitness	Av. Pop.	Av. 1000r	Inc. (%)
ODD80_1	0.87	16775	400	405.05	430.17	7.54
ODD80_2	1.24	31903	349	353.90	373.00	6.88
ODD80_3	0.64	15606	348	354.19	371.25	6.68
ODD80_4	0.64	14505	411	417.14	431.00	4.87
ODD80_5	1.29	25518	307	312.76	337.28	9.86
ODD80_6	0.52	13217	329	332.62	340.99	3.64
ODD80_7	0.96	20278	331	336.86	361.88	9.33
ODD80_8	0.70	12646	354	357.67	368.84	4.19
ODD80_9	0.99	18863	317	321.43	343.06	8.22
ODD80_10	0.69	14645	344	347.67	366.58	6.56
ODD80_11	0.73	17557	394	398.19	417.80	6.04
ODD80_12	0.75	15224	363	374.52	385.68	6.25
ODD80_13	0.55	10500	317	322.81	338.57	6.81
ODD80_14	0.54	10104	364	368.86	389.80	7.09
ODD80_15	0.65	11040	369	373.86	385.32	4.42
ODD80_16	0.52	10418	370	375.90	384.52	3.92
ODD80_17	0.52	10982	325	331.48	342.11	5.26
ODD80_18	0.84	13599	307	312.81	324.60	5.73
ODD80_19	0.76	15434	357	363.43	376.29	5.40
ODD80_20	0.63	14365	365	372.33	391.88	7.36

best individual is used to evaluate these 1000 replications and the average value is reported in the 'Av. 1000r' column. Column 'Inc.' indicates the increase (in %) of the 1000 modified instances compared to the best solution – i.e., (Av.1000r-Fitness)/Fitness – (e.g., for instance `ODD80_1`, a modification in the release dates could lead to an increasing of 7.54 % of the objective function value, on average).

Similar results are reported in Table 5. In addition we have the robust fitness value (column 'Robust Fit.') which was used during the search to guide the RGA. This value is used as a reference instead of the Fitness value. Note that the average population value was computed on the fitness values and not on the robust fitness values. A column (named SGA-RGA) compares the two algorithms. The value computed here is the (SGA:Av1000r - RGA:Av1000r) / SGA:Av1000r which denotes the difference between the two algorithms in percentage after disruption.

For instance `ODD80_20`, for example, the best objective function value obtained by the standard GA is 365 after 14365 iterations in 0.63 seconds. If we keep this sequence knowing that release dates can be modified, we could expect an objective value of 391.88 (on average) which is an increase of 7.36% of the cost. The Robust GA gives a sequence with a solution of 370 (or more likely with 371.78, the robust fitness value) after 23057 iterations in 151.64 seconds. If now we use this sequence for the modified data, the expected objective value will be 377.56 (on average) which is an increase of only 2.04%. Although, the RGA improves almost always the average solution after disruption of the SGA method. The solution given by the RGA is then more *quality robust*.

Table 5. Results of the Robust GA for 80 jobs

Name	Cpu (s)	# It.	Fitness	Robust Fit.	Av. Pop	Av. 1000r	Inc. (%)	SGA-RGA (%)
ODD80_1	158.33	22058	406	406.36	411.76	417.60	2.86	2.92
ODD80_2	72.52	9956	357	358.91	361.90	366.73	2.72	1.68
ODD80_3	108.76	15687	369	363.37	365.62	374.23	1.42	−0.80
ODD80_4	152.49	20818	429	422.61	424.57	426.65	−0.55	1.01
ODD80_5	224.35	29705	310	314.94	317.48	328.88	6.09	2.49
ODD80_6	100.98	15109	327	328.24	335.24	335.49	2.60	1.61
ODD80_7	147.44	20249	342	345.39	344.33	358.54	4.83	0.92
ODD80_8	96.03	11853	358	360.67	363.67	365.09	1.98	1.02
ODD80_9	120.26	14271	325	327.12	330.67	337.71	3.91	1.56
ODD80_10	126.65	16910	353	355.95	359.57	358.47	1.55	2.21
ODD80_11	119.67	15989	403	408.82	414.05	419.97	4.21	−0.52
ODD80_12	94.76	13801	369	371.24	375.29	377.26	2.24	2.18
ODD80_13	62.12	8559	341	333.90	337.33	340.49	−0.15	−0.57
ODD80_14	85.73	12212	381	378.84	379.76	383.52	0.66	1.61
ODD80_15	146.25	16869	373	377.04	382.81	378.48	1.47	1.78
ODD80_16	106.09	14472	377	370.44	377.38	380.52	0.93	1.04
ODD80_17	144.83	19637	329	329.30	332.00	333.23	1.28	2.60
ODD80_18	120.86	15289	313	316.72	315.86	321.17	2.61	1.06
ODD80_19	158.79	21159	364	363.40	363.81	368.20	1.16	2.15
ODD80_20	151.64	23057	370	371.78	376.43	377.56	2.04	3.65

Table 6. Global SGA and RGA results (deviation and CPU time)

Number	Gap to best solution		Gap (in %)	CPU time (s)	
of jobs	Std GA (%)	Rob GA (%)	SGA-RGA	Std GA	Rob GA
20	11.95	3.84	2.88	0.03	1.73
40	8.47	2.89	1.61	0.13	10.27
60	7.35	3.08	0.89	0.36	45.81
80	6.30	2.19	1.48	0.75	124.93
100	5.32	2.01	0.47	1.53	215.69

Table 6 gives the summarised results for the whole set of instances from 20 to 100 jobs. The columns "Gap to the best solution" indicates the average value in percentage between the sequence found by any method and the results after the 1000 replications of modified instances (e.g., for $n = 80$, the average value of column "Inc." in Table 4 and 5). The middle column "Gap SGA-RGA" is the average value of the column SGA-RGA of Table 5 and measures the improvement of the RGA over the SGA after disruption.

The results confirm that it is always better to take the stochastic nature of some of the data into account when determining a schedule. On average, modifications of the problem data have a much smaller effect on the objective function value of solutions found by the RGA. However, since the computation of the robust evaluation function requires the ordinary evaluation function to be used 100 times, CPU times are much larger for the RGA. A possible strategy to lower the computational effort is to use the robust evaluation function only when the standard evaluation of a solution is

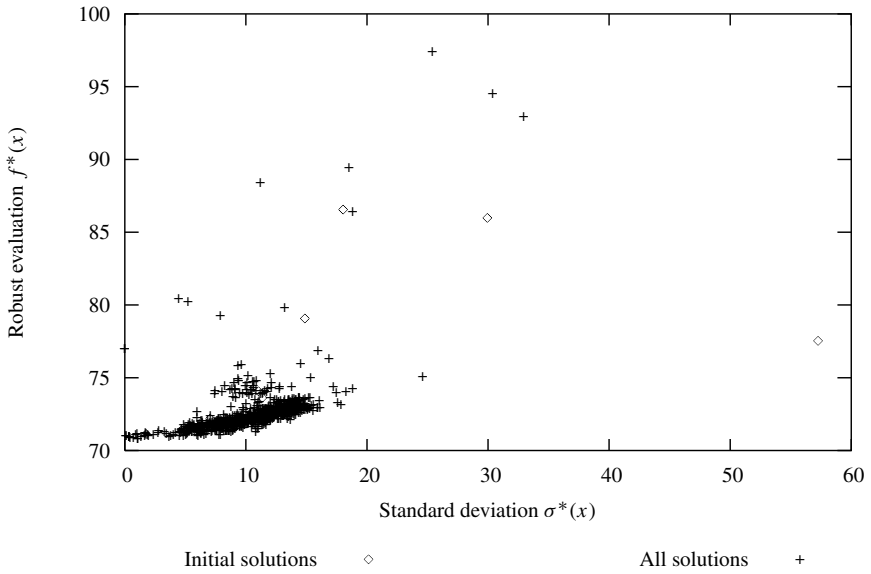


Fig. 3. Standard deviation of the robust value – all solutions

better than a certain threshold. We did not implement such strategies here since the computational times recorded for the RGA are still very reasonable.

8 Helping the decision maker

8.1 Risk analysis

As mentioned in Sect. 5.3, choosing a robust solution is not an easy task and can be done through a *multiple objective decision making process*. In Fig. 3, for a 20 job instance, we plot all the solutions (including the initial solutions) in the space "Standard Deviation of the robust value / Robust Value". The initial solutions are spread out in the space and all the other solutions seem to converge toward a good robust fitness value (to the bottom of the graph). At the same time, the standard deviation of the solutions decreases meaning that the solution are more and more robust. Figure 4 shows only the non-dominated solutions (Pareto solutions) that can be suggested to the decision maker.

8.2 Solution robustness

To obtain solutions that are solution robust, the edit distance between a base-line solution and all the solutions found by the RGA is measured. We assume that the

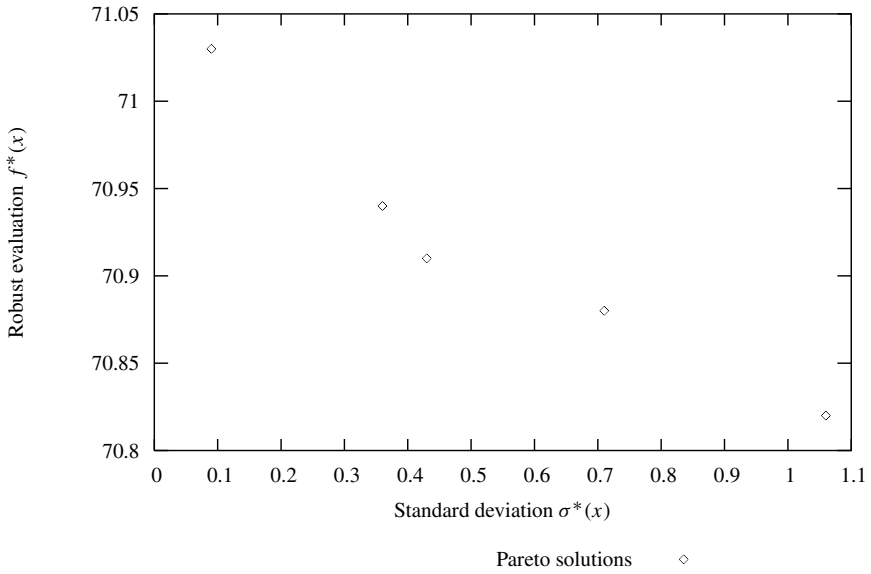


Fig. 4. Standard deviation of the robust value – efficient set

solution that is currently implemented (the base-line solution x_0) is the best solution obtained with the SGA. This choice is arbitrary and in real-life situations, x_0 would represent the solution actually used. For the same 20-job instance, Fig. 5 represents the initial, intermediate and final solutions obtained by the RGA. The number of intermediate solutions can be very large and only non-dominated solutions are kept for clarity.

For this particular instance, the minimum distance to x_0 that can be obtained by the GA is 12, indicating that all robust solutions are relatively "far" from the base-line solution. Using the information about the distance to the base-line solution and the robustness of a solution, the decision maker can make an informed decision.

9 Conclusion

In this paper, a genetic algorithm for a single machine scheduling problem was modified to find robust solutions. Two types of robustness were defined and it was shown how our GA can be used to find solutions that are both solution robust and quality robust. It was also shown how information about the risk associated with a specific solution can be incorporated into the decision making process. Our GA was applied to a just-in-time production planning problem. Computational results show that robust solutions can be found by applying some simple modifications of a genetic algorithm for a deterministic problem. This approach is very flexible

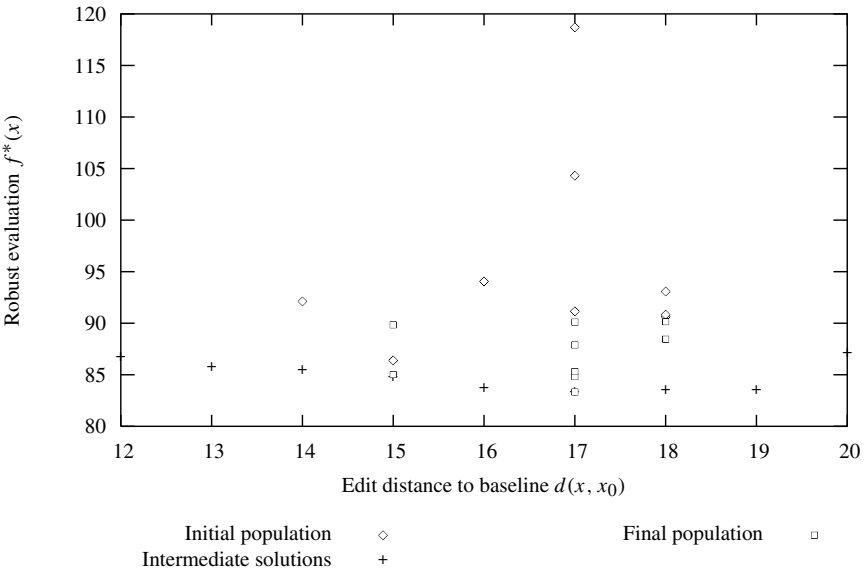


Fig. 5. Distance between solutions

in that it does not impose any requirements on the number and type of stochastic information that is incorporated. Without changing the solution algorithm in an extensive way, any information that is available about the problem data can be used in the decision making process.

Acknowledgement. The authors would like to thank the anonymous referees and the editor for their constructive comments and suggestions.

References

Baptiste Ph (1999) An $\mathcal{O}(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Operations Research Letters* 24: 175–180

Baptiste Ph, Péridy L, Pinson E (2003) A branch and bound to minimize the number of late jobs on a single machine with release time constraints. *European Journal of Operational Research* 144: 1–11

Branke J (1998) Creating robust solutions by means of evolutionary algorithms. In: *Parallel Problem Solving from Nature V* (LNCS, vol. 1498). Springer, Berlin Heidelberg New York, pp 119–128

Branke J (2001) Reducing the sampling variance when searching for robust solutions. In: Spector L et al. (ed), *GECCO 2001 – Proceedings of the genetic and evolutionary computation conference*. Morgan Kaufmann Publishers, pp 235–242

Christopher M (1998) *Logistics and supply chain management – strategies for reducing cost and improving service*. Financial Times Prentice Hall, London

Dauzère-Pérès S (1995) Minimizing late jobs in the general one machine scheduling problem. *European Journal of Operational Research* 81: 134–142

Dauzère-Pérès S, Sevaux M (2003) Using lagrangean relaxation to minimize the weighted number of late jobs. *Naval Research Logistics* 50 (3): 273–288

Dauzère-Pérès S, Sevaux M (2001) An exact method to minimize the number of late jobs in single machine scheduling. Technical report, Ecole des Mines de Nantes, France, submitted

Davenport AJ, Beck JC (2000) A survey of techniques for scheduling with uncertainty. Unpubl. manuscript

Goldberg DE (1989) *Genetic algorithms in search, optimization and machine learning*. Addison Wesley, ♣ ((insert publ.'s place))

Hart E, Ross P (1999a) The evolution and analysis of a potential antibody library for job-shop scheduling. In: Corne D, Dorigo M, Glover F (eds) *New ideas in optimization*. London, McGraw Hill, pp 185–202

Hart E, Ross P (1999b) An immune system approach to scheduling in changing environments. In: *GECCO 99. Proceedings of the genetic and evolutionary computation conference*. Morgan Kaufmann publishers, ♣, pp 1559–1566

Herroelen W, Leus R (2002) Project scheduling under uncertainty – survey and research potentials. Invited paper to be published in the special issue of EJOR containing selected papers from PMS2002

Holland JH (1975) Adaptation in natural and artificial systems. Technical report, University of Michigan, Ann Arbor

Jensen MT *Robust and flexible scheduling with evolutionary computation*. PhD thesis, University of Aarhus, Dept. of Computer Science, Denmark

Kouvelis P, Yu G (1997) *Robust discrete optimisation and its applications*. (Nonconvex Optimization and its Applications, vol. 14). Kluwer Academic Publishers, Dordrecht

Lenstra JK, Rinnooy Kan AHG , Brucker P (1977) Complexity of machine scheduling problems. *Annals of Discrete Mathematics* 1: 343–362

Leon VJ, Wu SD, Storer RH (1993) Robust scheduling and game-theoretic control for short-term scheduling of job-shops. In: Fandel G, Gulledge T, Jones A (eds) *Operations research in production planning and control*. Springer, Berlin Heidelberg New York, pp 321–335

Leon VJ, Wu SD, Storer RH (1994a) A game-theoretic control approach for job shops in the presence of disruptions. *International Journal of Production Research* 32 (6): 1451–1476

Leon VJ, Wu SD, Storer RH (1994b) Robustness measures and robust scheduling for job shops. *IIE Transactions* 26: 32–43

Levenshtein VI (1966) Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics – Doklady* 10: 707–710

Portmann MC (1996) Genetic algorithm and scheduling: a state of the art and some propositions. In: *Proceedings of the workshop on production planning and control*. Mons, Belgium, pp I–XIV

Reeves CR (ed) (1993) *Modern heuristic techniques for combinatorial problems*. Wiley, New York

Reeves CR (1997) Genetic algorithms for the operations researcher. *Informs Journal on Computing* 9 (3): 231–250

Sevaux M, Dauzère-Pérès S (2002) Genetic algorithms to minimize the weighted number of late jobs on a single machine. *European Journal of Operational Research* (forthcoming)

Sörensen K (2001) Tabu searching for robust solutions. In: *Proceedings of the* 4^{th} *metaheuristics international conference*. Porto, Portugal, pp 707–712

Tsutsui S (1999) A comparative study on the effects of adding perturbations to phenotypic parameters in genetic algorithms with a robust solution searching scheme. In: *Proceedings of the 1999 IEEE systems, man, and cybernetics conference (SMC'99 Tokyo)*. pp III–585–591

Tsutsui S, Ghosh A (1997) Genetic algorithms with a robust solution searching scheme. *IEEE Transactions on Evolutionary Computation* 1: 201–208

Tsutsui S, Ghosh A, Fujimoto Y (1996) A robust solution searching scheme in genetic search. In: Voigt HM, Ebeling W, Rechenberg I, Schwefel HP (eds) *Parallel problem solving from nature – PPSN IV*, vol. 10. Springer, Berlin Heidelberg New York, pp 543–552,

Tsutsui S, Jain JC (1998) Properties of robust solution searching in multi-dimensional space with genetic algorithms. In: *Proceedings of the 2nd International Conference on knowledge-based electronic systems (KES-98)*.

Ukkonen E (1985) Finding approximate patterns in strings. *Journal of Algorithms* 6: 132–137

Wagner RA, Fischer MJ (1974) The string-to-string correction problem. *Journal of the ACM* 21: 168–173

GA|PM: genetic algorithms with population management

Kenneth Sörensen and Marc Sevaux

February 2004

Abstract

We present a new type of evolutionary algorithms called genetic algorithms with population management or GA|PM. The main distinguishing feature of GA|PM is the use of distance measures to control the diversity of a small population. Combined with a local search procedure to ensure the high quality of individual solutions in the population, population management strategies can be developed to avoid slow or premature convergence and achieve excellent performance.

The new algorithm is tested on two problems: the multidimensional knapsack problem and the weighted tardiness single machine scheduling problem. On both problems, population management is shown to be able to improve the performance of an evolutionary algorithm.

1 Introduction

"Classical" genetic algorithms (GA) —pioneered by among others Holland [15] and Goldberg [12]— have been shown experimentally to perform rather poorly on many combinatorial optimization problems. The performance of many GA implementations is characterized by large memory requirements, large running times and poor performance in terms of solution quality.

Commonly referred to as evolutionary algorithms (EA), many variants of genetic algorithms have been proposed to improve their performance. Perhaps most successful in combinatorial optimization are *memetic algorithms* [20], also called *hybrid genetic algorithms* or *genetic local search*. These algorithms apply *local search operators* to improve the quality of individual solutions in the population. Besides hybridizing genetic algorithms, the use of more natural encodings (instead of binary ones) is also widely recognized as contributing to the quality of an EA, as is the use of crossover operators that preserve the structure of good parts of solutions (called *building blocks*). That this is not a trivial task is exemplified by van Kampen and Buydens [30], who show that their genetic algorithm is easily outperformed by a simulated annealing approach because the crossover operator is unable to preserve the fitness of high-quality building blocks.

Nevertheless, even EA with all these improvements may still fall victim to either slow or premature convergence. Many researchers agree that the quality of a metaheuristic optimization approach is largely a result of the interplay between *intensification* and *diversification* strategies (see e.g. Ferland et al. [7], Laguna et al. [16]). One of the main motivations for this paper is the observation that the design of evolutionary algorithms makes it particularly difficult to control the balance between intensification and diversification. As Hertz and Widmer [14] point out, preserving the diversity of the population of an evolutionary algorithm is necessary. Although EA have the operators to increase or decrease the diversity of the population, most lack the means to *control* this diversification. On the one hand, crossover and mutation operators tend to diversify the population by introducing new or recombining existing genetic material. As a result of too much diversification, some EA exhibit extremely slow convergence. On the other hand, the hybridization (local search) operators tend to focus the search in specific regions. This may even lead to so-called *premature convergence*: a total loss of diversity in the population.

