Contributions à la compréhension de problèmes d'optimisation combinatoire et études d'extensions de la méthode B

Vincent Poirriez

Contributions à la compréhension de problèmes d'optimisation combinatoire et études d'extensions de la méthode B.

Mémoire - Synthèse des travaux scientifiques

pour l'obtention de l'

Habilitation à diriger les recherches

(spécialité informatique)

présentée et soutenue publiquement le 6 décembre 2006, par

Vincent Poirriez

Composition du jury

Président :	Arnaud FREVILLE	UVHC : LAMIH (Valenciennes)
Rapporteurs :	Jeanine SOUQUIERES	NANCY2 : LORIA (Nancy)
	Denis TRYSTRAM	ENSGI-INPG : IMAG (Grenoble)
	Nicola YANEV	SOFIA (Bulgarie)
Examinateurs :	Rumen ANDONOV	RENNES1 : IRISA (Rennes)
	Sylvain LECOMTE	UVHC : LAMIH (Valenciennes)
	Frédéric SEMET	UVHC : LAMIH (Valenciennes)
	Pascal YIM	Ecole Centrale : LAGIS (Lille)

Habilitation à Diriger Les Recherches préparée au sein

du Laboratoire d'Automatique, de Mécanique et d'Informatique, industrielles et Humaines UMR 8530

et de l'Université de Valenciennes et du Hainaut Cambrésis

LAMIH
LABORATOIRE D'AUTOMATIQUE ET DE MECANIQUE INDUSTRIELLES ET HUMAINES

UVHC

Remerciements

J'ai plaisir à remercier ici celles et ceux, collègues ou étudiants, qui ont jalonné les étapes du parcours d'enseignant-chercheur dont ce document fait le bilan.

Je suis redevable à **Jeanine Souquières**, **Denis Trystram** et **Nicola Yanev** du travail d'évaluation de mon manuscrit qu'ils ont accepté d'accomplir. Travail dont j'ai conscience qu'il s'est cumulé avec tant d'autres activités. Les remarques de chacun sont des indications pertinentes pour les perspectives de ce travail.

Arnaud Fréville, tour à tour initiateur, catalyseur ou facilitateur de nombreux projets collectifs qui me fait l'amitié de libérer de son temps si rare pour présider ce jury.

Rumen Andonov, qui bien avant l'étape matérialisée par ce document m'a incité à prendre mon sac-à-dos. Nombre de mes travaux de recherche sont des travaux que nous avons en commun. Je n'oublie pas non plus son invitation à découvrir la Bulgarie.

Pascal Yim, qui accepte d'évaluer la présentation de cette synthèse traitant d'optimisation combinatoire et de sûreté de fonctionnement.

L'équipe ROI, avec une mention particulière pour les collègues du thème *SID*, qui réussi à conserver des conditions de travail confortables et sympathiques tout en préservant la diversité des thématiques de recherche de ses membres. **Frédéric Semet** (ROI) et **Sylvain Lecomte** (SID), en charge respectivement de l'équipe et du thème, me font l'amitié de les représenter dans ce jury de soutenance.

Guy Cousineau, **Thérèse Hardin**, **Xavier Leroy** ont initié ou enrichi ma première vie (parisienne) de chercheur en informatique, il en reste des traces.

À mon arrivée à l'UVHC, les collègues présents antérieurement m'y ont accueilli. C'est sur leurs labours que la recherche et des formations en informatique à Valenciennes peuvent grandir. En particulier, **Jean-Marie Raviart**, laboureur persévérant de l'enseignement de l'informatique à l'UVHC. L'informatique à Valenciennes (étudiants, chercheurs, enseignants-chercheurs) lui en est grandement redevable.

Toutes les personnes du département informatique de l'UVHC par leur diversité sont sources d'échanges intéressants et fructueux. Les noms cités dans ce document en constituent une trace.

Dorian et **Samuel** qui m'ont confirmé que patience, confiance et échanges permettent d'aider à éclore des chercheurs autonomes.

J'aimerai enfin remercier les cohortes d'étudiants, qui ont chaque année, de manière différente, su renouveler mon plaisir d'enseigner. Sans eux mon travail aurait perdu la moitié de son attrait.

À la mémoire d'Hafid Bourzoufi.