This problem is tackled by a new class of evolutionary algorithms that we propose: GA|PM or genetic algorithms with population management [28]. *Population management* works by measuring and controlling the diversity of a small population of high-quality solutions, and makes ample use of *distance measures*. The distance measure to use depends on the representation of a solution. For binary problems, the Hamming distance can be used. For permutation problems, several distance measures have been proposed in the literature, e.g. [2, 25, 26]. The distance measures used in this paper are based on the *edit distance* (see e.g. Wagner and Fischer [31]), that is well known and easily adaptable to a large number of combinatorial problems [27].

Working with a small population (Greistorfer and Voß [13] call it a pool) is an idea that is incorporated in a number of metaheuristic approaches, but a common framework for creating and maintaining such a population has not yet been established. *Scatter search* and *path relinking* [9] are approaches that work by combining solutions from a small set of high-quality (called *elite*) solutions. As mentioned, memetic algorithms [20] combine genetic algorithms with local search to obtain a small population of high- quality solutions. Some work on standard genetic algorithms using a small population is due to Reeves [23].

GA|PM show some resemblance to memetic algorithms [20] but they differ from them in that GA|PM use population management. Other approaches that use a small population of high quality solutions include *scatter search* and *path relinking* [9]. GA|PM offer the advantage of being closer to classical EA in terms of algorithm structure and therefore considerably easier to implement. The use of distance measure for GA design has been proposed for maintaining a diverse population [19] or to locate a set of dif-

ferent solutions of a multimodal problem such as in crowding [18] or fitness sharing [11]. GA|PM differ from these approaches in that they maintain a small population of locally improved solutions and use adaptive population management strategies.

The rest of this paper is structured as follows. Section 2 describes the basic structure of a GA|PM and discusses how distance measures can be used to control the diversity of a population. Some possible population management strategies are also discussed. In section 3, the algorithm is tested on the multidimensional knapsack problem and the total weighted tardiness single-machine scheduling problems. For both problems, we compare the GA|PM to a hybrid evolutionary algorithm without population management. Although the population management improves the performance of the EA in a big way in both cases, we should remark that the main goal of these applications was to show the effectiveness of population management, not to create solution methods that could compete with the best-performing approaches. That GA|PM can be used to create first-class optimization methods, is currently being shown in ongoing research. In Sörensen [28], a GA|PM for the vehicle routing problem (VRP) is shown to produce results competitive to the best-known approaches. In Prins et al. [22], a GA|PM is developed for the capacitated arc routing problem (CARP). To date, this GA|PM is the best-known approach for this problem.

2 GA|PM

This section describes the new genetic algorithm with population management. Its main distinguishing features are

- a small population (typically 10 to 30 solutions),
- a local improvement operator (e.g. a local search, or simple tabu search procedure),
- *population management* to control the diversity of the population.

2.1 Algorithm overview

A genetic algorithm with population management is structured much like a standard genetic algorithm, but differs in the use of population management and local search. An outline is given in algorithm 1.

Initially, a small population is built randomly or using initial heuristics. From this population, two *parent* solutions are selected and subjected to the crossover operator, forming one or two new *offspring* solutions. These solutions are improved by a local search operator and added to the population, after being subjected to population management.

Algorithm 1 GA|PM outline

1: *initialize* population P
2: set population diversity parameter Δ
3: **repeat**
4: *select:* p_1 and p_2 from P
5: *crossover:* $p_1 \otimes p_2 \rightarrow o_1, o_2$
6: *local search:* improve o_1 and o_2
7: **for** each offspring o **do**
8: **while** o does not satisfy conditions for addition (*input function*) **do**
9: mutate o
10: **end while**
11: *remove solution:* $P \leftarrow P \backslash b$
12: *add solution:* $P \leftarrow P \cup o$
13: **end for**
14: update diversity parameter Δ
15: **until** stopping criterion satisfied

2.2 Population management

Population management controls the diversity of a small population of high-quality solutions. It uses an *input function* that determines whether an offspring solution is added to the population or not. In GA|PM, this input function takes the following two factors into account:

- the quality of the solution, and
- the diversity of the population after addition of the solution, or —in other words— the contribution that the solution makes to the diversity of the population; this is measured as the "distance" of the solution to the population, see section 2.2.2.

2.2.1 Distance measures

To evaluate whether a candidate solution sufficiently diversifies the population, a distance measure d is used that determines for each pair of solutions their relative distance (or similarity). The distance should be measured *in the solution space* and not —as is commonly done— in the objective function space. Distance measures cannot be developed independent of the problem or even the representation (encoding) of the solution. For binary representations, the Hamming distance can be used. For solutions that are encoded as vectors of real numbers, some form of the Minkowsky-r-distance[1]

[1] $d(\vec{x}, \vec{y}) = \left(\sum_{i=1}^{n} |x_i - x_j|^r \right)^{\frac{1}{r}}$

is appropriate (e.g. Euclidean, Manhattan, Chebychev).

For permutation problems, several distance measures have been developed in the literature. A GA|PM developed in this paper (see section 3.2.1) uses the so-called *edit distance*. The edit distance can be calculated between two strings composed of characters from a finite alphabet. The edit distance is the number of *edit operations* required to transform the first string into the second one[2]. Three *edit operations* are defined: insertion of a character, deletion of a character and substitution of a character by another one. Using a simple dynamic programming algorithm [31], the edit distance can be calculated in $O(n^2)$. Other, more efficient algorithms have been developed, e.g. [29]. The edit distance can be modified for different types of permutation problems, where solutions cannot be represented as a simple string. An example is the traveling salesman problem, in which a solution does not have a fixed starting position. For a more elaborate discussion, we refer to Sörensen [27].

2.2.2 Distance of a solution to the population

Given a distance measure that can calculate the distance between any pair of solutions, the distance of a given solution s_k to the population can be calculated as follows:

$$d_P(s_k) = \min_{s_i \in P} d(s_k, s_i). \tag{1}$$

Calculating the distance of a solution to the population requires calculating $|P|$ distance measures (where $|P|$ is the cardinality of the population). This high computational requirement is one of the reasons why population management techniques are more effective when applied to small populations.

2.2.3 Input function and diversity parameter

It is obvious that a solution that has a small distance to another solution already in the population, will not contribute much to the diversity of a population. Therefore, a solution is not added to the population if its distance to the population is below a certain threshold Δ. We call Δ the *diversity parameter*. Assuming that the quality of s_k is sufficient, a solution can be added to the population if the following holds:

$$d_P(s_k) = \min_{s_i \in P} d(s_k, s_i) \geq \Delta. \tag{2}$$

Using the distance $d_P(s_k)$ and the fitness or objective function value $f(s_k)$, the input function can also use a multi-objective decision method to

[2]Often, a cost is assigned to each possible edit operation and the edit distance is defined as the minimum total cost of all edit operations required to transform the first string into the second.

determine whether s_k should be added to the population or not. A very simple way of doing this (assuming that f should be minimized) is to calculate $f(s_k) + \lambda d_P(s_k)$. If this value does not exceed a certain threshold, the solution is added, otherwise it is discarded. λ is a parameter that determines the relative importance of the diversity with respect to the solution quality. Of course, more elaborate multi-objective decision methods can be used.

If the local search procedure is effective enough to always ensure the quality of solutions it produces, a solution can be added if eq. (2) holds, without taking the objective function value of the solution into account.

As shown in algorithm 1, a solution that does not have a sufficiently large distance to the population, is randomly mutated until it does. Of course, other strategies are possible, such as simply discarding the solution.

2.2.4 Population management strategies

Using the diversity parameter Δ, the diversity of the population can be controlled as higher values for Δ will increase the diversity of the population while lower values will decrease it. A high value of Δ will allow only solutions that have a large distance to all solutions in the population and will lead —perhaps after a few iterations— to a population that consists of very different solutions. A low value of Δ will allow solutions in the population that are relatively similar to solutions already in the population. This will result in a less diverse population.

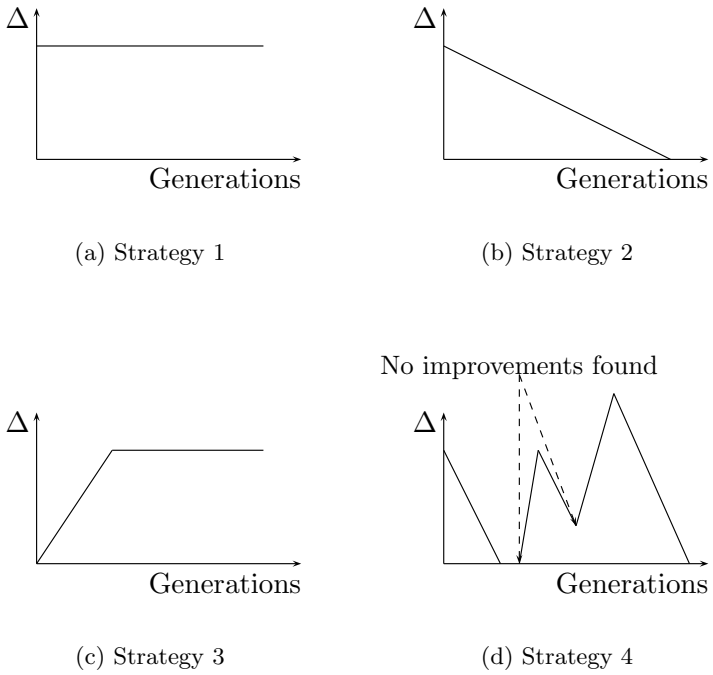
Several population management strategies can be proposed, using only the diversity parameter Δ. The following lists several potential strategies in increasing order of complexity.

Strategy 1: Δ is set to a constant level. This strategy prevents population convergence, and introduces a constant level of diversification into the population.

Strategy 2: Δ is set to a high level in the beginning of the algorithm. The value of Δ is decreased steadily throughout the working of the algorithm, allowing increased intensification near the end of the run.

Strategy 3: A closely related alternative strategy is to create a few good solutions (low Δ) in the beginning of the algorithm and allow more diversification near the end. A variant of this strategy is used in the experiments.

Strategy 4: Δ is set to a high level in the beginning of the algorithm. It is steadily decreased as better solutions are found. When no improvement of the best solution is found for a set number of iterations, the diversity is increased by increasing Δ, thus introducing new genetic material in the population. After this, Δ is steadily decreased again,



(a) Strategy 1

(b) Strategy 2

(c) Strategy 3

(d) Strategy 4

Figure 1: Population management strategies

etcetera. This strategy can be called *adaptive* because it uses information about the effectiveness of the search to dynamically control the diversity of the population.

These strategies are graphically represented in figure 1. A higher complexity of a population management strategy also implies an increase in the number of parameters that has to be determined. Elaborate strategies like 3 and 4 are especially useful when the search space is large and algorithm runs are long.

2.3 Selection, crossover and other genetic operators

Several genetic operators have to be defined in order for the algorithm to work. The selection operator —used for the selection of parent solutions and for the selection of solutions to remove from the population when new solutions need to be added— can be a binary tournament selection, a roulette wheel selection or any other selection method.

Crossover operators should be designed to preserve the good features of the parent solutions as much as possible. A bad choice of crossover operator will result in offspring solutions that most likely have a rather poor quality.

For the design of efficient selection and crossover operators, we refer to the specialized literature. See e.g. Reeves [24] for a discussion.

2.4 Intensification

A local search procedure is necessary to maintain a population of high-quality solutions. This local search procedure should be able to quickly improve the quality of a solution produced by the crossover operator, without diversifying it into other regions of the search space. Neighbourhood -search methods like simple tabu search approaches are particularly useful for this purpose.

3 Experiments

In this section, we apply the principles of GA|PM to two problems: the multidimensional knapsack problem and the total weighted tardiness single machine scheduling problem. We should note that in none of the two problems, it is our intention to compete with the best approaches in the literature as this would involve developing specialized crossover operators and advanced local search operators and would distract the attention from the main point: the population management of GA|PM. Instead, we focus on the population management and compare the GA|PM we develop to comparable EA that lack population management.

3.1 The multidimensional knapsack problem

The $\mathcal{NP}$-hard 0/1 multidimensional knapsack problem (MKP01) is a pure 0/1 integer programming problem. Given a set of n items each having an associated *profit* c_i, the objective of this problem is to select the subset of items that maximizes the total profit, while satisfying a set of knapsack constraints. For each knapsack j, each item i has a given *weight* a_{ij} and the sum of the weights of the chosen items is not allowed to exceed the knapsack capacity b_j. More formally, the MKP01 can be stated as follows:

$$\begin{aligned} \max & \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} & \sum_{i=1}^{m} a_{ij} x_i \leq b_j \quad \forall j \in [1,m] \\ & x_i \in 0,1 \qquad\quad \forall i \in [1,n] \end{aligned} \tag{3}$$

Several approaches have been proposed for this problem, including genetic algorithms [3] and tabu search [6, 10].

3.1.1 A GA|PM for the MKP01

A solution of the MKP01 can be represented as a binary string of length n. We set the objective function value of a feasible solution equal to the total profit (i.e. $\sum_i c_i x_i$). For infeasible solutions, we set the objective function value equal to the (negative) total violation of all constraints combined, i.e.

$$\sum_j (b_j - \sum_i a_{ij} x_i) \tag{4}$$

Since feasible solutions always have a positive evaluation function value and infeasible solutions a negative one, a feasible solution is always preferred and the search will not return an infeasible solution unless no feasible solution could be found. Assigning a negative objective function value to an infeasible solution allows us to use the local search procedure to improve such solutions, assuming that the quality of an infeasible solution is determined by the total violation of the constraints.

A simple *steepest descent* local search procedure is used. This procedure attempts to change the status of every item of a given solution and changes the status of the item that yields the largest increase in objective function value. The search continues until the objective function value cannot be increased by changing the status of a single item.

The distance between two solutions is measured as the Hamming distance between their binary strings. As usual, the distance of a solution to the population is given by the minimum distance of this solution to any solution already in the population.

Selection is done by a binary tournament method: two solutions are randomly chosen and the best one is selected. A simple one-point crossover is used to combine solutions. Each crossover produces two offspring solutions that are locally improved using the local search procedure and then subjected to population management.

Population management determines the distance $d_P(s)$ of a given solution to the population and compares this value to the population diversity parameter Δ. If this distance is smaller than Δ, the solution s is mutated. This is done by randomly flipping $\Delta - d_P(s)$ bits of s. This procedure does not ensure that the mutated solution has a sufficiently large distance to the population, but it diversifies it without causing too much overhead.

The performance of the GA|PM is compared to the performance of the evolutionary algorithm that results from removing the population management component from the GA|PM and replacing it with random mutation. From now on, we refer to this procedure as HGA (hybrid GA).

3.1.2 Experiments

The GA|PM is tested on 48 well-known test problems available from the OR library[3].

A first experiment uses the simplest population management for the GA|PM, i.e. $\Delta = 1$. A second experiment uses more advanced population management (a variant of strategy 3 in figure 1) with

$$\Delta = \frac{\text{current iteration}}{\text{maximum iterations}} \times \frac{n}{5},$$

where n is the number of items. This strategy sets Δ to 0 in the beginning of the algorithm and linearly increases it to a maximum of $n/5$.

A population of 10 is used in all experiments. The HGA uses a 10% random mutation rate. Each algorithm is awarded 10000 generations to find the optimal solution. Time is measured when the optimal solution is found or at the end of the 10000 generations when it is not found. Each experiment is repeated 5 times.

3.1.3 Numerical comparison

Results show that the GA|PM outperforms the HGA. The average quality over the 5 runs of the solutions produced by the GA|PM is better in all cases. In 46 out of 55 cases (84%), the worst solution found by the GA|PM is better than the best solution found by the HGA. More detailed results can be found in table 1.

	Nr. opt.	Min. 1 opt.	Avg. it.	Avg. CPU
GA|PM strategy 1	3.84	87.27%	3537	6.43
GA|PM strategy 3	3.73	92.59%	3879	10.00
HGA	0.51	25.45%	9206	9.38

Table 1: Comparison GA|PM and HGA for the multidimensional knapsack problem

This table should be read as follows. "Nr. opt." is the average number of times the optimal solution was found over the 5 runs, "Min. 1 opt." is the average percentage of instances for which the optimal solution was found at least in one of the 5 runs, "Avg. it." is the average number of iterations, and "Avg. CPU (s)" is the average time required in seconds.

As can be seen, the GA|PM performs much better than the HGA without population management. The average time per iteration is somewhat higher, but the solutions found are much better and the GA|PM therefore performs less iterations. The more advanced population management increases the average number of iterations required and also the average CPU time. It

[3] `http://www.ms.ic.ac.uk/jeb/pub/mknap2.txt`

does however improve the robustness of the algorithm in that the optimal solution is now found in over 92% of the problem instances.

3.2 The total weighted tardiness single machine scheduling problem

A set of n jobs has to be sequenced on a single machine. Preemption is not allowed. Jobs are not available before a release date r_j and are processed for p_j units of time. For each job, a due date d_j and a weight w_j is given. If C_j denotes the completion time of job j, the tardiness is defined by $T_j = \max(0, C_j - d_j)$. The objective is to minimize the total weighted tardiness $(1|r_j|\sum w_j T_j)$. This problem is $\mathcal{NP}$-Hard in a strong sense.

The total weighted tardiness problem with release dates is probably one of the most difficult one machine scheduling problem. If all the weights are equal to one $(1|r_j|\sum T_j)$ the problem is $\mathcal{NP}$-Hard in a strong sense [8]. For different weights and the release dates all equal to zero $(1||\sum w_j T_j)$, the problem is also $\mathcal{NP}$-Hard in a strong sense [17].

Recent approaches on the $1||\sum w_j T_j$ problem are proposed in the literature. In [5], a comparison between simulated annealing, tabu search and genetic algorithms is discussed and in [4] the iterated dynasearch algorithm is presented and shown to yield the best solutions. Results are reported in the OR-Library. A recent constraint-based approach [1] can solve problem instances with up to 40 jobs.

3.2.1 A GA|PM for the total weighted tardiness problem

In this section, we develop a simple GA|PM for the total weighted tardiness problem and compare it to a similar hybrid GA that lacks population management. To perform a fair comparison between the two approaches, common components are used for the HGA and GA|PM algorithms.

The *selection* process used is a binary tournament selection, the *crossover* operator is the standard LOX operator [21] and the *mutation* operator is the GPI (general pairwise interchange). The local search procedure is also based on the GPI and a steepest descent algorithm is applied. The algorithm is stopped after 60 seconds of CPU time.

For the numerical experiments, we use a variation of strategy 3. The distance measure used is the *edit distance* discussed in section 2.2.1. In the implementation, Δ is measured in percentage (result of the distance measure divided by the maximum distance value). At the initialization step, Δ is fixed to 1%. Every 15 iterations, Δ is multiplied by a growth factor (1.05). When an upper limit of 50% is reached, Δ is not increased anymore, since this would not allow the algorithm to find a new solution that is accepted during the search.

In the HGA, not all solutions are subjected to local search as this almost always induces premature convergence. Instead, the local search rate p_{ls} is set to 10%, the mutation rate p_m is fixed to 10%. The size of the population is set to 25. For the GA|PM, the size of the population is only 10 individuals.

3.2.2 Numerical comparison

Results are reported in different tables. Table 2 summarizes the results for the ODD instances (these instances can be obtained by email request). Optimal solutions are not known for this set. The first column indicates the set of instances (ODD20 is the ODD instances with 20 jobs), the following three columns provides the results for the GA|PM method and the next three columns, the results for the HGA method. Note that for a fair comparison, the same amount of time (60s) has been given to the GA|PM and the HGA.

	GAPM results			HGA results		
Set of inst.	First pos.	Avg. gap (in %)	Avg. iter.	First pos.	Avg. gap (in %)	Avg. iter.
ODD20	0	0.000	32375	0	0.000	952757
ODD40	0	0.000	4142	0	0.000	258526
ODD60	6	0.088	1685	1	0.251	88106
ODD80	6	0.003	1153	1	0.083	39040
ODD100	8	0.064	844	4	0.118	19854
Global	20	0.057	8040	6	0.148	271657

Table 2: Numerical results for ODD instances, CPU time = 60s

For each of the methods, the first column gives the number of times the method reports the best solution (among the two methods), the second column is the average deviation from the best solution. The last column gives the average number of iterations done in 60 seconds.

The OR-Library instances are generated for the $1||\sum w_j T_j$ problem and can be solved by our approach. Optimal values of solutions are available for 124 and 115 out of 125 of the 40 and 50 job problem instances, respectively. When the optimal solution is unknown, best solutions provided by [5] are used instead (by extension they are also called optimal solutions).

ORLib40 results

Method used	Opt. sol.	First pos.	Avg. Gap (in %)	Iterations Avg.	Max
GAPM	125	4	0.000	5682	8961
HGA	121	0	0.284	327763	470984

ORLib50 results

Method used	Opt. sol.	First pos.	Av. Gap (in %)	Iterations Aver.	Max
GAPM	123	11	0.002	3334	5298
HGA	113	0	0.595	196024	287802

ORLib100 results

Method used	Best. sol.	First pos.	Av. Gap (in %)	Iterations Aver.	Max
GAPM	94	27	0.276	1030	1480
HGA	85	16	2.110	34164	56521

Table 3: Numerical results for OR-Library instances, CPU time = 60s

In table 3 the column "Opt. sol." ("Best sol." for 100 job instances) counts the number of times the algorithm find the optimal solution (or best known). The rest of the columns is identical to the previous table except that the deviation is measured from the optimal solution. The maximum number of iterations is added too. Again, the same amount of time has been allocated to the two methods for a fair comparison.

These results show that the GA|PM approach performs better than the HGA approach, even though it operates on a smaller population. For the OR-Library instances, more optimal solutions are found and the approach is ranked first in more cases. Moreover, the deviation from the optimal (or best known) solution is always smaller with the proposed method. When we use the same stopping conditions (fixed number of iterations without improvement of the best solution) the difference between the two algorithms is even greater, but the GA|PM requires more time.

4 Conclusions

It is commonly known that the performance of an evolutionary algorithm (and any metaheuristic) on a hard combinatorial optimization problem depends in a big way on the balance between intensification and diversification. Although all evolutionary algorithms possess the necessary operators for intensification and diversification, many EA implementations lack a mechanism to control the balance between these two factors. This paper introduced a new class of evolutionary algorithms, GA|PM or genetic algorithms with population management, that addresses this issue. GA|PM are hybrid evolutionary algorithms that —like scatter search— operate on a small population of high-quality individuals. The main distinguishing feature of

a GA|PM is the application of *population management strategies* to control the diversity of the population. We discussed how such strategies can be easily implemented by controlling the value of only a single parameter, the population diversity parameter Δ.

The proposed approach was tested by applying it to two different combinatorial problems: the multidimensional knapsack problem (a pure 0/1 integer programming problem) and the total weighted tardiness single machine scheduling problem, a difficult permutation problem. We discussed that different problems require different distance measures and proposed to use the edit distance for permutation problems. For binary problems, the Hamming distance was used. Numerical comparison showed that the GA|PM outperformed very similar hybrid genetic algorithms without population management (but with random mutation instead).

References

[1] P. Baptiste, A. Jouglet, and J. Carlier. A branch-and-bound procedure to minimize total tardiness on one machine with arbitrary release dates. *European Journal of Operational Research*, 2004. To appear.

[2] V. Campos, M. Laguna, and R. Martí. Context-independent scatter and tabu search for permutation problems. *INFORMS Journal on Computing*, to appear, 2003. URL `http://leeds.colorado.edu/Faculty/Laguna/publications.htm`.

[3] P.C. Chu and J.E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristic*, 4:63–86, 1998.

[4] R. Congram, C.N. Potts, and S. van de Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness schedulign problem. *Informs journal on computing*, 14(1):52–67, 2002.

[5] H.A.J. Crauwels, C.N. Potts, and L.N. Van Wassenhove. Local search heuristics for the single machine total weighted tardiness scheduling problem. *Informs Journal of Computing*, 10(3):341–350, 1998.

[6] F. Dammeyer and S. Voß. Dynamic tabu list management using the reverse elimination method. *Annals of Operations Research*, 41:31–46, 1993.

[7] J.A. Ferland, S. Ichoua, A. Lavoie, and E. Gagné. Scheduling using tabu search methods with intensification and diversification. *Computers and Operations Research*, 28:1075–1092, 2001.

[8] M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to theory of np-completeness*. Freeman, San Franscisco, USA, 1979.

[9] F. Glover. A template for scatter search and path relinking. In J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Artificial Evolution*, volume 1363 of *Lecture Notes in Computer Science*, pages 13–54, Springer, Berlin, 1998.

[10] F. Glover and G.A. Kochenberger. Critical event tabu search for the multidimensional knapsack problem. In I.H. Osman and J.P. Kelly, editors, *Metaheuristics: The Theory and Applications*, pages 407–427. Kluwer Academic Publishers, Boston, 1996.

[11] D. Goldberg. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, 1987.

[12] D.E. Goldberg. *Genetic Algorithms in search, Optimization and Machine Learning.* Addison Wesley, 1989.

[13] P. Greistorfer and S. Voß. Controlled pool maintenance in combinatorial optimization. In C. Rego and B. Alidaee, editors, *Adaptive Memory and Evolution: Tabu Search and Scatter Search.* Kluwer Academic Publishers, Boston, To appear.

[14] A. Hertz and M. Widmer. Guidelines for the use of meta-heuristics in combinatorial optimization. *European Journal of Operational Research*, 151(2):247–252, 2003.

[15] J.H. Holland. Adaptation in natural and artificial systems. Technical report, University of Michigan, Ann Arbor, 1975.

[16] M. Laguna, R. Martí, and V. Campos. Intensification and diversification with elite tabu search solutions for the linear ordering problem. *Computers and Operations Research*, 26:1217–1230, 1999.

[17] E.L. Lawler. A 'pseudo-polynomial' algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, 1:331–342, 1977.

[18] S.W. Mahfoud. Crowding and preselection revisited. In R. Manner and B. Manderick, editors, *Parallel Problem Solving from Nature*, pages 27–36, Elsevier, Amsterdam, 1992.

[19] M. Mauldin. Maintaining diversity in genetic search. In *Proceedings of the National Conference on Artificial Intelligence*, pages 247–250, 1984.

[20] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P 826, Caltech Concurrent Computation Program, 1989.

[21] M-C. Portmann. Genetic algorithm and scheduling: a state of the art and some propositions. In *Proceedings of the workshop on production planning and control*, pages I–XIV, Mons, Belgium, 1996.

[22] C. Prins, M. Sevaux, and K. Srensen. A genetic algorithm with population management for the CARP. In *Proceedings of TRISTAN V*, Guadaloupe, 2004.

[23] C.R. Reeves. Using genetic algorithms with small populations. In S. Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 92–99, Morgan Kaufman, San Mateo, 1993. Morgan Kaufmann.

[24] C.R. Reeves. Genetic algorithms for the operations researcher. *INFORMS Journal on Computing*, 9:231–250, 1997.

[25] S. Ronald. Distance functions for order-based encodings. In D. Fogel, editor, *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 641–646, IEEE Press, New York, 1997.

[26] S. Ronald. More distance functions for order-based encodings. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 558–563, IEEE Press, New York, 1998.

[27] K. Sörensen. Distance measures based on the edit distance for permutation-type representations. In A. Barry, editor, *Proceedings of the Workshop on Analysis and Design of Representations and Operators (ADoRo), GECCO Conference*, pages 15–21, Chicago, 2003.

[28] K. Sörensen. *A framework for robust and flexible optimisation using metaheuristics with applications in supply chain design.* PhD thesis, University of Antwerp, 2003.

[29] E. Ukkonen. Finding approximate patterns in strings. *Journal of Algorithms*, 6:132–137, 1985.

[30] A.H.C. van Kampen and L.M.C. Buydens. The ineffectiveness of recombination in a genetic algorithm for the structure elucidation of a heptapetide in torsion angle space. A comparison to simulated annealing. *Chemometrics and Intelligent Laboratory Systems*, 36:141–152, 1997.

[31] R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21:168–173, 1974.

A genetic algorithm for a bi-objective capacitated arc routing problem

P. Lacomme[1], C. Prins[2] and M. Sevaux[3*]

[1] *Université Blaise Pascal, LIMOS, Campus Universitaire des Cézeaux, BP 10125, F-63177 Aubière Cedex, France*

[2] *Université de Technologie de Troyes, LOSI, 12 Rue Marie Curie, BP 2060, F-10010 Troyes Cedex, France*

[3] *Université de Valenciennes, LAMIH/SP, Le Mont Houy, F-59313 Valenciennes Cedex 9, France*

June 2003

Abstract

The Capacitated Arc Routing Problem (CARP) is a very hard vehicle routing problem for which the objective – in its classical form – is the minimisation of the total cost of the routes. In addition, one can seek to minimize also the cost of the longest trip. In this paper, a multi-objective genetic algorithm is presented for this more realistic CARP. Inspired by the second version of the Non-dominated Sorted Genetic Algorithm framework (NSGA-II), the procedure is improved by using good constructive heuristics to seed the initial population and by including a local search procedure. The new framework and its different flavours are appraised on three sets of classical CARP instances comprising 81 files. Yet designed for a bi-objective problem, the best versions are competitive with state-of-the-art metaheuristics for the single objective CARP, both in terms of solution quality and computational efficiency: indeed, they retrieve a majority of proven optima and improve two best-known solutions.

Keywords: Capacitated Arc Routing Problem; Multi-objective optimization; Metaheuristic; Genetic algorithm.

1 Introduction: the CARP and its bi-objective version

The Capacitated Arc Routing Problem (CARP) in its undirected version is defined as follows. Let $G = (V, E)$ be an undirected graph with a set V of n nodes and a set E of m edges. A fleet of identical vehicles of capacity W is based at a depot node s. The number of vehicles is either fixed or left as a decision variable. A subset of τ edges, denoted as *required edges* or *tasks*, require service by a vehicle. All edges can be traversed any number of times. Each edge (i, j) has a traversal cost c_{ij} and a demand q_{ij}. All costs and demands are integers. The goal is to determine a set of vehicle trips (routes) of minimum total cost, such that each trip starts and ends at the depot, each required edge is serviced by one single trip, and the total demand handled by any vehicle does not exceed W.

*Corresponding author. Tel: +33-327-51-1326, fax: +33-327-51-1310.
E-mail address: Marc.Sevaux@univ-valenciennes.fr (M. Sevaux)

Since the CARP is $\mathcal{NP}$-hard, large scale instances must be solved in practice with heuristics. Among fast constructive methods, one can cite for instance Path-Scanning [1], Augment-Merge [2] and Ulusoy's splitting technique [3]. Available metaheuristics include tabu search algorithms [4,5], a tabu search combined with scatter search components [6], a variable neighborhood search [7], a guided local search [8], and hybrid genetic algorithms [9, 10]. All these heuristics can be evaluated thanks to tight lower bounds [11, 12].

CARP problems are raised by operations on street networks, e.g. urban waste collection, snow plowing, sweeping, gritting, etc. Economically speaking, the most important application certainly is municipal refuse collection. In that case, nodes in G correspond to crossroads, while edges in E model street segments that connect crossroads. The demands are amounts of waste to be collected. The costs correspond either to distances or durations.

The single objective CARP only deals with minimizing the total cost of the trips. In fact, most waste management companies are also interested in balancing the trips. For instance, in Troyes (France), all trucks leave the depot at 6 am and waste collection must be completed as soon as possible to assign the crews to other tasks, e.g. sorting the waste at a recycling facility. Hence, the company wishes to solve a bi-objective version of the CARP, in which both the total duration of the trips and the duration of the longest trip (corresponding to the *makespan* in scheduling theory) are to be minimized. A multiobjective genetic algorithm (MOGA) for this bi-objective CARP is investigated in this paper which extends preliminary results presented at the EMO 2003 conference [13]. As the two objectives are conflicting, the fleet size (equal to the number of trips) is not imposed, in order to get smaller makespan values. This is not a problem because fleet size is naturally bounded above by τ (when each trip contains one single task).

For a complete introduction to multi-objective optimisation, we refer the reader to a recent annotated bibliography [18] which provides a suitable entry point for general definitions and good references. Concerning the MOGAs, two books are entirely devoted to these algorithms [19, 20].

In the last decade, there has been a growing interest in multi-objective vehicle routing, but all published papers deal with node routing problems, e.g. [14–17]. Moreover, apart from the total cost of the trips, they only consider additional objectives such as minimizing time window violations or balancing vehicle loads, but not makespan. To the best of our knowledge, this paper presents the first study devoted to a multi-objective arc routing problem and with one objective related to the makespan. It is organized as follows. Section 2 recalls a general framework of multi-objective genetic algorithm, called NSGA-II, and describes how to adapt it for the bi-objective CARP. The inclusion of a local search procedure is discussed in Section 3. Numerical experiments are conducted in Section 4. Section 5 brings some concluding remarks.

2 A genetic algorithm for the bi-objective CARP

Today, several MOGA frameworks are available in literature and selecting the best one for a given MOOP is not obvious. A recent survey [21] and two comparative studies [22, 23] try to provide guidelines for selection but these papers consider unconstrained problems, continuous objective functions, and specific sets of benchmarks. It is very difficult to draw conclusions for constrained combinatorial optimization problems.

Our choice for the bi-objective CARP finally turned to a MOGA framework called NSGA-II (Non-Dominated Sorting GA) and designed by Deb [20,24]. The reasons are: a) its modular

and flexible structure, b) the possibility of upgrading a single-objective GA to NSGA-II (the encoding of chromosomes and crossover operators can be recycled) and c) its successful applications to a wide range of problems. The first subsection is necessary to introduce the non-trivial framework of NSGA-II, while correcting minor bugs and typos from [20, 24]. The adaptation to the bi-objective CARP is described in the second subsection.

2.1 The NSGA-II framework

Non-dominated sorting algorithm The NSGA-II algorithm computes successive generations on a population of solutions partitioned into *non-dominated fronts*. The non-dominated set is identified and constitutes the non-dominated front of level 1 or front 1. This set is temporarily disregarded from the population. The non-dominated set of the remaining population gives the non-dominated front of level 2, and so on. This process called *non-dominated sorting* is repeated until the partition is complete, see the left part of Figure 1 for an example.

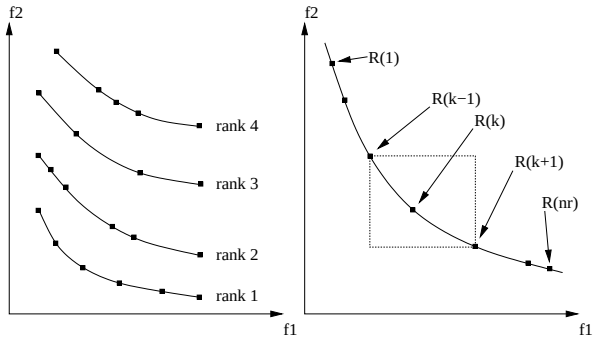


Figure 1: Non-dominated sorting (left) and crowding distances (right)

The method is detailed in Algorithm 1. The population is implemented as a table *pop* of *ns* solutions and $pop(i)$ denotes the i^{th} solution. The partition is a vector *front* of *ns* lists of solutions indexes, i.e. $i \in front(k)$ means that solution $pop(i)$ is currently assigned to the front of level k. Dominance between two solutions x and y is computed by the $dom(x, y)$ boolean function. $nb\_better(i)$ counts the solutions which dominate solution $pop(i)$ and $set\_worse(i)$ is the set of solutions dominated by $pop(i)$.

The non-dominated sorting algorithm returns the number of fronts *nf*, the partition $front(1), front(2), \ldots, front(nf)$ and a vector *rank* in which $rank(i)$ denotes the index of the front that stores solution $pop(i)$. A first phase computes in $O(nc \cdot ns^2)$ *nb_better* and *set_worse* values, a second one in $O(ns^2)$ assigns solutions to their respective fronts and computes *nf* and *rank*. The overall complexity is then $O(nc \cdot ns^2)$.

Crowded tournament operator In NSGA-II, each solution is assigned a fitness equal to its non-domination level (1 being the best level). Thus, minimization of the fitness is assumed. Each generation generates new solutions by selecting parent-solutions and applying crossover and mutation operators. Each parent is selected using *binary tournament*: two solutions are randomly selected from the population and the fittest one (with the smallest rank) is kept.

This classical selection technique is modified as follows in NSGA-II: when two solutions belong to the same front, the tournament prefers the most isolated one, using a *crowding distance* described below. The crowding distance $margin(i)$ of a solution $pop(i)$ is a kind of

Algorithm 1 procedure non_dominated_sort(pop,ns,front,nf,rank)

```
1: front(1) ← ∅
2: for i ← 1 to ns do
3:    nb_better(i) ← 0
4:    set_worse(i) ← ∅
5:    for j ← 1 to ns do
6:       if dom(pop(i),pop(j)) then
7:          add j to set_worse(i)
8:       else if dom(pop(j),pop(i)) then
9:          nb_better(i) ← nb_better(i)+1
10:      end if
11:   end for
12:   if nb_better(i) = 0 then
13:      add i to front(1)
14:   end if
15: end for
16: nf ← 1
17: loop
18:   for all i in front(nf) do
19:      rank(i) ← nf
20:      for all j in set_worse(i) do
21:         nb_better(j) ← nb_better(j)-1
22:         if nb_better(j) = 0 then
23:            add j to front(nf+1)
24:         end if
25:      end for
26:   end for
27:   exit when front(nf+1) = ∅
28:   nf ← nf+1
29: end loop
```

measure of the search space around it which is not occupied by any other solution in the population. The winner of a tournament between two solutions $pop(i)$ and $pop(j)$ can then be computed thanks to a *crowded tournament operator*, implemented as a boolean function $better(i, j)$ which returns *true* if $rank(i) < rank(j)$ or $(rank(i) = rank(j))$ and $(margin(i) > margin(j))$. The idea is to favor the best solutions, while keeping the fronts well scattered to prevent clusters of solutions.

To simplify, consider $nc = 2$ criteria like in the bi-objective CARP and a front R of nr solutions. Let f_c^{min} and f_c^{max} be the minimum and maximum values of criterion f_c in R, $c = 1, 2$. Sort R in increasing values of the first criterion and let $R(k)$ be the k^{th} solution in the sorted front. The crowding distance of $R(k)$ is defined as follows for $1 < k < nr$:

$$margin(R(k)) = \frac{f_1(R(k)+1) - f_1(R(k)-1)}{f_1^{max} - f_1^{min}} + \frac{f_2(R(k)-1) - f_2(R(k)+1)}{f_2^{max} - f_2^{min}} \qquad (1)$$

For $k = 1$ or $k = nr$, $margin(R(k)) = \infty$. The goal of this convention is to favour the

two extreme points of the front, to try to enlarge it. The right part of Figure 1 depicts the computation of margins. The margin of $R(k)$ is nothing but half of the perimeter of the rectangle indicated by dashed lines.

Generation of new solutions The computation of new solutions at each iteration of NSGA-II can be implemented as a procedure *add_children*, specified by Algorithm 2. The vector *rank* computed by the non_dominated sorting procedure and the vector of crowding distances *margin* are required for the crowded tournament operator. The procedure creates *ns* offsprings which are added at the end of *pop*, thus doubling its size. Any crossover or mutation operator can be used, depending on the problem at hand.

Algorithm 2 procedure add_children(pop,ns,rank,margin)

```
1: input_ns ← ns
2: for ns ← input_ns to 2×input_ns do
3:    draw two distinct parents P1, P2 using the crowded tournament operator
4:    combine P1 and P2, using a crossover operator, to get one new solution S
5:    mutate S with a fixed probability
6:    pop(ns) ← S
7: end for
```

NSGA-II overall structure The general structure of NSGA-II is given in Algorithm 3. An initial population *pop* of *ns* random solutions is built by the *first_pop* procedure and sorted by non-domination, using Algorithm 1. The procedure *get_margins* computes for each solution $pop(i)$ its crowding distance $margin(i)$. Then, each iteration of the main loop starts by calling *add_children*, to create *ns* children which are added at the end of *pop*, see Algorithm 2. Finally, the resulting population with $2 \cdot ns$ solutions is reduced to a new population *newpop* by keeping the *ns* best solutions. To do this, fronts and margins must be updated, using *non_dominated_sort* and *get_margins*. Starting from the front of level 1, complete fronts are then transferred to *newpop* as long as possible. The first front $front(i)$ which could not be accomodated fully is truncated by keeping its most widely spread solutions. This is achieved by arranging its solutions in descending order of the crowding distance values, thanks to the *margin_sort* procedure, and by copying the best solutions until *newpop* contains exactly *ns* solutions. Finally, *pop* receives the contents of *newpop* for the next iteration of the GA.

2.2 GA components for the bi-objective CARP

This section describes the required components to instanciate the NSGA-II framework for our bi-objective CARP. The representation of solutions as chromosomes and their evaluation come from a GA [10] which is currently one of the most effective solution methods for the single objective CARP. Therefore, only a short description is provided here: the reader is invited to refer to [9] or [10] for more details.