Résumé

Je présente dans ce mémoire un bilan de mon activité scientifique effectuée au sein des groupes POC (Parallèlisation et Optimisation Combinatoire) et SID (Systèmes d'Information Distribués) de l'équipe ROI (Recherche Opérationnelle et Informatique) du laboratoire LAMIH à l'UVHC. Mes travaux de recherche se divisent en trois parties :

- l'étude du problème du *sac-à-dos non borné*, problème classique de l'optimisation combinatoire, dont nous mettons en évidence des propriétés fondamentales et pour lequel nous avons dérivé, implanté et mis à disposition deux algorithmes qui tirent avantage des propriétés découvertes ;
- une approche algorithmique parallèle/distribuée pour la bio-informatique notamment le problème de *repliement de protéïnes* qui est un problème reconnu comme l'un des plus difficiles posés à la science informatique dans le contexte de la bio-informatique ;
- le développement d'une plate-forme ouverte d'expérimentations pour la méthode formelle B, l'étude de la *modularité du langage B* et d'extensions de B pour générer des composants logiciels ainsi que l'étude de l'*adjonction au langage B d'une logique temporelle*.

Nous montrons comment les approches utilisées dans une recherche en optimisation combinatoire d'une part et en spécification formelle d'autre part peuvent s'enrichir et se féconder mutuellement.

Mots-clés: Optimisation Combinatoire, Sac-à-dos, Repliement de protéïnes, Développement formel

Table des matières

Avant-propos

Préliminaires

Ce mémoire est rédigé en vue d'obtenir le titre de *docteur habilité en sciences*. Conformément à l'arrêté du 23 novembre 1988 (article 4) [97] il est constitué *"d'un dossier de travaux accompagné d'une synthèse de mon activité scientifique permettant de faire apparaître mon expérience dans l'animation d'une recherche"*.

Motivations Je voudrais commencer par te remercier, lecteur qui ouvre ce document. Par cet acte tu manifestes un intérêt pour mon travail. Peut-être ta lecture s'arrêtera rapidement ou peut-être iras tu piocher telle ou telle information, ou sûrement dans de plus rares cas, liras tu ce document in extenso. Peu importe, c'est parce que tu existes (virtuellement au moment où ces lignes sont écrites) que ce mémoire a un sens.

Pourquoi rédiger un tel mémoire et présenter mes travaux devant un jury de collègues ? Je me suis longtemps posé cette question. Alors certes il y a ce que permet la possession du titre de docteur habilité, les textes officiels sont explicites : Circulaire d'application 89-004 du 5 janvier 1989 : *L'habilitation à diriger des recherches est un diplôme dont la finalité essentielle, sinon exclusive, est de permettre l'accès au corps des professeurs d'universités* [99]. Bien, il reste que je demeure convaincu que l'essentiel de mon activité professionnelle est caractérisée par le fait que je suis *enseignant-chercheur* et non pas maître-de-conférences ou professeur-des-universités. Les textes étant ce qu'ils sont aujourd'hui, il y a aussi cette possibilité d'encadrer des doctorants en en étant pleinement responsable, c'est un argument important. En fait, ultimement,

faire le point sur mon activité scientifique depuis l'obtention de ma thèse est l'argument décisif.

Avec de la méthode et de la logique on peut arriver à tout aussi bien qu'à rien.
Pierre Dac

Jeune professeur certifié de mathématiques exerçant à Reims en 1984/85, c'est sur l'incitation de Jean-Pierre Steen, alors professeur d'informatique à l'Université de Reims, que je débutai des travaux de recherche en informatique. Le DEA d'Informatique Fondamentale de Paris 7 avec un stage encadré par Pierre-Louis Curien m'a donné le goût de la recherche à la confluence des mathématiques et de l'informatique. Le travail de thèse confié par Guy Cousineau qui consistait à étudier la faisabilité d'une extension d'un langage fortement typé (ML) avec des fonctionnalités logiques m'a confirmé que s'affronter à des questions ouvertes avec l'exigence de fondations théoriques (il ne s'agit pas de bidouille ni d'empirisme) et l'objectif d'une réalisation concrète est une tâche passionnante et plaisante. Recruté en septembre 1992 comme enseignant-chercheur à l'Université de Valenciennes et du Hainaut Cambrésis (**UVHC**) c'est donc au Laboratoire d'Informatique et de Mathématiques Appliquées de Valenciennes (**LIMAV**), devenu ensuite l'équipe Recherche Opérationnelle et Informatique (**ROI**) du Laboratoire d'Automatique, de Mécanique et d'Informatique industrielles et Humaines (**LAMIH**), et à l'Institut des Sciences et Techniques de Valenciennes (**ISTV**) que j'ai eu à exercer une activité de recherche en informatique et à former des étudiants. Le LIMAV (puis le LAMIH-ROI) ainsi que l'ISTV étaient alors sous la responsabilité d'Arnaud Fréville qui m'a fait confiance tant dans l'aspect animation de la recherche dans une thématique éloignée des siennes que dans les aspects de la vie pédagogique d'une formation universitaire.