2.2.1 Chromosome representation and evaluation

A chromosome is an ordered list of the τ tasks, in which each task may appear as one of its two directions. Implicit shortest paths are assumed between successive tasks. The chromosome

Algorithm 3 procedure nsga2()

```
1: first_pop(pop,ns)
2: non_dominated_sort(pop,ns,front,nf,rank)
3: get_margins(pop,ns,front,nf,margin)
4: repeat
5:    add_children(pop,ns,rank,margin)
6:    non_dominated_sort(pop,ns,front,nf,rank)
7:    get_margins(pop,ns,front,nf,margin)
8:    newpop ← ∅
9:    i ← 1
10:   while |newpop|+|front(i)| ≤ ns do
11:      add front(i) to newpop
12:      i ← i+1
13:   end while
14:   missing ← ns-|newpop|
15:   if missing ≠ 0 then
16:      margin_sort(front,i,margin)
17:      for j ← 1 to missing do
18:         add the j-th solution of front(i) to newpop
19:      end for
20:   end if
21:   pop ← newpop
22: until stopping_criterion
```

does not include trip delimiters and can be viewed as a giant tour for a vehicle with infinite capacity. An $O(\tau^2)$ procedure *Split* described in [10] is used to derive a least total cost CARP solution (subject to the given sequence), by splitting the chromosome (giant tour) into capacity-feasible tours. This technique is based on the computation of a shortest path in an auxiliary graph, in which each arc models one possible feasible tour that can be extracted from the giant tour. The makespan (second objective) corresponds to the maximum duration of the trips computed by *Split*.

In practice, each chromosome is stored in *pop* as a record with three fields: the sequence of tasks and the values computed by *Split* for the total cost and the makespan. The associated solution with its detailed trips is not stored because it is required only at the end of the GA, to print the results. It can be extracted at that time by calling *Split* again.

Following GA terminology, each of our chromosomes represents the *genotype* of a solution, i.e. one abstraction of a solution instead of the solution itself. The actual solution (*phenotype*) is materialized by a evaluation/decoding procedure, here *Split*. The reader may be surprised by the priority given by *Split* to the total cost over the makespan. Indeed, it is possible to adapt *Split* to minimize makespan [10] or even to minimize total cost, subject to an upper bound on the makespan. In fact, all these evaluations are pertinent because they return a non-dominated solution among the possible interpretations of the same chromosome. An example of non-pertinent evaluation is to scan the chromosome, starting from the first task, and to create a new trip each time the current task does not fit vehicle capacity. Clearly, such an evaluation "wastes" the chromosome because it is dominated by *Split*.

A version of *Split* that minimizes total cost was adopted because no good rule is available

to choose amongst pertinent evaluations. Giving priority in that way to the total cost has no negative impact on the overall performance of the GA. Indeed, the aim of a decoding procedure in general is to define a mapping from the set of chromosomes into the set of solutions and to assign reproducible values to the two objectives, but not to optimize in some way. Optimizing is the role of the search mechanism of the GA, with its selections that favour the fittest parents for reproduction.

2.2.2 Initial population and clone management

Most published MOGAs start from an initial population *pop* made of *ns* random chromosomes, but including a few good solutions always accelerates the convergence. The procedure *first_pop* in Algorithm 3 uses three such solutions computed by classical CARP heuristics, namely Path-Scanning [1], Augment-Merge [2] and Ulusoy's heuristic [3]. Each heuristic solution is converted into a chromosome by concatenating its trips. The chromosome is then re-evaluated by *Split* because the resulting solution sometimes dominates the heuristic solution.

Although this is not part of the NSGA-II framework, a sufficient condition is used to prevent identical solutions (*clones*): in the initial population and in the subsequent iterations of the GA, there can be at most one solution with two given values of the objectives, i.e. two solutions cannot occupy the same point in the graphical representation of the fronts (remember that all costs are integers). This technique was included to avoid a progressive colonization of the population by clones (typically 20 to 30% after 100 iterations).

2.2.3 Crossover operators

The advantage of chromosomes without trip delimiters is to allow the use of classical crossovers designed for permutation chromosomes, like the *Linear Order Crossover* (LOX) and the *Order Crossover* (OX). Like the single-objective CARP studied in [10], the OX crossover gave the best results after some preliminary testing. This crossover must be slightly adapted because the chromosomes are not exactly permutations: each required edge may appear as one of its two directions. Implementation details can be found in [10].

Hence, the crossover used in the *add_children* procedure of Algorithm 2 is in fact OX. One of the two children is selected at random and is evaluated by *Split*. It is added to *pop* only if no solution exists with the same objective values, else it is discarded. Because of such rejections, more than *ns* crossovers may be required to double the size of *pop*: the FOR loop of Algorithm 2 must be replaced by a REPEAT ... UNTIL $ns = 2 \times input\_ns$.

2.2.4 Stopping conditions

There is no standard technique to define reliable stopping conditions for stopping a MOGA. In combinatorial optimization and in the single-objective case, metaheuristics can be stopped after a fixed number of iterations without improving the current best solution, or by using a variance criterion. The ways of extending these criteria to several objectives still raise endless discussions in the MOO community. The underlying problem, discussed in the numerical experiments of Section 4, is to define a pertinent comparison between the final sets of efficient solutions computed by two different algorithms on the same instance. For this study, the MOGA is simply stopped after a fixed number of iterations. More sophisticated stopping criteria had no noticeable effect on the final results.

3 Local search procedures

In single-objective optimization, it is well known that a standard GA must be hybridized with a local search procedure to be able to compete with state-of-the-art metaheuristics like tabu search. Such GAs, called *memetic algorithms* by Moscato [25], have raised a growing interest in the last decade: for instance, a dedicated conference (WOMA) is now entirely devoted to them. Hybridization is still rarely used in multi-objective optimization, and only a few authors like Jaszkiewicz [26] start investigating this area. Several difficulties are raised, for instance: a) the notion of local search must be clarified when several objectives must be optimized, b) several locations are possible in the existing MOGA frameworks for calling a local search, and c) the local search must not disturb the search mechanism of the MOGA, for instance by concentrating the solutions into small clusters. This section describes the moves and the general structure of our local search procedure, the criteria used to accept a move, and the possible ways of integrating local search in the MOGA.

3.1 Moves tested and general structure

Our local search procedures work on the individual routes, not on the sequence of tasks defined by a chromosome. Hence, the input chromosome is first converted into a set of routes. The moves include the removal of one or two consecutive tasks from a route, with reinsertion at another position, and 2-opt moves. All moves may involve one or two routes and the two traversal directions of an edge are tested in the reinsertions. Each iteration of the local search scans all these moves in $O(\tau^2)$ to find the *first* improving move (the criteria for deciding if there is an improvement are discussed in the next subsection). The whole local search stops when no more improvement can be found. The trips are then concatenated into a chromosome (giant tour), which is reevaluated by *Split*.

3.2 Acceptance criteria for a move

Four acceptance criteria $accept(S, S')$ were tested to accept a move that transforms the incumbent solution S into a neighbor solution S'. This has lead to four local search procedures called LS1 to LS4. In what follows, $f_1(S)$ and $f_2(S)$ respectively denote the total cost and the makespan of solution S and w_1 is a real number in $[0, 1]$.

LS1: $accept(S, S') = f_1(S') - f_1(S) < 0$;

LS2: $accept(S, S') = f_2(S') - f_2(S) < 0$;

LS3: $accept(S, S') = ((f_1(S') - f_1(S) \leq 0) \text{ and } (f_2(S') - f_2(S) < 0))$
$\text{or } ((f_1(S') - f_1(S) < 0) \text{ and } (f_2(S') - f_2(S) \leq 0))$;

LS4: $accept(S, S') = w_1 \cdot (f_1(S') - f_1(S)) + (1 - w_1) \cdot (f_2(S') - f_2(S) < 0$.

LS1 performs its descent on the total cost, LS2 on the makespan, while LS3 accepts the new solution if and only if it *dominates* the current one. The last local search is a weighted sum method which scalarizes the two objectives into a single objective, by multiplying each objective by a user-supplied weight. In these four versions, no normalization of the objectives is required because the total duration of the trips and the makespan are commensurable and of the same order of magnitude.

3.3 Location of local search in the MOGA

The easiest way to include a local search procedure in the MOGA probably is to replace the mutation in Algorithm 2. The mutation can be suppressed because diversification is inherent to the NSGA-II framework, thanks to crowding distances. However, a systematic local search plays against a good dispersal of solutions in the population. This is why, in practice, an offspring undergoes a non-systematic local search with a fixed probability p_{LS}, typically 10%. This location of the local search was the only one tested for LS1, LS2 and LS3. It is called *local search on children* in the numerical experiments of Section 4.

In LS4, the weight w_1 that determines the descent direction must be computed. A random selection as suggested in [27] gives poor results. Murata et al. [28] have designed a MOGA (not based on NSGA-II) for a pattern classification problem, in which a local search like LS4 is applied to all solutions of front 1, with a descent direction which depends on the location of solutions on this front, see Figure 2. The aim is to improve the front, with emphasis on the two extreme solutions, while preserving the spacing between solutions. To compute w_1 for a given solution S, these authors use Equation 2 in which f_c^{min} and f_c^{max} respectively denote the minimum and maximum values of criterion f_c, $c = 1, 2$.

$$w_1 = \left(\frac{f_1(S) - f_1^{min}}{f_1^{max} - f_1^{min}} \right) \Bigg/ \left(\frac{f_1(S) - f_1^{min}}{f_1^{max} - f_1^{min}} + \frac{f_2(S) - f_2^{min}}{f_2^{max} - f_2^{min}} \right) \tag{2}$$

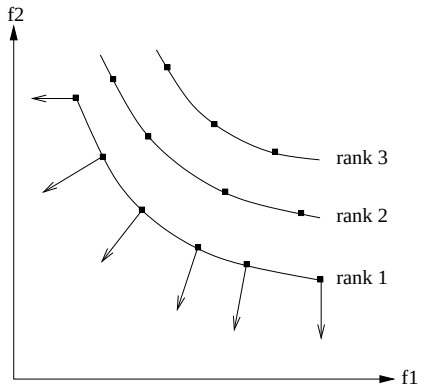


Figure 2: Variable descent directions in Murata's MOGA

For the instances tested in Section 4 for the bi-objective CARP, the first front contains on average less than 1/10 of the solutions and applying LS4 to the first front only is not agressive enough. However, by computing f_1^{min}, f_1^{max}, f_2^{min} and f_2^{max} on the whole population instead of the first front, LS4 can be applied to any solution and with a descent direction that depends on its position in *pop*. This option was finally selected for our MOGA.

Two possible locations can be used for LS4. As for LS1, LS2 and LS3, it can be called in *add_children* with a given probability p_{LS}, in lieu of mutation (*local search on children*). In that case, f_1^{min}, f_1^{max}, f_2^{min} and f_2^{max} are computed on the whole population, at the beginning of the procedure, and the descent direction for child S is computed using Equation 2.

The other location consists in applying periodically LS4 to a fraction of the fronts, just after the non-dominated sort in Algorithm 3. Since this local search phase may change the fronts, it must be followed by a second call to *non_dominated_sort*. To keep approximately an overall local search rate of 10%, LS4 is applied every k iterations of the MOGA to $k \times ns/10$

solutions of *pop*, for instance 10% of solutions every iteration or 20% every two iterations. In fact, better results were obtained by applying LS4 to all solutions every $k = 10$ iterations, probably because of a better oscillation between diversification and intensification. We call this way of including LS4 *periodic local search*.

4 Computational experiments

All developpements were done in Delphi 7 (Borland®) and tested on a Pentium IV laptop computer clocked at 1.8 GHz under Windows 2000. The following subsections describe the instances used, the MOGA versions selected, the evaluation criteria, the results, and two graphical examples.

4.1 Instances tested

The computational evaluation is based on three sets of standard CARP instances which are used by most authors working on the single-objective CARP [4–12]. The first set corresponds to the 23 *gdb* files from Golden, DeArmon and Baker [1], with 7 to 27 nodes and 11 to 55 edges, all required. The original set contains 25 files but two instances (*gdb*8 and *gdb*9) are inconsistent and never used. The second set gathers the 34 *val* files proposed by Belenguer and Benavent in [12], which have 24 to 50 nodes and 34 to 97 edges, all required. The last set provides the 24 *egl* instances generated by Belenguer and Benavent [12], by perturbing the demands in three rural networks supplied by Eglese. These *egl* files are bigger (77 to 140 nodes, 98 to 190 edges) and in some instances not all edges are required. All these files can be downloaded from the internet, at address `http://www.uv.es/~belengue/carp.html`.

4.2 Test protocol and parameters

A preliminary testing phase was required to fix the population size, the crossover (OX or LOX), the number of iterations and to evaluate the impact of good heuristic solutions in the initial population. This has lead to the following features which are shared by all MOGA versions in the sequel: a population of $ns = 60$ solutions with three good initial solutions, the OX crossover and 100 to 200 iterations corresponding to 6,000 to 12,000 crossovers. Path-Scanning [1], Augment-Merge [2] and Ulusoy's heuristic [3] were selected to provide good initial solutions. They can be discarded without changing the quality of final solutions, but in that case the MOGA converges more slowly, in 300 to 400 iterations.

A basic version of the MOGA without local search, called MO1, was selected as a reference algorithm. Eight other versions listed in Table 1 were prepared for comparison. They differ from MO1 by the number of iterations and the kind of local search.

4.3 Evaluation criteria

Comparing two different algorithms for solving the same problem is a central issue in multi-objective optimization, especially when the final choices of the decision maker are unknown. Several criteria were published, either to evaluate the quality of front 1 at the end of one given algorithm, or to provide a relative comparison between the final fronts computed by two algorithms.

Table 1: MOGA versions tested

Version	Iterations	LS type	LS location
MO1	100	None	Irrelevant
MO2	100	1	On children, rate 10%
MO3	100	2	On children, rate 10%
MO4	100	3	On children, rate 10%
MO5	200	3	On children, rate 10%
MO6	100	4	On children, rate 10%
MO7	200	4	On children, rate 10%
MO8	100	4	Periodic, every 10 iterations
MO9	200	4	Periodic, every 10 iterations

This study uses a simple measure proposed by Riise [29] to compare two fronts. Given a front F to be evaluated and a reference front R (defined by MO1 in our case), a kind of distance is measured between each solution of F and its projection onto the extrapolated reference front. This extrapolated front is a piecewise continuous line (in the bi-objective case) going through each solution and extended beyond the two extreme solutions, like in Figure 3. These distances are multiplied by -1 when the solution to compare is below the extrapolated front, like d_4 in the figure. The proposed measure $\mu(F, R)$ is the sum of the signed distances between the solutions of F and their respective projections on the reference front, e.g. $\mu = d_1 + d_2 + d_3 - d_4$ in the figure.

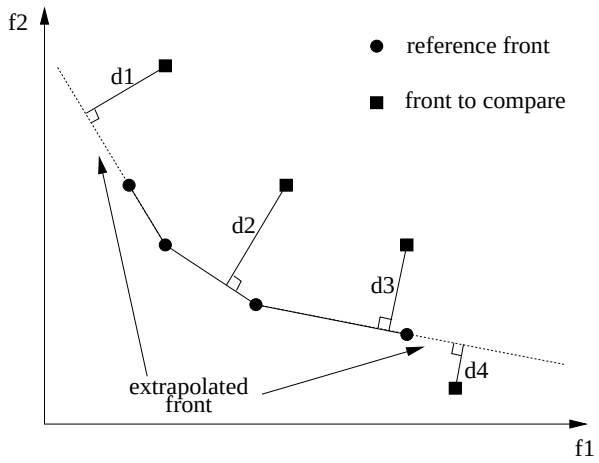


Figure 3: Examples of distances to the extrapolated front

Hence, for one given instance, a version of the MOGA with a final front F outperforms MO1 with its final front R if $\mu(F, R) < 0$. Since μ depends on the number of solutions in F, a normalized measure $\bar{\mu}(F, R) = \mu(F, R)/|F|$ will be also used. The measure μ is a bit arbitrary and of course subject to criticism. For example, how can we measure the difference between two fronts, when one is well spread while the other displays several gaps between clusters of solutions? Nevertheless, its merit is to exist and to help us rank the tested protocols. Because of no a-priori knowledge of the solutions which lie on the efficient front, such measures should be completed by a visual inspection.

Very recently, some authors coming from the combinatorial optimization community like

Jaszkiewicz [27] have claimed that the authors of multi-objective metaheuristics should prove to be competitive to single objective metaheuristics in terms of quality of solutions and computational efficiency. Since several metaheuristics are available for the single objective CARP [4–10], they can be used to evaluate the total cost of the leftmost solution of front 1, thus completing the measures μ and $\bar{\mu}$. Two metaheuristics were selected for that purpose in our computational evaluation: CARPET, the most efficient tabu search algorithm, designed by Hertz *et al.* [5], and the memetic algorithm (MA) proposed by Lacomme *et al.* [10].

Our tables of results compare also the leftmost and rightmost solutions with two lower bounds. The lower bound LB_1 for the total cost is the one proposed by Belenguer and Benavent [12]. Concerning makespan, a trivial lower bound LB_2 corresponds to the longest trip, when each trip performs one single task: $LB_2 = \max\{D_{si} + c_{ij} + D_{js} \mid [i,j] \in E\}$, where s is the depot node and D_{ij} the cost of a shortest path from node i to node j in G.

4.4 Results

4.4.1 Format of tables

For each set of benchmarks, the next subsections present one synthetic table with the results obtained by CARPET, the MA and each MOGA version, see Table 2 for instance. This table indicates for each algorithm the following *average* values: the number of solutions in the final set of non-dominated solutions F, the deviation $Dev\ LB_1$ of the leftmost solution to the lower bound on the total cost LB_1, the number of times $LB_1\ hits$ this bound is reached (number of proven optima), the deviation $Dev\ LB_2$ of the rightmost solution to the makespan lower bound LB_2, the number of proven optimal makespans $LB_2\ hits$, the measure μ for a reference front provided by MO1, the normalized measure $\bar{\mu}$ and the CPU time in seconds on a 1.8 GHz Pentium-IV PC. For CARPET and the MA, the only applicable columns are $Dev\ LB_1$, $LB_1\ hits$ and the running time. These fields are in boldface when a MOGA is at least as good as a single objective metaheuristic.

For each synthetic table, the results of the best MOGA version in terms of $\bar{\mu}$ are listed instance by instance in a detailed table given in the Appendix, see Table 5 for instance. The three first columns indicate the instance name, the number of nodes n and the number of edges m. The number of tasks τ is cited only for *egl* files, because it is always equal to the number of edges in the other benchmarks. These columns are followed by the total cost and the running time achieved by CARPET (listed in [5], except the *egl* files whose solution values are given in [12], but without running times) and by the memetic algorithm [10].

In order to have a fair comparison with the MOGAs, the solution values are given for the standard setting of parameters of these metaheuristics and the running times are scaled for the 1.8 GHz PC used in this study. The MA in [10] was executed on a 1 GHz Pentium-III PC and its running times are here multiplied by 0.555. CARPET was tested on a 195 MHz Silicon Graphics Indigo-2 workstation which is 5.43 times slower than the Pentium-III. Hence, the execution times provided by Hertz et al. are multiplied by $0.555/5.43 = 0.1022$ in our study. The best-known solution values (total cost) found by CARPET and the MA with various settings of parameters are given in the *BKS* column.

The eight remaining columns concern the MOGA: the number of efficient solutions $|F|$, the total costs and makespans for the leftmost and rightmost solutions (f_1^{left}, f_2^{left}, f_1^{right} and f_2^{right}), the running time in seconds and the measures μ and $\bar{\mu}$. Each detailed table ends with three lines that provide for each column its average value (deviation to a bound, running

time or measure), the worst value and the number of proven optima.

4.4.2 Results for *gdb* files

They are given in Table 2. All versions with a local search (MO2 to MO9) are much better than MO1. As expected, MO2 with its local search LS1 on the total cost yields a good leftmost point but a poor rightmost point, while the contrary holds for MO3 with its LS2 on makespan. MO4 with its local search LS3 (accepting a move if the result dominates the incumbent solution) is almost as good as CARPET but performing 200 iterations instead of 100 (MO5) brings no improvement. The versions which periodically apply the directional local search LS4 to all solutions (MO8 and 9) respectively outperform MO6 and MO7 in which LS4 is called on children. The best versions for 100 and 200 iterations are respectively MO8 and MO9, they have little effect on the two measures but improve the two extreme solutions. They outperform CARPET for the average deviation to LB_1 but do not find more optima and are a bit slower.

Table 2: Average results for the 23 *gdb* files

Method	$\|F\|$	Dev LB_1	LB_1 hits	Dev LB_2	LB_2 hits	μ	$\bar{\mu}$	Time (s)
CARPET	–	0.48	18	–	–	–	–	5.01
MA	–	0.15	21	–	–	–	–	2.94
MO1	3.87	3.81	5	29.64	9	0.00	0.00	1.94
MO2	3.52	0.47	17	30.43	7	-18.15	-4.61	6.01
MO3	3.78	3.40	5	24.97	11	-5.56	-1.74	4.56
MO4	3.26	0.56	16	22.07	11	-22.82	-6.48	6.53
MO5	3.30	0.56	16	20.28	11	-23.64	-6.45	11.53
MO6	3.39	0.56	16	22.00	10	-25.14	-6.39	6.46
MO7	3.43	**0.48**	16	21.33	11	-24.07	-6.35	12.94
MO8	3.39	**0.39**	**18**	21.18	12	-23.94	-6.45	7.17
MO9	3.43	**0.36**	**18**	20.95	12	-23.49	-6.53	13.79

The results of the best version MO9 are detailed in Table 5, in the appendix. They show that MO9 is very robust, since its worst deviation to LB_1 is 2.23% vs 4.62% for CARPET. No total cost obtained by MO9 is improved by the other versions. In particular, the two instances *gdb*10 and *gdb*14 remain open.

4.4.3 Results for *val* files

The synthetic results of Table 3 show the same hierarchy as for *gdb* files: the MOGAs with LS4 outperform the MOGAs with LS3, which in turn outperform the versions with single objective local searches and the version without local search. But, this time, LS4 gives better results when it is called on children (MO6-MO7). The convergence seems slower, since noticeable improvements can be found by allowing 200 iterations. The average running time strongly increases but is still reasonable (1.3 minutes maximum). It is mainly consumed by the calls to local searches. All MOGAs with LS3 or LS4 (MO4 to MO9) outperform CARPET for the average deviation to LB_1 (MO4 is even a bit faster) but 200 iterations are required to find more optima.

Roughly speaking, the best version MO7 halves the average deviation to LB_1 obtained by CARPET, at the expense of a double running time. Its results are detailed for each instance in Table 6, in the appendix. In addition to the 18 optimal total costs listed in this table, the

Table 3: Average results for the 34 *val* files

Method	$\|F\|$	Dev LB_1	LB_1 hits	Dev LB_2	LB_2 hits	μ	$\bar{\mu}$	Time (s)
CARPET	–	1.90	15	–	–	–	–	35.48
MA	–	0.61	22	–	–	–	–	21.31
MO1	5.91	8.52	1	37.73	5	0.00	0.00	3.21
MO2	4.76	**1.23**	**17**	32.06	5	-85.43	-16.18	27.80
MO3	6.06	7.99	3	30.90	7	-7.33	-2.31	11.74
MO4	4.88	**1.36**	**15**	21.65	8	-109.05	-20.87	32.50
MO5	5.29	**1.01**	**19**	20.59	9	-125.98	-21.37	65.20
MO6	5.00	**1.16**	**15**	21.50	10	-113.07	-21.30	38.20
MO7	5.24	**0.99**	**18**	19.62	10	-121.70	-21.85	76.17
MO8	5.09	**1.33**	**16**	20.19	10	-114.11	-20.61	44.53
MO9	5.32	**1.13**	**17**	19.43	10	-124.03	-21.51	83.06

other versions have found optima for the following instances: *val*2*b* (259), *val*2*c* (457) and *val*10*c* (446). Moreover, they improve MO7 for *val*4*d* (536 vs 539), *val*5*d* (593 vs 595) and *val*9*d* (397 vs 399). Unfortunately, no open instance is broken and best-known solutions are not improved.

4.4.4 Results for *egl* files

The lower bound LB_1 proposed by Belenguer and Benavent is never tight on the *egl* files. This is why the number of optimal total costs is replaced in Table 4 by the number of best-known solutions retrieved or (in brackets) improved (column *BKS*). All MOGAs except MO1 and MO3 outperform CARPET in terms of average deviation to the lower bound on the total cost and retrieve some best-known solutions (CARPET: 0). MO9 is even able to improve two best-known solutions, namely *egl-e3-c* (10349 vs 10369) and *egl-s3-b* (14004 vs 14028). Contrary to *gdb* and *val* files, the MOGAs reach the optimal makespan for a majority of instances and become faster than the memetic algorithm. In particular, the last MOGA finds all optimal makespans except one. Average running times are still acceptable: less than 5 minutes. For the detailed results in appendix (Table 7), we preferred to select MO9 with the lowest deviation to LB_1 and its two improved instances, rather than MO5 which has a slightly better measure $\bar{\mu}$.

Table 4: Average results for the 24 *egl* files

Method	$\|F\|$	Dev LB_1	BKS	Dev LB_2	LB_2 hits	μ	$\bar{\mu}$	Time (s)
CARPET	–	4.74	0	–	–	–	–	Unknown
MA	–	2.47	19	–	–	–	–	292.77
MOGA1	5.83	13.24	0(0)	1.87	15	0.00	0.00	2.77
MOGA2	5.29	**4.24**	**1(0)**	0.94	16	-1234.01	-256.65	158.61
MOGA3	6.67	13.32	0(0)	0.17	20	-63.93	-15.98	35.80
MOGA4	5.21	**4.47**	0(0)	0.27	19	-1333.52	-335.11	148.50
MOGA5	6.00	**3.84**	**1(0)**	0.10	21	-1586.62	-360.97	263.49
MOGA6	5.33	**4.47**	**1(0)**	0.08	22	-1435.73	-324.27	153.48
MOGA7	5.83	**4.00**	**1(0)**	0.07	22	-1573.80	-302.97	290.55
MOGA8	5.96	**4.11**	**1(0)**	0.06	22	-1592.01	-344.66	159.01
MOGA9	5.79	**3.69**	**2(2)**	0.05	23	-1531.94	-350.76	268.99

BKS gives the number of best-known solutions retrieved or (in brackets) improved by the MOGA.

4.5 Graphical examples

Graphical representations are well suited to visualize the behavior of bi-objective optimization methods. The kind of progression performed by our MOGAs is illustrated in Figure 4. The graphic represents in the objective space the random initial solutions (symbol +), the three good heuristic solutions (symbol ×) and the final population (symbol ∗) computed by MO9 on the *egl-e3-a* instance (77 nodes, 98 edges and 87 tasks). Remember that MO9 is the version which performs 200 iterations and applies LS4 to all solutions, every 10 iterations.

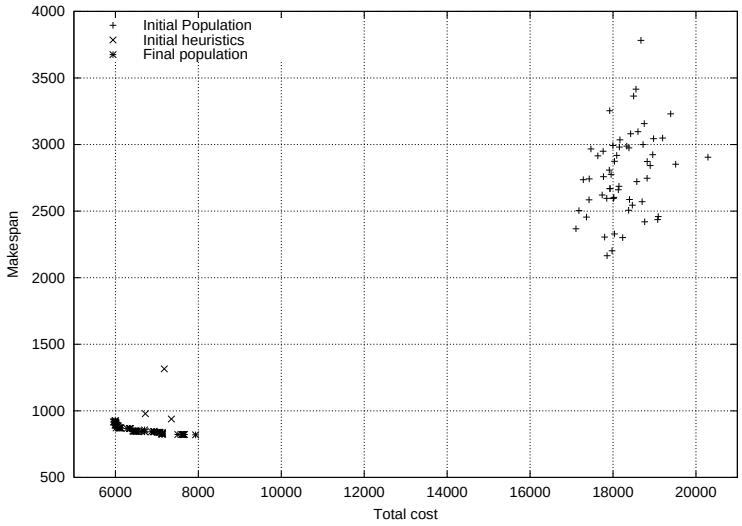


Figure 4: Convergence of MO9 on instance *egl-e3-a*

One can notice that the initial heuristic solutions are already quite close to the final population. As already mentioned, adding such solutions is very useful to accelerate the search. Figure 5 provides a magnified representation of the final population. A dashed line is used to emphasize efficient solutions.

The impact of the other key-component, the local search procedure, can be illustrated on the same instance by comparing the efficient set computed by the basic version MO1 (no local search, 100 iterations) with the one obtained by the best version MO9 (periodic local search LS4, 200 iterations). See Figure 5. Even if the total number of iterations is increased for MO1, the results are only slightly improved. As for single objective GAs, hybridization with local search is necessary to obtain efficient MOGAs. Here for instance, MO9 decreases the best total cost by 10.55% (from 6659 to 5956) and the best makespan by 10.97% (from 921 to 820). Moreover, more efficient solutions are obtained and they are better spread, thus providing the decision maker with a wider choice.

5 Concluding remarks

In industry, most decision makers wish to take several criteria into account and the bi-objective CARP addressed in this article is a typical example. The combinatorial optimization community can help research in multi-objective optimization, by bringing a rich toolbox, very efficient algorithms, and a stricter methodology to evaluate and compare different algorithms.

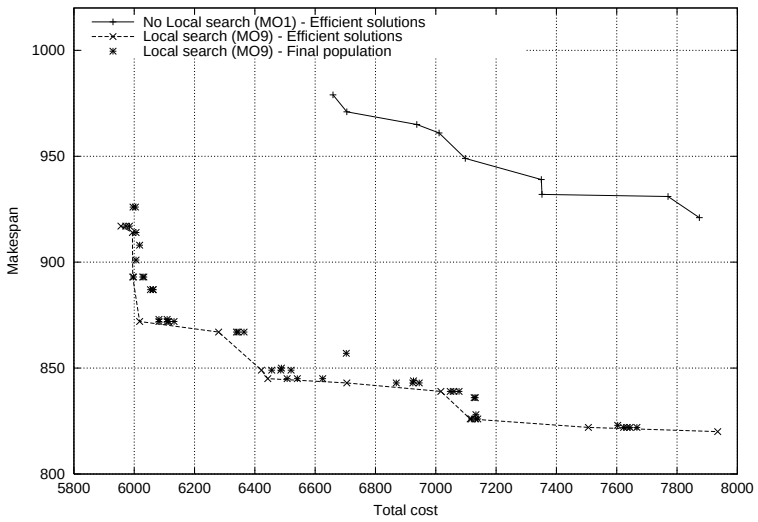


Figure 5: Impact of local search for instance *egl-e3-a*

In this paper, the implementation of the NSGA-II multi-objective genetic algorithm alone could not efficiently solve the bi-objective CARP. Two key-components were required to enhance performance: using good heuristics (Path-Scanning, Augment-Merge and Ulusoy's heuristic) in the initial population and adding a local search able to improve solutions for both criteria. It is worth noticing that the chromosome encoding, the crossover operator and the two proposed improvements come from a memetic algorithm successfully developped for the single objective CARP. The main difficulty was to select a local search adapted to the multi-objective case and to find its best location in the MOGAs.

An intensive testing on three sets of benchmarks, with a strict comparison completing a distance measure with lower bounds and metaheuristics for the single objective case, proves the validity and the efficiency of the proposed approach. In particular, the best versions of our MOGAs are still competitive with single objective metaheuristics. We even believe that any GA or memetic algorithm for a single objective problem could be generalized in the same way to handle several objectives, while keeping its efficiency, although this should be confirmed with other combinatorial optimization problems.

References

[1] B.L. Golden, J.S. DeArmon, and E.K. Baker. Computational experiments with algorithms for a class of routing problems. *Computers and Operations Research*, 10(1):47–59, 1983.

[2] B.L. Golden and R.T. Wong. Capacitated arc routing problems. *Networks*, 11:305–315, 1981.

[3] G. Ulusoy. The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research*, 22:329–337, 1985.

[4] J.M. Belenguer, E. Benavent, and F. Cognata. Un metaheuristico para el problema de rutas por arcos con capacidades. In *Proceedings of the* 23^{th} *national SEIO meeting*, Valencia, Spain, 1997.

[5] A. Hertz, G. Laporte, and M. Mittaz. A tabu search heuristic for the Capacitated Arc Routing Problem. *Operations Research*, 48(1):129–135, 2000.

[6] P. Greistorfer. A tabu-scatter search metaheuristic for the arc routing problem. *Computers and Industrial Engineering*, 44(2):249–266, 2003.

[7] A. Hertz and M. Mittaz. A variable neighborhood descent algorithm for the undirected Capacitated Arc Routing Problem. *Transportation Science*, 35(4):425–434, 2001.

[8] P. Beullens, L. Muyldermans, D. Cattrysse, and D. Van Oudheusden. A guided local search heuristic for the Capacitated Arc Routing Problem. *European Journal Of Operational Research*, 147(3):629–643, 2003.

[9] P. Lacomme, C. Prins, and W. Ramdane-Chérif. A genetic algorithm for the Capacitated Arc Routing Problem and its extensions. In E.J.W. Boers et al., editor, *Applications of evolutionnary computing*, volume 2037 of *Lecture Notes in Computer Science*, pages 473–483. Springer, 2001.

[10] P. Lacomme, C. Prins, and W. Ramdane-Chérif. Competitive memetic algorithms for arc routing problems. Technical Report LOSI-2001-01, Laboratory for Industrial Systems Optimization, Université de Technologie de Troyes, France, 2001. To appear in *Annals of Operations Research*.

[11] A. Amberg and S. Voß. A hierarchical relaxations lower bound for the Capacitated Arc Routing Problem. In R.H. Sprague (Hrsg.), editor, *Proceedings of the* 35^{th} *Annual Hawaii International Conference on Systems Sciences*, pages DTIST02:1–10, Piscataway, 2002. IEEE.

[12] J.M. Belenguer and E. Benavent. A cutting plane algorithm for the Capacitated Arc Routing Problem. *Computers and Operations Research*, 30(5):705–728, 2003.

[13] P. Lacomme, C. Prins, and M. Sevaux. Multi-objective Capacitated Arc Routing Problem. In C.M. Fonseca et al., editor, *Evolutionary multi-criterion optimization (Proceedings of EMO 2003, Faro, Portugal)*, volume 2632 of *Lecture Notes in Computer Science*, pages 550–564. Springer, 2003.

[14] Y.B. Park and C.P. Koelling. An interactive computerized algorithm for multicriteria vehicle routing problems. *Computers and Industrial Engineering*, 16:477–490, 1989.

[15] A. Corberan, E. Fernandez, M. Laguna, and R. Martí. Heuristic solutions to the problem of routing school buses with multiple objectives. *Journal of the Operational Research Society*, 53(4):427–435, 2002.

[16] S.C. Hong and Y.B. Park. A heuristic for bi-objective vehicle routing problem with time windows constraints. *International Journal of Production Economics*, 62:249–258, 1999.

[17] W. Sessomboon, K. Watanabe, T. Irohara, and K. Yoshimoto. A study on multi-objective vehicle routing problem considering customer satisfaction with due-time (the creation of pareto optimal solutions by hybrid genetic algorithm). *Transaction of the Japan Society of Mechanical Engineers*, 1998.

[18] M. Ehrgott and X. Gandibleux. *Multiobjective Combinatorial Optimization*, M. Ehrgott and X. Gandibleux (eds) , volume 52 of *International Series in Operations Research and Management Science*, pages 369–444. Kluwer, 2002.

[19] C.A. Coello Coello, D.A. Van Veldhuizen, and G.B. Lamont. *Evolutionary algorithms for solving multi-objective problems*. Kluwer, New York, 2002.

[20] K. Deb. *Multi-objective optimization using evolutionary algorithms*. Wiley, Chichester, UK, 2001.

[21] C.A. Coello Coello. An updated survey of GA-based multiobjective optimization techniques. *ACM Computing Surveys*, 32(2):109–143, 2000.

[22] K. Deb. Multi-objective genetic algorithms: problem difficulties and construction of test problems. *Evolutionary Computation*, 7(3):205–230, 1999.

[23] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.

[24] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[25] P. Moscato. *New ideas in optimization* D. Corne, M. Dorigo and F. Glover (eds), chapter Memetic algorithms: a short introduction, pages 219–234. McGraw-Hill, Maidenhead, UK, 1999.

[26] A. Jaszkiewicz. Genetic local search for multiple objective combinatorial optimization. *European Journal of Operational Research*, 137(1):50–71, 2001.

[27] A. Jaszkiewicz. Do multiple objective metaheuristics deliver on their promises? a computational experiment on the set covering problem. *IEEE Transactions on Evolutionary Computation*, 7(2):133–143, 2003.

[28] T. Murata, H. Nozawa, H. Ishibuchi, and M. Gen. Modification of local search directions for non-dominated solutions in cellular multiobjective genetic algorithms for pattern classification problems. In C.M. Fonseca et al., editor, *Evolutionary multi-criterion optimization (Proceedings of EMO 2003, Faro, Portugal)*, volume 2632 of *Lecture Notes in Computer Science*, pages 593–607. Springer, 2003.

[29] A. Riise. Comparing genetic algorithms and tabu search for multi-objective optimization. In *Abstract conference proceedings*, page 29, Edinburgh, UK, July 2002. IFORS.

Table 5: Detailed results of MO9 for *gdb* files

File	n	m	LB_1	LB_2	CARPET	Time	MA	Time	BKS	$\|F\|$	f_1^{left}	f_2^{left}	f_1^{right}	f_2^{right}	Time	μ	$\bar{\mu}$
gdb1	12	22	316	63	316*	1.75	316*	0.00	316*	3	316*	74	337	63*	8.36	-30.08	-10.03
gdb2	12	26	339	59	339*	2.87	339*	0.24	339*	3	339*	69	395	59*	10.27	-18.86	-6.29
gdb3	12	22	275	59	275*	0.04	275*	0.03	275*	4	275*	65	339	59*	8.57	-31.44	-7.86
gdb4	11	19	287	64	287*	0.05	287*	0.00	287*	3	287*	74	350	64*	7.07	-11.79	-3.93
gdb5	13	26	377	64	377*	3.10	377*	0.06	377*	6	377*	78	447	64*	10.46	-12.78	-2.13
gdb6	12	22	298	64	298*	0.47	298*	0.09	298*	4	298*	75	351	64*	8.16	-40.47	-10.12
gdb7	12	22	325	57	325*	0.00	325*	0.03	325*	3	325*	68	381	61	8.55	-14.02	-4.67
gdb10	27	46	344	38	352	33.85	350	22.10	348	4	350	44	390	38*	22.45	-19.35	-4.84
gdb11	27	51	303	37	317	29.92	303*	3.93	303*	3	309	43	333	37*	27.17	-31.77	-10.59
gdb12	12	25	275	39	275*	0.86	275*	0.03	275*	4	275*	71	297	54	10.24	-33.43	-8.36
gdb13	22	45	395	43	395*	1.27	395*	0.70	395*	5	395*	81	421	64	33.68	-105.14	-21.03
gdb14	13	23	448	93	458	11.45	458	5.43	458	4	458	97	547	93*	7.26	-64.54	-16.13
gdb15	10	28	536	128	544	1.34	536*	4.12	536*	1	544	128*	544	128*	9.03	-10.22	-10.22
gdb16	7	21	100	15	100*	0.27	100*	0.03	100*	3	100*	21	112	17	7.66	-1.36	-0.45
gdb17	7	21	58	8	58*	0.00	58*	0.00	58*	2	58*	15	60	13	9.66	-4.36	-2.18
gdb18	8	28	127	14	127*	0.94	127*	0.03	127*	4	127*	27	135	19	10.75	-13.78	-3.44
gdb19	8	28	91	9	91*	0.00	91*	0.03	91*	1	91*	15	91*	15	12.81	-2.77	-2.77
gdb20	9	36	164	19	164*	0.16	164*	0.06	164*	3	164*	33	178	27	18.05	-19.44	-6.48
gdb21	8	11	55	17	55*	0.11	55*	0.00	55*	2	55*	21	63	17*	4.80	-3.47	-1.73
gdb22	11	22	121	20	121*	5.27	121*	0.18	121*	5	121*	36	131	20*	8.10	-16.61	-3.32
gdb23	11	33	156	15	156*	0.63	156*	0.09	156*	4	156*	30	160	22	14.57	-10.69	-2.67
gdb24	11	44	200	12	200*	1.88	200*	1.86	200*	4	200*	26	207	20	24.03	-21.02	-5.25
gdb25	11	55	233	13	235	19.08	233*	28.41	233*	4	235	23	241	20	35.45	-22.96	-5.74
Average					0.48%	5.01	0.15%	2.94	0.13%	3.43	0.36%	47.09%	10.70%	20.95%	13.79	-23.49	-6.53
Worst					4.62%	33.85	1.78%	28.41	1.78%	6	2.23%	116.67%	23.27%	66.67%	35.45	-1.36	-0.45
Optima					18		21		21		18	1	1	12			

BKS is the best solution value (total cost) found by CARPET and the MA using several settings of parameters.
CPU times in seconds on a 1.8 GHz Pentium-IV PC. Times for CARPET and MA have been scaled. See section 4.4.1 for details