Lors de ma brève audition de recrutement, j'ai eu à répondre à la question : *envisagez vous d'adapter vos thématiques de recherche vers des aspects pratiques ?* Ce que j'ai traduis en *envisagez vous d'adapter vos recherches au contexte du laboratoire ?* Le présent document a pour objet de faire un flashback sur quatorze années d'adaptation dynamique des différents composants de mon activité scientifique aux évolutions du contexte.

Le document est organisé ainsi : dans la partie **Synthèse d'activités scientifiques** (p.5) je positionne mes travaux dans le contexte scientifique dans lequel j'évolue depuis l'obtention de ma thèse ; dans la partie **Recueil de publications**(p.47) est rassemblée une sélection d'articles publiés (ou soumis) depuis l'année 2000 dans des journaux ou conférences d'audience internationale reconnue.

Synthèse d'activités scientifiques

Cette partie constitue la synthèse de mes activités scientifiques depuis l'obtention de mon doctorat. Je décompose ces activités en un pôle recherche d'une part (p.9) et un pôle enseignement d'autre part (p.27).

Je commence par donner, page suivante un tableau d'indicateurs de cette activité ainsi qu'un résumé de mon parcours professionnel.

Un tableau d'indicateurs

Année de naissance	1958
Maîtrise de Mathématiques(Reims)	1980
Capes es mathématiques	1982
Thèse de Doctorat d'Informatique Fondamentale de l'Université de Paris 7	1991
Nombre de thèses encadrées(taux d'enc : 90%) et soutenues	2
Nombre de DEA-Master 2 recherche encadrés (3 à 100% ; 3 à 50%)	6
Nombre de séjours de recherche à l'étranger	2(Bulgarie 1 mois, Espagne 1 sem.)
Nombre de publications en revues internationales	4
Nombre de publications en soumission à des revues internationales	2
Nombre de publications en conférences internationales avec actes	20
Nombre de publications en conférences nationales ou internationales sans acte	5
Lecteur arbitre pour	EJOR, IST, ICHPC, DATE
Membre de jury de thèses	4
Nombre de (co)-responsabilité de projets avec collaboration régionale ou nationale	5
Responsabilités électives nationales	CNU 27
Responsabilités électives locales	CA UVHC et CSE 27 UVHC
Membre du bureau du Département Informatique de l'UVHC	1996-1998
Cours créés	10 dont 3 de cycle 3, 3 de cycle 2 et 4 de cycle 1
Missions collectives importantes assurées	Coordination passage Deug A MP en Deug Mias(1994), et passage licence-maitrise informatique en IUP GMI(2000-2002)
Participation à la visibilité de l'informatique à l'UVHC	Co-fondateur du département informatique de lUVHC, Responsable des relations internationales de la filière informatique (1995 à 2003)
Associations professionnelles	SPECIF et SGEN

Parcours professionnel

Maître de conférences Hors Classe	2006
Titulaire de la PEDR	depuis 2004
Délégation CNRS(LAMIH-UMR 8530)	1 an : 2004-2005
Maître de conférences 1ERE Classe	1996
Maître de conférences(Université de Valenciennes)	1992 - ...
ATER (IUT de Reims, Dépt. Informatique)	1989-1992
Vacataire (Université de Paris I)	1988-1989
Allocataire de Recherche (Ministère de la Recherche)	1986-1988
Vacataire (Université de Reims)	1985-1986
Enseignant de Mathématiques dans le secondaire(Académies de Reims et Amiens)	1983-1985
Enseignant de Mathématiques en lycée et collège dans le cadre du VSNA(Burkina Faso)	1981-1983

Chapitre 1

Activités de recherche

1.1 Introduction

J'ai commencé mon travail de chercheur par des travaux de thèse [30] à l'Université de Paris 7 dans l'équipe qui s'appelait alors "projet FORMEL". Les thématiques de recherche de l'équipe s'articulaient autour des questions ouvertes posées par la conception de langages de programmation. Sous la direction de Guy Cousineau, j'ai exploré la possibilité d'étendre le langage ML avec des fonctionnalités logiques "à la PROLOG". Ces travaux aboutirent à la définition de MLOG, une extension de ML avec variables logiques, unification et un mécanisme de suspension d'évaluation, ainsi qu'à la réalisation d'un compilateur pour cette extension. Nous avons montré que le noyau de ML ainsi étendu conserve la propriété de confluence (Church-Rosser) des langages fonctionnels[1]. Le compilateur est une extension de la première mouture du compilateur de caml-light développé alors par Xavier Leroy [48].

Recruté en 1992 à l'UVHC, il m'a fallu :

- prendre une part conséquente du travail administrativo-pédagogique (cf. chapitre 2),
- m'intégrer aux thématiques du laboratoire d'accueil (le LIMAV[2]), la principale étant alors la recherche opérationnelle.

Mon activité de recherche s'inscrit dans l'activité plus large du groupe Recherche Opérationnelle et Informatique (ROI) du LAMIH animé initialement par Arnaud Fréville et aujourd'hui par Frédéric Semet, et plus précisément dans les sous-groupes : *Parallélisme et optimisation combinatoire(POC)* animé par Rumen Andonov jusqu'en 2005 et *Systèmes d'Informations Distribués(SID)* initialement créé par Didier Donsez et aujourd'hui animé par Sylvain Lecomte. En 1997, ce sont deux sollicitations concomitantes : le recrutement de Rumen Andonov comme professeur et un contact avec l'équipe ESTAS de l'INRETS (Lille) qui m'ont amené à travailler sur ces deux thématiques disjointes qui ne sont pas dans le prolongement direct de mes travaux de thèse.

Je présente donc dans deux sections distinctes ces deux thèmes de recherche sans chercher à procéder à une unification thématique illusoire. La cohérence que j'y trouve réside dans la démarche employée avec une attention portée à la spécification des questions et, d'autre part, l'expérimentation de la fécondité des aller-retours entre spécification et formalisation de propriétés d'un côté et implantation de l'autre aboutissant à des réalisations efficaces fondées théoriquement.

[1] Ces travaux ont donné lieu aux communications [16, 17, 18, 19] et sont présentés dans un article paru dans le journal Theoretical Computer Sciences [4].

[2] La partie informatique du LIMAV a intégré le LAMIH UMR 8530 en 1997

1.2 Parallélisme et optimisation combinatoire

1.2.1 Sac-à-dos non borné

À l'arrivée de Rumen Andonov, nous avons commencé à travailler sur un problème classique d'optimisation combinatoire : le sac-à-dos non borné (*Unbounded Knapsack Problem* **(UKP)**). Nous avons obtenu plusieurs résultats intéressants dont la mise en évidence de propriétés jusqu'alors inconnues de UKP.

La résolution d'un UKP consiste à maximiser la valeur du contenu d'un récipient de capacité fixée, *le sac-à-dos*, en le remplissant avec des articles de différents types (encombrement ou poids, valeur ou profit)[3]. Ce qui caractérise le sac-à-dos non borné c'est que le nombre d'articles pour chaque type est illimité. D'autres versions du sac-à-dos sont plus fréquemment étudiées : le sac-à-dos 0/1 où l'on peut prendre au plus un article de chaque type, et le sac-à-dos borné où le nombre d'articles de chaque type est limité. Le lecteur intéressé par une revue exhaustive des problèmes de sac-à-dos est renvoyé à la lecture d'un livre publié récemment, dédié aux différentes versions du sac-à-dos [88]. Les **Article 1** (il s'agit de la publication [2]) et **Article 2** (actuellement soumis) de la partie **Recueil de publications** regroupent nos résultats. Je résume ci-dessous ces contributions et leur historique.

Une formulation possible de la valeur optimale du sac-à-dos non borné de capacité c avec des types d'articles de poids $\mathbf{w}$ et profits $\mathbf{p}$ est :

$$f(\mathbf{w}, \mathbf{p}, c) = \max \{\mathbf{px} \text{ tel que } \mathbf{wx} \leq c, \mathbf{x} \text{ vecteur d'entiers naturels}\} \tag{1.1}$$

où $\mathbf{w}, \mathbf{p}$ et $\mathbf{x}$ sont des vecteurs de taille n.

Rumen Andonov est arrivé avec une question issue de ses travaux antérieurs à l'IRISA avec Sanjay Rajopadhye : *dans [66] nous avons décrit un algorithme utilisant une représentation creuse et une approche de programmation fonctionnelle avec des flots ciblant la conception d'un circuit VLSI dédié pour la résolution de UKP, comment aller plus loin ?* Un premier prototype rapidement écrit (grâce au langage caml [47]) nous a montré que non seulement cette idée était utilisable, mais que de plus elle ouvrait d'intéressantes perspectives.