Table 6: Detailed results of MO7 for *val* files

File	n	m	LB_1	LB_2	CARPET	Time	MA	Time	BKS	$\|F\|$	f_1^{left}	f_2^{left}	f_1^{right}	f_2^{right}	Time	μ	$\bar{\mu}$
val1a	24	39	173	40	173*	0.02	173*	0.00	173*	1	173*	58	173*	58	20.82	-21.08	-21.08
val1b	24	39	173	40	173*	9.26	173*	8.02	173*	6	173*	61	204	42	21.39	-46.67	-7.78
val1c	24	39	235	40	245	93.20	245	28.67	245	2	245	41	248	40*	14.25	-60.72	-30.36
val2a	24	34	227	71	227*	0.17	227*	0.05	227*	6	227*	114	270	90	15.92	-39.62	-6.60
val2b	24	34	259	71	260	13.02	259*	0.22	259*	5	260	101	306	78	15.83	-26.45	-5.29
val2c	24	34	455	71	494	31.66	457	21.76	457	1	463	71*	463	71*	9.49	-7.69	-7.69
val3a	24	35	81	27	81*	0.77	81*	0.05	81*	4	81*	41	88	31	16.40	-40.05	-10.01
val3b	24	35	87	27	87*	2.79	87*	0.00	87*	4	87*	32	105	27*	14.88	-23.83	-5.96
val3c	24	35	137	27	138	41.66	138	28.23	138	1	138	27*	138	27*	10.39	-14.00	-14.00
val4a	41	69	400	80	400*	28.32	400*	0.72	400*	4	400*	134	446	92	89.86	-187.06	-46.77
val4b	41	69	412	80	416	75.66	412*	1.21	412*	9	412*	105	468	83	79.25	-402.04	-44.67
val4c	41	69	428	80	453	70.06	428*	19.11	428*	10	430	99	482	80*	73.68	-474.05	-47.40
val4d	41	69	520	80	556	233.56	541	103.26	530	1	539	80*	539	80*	50.42	-24.43	-24.43
val5a	34	65	423	72	423*	3.80	423*	1.86	423*	4	423*	141	474	96	72.75	-143.50	-35.88
val5b	34	65	446	72	448	41.40	446*	1.04	446*	8	446*	112	506	86	73.83	-151.41	-18.93
val5c	34	65	469	72	476	53.27	474	101.01	474	9	474	96	541	80	67.82	-189.67	-21.07
val5d	34	65	571	72	607	224.11	581	90.74	581	4	595	81	686	72*	51.83	-44.57	-11.14
val6a	31	50	223	45	223*	3.89	223*	0.17	223*	5	223*	75	259	56	37.08	-33.93	-6.79
val6b	31	50	231	45	241	26.94	233	67.34	233	7	233	68	263	50	34.93	-50.39	-7.20
val6c	31	50	311	45	329	85.18	317	52.23	317	5	317	55	329	45*	21.88	6.48	1.30
val7a	40	66	279	39	279*	6.59	279*	1.97	279*	3	279*	85	289	59	72.00	-44.27	-14.76
val7b	40	66	283	39	283*	0.02	283*	0.44	283*	4	283*	58	299	51	79.89	-81.59	-20.40
val7c	40	66	333	39	343	121.44	334	101.17	334	5	335	50	352	40	61.00	-9.03	-1.81
val8a	30	63	386	67	386*	3.84	386*	0.66	386*	7	386*	129	429	87	70.02	-172.40	-24.63
val8b	30	63	395	67	401	81.46	395*	9.95	395*	9	395*	100	455	79	66.16	-227.49	-25.28
val8c	30	63	517	67	533	147.40	527	71.46	527	7	545	74	610	67*	43.29	-180.52	-25.79
val9a	50	92	323	44	323*	28.51	323*	18.29	323*	3	326	82	333	68	171.97	-109.49	-36.50
val9b	50	92	326	44	329	59.89	326*	29.39	326*	3	326*	82	340	58	170.31	-108.22	-36.07
val9c	50	92	332	44	332*	56.44	332*	71.19	332*	12	332*	69	389	51	175.60	-330.71	-27.56
val9d	50	92	382	44	409	353.28	391	211.13	391	6	399	50	434	44*	134.13	-106.55	-17.76
val10a	50	97	428	47	428*	5.52	428*	25.48	428*	5	428*	143	449	91	203.20	-203.37	-40.67
val10b	50	97	436	47	436*	18.43	436*	4.67	436*	5	436*	111	459	77	194.28	-175.23	-35.05
val10c	50	97	446	47	451	93.47	446*	17.30	446*	7	448	93	498	66	205.89	-169.32	-24.19
val10d	50	97	524	47	544	156.31	530	215.04	528	6	537	61	595	54	149.31	-245.01	-40.83
Average					1.90%	63.87	0.61%	38.35	0.54%	5.24	0.99%	52.68%	10.69%	19.62%	76.17	-121.70	-21.85
Worst					8.57%	353.28	4.26%	215.04	4.26%	12	5.42%	204.26%	20.69%	93.62%	205.89	6.48	1.30
Optima					15		22		22		18	3	1	10			

BKS is the best solution value (total cost) found by CARPET and the MA using several settings of parameters.
CPU times in seconds on a 1.8 GHz Pentium-IV PC. Times for CARPET and MA have been scaled. See section 4.4.1 for details.

Table 7: Detailed results of MO9 for *egl* files

File	n	m	τ	LB_1	LB_2	CARPET	MA	Time	BKS	$\|F\|$	f_1^{left}	f_2^{left}	f_1^{right}	f_2^{right}	Time	μ	$\bar{\mu}$
egl-e1-a	77	98	51	3515	820	3625	3548	74.26	3548	4	3548	943	3824	820*	31.39	-1431.21	-357.80
egl-e1-b	77	98	51	4436	820	4532	4498	69.48	4498	3	4525	839	4573	820*	25.70	-753.88	-251.29
egl-e1-c	77	98	51	5453	820	5663	5595	71.18	5595	2	5687	836	5764	820*	23.33	-177.03	-88.51
egl-e2-a	77	98	72	4994	820	5233	5018	152.58	5018	5	5018	953	6072	820*	77.93	-1269.08	-253.82
egl-e2-b	77	98	72	6249	820	6422	6340	153.41	6340	6	6411	864	6810	820*	65.47	-1581.27	-263.54
egl-e2-c	77	98	72	8114	820	8603	8415	129.63	8395	4	8440	854	8651	820*	56.02	-230.42	-57.61
egl-e3-a	77	98	87	5869	820	5907	5898	242.00	5898	12	5956	917	7935	820*	149.30	-1860.85	-155.07
egl-e3-b	77	98	87	7646	820	7921	7822	255.35	7816	7	7911	872	8455	820*	104.96	-1592.97	-227.57
egl-e3-c	77	98	87	10019	820	10805	10433	206.35	10369	4	10349	864	10511	820*	95.80	-827.25	-206.81
egl-e4-a	77	98	98	6372	820	6489	6461	291.87	6461	11	6548	890	7362	820*	166.66	-2003.50	-182.14
egl-e4-b	77	98	98	8809	820	9216	9021	312.85	9021	5	9116	874	9584	820*	145.44	-389.35	-77.87
egl-e4-c	77	98	98	11276	820	11824	11779	252.38	11779	1	11802	820*	11802	820*	111.18	-1041.39	-1041.39
egl-s1-a	140	190	75	4992	912	5149	5018	208.61	5018	11	5102	1023	6582	924	91.23	-1895.27	-172.30
egl-s1-b	140	190	75	6201	912	6641	6435	208.77	6435	7	6500	984	8117	912*	83.35	-1486.99	-212.43
egl-s1-c	140	190	75	8310	912	8687	8518	165.55	8518	5	8694	946	9205	912*	71.72	-1413.13	-282.63
egl-s2-a	140	190	147	9780	979	10373	9995	874.36	9995	13	10207	1058	12222	979*	497.49	-1233.84	-94.91
egl-s2-b	140	190	147	12886	979	13495	13174	760.50	13174	10	13548	1058	14334	979*	516.85	-496.91	-49.69
egl-s2-c	140	190	147	16221	979	17121	16795	746.93	16715	4	16932	1040	16975	979*	399.14	-4042.88	-1010.72
egl-s3-a	140	190	159	10025	979	10541	10296	1070.50	10296	10	10456	1099	12605	979*	699.12	-3899.11	-389.91
egl-s3-b	140	190	159	13554	979	14291	14053	1064.01	14028	6	14004	1040	15103	979*	609.54	-6439.40	-1073.23
egl-s3-c	140	190	159	16969	979	17789	17297	874.30	17297	4	17825	998	18043	979*	493.60	-748.77	-187.19
egl-s4-a	140	190	190	12027	1027	13036	12442	1537.59	12442	3	12730	1040	12912	1027*	838.24	-255.12	-85.04
egl-s4-b	140	190	190	15933	1027	16924	16531	1430.26	16531	1	16792	1027*	16792	1027*	720.31	-271.94	-271.94
egl-s4-c	140	190	190	20179	1027	21486	20832	1495.02	20832	1	21309	1027*	21309	1027*	381.86	-1425.00	-1425.00
Average						4.74%	2.47%	526.99	2.40%	5.79	3.69%	6.31%	12.94%	0.05%	268.99	-1531.94	-350.77
Worst						8.61%	4.46%	1537.59	4.46%	13	5.85%	16.22%	35.20%	1.32%	838.24	-177.03	-49.69
Nb BKS retrieved						0	19		24		2						
Nb BKS improved											2						
Nb Optima												3		23			

BKS is the best solution value (total cost) found by CARPET and the MA using several settings of parameters.
CPU times in seconds on a 1.8 GHz Pentium-IV PC. Times for CARPET and MA have been scaled.

Using Lagrangean Relaxation to Minimize the Weighted Number of Late Jobs on a Single Machine

Stéphane Dauzère-Pérès,[1,*] Marc Sevaux[2,†]

[1] *IRCCyN/Ecole des Mines de Nantes, La Chantrerie, BP 20722, F-44307 Nantes Cedex 03, France*

[2] *Université de Valenciennes, LAMIH/SP, Le Mont-Houy, F-59313 Valenciennes Cedex, France*

Received 30 April 2001; revised 21 February 2002; accepted 15 July 2002

DOI 10.1002/nav.10056

Abstract: This paper tackles the general single machine scheduling problem, where jobs have different release and due dates and the objective is to minimize the weighted number of late jobs. The notion of *master sequence* is first introduced, i.e., a sequence that contains at least an optimal sequence of jobs on time. This *master sequence* is used to derive an original mixed-integer linear programming formulation. By relaxing some constraints, a Lagrangean relaxation algorithm is designed which gives both lower and upper bounds. The special case where jobs have equal weights is analyzed. Computational results are presented and, although the duality gap becomes larger with the number of jobs, it is possible to solve problems of more than 100 jobs.
 Naval Research Logistics 50: 273–288, 2003.

1. INTRODUCTION

A set of n jobs $\{J_1, \ldots, J_n\}$, subject to *release dates* r_i and *due dates* d_i, have to be scheduled on a single machine. The *processing time* of jobs on the machine is denoted by p_i, and a weight w_i is associated with each job. The machine can only process one job at a time. A scheduled job completed before its due date is said to be *early* or *on time,* and *late* otherwise. Let U_i be equal to 1 if job J_i is late in a schedule, and to 0 otherwise. The objective is to minimize the weighted number of late jobs, or equivalently to maximize the weighted number of early jobs. A well-known and important remark is that there is always an optimal schedule in which late jobs are sequenced after all the early jobs.

This single-machine scheduling problem, noted $1|r_j|\Sigma\ w_jU_j$ in the standard classification, is strongly $\mathcal{NP}$-hard [8]. When all weights are equal ($1|r_j|\Sigma\ U_j$), the problem remains $\mathcal{NP}$-Hard, but becomes polynomially solvable if all release dates are equal ($1\|\Sigma\ U_j$) [9] ($O(n \log n)$), or

* This paper was written while this author was invited professor at the Department of Finance and Management Science, Norwegian School of Economics and Business Administration, Helleveien 30, N-5035 Bergen-Sandviken, Norway.

† This research was performed while this author was at Ecole des Mines de Nantes.

Correspondence to: S. Dauzère-Pérès

if release and due dates are similarly ordered ($r_i < r_j \Rightarrow d_i \leq d_j \; \forall(J_i, J_j)$) [6] ($O(n^2)$), [7] ($O(n \log n)$). However, some exact approaches have recently been proposed for this problem [1, 5]. Lawler [7] showed that the Moore's algorithm [9] could be applied when processing times and weights are agreeable, i.e., $p_i < p_j \Rightarrow w_i \geq w_j \; \forall(J_i, J_j)$. Finally, branch-and-bound procedures have been developed to solve the case where all release dates are equal ($1\|\sum w_j U_j$) in [12] and [11]. To our knowledge, no algorithm has been proposed to solve the general problem $1|r_j| \sum w_j U_j$.

In this paper, based on the notion of *master sequence,* i.e., a sequence from which an optimal sequence can be extracted, a new mixed-integer linear programming formulation is introduced. Using this formulation, a Lagrangean relaxation algorithm is derived. Lagrangean relaxation is a powerful optimization tool from which heuristic iterative algorithms can be designed, where both upper and lower bounds are determined at every iteration. It is thus possible to always know the maximum gap between the best solution found and the optimal solution, and stop the algorithm when this gap is small enough. One condition that is often associated with the efficiency of Lagrangean relaxation approaches is to relax as few constraints as possible, in order to obtain good bounds when solving the relaxed problem. This is why our formulation compares very favorably to other known ones (see [4] for a study of classical formulations for this problem). Only one constraint type, coupling variables of different jobs, needs to be relaxed to obtain an easily solvable problem, which can be solved independently for each job. This is not the case for other formulations, including discrete-time formulations, which often have many coupling constraints.

The master sequence is introduced in Section 2, and the resulting mixed-integer linear programming formulation is given and discussed in Section 3. Section 4 shows how the size of the master sequence, and thus the size of the model, can be reduced. Section 5 presents the Lagrangean relaxation algorithm, and Section 6 several improvements of the algorithm. The nonweighted case is analyzed in more details in Section 7. Numerical results on a large set of test instances are given and discussed in Section 8. Finally, some conclusions and perspectives are drawn in Section 9.

2. THE MASTER SEQUENCE

In the remainder of this paper, because we are only interested in sequencing jobs on time (late jobs can be set after the jobs on time), the sequence of jobs will often mean the sequence of *early* jobs. Many results in this paper are based on the following theorem.

THEOREM 1: There is always an optimal sequence of jobs on time that solves the problem $1|r_j| \sum w_j U_j$, in which every job J_j is sequenced just after a job J_i such that either condition (1) $d_i < d_j$, or (2) $d_i \geq d_j$ and $r_k \leq r_j \; \forall J_k$ sequenced before J_j, holds.

PROOF: First, note that conditions (1) or (2) being satisfied is equivalent to condition (3), $d_i \geq d_j$ and $\exists J_k$ sequenced before J_j such that $r_k > r_j$, not being satisfied.

The proof goes by showing that, by construction, it is possible to change any optimal sequence into an optimal sequence that satisfies conditions (1) or (2).

Suppose that we have a sequence in which some (or all) ready jobs do not satisfy one of the conditions. Starting from the beginning of the sequence, find the first pair of jobs (J_i, J_j) in the sequence that does not satisfy the two conditions, i.e., for which condition (3) holds. If t_i and t_j denote the start times of the two jobs, the latter condition ensures that, after interchanging the

two jobs, J_j can start at t_i (since $\exists J_k$ sequenced before J_j such that $r_j < r_k \leq t_i$). Hence, J_i will end at the same time as J_j before the interchange ($t_i + p_i + p_j$), and thus will still be on time (since $t_i + p_i + p_j \leq d_j \leq d_i$).

The interchange must be repeated if J_j and the new job just before it do not satisfy conditions (1) or (2), until one of these conditions is satisfied for J_j and the job just before it, or J_j is sequenced first.

The procedure is repeated for all jobs until the conditions are satisfied for all jobs. Because once a job has been moved, it will never go back again, one knows that the procedure will not be repeated more than n times, i.e., takes a finite amount of time. □

We will denote by $\mathcal{S}$ the subset of sequences in which jobs satisfy the conditions in Theorem 1. In the sequel, we will only be interested in sequences in $\mathcal{S}$, since we know that it always contains an optimal sequence.

THEOREM 2: If, in a sequence of $\mathcal{S}$, job J_j is after jobs J_i such that $r_j < r_i$, then there is at least one job J_i such that $d_i < d_j$.

PROOF: By contradiction, if all jobs J_k before J_j such that $r_k < r_i$ verify $d_k \geq d_j$, then neither of the conditions (1) and (2) is satisfied. Thus, the sequence is not in $\mathcal{S}$. □

COROLLARY 1: Consider a job J_j. If, for every job J_i such that $r_j < r_i$, condition $d_j \leq d_i$ holds, then, in every sequence of $\mathcal{S}$ (i.e., in an optimal sequence), job J_j is sequenced before all jobs J_i.

COROLLARY 2: Consider a job J_i. If, for every job J_j such that $d_j < d_i$, condition $r_j \leq r_i$ holds, then, in every sequence of $\mathcal{S}$ (i.e., in an optimal sequence), job J_i is sequenced after all jobs J_j.

We want to show that it is possible to derive what will be called a *master sequence,* denoted by σ, which "contains" every sequence in $\mathcal{S}$. A master sequence is a sequence in which each job may appear several times. A sequence in $\mathcal{S}$ is "created" from the master sequence by selecting at most one position for each job. The largest possible master sequence is $(J_1, J_2, \ldots, J_n, J_1, J_2, \ldots, J_n, \ldots, J_1, J_2, \ldots, J_n)$, where $(J_1, J_2, \ldots, J_n)$ is repeated n times, and thus has n^2 positions (this number is reduced in the sequel). Corollary 1 implies that there is only one position for J_j in the master sequence, and Corollary 2 that there is only one position for J_i.

THEOREM 3: In a master sequence, if $r_i < r_j$ and $d_i > d_j$, then there is a position for J_i before J_j and a position for J_i after J_j.

PROOF: Because $r_i < r_j$, Condition (2) in Theorem 1 is satisfied for the pair of jobs (J_i, J_j), and because $d_i > d_j$, Condition (1) is satisfied for the pair (J_j, J_i). Hence, there is a position in the master sequence for J_i before and after J_j. □

Hence, there must be a position in a master sequence for J_i after every job J_j such that $r_i < r_j$ and $d_i > d_j$. This shows that there will be *at most* $\frac{n(n+1)}{2}$ positions in the master sequence.

Table 1. Data for a 5-job problem.

Jobs	J_1	J_2	J_3	J_4	J_5
r_i	0	5	8	12	14
p_i	8	6	5	6	10
d_i	16	26	24	22	32

COROLLARY 3: If, for every job J_j such that $r_i < r_j$, the condition $d_i \leq d_j$ holds, then there is only one position for job J_i in the master sequence.

Corollary 3 shows that, when release and due dates are similarly ordered (as in Kise, Ibaraki, and Mine [6]), the master sequence will be the sequence of jobs in increasing order of their release dates (or due dates if some jobs have equal release dates). In the nonweighted case ($w_i = 1$, $\forall J_i$), the problem is then polynomially solvable using the algorithm proposed in [6] (in $O(n^2)$) or in [7] (in $O(n \log n)$).

An interesting and important property of the master sequence is a kind of transitivity property. If job J_i is set before and after J_j in the master sequence because either Condition (1) or (2) of Theorem 1 holds, and if J_j is set before and after J_k in the master sequence because either Condition (1) or (2) holds, then either Condition (1) or (2) of Theorem 1 holds and J_i is set before, and after J_k in the master sequence.

EXAMPLE 1: Let us consider a 5-job problem with the data of Table 1.

Considering sequences in $\mathcal{S}$, by Corollary 1, one knows that J_1 is set before all jobs (conditions $r_1 < r_i$ and $d_1 < d_i$ are satisfied for every job $J_i \neq J_1$), and all jobs are set before J_5 (conditions $r_i < r_5$ and $d_i < d_5$ are satisfied for every job $J_i \neq J_5$). Hence, in the master sequence σ, job J_1 will be set first and job J_5 last.

The master sequence has the following form:

$$\sigma = (J_1, J_2, J_3, J_2, J_4, J_3, J_2, J_5).$$

Every sequence of jobs in $\mathcal{S}$ can be constructed from σ. In this example, there are numerous sequences or early jobs (more than 40). For instance, the subset of sequences containing 5 jobs is:

$$\{(J_1, J_2, J_3, J_4, J_5), (J_1, J_2, J_4, J_3, J_5), (J_1, J_3, J_2, J_4, J_5), (J_1, J_3, J_4, J_2, J_5), (J_1, J_4, J_3, J_2, J_5)\}$$

One can check that each of these sequences is included in $\mathcal{S}$.

The algorithm to create the master sequence σ is sketched below. We suppose that the jobs are preordered in nondecreasing order of their release dates, and $\bar{J}$ denotes the set of jobs already sequenced. In Step 3, jobs that are already in the master sequence (set $\bar{J}$) and have a position after the new job J_i in σ (i.e., satisfying the conditions of Theorem 3) are added in nondecreasing order of their due dates. Hence, to speed up the algorithm, jobs added to $\bar{J}$ in Step 2 are ordered in nondecreasing order of their due dates.

FOR every job $J_i \in J$ DO

1. $\sigma \leftarrow \sigma \cup J_i$.
2. $\bar{J} \leftarrow \bar{J} \cup J_i$.
3. FOR every job $J_j \in \bar{J}$, $J_j \neq J_i$, such that $d_j \geq d_i$ DO $\sigma \leftarrow \sigma \cup J_j$.

The algorithm has a time complexity of $O(n^2)$. The job set at position k in σ is denoted $\sigma(k)$. The positions added in Step 3 of the algorithm are said to be *generated* by J_i. In the previous example, the 4th position of σ (J_2) is generated by J_3, and the 6th and 7th positions (J_3 and J_2) are generated by J_4. The number of positions in the master sequence is denoted by P. Recall that $P \leq \frac{n(n+1)}{2}$. Actually, P will only be equal to its upper bound if the job with the smallest release date has also the largest due date, the job with the second smallest release date has the second largest due date, and so on (see Theorem 3). This is clearly a very special case and, in practical experiments, P will be much smaller than $\frac{n(n+1)}{2}$.