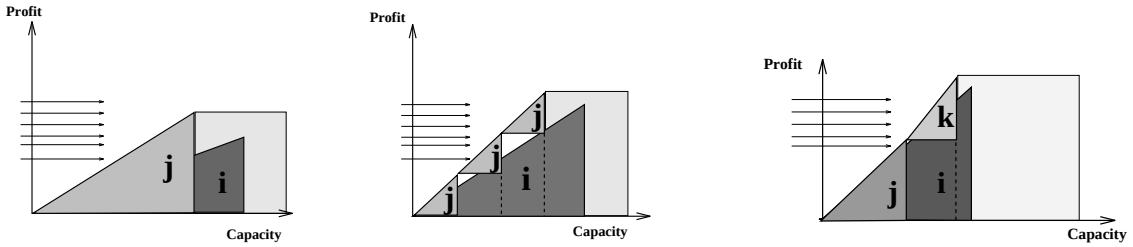
Résultats obtenus

Relations de dominance, ***Threshold dominance : Dominance de seuil.*** L'observation du comportement du prototype sur un grand nombre d'instances nous a intrigué. Les solutions optimales construites faisaient intervenir très peu de types distincts, rarement plus de trois, et pratiquement jamais au delà de dix. Cela malgré des nombres d'objets distincts de l'ordre de plusieurs milliers ou même centaine de milliers. L'analyse nous a montré que cette observation met en évidence l'effet de relations de dominance entre objets.

Intuitivement un objet dominé par un (ou des) autre(s) peut être ignoré, il n'est pas nécessaire pour l'obtention d'une valeur optimale. Un certain nombre de ces relations de dominance étaient connues : la dominance simple [78] la dominance multiple [86], et la dominance modulaire [95]. Ces trois relations concernent des paires d'objets (un objet dominé par un autre). Nous avons formalisé deux nouvelles relations de dominance : la relation de *dominance collective*, qui a été indépendemment mise en évidence dans un rapport de recherche par Pisinger [89], et la relation que nous avons appelée *dominance de seuil* qui la généralise.

Les relations de dominance entre deux objets sont utilisables dans une phase de pré-calcul pour éliminer des objets. Il n'est pas envisageable d'utiliser les relations de dominance de seuil et de dominance collective en phase de pré-calcul en raison de leur coût en calculs, par contre leur

[3] J'utiliserai indifféremment *type d'article* ou *objet*.



Dominance simple , $i \ll_s j$ Dominance multiple, $i \ll_m j$ Dominance collective, $i \ll \{j,k\}$

Un type d'objet est représenté par un triangle de largeur w et hauteur p. Son "ombre" est la région à sa droite sous sa hauteur. À gauche : i est dans l'ombre de j. Au milieu : i est dans l'ombre combinée de $3j$. À droite : i est dans l'ombre combinée de $j+k$.

FIG. *1.1 – Illustration des relations de dominances simple, multiple et collective.*

intégration est naturelle dans un algorithme basé sur la résolution de sous-problèmes. Ces deux nouvelles relations peuvent donc être utilisées dans une approche de programmation dynamique pour *éliminer dynamiquement* de nombreux objets devenus inutiles. Les figures **Fig.1.1** et **Fig.1.3**, reprisent de l'**Article 1** de la partie **Recueil de publications**, illustrent les relations de dominance préalablement connues et la relation de dominance de seuil. Voici une présentation unifiée des relations de dominance concernant l'objet i dominé par un sous-ensemble d'objets J. Elles peuvent être toutes exprimées à l'aide des inégalités suivantes :

$$\sum_{j \in J} x_j w_j \leq \alpha w_i, \text{ et } \sum_{j \in J} x_j p_j \geq \alpha p_i \text{ pour un vecteur } \mathbf{x} \in Z_+^n \tag{1.2}$$

où $\alpha \in Z_+$, $J \subseteq N$ et $i \notin J$.