3. A NEW MIXED-INTEGER LINEAR PROGRAMMING FORMULATION

The variable u_k is equal to 1 if the job $\sigma(k)$ is sequenced on time at position k in the master sequence, and is equal to 0 otherwise. Let us denote by t_k the start time of the job at position k in the master sequence (t_k is fictitious if $u_k = 0$). Then, one can derive the following model:

$$
\begin{cases}
c^* = \min \sum_{i=1}^{n} w_i U_i, & & (1)\\
t_k - t_{k-1} - p_{\sigma(k-1)} u_{k-1} \geq 0 & k = 2, \ldots, P, & (2)\\
t_k - r_{\sigma(k)} u_k \geq 0 & \forall\, k, & (3)\\
t_k + p_{\sigma(k)} u_k - d_{\sigma(k)} - D_k(1 - u_k) \leq 0 & \forall\, k, & (4)\\
\sum_{\substack{k=1 \\ \sigma(k)=i}}^{P} u_k + U_i = 1 & \forall\, i, & (5)\\
u_k \in \{0, 1\} & \forall\, k, & (6)\\
U_i \in \{0, 1\} & \forall\, i, & (7)
\end{cases}
$$

where D_k is chosen big enough to not constrain the jobs sequenced before k, for instance,

$$
D_k = \max_{\substack{l=1,\ldots,k-1 \\ d_{\sigma(l)} > d_{\sigma(k)}}} (d_{\sigma(l)} - d_{\sigma(k)}) \qquad \left(= \max_{l=1,\ldots,k-1} (0, d_{\sigma(l)} - d_{\sigma(k)}) \right).
$$

By Constraint (2) we ensure that, if the job at the kth position in the master sequence is set on time ($u_k = 1$), then the job at position $k + 1$ cannot start before the completion of the job at position k. If $u_k = 0$, the constraint only ensures that $t_{k+1} \geq t_k$. Constraint (3) specifies that, if the job is scheduled on time, it cannot start before its release date. By Constraint (4), if the job at position k is set on time ($u_k = 1$), then it has to be completed before its due date. If $u_k = 0$, the constraint is redundant. Finally, Constraint (5) ensures that at most one position is used for each job, or the job is late ($U_i = 1$).

In the previous model, it is possible to replace Constraint (3) by $t_k - r_{\sigma(k)} \geq 0$ [or equivalently to remove u_k from Constraint (3)]. The new constraint is numbered (3′). Theorem 4 will prove the validity of the resulting model.

In the nonweighted case ($w_j = 1$, $\forall J_j$), if Constraint (4) is replaced by $t_k + p_{\sigma(k)}u_k - d_{\sigma(k)} \leq 0$ [or equivalently $D_k = 0$ in Constraint (4)], then the resulting formulation still provides an optimal solution to the problem. The new constraint is numbered (4′). Although the nonweighted case will be analyzed in more details in Section 7, the following theorem is introduced here because it is also useful for the weighted case.

THEOREM 4: In the nonweighted case, there is always an optimal sequence of $\mathcal{S}$ that satisfies Constraints (2), (3′), (4′), and (5)–(7).

PROOF: The proof goes by showing that the only case where there is a problem is when J_j can be sequenced before and after J_i in the master sequence, and $r_j < r_i$ and $d_j > d_i$, and J_i is not sequenced in the optimal sequence. It can be shown that Constraints (2), (3′), and (4′) prevent job J_j to start between $d_i - p_j$ [Constraint (4′)] and r_i [Constraint (3′)]. This is only a problem if $d_i - p_j < r_i$. If this is the case, then $p_i < p_j$ (since J_i is not late if started at its release date r_i). Hence, in an optimal solution where J_j starts in the interval $[d_i - p_j, r_i]$, i.e., ends in the interval $[d_i, r_i + p_j]$, J_j can be replaced by J_i, and the sequence will remain optimal since J_i starts after r_i and ends before d_i. □

The proof of Theorem 4 is based on equal weight for jobs. In the weighted case, following the proof of Theorem 4, D_k can be chosen as follows:

$$D_k = \max_{l=1,\ldots,k-1} (0, r_{\sigma(l)} - d_{\sigma(k)})$$

Hence, the case where $d_i - p_j < r_i$, discussed in the proof of Theorem 4, is avoided. In numerical experiments, D_k is very often equal to zero.

4. REDUCING THE MASTER SEQUENCE

Because the size of the model is directly linked to the length of the master sequence, it is desirable to remove as many positions as possible from σ. Not only solution procedures will be more efficient, but the model will be tighter and will give better lower bounds by Lagrangean relaxation.

By Constraints (2) and (3), $t_k \geq \max_{l=1,\ldots,k-1} r_{\sigma(l)}$. Hence, the first reduction will be done by removing positions k such that $\max_{l=1,\ldots,k-1} r_{\sigma(l)} + p_{\sigma(k)} > d_{\sigma(k)}$.

Several dominance rules are proposed in [5] for the nonweighted case. However, if parameter D_k is changed according to Theorem 4, all of them do not apply. This is because, in the resulting formulation, when job J_j is before and after J_i in the master sequence and J_i is late, the position of J_j after J_i might need to be occupied in an optimal solution. One could show that this is not the case with the initial formulation. Our preliminary numerical experiments showed that reducing parameter D_k was more important than using the lost dominance rules.

We will describe here the dominance rules that still apply to our formulation, and which have been modified for the weighted case (see [5] for details).

In the master sequence, if Conditions (1) $r_i < r_j$, (2) $r_i + p_i \geq r_j + p_j$, (3) $r_i + p_i + p_j > d_j$, (4) $r_j + p_j + p_i > d_i$, (5) $d_i - p_i \leq d_j - p_j$, and (6) $w_j \leq w_i$ hold, then J_j dominates

J_i and all positions of job J_i can be removed from the master sequence. Because of Conditions (3) and (4), only one of the two jobs can be scheduled on time. In an optimal solution, either both jobs are late, or it is always possible to find a solution in which job J_j is on time and the total weight of late jobs is as small as a solution with job J_i on time.

Another dominance rule is based on the fact that if there is a position l and a job J_j [$J_j \neq \sigma(l)$] such that Conditions (1) $r_{\sigma(l)} + p_{\sigma(l)} \geq r_j + p_j$, (2) $p_{\sigma(l)} \geq p_j$, (3) $r_{\sigma(l)} + p_{\sigma(l)} + p_j > d_j$, (4) $r_j + p_j + p_{\sigma(l)} > d_{\sigma(l)}$, (5) $d_{\sigma(l)} - p_{\sigma(l)} \leq d_j - p_j$, and (6) $w_{\sigma(l)} \geq w_j$ are satisfied, then J_j dominates position l, and thus the latter can be removed. This is because, if there is an optimal solution in which position l is occupied (i.e., job $J_{\sigma(l)}$ is on time), then, by Condition (3), J_j is late. The solution can be changed to another optimal solution in which $J_{\sigma(l)}$ is replaced by J_j.

5. A LAGRANGEAN RELAXATION ALGORITHM

Following Theorem 4 and the remarks in Section 3, the mixed-integer linear programming formulation is now

$$
\begin{cases}
c^* = \min \sum_{i=1}^{n} w_i U_i, & & (8)\\
t_k - t_{k-1} - p_{\sigma(k-1)} u_{k-1} \geq 0 & k = 2, \ldots, P, & (9)\\
t_k - r_{\sigma(k)} \geq 0 & \forall\, k, & (10)\\
t_k + p_{\sigma(k)} u_k - d_{\sigma(k)} - D_k(1 - u_k) \leq 0 & \forall\, k, & (11)\\
\sum_{\substack{k=1 \\ \sigma(k)=i}}^{P} u_k + U_i = 1 & \forall\, i, & (12)\\
u_k \in \{0, 1\} & \forall\, k, & (13)\\
U_i \in \{0, 1\} & \forall\, i. & (14)
\end{cases}
$$

By relaxing Constraint (9) using Lagrangean multipliers λ_k ($k = 2, \ldots, P$), the model becomes

$$
\begin{cases}
\max_{\lambda_k \geq 0} \min_{t_k, u_k, U_i} \left[\sum_{i=1}^{n} w_i U_i - \sum_{k=2}^{P} \lambda_k (t_k - t_{k-1} - p_{\sigma(k-1)} u_{k-1}) \right], & & (15)\\
t_k - r_{\sigma(k)} \geq 0 & \forall\, k, & (10)\\
t_k + p_{\sigma(k)} u_k - d_{\sigma(k)} - D_k(1 - u_k) \leq 0 & \forall\, k, & (11)\\
\sum_{\substack{k=1 \\ \sigma(k)=i}}^{P} u_k + U_i = 1 & \forall\, i, & (12)\\
u_k \in \{0, 1\} & \forall\, k, & (13)\\
U_i \in \{0, 1\} & \forall\, i. & (14)
\end{cases}
$$

To use Lagrangean relaxation, one needs to solve the above model for given values of λ_k ($k = 2, \ldots, P$). The objective function can be written:

$$\min_{t_k,u_k,U_i}\left[\sum_{i=1}^{n} w_i U_i + \sum_{k=2}^{P} \lambda_k p_{\sigma(k-1)} u_{k-1} + \lambda_2 t_1 + \sum_{k=2}^{P-1} (\lambda_{k+1} - \lambda_k) t_k - \lambda_P t_P\right] \tag{16}$$

Because Constraint (9) has been relaxed, variables t_k are now independent and bounded through Constraints (10) and (11). Hence, if the coefficient of t_k $(\lambda_{k+1} - \lambda_k)$ is positive, t_k will be chosen as small as possible to minimize the cost, i.e., $r_{\sigma(k)}$ [because of (10)], and if the coefficient is negative, t_k will be chosen as large as possible, i.e., $d_{\sigma(k)} + D_k - (p_{\sigma(k)} + D_k)u_k$ [because of (11)]. Moreover, using (12), U_i can be replaced by $1 - \sum_{\substack{k=1 \\ \sigma(k)=i}}^{P} u_k$ in the criterion. Hence, (16) becomes

$$\min_{u_k}\left[\sum_{i=1}^{n} w_i\left(1 - \sum_{\substack{k=1 \\ \sigma(k)=i}}^{P} u_k\right) + \sum_{k=2}^{P} \lambda_k p_{\sigma(k-1)} u_{k-1} + \lambda_2 r_{\sigma(1)} + \sum_{\substack{k=2 \\ (\lambda_{k+1}-\lambda_k)\geq 0}}^{P-1} (\lambda_{k+1} - \lambda_k) r_{\sigma(k)}\right.$$
$$\left. + \sum_{\substack{k=2 \\ (\lambda_{k+1}-\lambda_k)<0}}^{P-1} (\lambda_{k+1} - \lambda_k)(d_{\sigma(k)} + D_k - (p_{\sigma(k)} + D_k)u_k) - \lambda_P(d_{\sigma(P)} + D_P - (p_{\sigma(P)} + D_P)u_P)\right]$$

Note that the minimization now only depends on variables u_k. Since r_i and d_i are data, several terms of the previous expression can be ignored in the optimization:

$$\min_{u_k} \sum_{i=1}^{n}\left[\sum_{\substack{k=1;\sigma(k)=i \\ (\lambda_{k+1}-\lambda_k)\geq 0}}^{P} (\lambda_{k+1} p_i - w_i)u_k + \sum_{\substack{k=1;\sigma(k)=i \\ (\lambda_{k+1}-\lambda_k)<0}}^{P} (\lambda_{k+1} p_i - (\lambda_{k+1} - \lambda_k)(p_i + D_k - w_i))u_k\right]$$

or, after simplification,

$$\min_{u_k} \sum_{i=1}^{n}\left[\sum_{\substack{k=1;\sigma(k)=i \\ (\lambda_{k+1}-\lambda_k)\geq 0}}^{P} (\lambda_{k+1} p_i - w_i)u_k + \sum_{\substack{k=1;\sigma(k)=i \\ (\lambda_{k+1}-\lambda_k)<0}}^{P} (\lambda_k p_i - (\lambda_{k+1} - \lambda_k)D_k - w_i)u_k\right], \tag{17}$$

where λ_1 and λ_{P+1} are parameters such that $\lambda_1 = \lambda_{P+1} = 0$.

To minimize the cost, and to satisfy Constraint (12), one has to determine, for every job J_i, the position k' such that $\sigma(k') = i$ with the smallest coefficient in (17), i.e., $(\lambda_{k+1} p_i - w_i)$ or $(\lambda_k p_i + (\lambda_{k+1} - \lambda_k) D_k - w_i)$, depending on the sign of $(\lambda_{k+1} - \lambda_k)$. If the coefficient is positive, then $u_k = 0$ $\forall k$ such that $\sigma(k) = i$, and $U_i = 1$, and if the coefficient is negative, then $u_{k'} = 1$, $u_k = 0$ $\forall k \neq k'$ such that $\sigma(k) = i$, and $U_i = 0$.

REMARK 1: We shall show that solving the relaxed problem can be done in $\mathbb{O}(P)$ time.

The solution of the relaxed problem would be the same, i.e., integral, if Constraints (13) and (14) were to be replaced by $0 \leq u_k \leq 1$, $\forall k$. Hence, the Lagrangean relaxation bound is identical to the bound obtained by linear relaxation (see Parker and Rardin [10]). However, this bound can be determined much faster, because every subproblem can be trivially solved, and larger problems can be solved. Actually, before implementing the Lagrangean relaxation

algorithm, we performed some preliminary testing using linear relaxation with a standard and efficient LP package. The quality of the bound was better than all other formulations we had tested before, including time-indexed formulations (see [4]).

It is relatively easy to interpret the impact of the values of λ_k, p_i, or w_i. Increasing λ_k will force the associated Constraint (9) to be satisfied, i.e., t_k to be chosen as large as possible and equal to $d_{\sigma(k)} + D_k - (p_{\sigma(k)} + D_k)u_k$ (u_k to 0), and t_{k-1} as small as possible and equal to $r_{\sigma(k-1)}$. Intuitively, a job with a large processing time that is set on time might force more jobs to be late than a job with a smaller processing time. Hence, it is natural to favor jobs with small processing times. This is consistent with (17), where the coefficient of u_k will increase with $p_{\sigma(k)}$, and then has more chances to become positive, thus inducing $u_k = 0$, i.e., job $J_{\sigma(k)}$ is not set in position k. The exact opposite can be said about the weight, since the larger its weight, the more you want to sequence a job. Again, this is in accordance with (17), where the coefficient of u_k will decrease with $w_{\sigma(k)}$, and then has more chances to become negative, thus inducing $u_k = 1$, i.e., job $J_{\sigma(k)}$ is set in position k.

The following algorithm is proposed to solve our problem using Lagrangean relaxation and subgradient optimization (see Parker and Rardin [10]).

Step 1—Initialization of the Lagrangean variables λ_k: $\lambda_k^0 = f\frac{p_{\sigma(k)}}{n * p_{max} * w_{max} * w_{\sigma(k)}}\ \forall k$ (where p_{max}, resp. w_{max}, is the largest processing time, resp. weight, among all jobs, and f a parameter), and $s = 0$ (where s denotes the iteration counter).

Step 2—Initialize the various parameters: $U_i = 1$, $coef(i) = \infty$ and $pos(i) = -1\ \forall i$, $u_k = 0\ \forall k$, and $\lambda_1^s = \lambda_{P+1}^s = 0$.

Step 3—Solve the relaxed problem:

Step 3.1—For $k = 1, \ldots, P$, if $\lambda_{k+1}^s - \lambda_k^s \geq 0$ then $tempcoef = \lambda_{k+1}^s p_i - w_i$, else $tempcoef = \lambda_k^s p_i - (\lambda_{k+1}^s - \lambda_k^s) D_k - w_i$.

If $tempcoef < coef(\sigma(k))$, then $coef(\sigma(k)) = tempcoef$ and $pos(\sigma(k)) = k$.

Step 3.2—For $i = 1, \ldots, n$, if $coef(i) \leq 0$ then $u_{pos(i)} = 1$ and $U_i = 0$.

Step 4—Compute the *lower bound:*

$$LB = \sum_{i=1}^{n} \left[w_i + \sum_{k=1;\sigma(k)=i}^{P} I1(\lambda_k^s, \lambda_{k+1}^s, u_k) \right],$$

where $I1(\lambda_k^s, \lambda_{k+1}^s, u_k) = (\lambda_{k+1}^s - \lambda_k^s)r_i + (\lambda_{k+1}^s p_i - w_i)u_k$ if $\lambda_{k+1}^s - \lambda_k^s \geq 0$, and $I1(\lambda_k^s, \lambda_{k+1}^s, u_k) = (\lambda_{k+1}^s - \lambda_k^s)(d_i + D_k - D_k u_k) + (\lambda_k^s p_i - w_i)u_k$ otherwise.

Step 5—Compute an *upper bound* by sequencing as many jobs as possible among the jobs J_i that are set on time in the solution associated to the lower bound, i.e., such that $U_i = 0$.

Step 6—Update the Lagrangean variables λ_k:

$$\lambda_k^{s+1} = \max\left(0, \lambda_k^s - \rho_s \frac{t_k - t_{k-1} - p_{\sigma(k-1)}u_{k-1}}{|t_k - t_{k-1} - p_{\sigma(k-1)}u_{k-1}|}\right),$$

where $t_k = r_{\sigma(k)}$ if $(\lambda_{k+1}^s - \lambda_k^s) \geq 0$, and $t_k = d_{\sigma(k)} + D_k - (D_k + p_{\sigma(k)})u_k$ otherwise. Update ρ_{s+1}, and $s = s + 1$.

Step 7—If no stopping conditions are met, go to Step 2.

We use a simple and fast greedy algorithm to determine the upper bound in Step 5. From $k = 1$ to $k = P$, job $J_{\sigma(k)}$ is added to the sequence of early jobs if $u_k = 1$ and $J_{\sigma(k)}$ is on time. The finishing time of the current sequence is updated each time a new job is added.

Various parameters have to be initialized and adjusted to ensure the best convergence of the algorithm for different types of instances. After sd iterations without improvement, the parameter ρ_s is decreased by a factor of $100 \times (1 - red_\rho)\%$. Various stopping conditions are checked: Maximum number of iterations $IterMax$, step ρ smaller than or equal to ρ_{min}, and of course if the optimum is found, i.e., the lower and upper bounds are equal. The parameters could be adjusted to improve the results on some instances, but we have decided to use generic parameters instead. After some preliminary testing, we chose the following values: $f = 0.4$, $\rho_0 = 1.6$, $sd = 40$, and $red_\rho = 0.9$. For the stopping conditions, we used $IterMax = 100{,}000$ and $\rho_{min} = 10^{-5}$. Actually, in our numerical experiments, the number of iterations is never larger than 20,000.

As already shown, every relaxed problem in Step 3 is solved very quickly, in $\mathbb{O}(P)$ time where P is not larger than $\frac{n(n+1)}{2}$. Hence, many iterations can be performed, even for large instances.

6. IMPROVING THE ALGORITHM

Several improvements are proposed. The first one is based on a rewriting of the formulation. In the model, because of Constraint (9), Constraint (10) can be rewritten as

$$t_k - rr_k \geq 0 \qquad \forall\, k,$$

where $rr_k = \max_{l=1,\ldots,k} r_{\sigma(l)}$ are release dates *per position.* To include this change in the algorithm, it suffices to replace $r_{\sigma(k)}$ by rr_k.

A similar rewriting can be performed for Constraint (11) when $D_k = 0\ \forall k$, as follows:

$$t_k + p_{\sigma(k)}u_k - dd_k \leq 0,$$

where $dd_k = \min_{l=k,\ldots,P;D_k=0} d_{\sigma(l)}$ are due dates *per position.* This rewriting can be done for all positions in the nonweighted case since $D_k = 0\ \forall k$.

Although they do not improve the lower bound obtained by linear relaxation, and thus by Lagrangean relaxation, these changes often considerably speed up the algorithm by better updating the Lagrangean multipliers in Step 6. This is because the positions for a job are better differentiated whereas, in the original formulation, they all have similar Constraints (10). Hence, the algorithm will more quickly choose the best position(s) for a job, and will require less iterations to converge to the lower bound.

Another improvement uses the following property to tighten Constraint (9) in the model.

THEOREM 5: Let J_i and J_j be two jobs in a master sequence. If a position k of J_i is generated by J_j, then there is an optimal schedule in which k is either not occupied or occupied and such that $t_k \geq d_j - p_i$.

PROOF: We want to prove that if, in an optimal schedule S, the position k of J_i generated by J_j is occupied and $t_k \leq d_j - p_i$, then this schedule can be transformed into an equivalent optimal schedule S' in which J_i is sequenced before J_j (i.e., position k is not occupied).

Since a position of J_i is generated by J_j, we know that $r_i < r_j$ and $d_i > d_j$. Hence, moving J_i before J_j will just shift J_j and the jobs between J_j and J_i in S by p_i and, because $t_k \leq d_j - p_i$ in S, the completion time of the shifted jobs will not be larger than d_j. By definition of the master sequence, and because position k is generated by J_j (see the end of Section 2 for the definition of a generated position), the due dates of the jobs between J_j and J_i in S are larger than or equal to d_j. Thus, the schedule S' is feasible. □

Following Theorem 5, Constraints (10) can be tightened (the added term is positive) for generated positions k (i.e., such that $rr_k > r_{\sigma(k)}$) as follows:

$$t_k - rr_k - RR_k u_k \geq 0 \qquad \forall\, k$$

where $RR_k = \max(0,\, d_{\sigma(l)} - p_{\sigma(k)} - rr_k)$, and l is the position that generated position k (i.e., $r_{\sigma(k)} < r_{\sigma(l)}$ and $d_{\sigma(k)} > d_{\sigma(l)}$).

The relaxed problem in the Lagrangian relaxation changes accordingly by adding the new term in the objective function, and by considering the coefficient $(\lambda_{k+1}p_i + (\lambda_{k+1} - \lambda_k)RR_k - w_i)$ when $(\lambda_{k+1} - \lambda_k)$ is positive. Strengthening the constraints helps to improve the quality of the lower bound. Moreover, it also accelerates the algorithm by again better differentiating the positions.

7. THE NONWEIGHTED CASE

The mixed-integer linear programming model defined in Section 3 can be enhanced for the nonweighted case, i.e., $w_i = 1\ \forall i$ following Theorem 4 in Section 3. The new model is given below:

$$\begin{cases} c^* = \min \sum_{i=1}^{n} U_i, & & (18)\\ t_k - t_{k-1} - p_{\sigma(k-1)}u_{k-1} \geq 0 & k = 2, \ldots, P, & (19)\\ t_k - r_{\sigma(k)} \geq 0 & \forall\, k, & (20)\\ t_k + p_{\sigma(k)}u_k - d_{\sigma(k)} \leq 0 & \forall\, k, & (21)\\ \sum_{\substack{k=1\\ \sigma(k)=i}}^{P} u_k + U_i = 1 & \forall\, i, & (22)\\ u_k \in \{0, 1\} & \forall\, k, & (23)\\ U_i \in \{0, 1\} & \forall\, i. & (24) \end{cases}$$

Because $w_i = 1\ \forall J_i$ and $D_k = 0$, $\forall k$, the objective function (17) can equivalently be written as

$$\min_{u_k} \sum_{i=1}^{n} \sum_{\substack{k=1;\\ \sigma(k)=i}}^{P} (\max(\lambda_k, \lambda_{k+1})p_i - 1)u_k. \qquad (25)$$

REMARK 2: In the nonweighted case, for a given job J_i, finding the position k', $\sigma(k') = i$, with the smallest coefficient in (17) is equivalent to finding the position with the smallest coefficient λ_{k+1} or λ_k, depending on the sign of $(\lambda_{k+1} - \lambda_k)$.

In the Lagrangean relaxation algorithm described in Section 5, the following steps are modified:

Step 3.1—For $k = 1, \ldots, P$, if $\lambda^s_{k+1} - \lambda^s_k \geq 0$, then $tempcoef = \lambda^s_{k+1}p_i - 1$; else $tempcoef = \lambda^s_k p_i - 1$.
If $tempcoef < coef(\sigma(k))$, then $coef(\sigma(k)) = tempcoef$ and $pos(\sigma(k)) = k$.
Step 4—Compute the lower bound:

$$LB = n + \sum_{i=1}^{n} \sum_{k=1;\sigma(k)=i}^{P} I2(\lambda^s_k, \lambda^s_{k+1}, u_k)$$

where $I2(\lambda^s_k, \lambda^s_{k+1}, u_k) = (\lambda^s_{k+1} - \lambda^s_k)r_i + (\lambda^s_{k+1}p_i - 1)u_k$ if $\lambda^s_{k+1} - \lambda^s_k \geq 0$, and $I2(\lambda^s_k, \lambda^s_{k+1}, u_k) = (\lambda^s_{k+1} - \lambda^s_k)(d_i - p_i u_k) + (\lambda^s_k p_i - 1)u_k$ otherwise.

Moreover, Kise, Ibaraki, and Mine's algorithm [6] can be used to compute the upper bound associated with the current value of the multipliers λ^s in Step 5. This is because, when the sequence in which jobs can be sequenced is fixed, i.e., for a given permutation of the jobs, the optimal sequence of early jobs can be found using Kise et al.'s algorithm. In our case, the set of jobs from which jobs have to be sequenced is the set of jobs J_i such that $U_i = 1$, and the fixed sequence is given by the positions k such that $u_k = 1$.

It is better to adjust the parameters for the algorithm when $w_i = 1$, $\forall i$. After multiple trials, we decided to use the following for all tested instances: $f = 0.4$, $\rho_0 = 0.6$, $sd = 40$, and $red_\rho = 0.9$. The same parameters are kept for the stopping conditions ($IterMax = 100{,}000$ and $\rho_{min} = 10^{-5}$).

8. COMPUTATIONAL RESULTS

Many test problems were generated to evaluate our algorithm. For each value of n, the number of jobs, 160 instances were randomly generated. The test program, written in C, was run on a SUN UltraSparc workstation.

The Lagrangean relaxation algorithm was first compared with solutions obtained by using the new formulation in the state of the art mixed-integer linear programming (MIP) solver ILOG-CPLEX. However, even for "small" size problems of 80 jobs, very large CPU times were observed with the MIP solver in many cases. For example, on a given instance, the search was stopped after 19 hours with the optimal solution, but its optimality was not proved. If the accuracy level is reduced to the one of the Lagrangean relaxation algorithm (see results in the sequel), the CPU time of the MIP solver decreases but remains very large.

Random generator. For each job J_i, a processing time p_i is randomly generated from the uniform distribution [1, 100] and a weight w_i is generated from the uniform distribution [1, 10]. As in [3], two parameters K_1 and K_2 are used, and taken in the set {1, 5, 10, 20}. Because we want data to depend on the number of jobs n, the release date r_i is randomly generated from the

Table 2. Results of the nonweighted case.

No. of jobs	Optimum		CPU time (s)			Gap			Gap (%) mean
	No.	(%)	Mean	St dev	Max	Mean	St dev	Max	
$n = 20$	88	55.0%	2.67	1.55	8.06	0.53	0.66	3	2.65
$n = 40$	60	37.5%	11.26	8.80	42.73	0.87	0.87	4	2.18
$n = 60$	25	15.6%	29.55	20.60	106.43	1.93	1.64	9	3.21
$n = 80$	11	6.9%	54.32	35.95	143.96	2.76	2.05	10	3.45
$n = 100$	2	1.3%	85.66	57.45	230.73	3.46	2.45	11	3.46
$n = 120$	2	1.3%	127.01	84.87	350.26	4.02	2.92	16	3.35
$n = 140$	1	0.6%	184.07	121.45	497.96	4.56	3.52	15	3.26

uniform distribution $[0, K_1 n]$, and the due date from the uniform distribution $[r_i + p_i, r_i + p_i + K_2 n]$. The algorithm was tested for $n \in \{20, 40, 60, 80, 100, 120, 140\}$. For each combination of n, K_1, and K_2, 10 instances are generated, i.e., 160 instances for each value of n.

Results of the nonweighted case. The Lagrangean relaxation algorithm was first run on the $1|r_j|\sum U_j$ problem. In Table 2, results are reported for each value of n. The optimum is considered to be found when lower and upper bounds are equal. For $n = 60$, 25 out of 160 instances are optimally solved, i.e., 15.6%. The CPU time necessary to find the best bounds is also reported. For $n = 80$, the mean CPU time is less than 1 minute. To evaluate the efficiency of both bounds, the gap between the upper and lower bounds is also measured and reported in the last four columns of the table. This gap is expressed in number of jobs and in percentage in the last column. For $n = 100$, the average gap is 3.46 jobs. The standard deviation and maximum gap are also given in the table.

The results are good, and the average duality gap remains stable when n increases. Remember that we decided to use the same parameters for our algorithm for every test instance, independently of n, K_1, or K_2. The algorithm does not perform so well when the master sequence is long. Looking at Theorem 3, this happens when there are many pairs of jobs (J_i, J_j) such that $r_i < r_j$ and $d_i > d_j$. This is the case when K_2 is large, and even more when K_1 is also small. The same analysis holds for the CPU time, since the time to solve the relaxed problem at every

Table 3. Sensitivity of the results to parameter K_2.

No. of jobs	Value of K_2	Length of σ		CPU time (s)		Gap		Gap (%) mean
		Mean	St dev	Mean	St dev	Mean	St dev	
$n = 100$	1	97.38	21.40	18.77	7.20	2.70	1.68	2.70
	5	667.65	225.48	63.13	21.99	1.57	0.84	1.57
	10	1161.12	313.23	104.58	30.35	3.00	1.20	3.00
	20	1677.60	303.77	156.18	37.74	6.55	2.29	6.55
$n = 120$	1	148.05	34.45	28.21	14.55	2.70	2.04	2.25
	5	1002.65	341.77	94.84	32.98	2.17	0.93	1.81
	10	1716.35	462.34	154.90	40.73	3.80	1.32	3.17
	20	2459.25	436.33	230.08	60.36	7.40	3.31	6.17
$n = 140$	1	214.90	65.15	41.90	21.76	2.67	1.93	1.91
	5	1439.20	517.32	139.62	52.36	2.42	1.06	1.73
	10	2412.40	677.09	226.25	66.99	4.25	2.12	3.04
	20	3357.38	606.50	328.48	80.37	8.90	3.65	6.36

Table 4. Comparing with the optimal solution.

No. of jobs	Lagrangean lower bound			Lagrangean upper bound		
	Opt. found	Gap w/mean	Opt. (%) max	Opt. found	Gap w/mean	Opt. (%) max
$n = 20$	54.37%	2.56	15.00	98.12%	0.09	5.00
$n = 40$	42.50%	1.95	7.50	91.25%	0.22	2.50
$n = 60$	24.38%	2.44	10.00	62.50%	0.77	6.67
$n = 80$	15.00%	2.28	7.50	43.75%	1.17	6.25
$n = 100$	8.75%	2.11	8.00	34.38%	1.34	6.00
$n = 120$	6.25%	1.97	7.50	26.25%	1.38	9.17
$n = 140$	3.75%	1.86	6.43	28.12%	1.40	7.14

iteration directly depends on the length of the master sequence P. This is why the CPU time average and standard deviation increase with the number of jobs and the value of K_2. Table 3 reports the results and the length of the master sequence for $n \in \{100, 120, 140\}$ and $K_2 \in \{1, 5, 10, 20\}$. Note that, for $K_2 = 20$, the mean CPU time is more than three times larger and the mean gap is approximately two times larger than in Table 2.

In [5], we propose a branch-and-bound procedure which is only valid for the nonweighted problem. This exact method also uses the notion of master sequence, and has been tested on the same set of instances. In a maximum running time of 1 hour, more than 95% of 140-job instances are solved to optimality. Hence, it is possible to compare the bounds given by our Lagrangean relaxation algorithm to the optimal solution for test instances that are optimally solved by our exact procedure. In Table 4, we compare the two bounds for instances of more than 80 jobs with the optimal solution.

For both the lower and upper bounds, the results are reported as follows: The first column gives the percentage of cases where the bound and the optimal solution are equal, and the next two columns give the mean and the maximum of the gap between the bound and the optimal solution. Even for the largest instances ($n = 140$), the upper bound is very good on average, with an error of less than 1.5%.

Better results could be obtained for the number of upper bounds equal to the optimal solution by adjusting the parameters of the Lagrangean algorithm. We did it using the following parameters $f = 0.4$, $\rho_0 = 0.05$, $sd = 60$ and $red_\rho = 0.92$. For example, with $n = 80$, the upper bound is equal to the optimal solution for 84.1% of instances, for $n = 100$, 68.2%, for $n = 120$, 35.5%, and for $n = 140$, 26.9%. Even if these results are better, the mean gap increases to more than 2.7% for $n = 140$ and the maximum gap increases to more than 14%!

Using these new parameters, the number of optimal solutions also increases. Table 5 gives the percentage of optimum values found for different values of n. But to obtain these results, the average CPU time also increases considerably (almost 6 minutes for $n = 140$).

The mean CPU times of the Lagrangean relaxation algorithm and the exact procedure are not very different, but the standard deviation is much larger in the second approach. Recall also that the branch-and-bound procedure can only solve nonweighted problems.

Results of the weighted case. Weights are randomly generated in the interval [1, 10]. Results are reported in Table 6. The Lagrangean relaxation algorithm seems to be as efficient as in the

Table 5. Optimum values found.

No. of jobs n	20	40	60	80	100	120	140
Opt. found (%)	53.1	46.9	41.3	34.4	16.3	6.9	5.0

Table 6. Results of the weighted case.

No. of jobs	CPU time (s)			Gap			Gap (%)
	Mean	St dev	Max	Mean	St dev	Max	mean
$n = 20$	4.18	2.28	12.46	4.46	3.54	20	4.07
$n = 40$	13.20	8.58	37.88	7.65	7.65	31	3.36
$n = 60$	30.66	20.39	83.28	10.26	6.37	41	3.09
$n = 80$	56.49	38.79	184.43	11.40	7.31	36	2.56
$n = 100$	86.21	58.83	231.57	14.82	8.78	47	2.70
$n = 120$	130.96	90.47	378.59	18.17	11.49	72	2.74
$n = 140$	184.66	130.34	496.85	29.18	21.46	127	3.82

nonweighted case. CPU times are close because the parameters sd and red_ρ are the same and control the convergence of the algorithm.

Again, the last column of Table 6 gives the gap between the two bounds expressed in %. One can notice that the gap remains very stable when the number of jobs increases. This gap can also be compared to the one given in Table 2 and is very similar. The table equivalent to Table 3 is not given in the weighted case, since it is very similar.

As in the nonweighted case, better results could be obtained by adjusting the parameters of the Lagrangean algorithm. We did it again for $n = 140$, $K_1 = 1$, and $K_2 = 20$. The average duality gap for the 10 instances reduces from 8.7% (max = 16.65%), when using the generic parameters ($f = 0.4$, $\rho_0 = 1.6$, $sd = 40$ and $red_\rho = 0.9$), to only 2.07% (max = 4.19%) by modifying ρ_0 and sd ($\rho_0 = 2.6$ and $sd = 80$). To obtain these results, the average CPU time increases to 751.09 seconds vs. 383.32 seconds with the previous parameters.

Other instances. Different ranges for weights and processing times have been considered to generate other instances. The generic parameters used in the previous set of instances have to be adjusted. Since the weights have a direct influence on the value of the objective function, the initial step ρ_0 in the subgradient search should be chosen accordingly. After many tests, we decided to use $\rho_0 = w_{max}/8$.

Instances of 100 jobs are generated and solved using the previous parameters. Table 7 presents the resulting gaps, and each line corresponds to 160 test problems. The results are very similar to the ones obtained on the previous instances. All average gaps remain below 3%.

9. CONCLUSION

This paper considers a single-machine scheduling problem in which the objective is to minimize the weighted number of late jobs. Based on the definition of the *master sequence,* a

Table 7. Results with different ranges.

Range of p_i	Range of w_i	Gap			Gap (%)
		Mean	St dev	Max	mean
[1, 100]	[1,10]	13.76	9.12	48	2.51
	[1,99]	133.12	79.39	424	2.66
[25, 75]	[1,10]	14.53	10.23	61	2.63
	[1,99]	144.86	92.07	486	2.92
[1, 500]	[1,10]	12.68	5.49	30	2.30
	[1,99]	135.06	60.52	322	2.73

new and efficient mixed-integer linear programming formulation is derived. By relaxing some coupling constraints using Lagrangean multipliers, the resulting problem becomes easily solvable. A Lagrangean relaxation algorithm is proposed and improved. Numerical experiments have been performed on an extended set of test instances for both the nonweighted and the weighted cases, and the algorithm performs well for problems of more than 100 jobs.

To our knowledge, our Lagrangean relaxation algorithm is the first method proposed to solve the problem $1|r_j| \sum w_j U_j$. We would like to improve the algorithm, in particular the number of iterations required to obtain the lower bound, by for instance using dual ascent instead of subgradient optimization when updating the Lagrangean multipliers.

The master sequence has also been used in a branch-and-bound method to solve the $1|r_j| \sum U_j$ problem, i.e., the non-weighted case [5]. We plan to use the bound provided by the Lagrangian relaxation algorithm in an exact method for the weighted case. Finally, it would be interesting to investigate other problems where the notion of master sequence could be applied. For instance, we believe it can be used to tackle the case where jobs can be processed in batches (although not with families; see Crauwels, Potts, and Van Wassenhove [2]).

ACKNOWLEDGMENTS

The authors are grateful to the referees for their useful remarks and suggestions.

REFERENCES

[1] P. Baptiste, C. Le Pape, and L. Péridy, Global constraints for partial CSPs: A case study of resource and due-date constraints, 4th Int Conf Principles Practice Constraint Program, Pisa, Italy, 1998 [Lecture Notes in Computer Science, Springer-Verlag, New York, to appear].

[2] H.A.J. Crauwels, C.N. Potts, and L.N. Van Wassenhove, Local search heuristics for single machine scheduling with batching to minimize the number of late jobs, Eur J Oper Res 90 (1996), 200–213.

[3] S. Dauzère-Pérès, Minimizing late jobs in the general one machine scheduling problem, Eur J Oper Res 81 (1995), 134–142.

[4] S. Dauzère-Pérès and M. Sevaux, Various mathematical programming formulations for a general one machine sequencing problem, Research Report 98/3/AUTO, École des Mines de Nantes, Nantes, France, 1998.

[5] S. Dauzère-Pérès and M. Sevaux, An exact method to minimize the number of tardy jobs in single machine scheduling, Research Report 99/6/AUTO, École des Mines de Nantes, Nantes, France, 1999.

[6] H. Kise, T. Ibaraki, and H. Mine, A solvable case of the one-machine scheduling problem with ready and due times, Oper Res 26(1) (1978), 121–126.

[7] E.L. Lawler, Knapsack-like scheduling problems, the Moore-Hodgson Algorithm and the "Tower of Sets" property, Math Comput Model 20(2) (1994), 91–106.

[8] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker, Complexity of machine scheduling problems, Ann Discrete Math 1 (1977), 343–362.

[9] J.M. Moore, A n job, one machine sequencing algorithm for minimizing the number of late jobs, Manage Sci 15(1) (1968), 102–109.

[10] R.G. Parker and R.L. Rardin, Discrete optimization, Academic Press, New York, 1988.

[11] C.N. Potts and L.N. Van Wassenhove, Algorithms for scheduling a single machine to minimize the weighted number of late jobs, Manage Sci 34(7) (1988), 834–858.

[12] F.J. Villarreal and R.L. Bulfin, Scheduling a single machine to minimize the weighted number of tardy jobs, IIE Trans 15 (1983), 337–343.

MÉTAHEURISTIQUES :
STRATÉGIES POUR L'OPTIMISATION DE LA PRODUCTION DE BIENS ET DE SERVICES

Résoudre des problèmes d'optimisation est un point clé dans l'amélioration constante de la productivité des entreprises. Quand les méthodes traditionnelles échouent, il devient alors naturel de se tourner vers des techniques de résolution approchée.

Les métaheuristiques jouent, aujourd'hui, un rôle primordial dans la résolution des problèmes d'optimisation. Ces techniques sont devenues, en quelques années, des outils incoutournables et performants. Dans cette synthèse, nous présentons un panorama des métaheuristiques classiques (méthodes de descente, recuit simulé, recherche tabou, algorithmes génétiques), de certaines moins connues (recherche à voisinages variables, GRASP, *iterated local search*, *guided local search*, colonies de fourmis) et de techniques avancées (algorithmes mémétiques, *scatter search*, GA|PM).

Pour toutes ces méthodes, nous analysons les facteurs d'intensification et de diversification présents, les particularités de chacune d'elle et notre retour d'expérience sur les applications que nous avons traités. De cette analyse, nous pouvons proposer ce que sont, selon nous, les caractéristiques indispensables à une bonne métaheuristique.

METAHEURISTICS :
STRATEGIES FOR THE OPTIMISATION OF THE PRODUCTION OF GOODS AND SERVICES

Solving optimisation problems is a key point in the constant improving of productivity in the industries. When traditional methods fail, it is natural to look towards some approximate resolution methods.

Metaheuristics play, today, an important role in solving optmisation problems. Those techniques became, in few years, necessary and powerful tools. In this synthesis, we present an overview of the classical metaheuristics (descent methods, simulated annealing, tabu search, genetic algorithms), some less known (variable neighbourhood search, GRASP, iterated local search, guided local search, ant colonies) and some advanced techniques (memetic algorithms, scatter search, GA|PM).

For all of these methods, we analyse intensification and diversification factors, specific features of each of them and our experiment on various applications we dealt with. From this analysis, we suggest what are, from our point of view, the necessary caracteristics of a good metaheuristic.

Marc Sevaux, Mai 2000
Marc.Sevaux@univ-valenciennes.fr