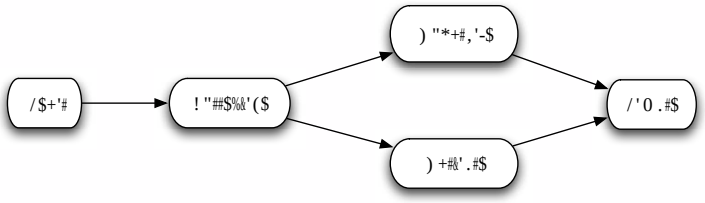
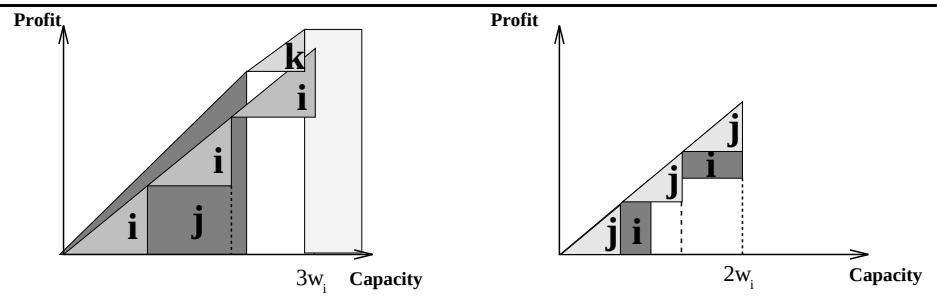


FIG. 1.2 – Ordre partiel entre relations de dominance

Relations de dominance

1. *De seuil* [2]. Le $i$ème objet est **threshold dominated** par le sous-ensemble d'objets J, décrit par $i \prec\!\!\prec J$ ssi (1.2) est vérifiée quand $\alpha \geq 1$. La plus petite valeur d'α qui vérifie cela définit le **seuil** (ou **threshold**) de l'objet i, décrit par t_i. $t_i = min\{\alpha w_i \text{ s.t. } \exists J, i \prec\!\!\prec J\}$.
   Intuition : *toute solution construite avec au moins $\alpha+1$ utilisations de l'objet i est égalable ou améliorable en se limitant à α utilisations de i combinées avec les autres objets de J.*
   Le plus petit objet parmi ceux qui ont le plus grand rapport profit/poids est appelé *meilleur objet* (ou *best item*) et noté b. Il est immédiat de constater que $t_i \leq w_b w_i$ ou même plus précisément $t_i \leq ppcm(w_b, w_i)$ où $ppcm(w_b, w_i)$ est le plus petit commun multiple de w_i et w_b.
2. *Collective* [2, 89, 95]. Le $i$ème objet est **dominé collectivement** par J, décrit par $i \ll J$ ssi (1.2) est vérifiée quand $\alpha = 1$, c'est à dire quand $t_i = w_i$.
   Intuition : *il est possible de faire mieux que i avec une combinaison des autres objets de J.*



$i \prec\!\!\prec \{j,k\}$ $\qquad$ $i \prec\!\!\prec \{j\}$ and $j \prec\!\!\prec \{i\}$

À gauche : $t_i = 3w_i$ *car 3 copies de* i *sont dans l'ombre de l'accumulation* $j+k$.
À droite : des objets equiprofitables *:* $2w_i = 3w_j = t_i = t_j$.

FIG. *1.3 – Illustration du seuil d'un type d'article.*

3. *Multiple* [86]. L'objet i est dominé de façon multiple (**multiply dominated**) par j, noté $i \ll_m j$, ssi pour $J = \{j\}, \alpha = 1$, $x_j = \lfloor \frac{w_i}{w_j} \rfloor$ les inégalités (1.2) sont valides.

   Intuition : *il est possible de faire mieux que* i *en empilant plusieurs articles du type* j.

4. *Modulaire* [95]. L'objet i est modulairement dominé (**modularly dominated**) par j, noté $i \ll_{\equiv} j$ ssi pour $J = \{b, j\}, \alpha = 1, w_j = w_i + tw_b, t \leq 0$, $x_b = -t, x_j = 1$ les inégalités (1.2) sont valides.

   Intuition : *il est possible de faire mieux que* i *avec une combinaison de* j *avec des articles du meilleur type* b.

5. *Simple*[78]. L'objet i est simplement dominé (**(simply) dominated**) par j, noté $i \ll_s j$, ssi pour $J = \{j\}, \alpha = 1$, $x_j = 1$ les inégalités (1.2) sont valides.

   Intuition : *il est possible de faire mieux que* i *avec un article du type* j.

Il est facile de constater qu'une relation d'ordre partiel relie ces cinq relations (voir le graphe figure **Fig 1.2**). Utilisant les relations de dominance de seuil et collective en combinaison avec une technique de programmation dynamique creuse nous avons conçu et mis à disposition de la communauté l'algorithme : *Efficient Dynamic programming for the Unbound Knapsack problem* (**EDUK**) cité dans [88] comme étant l'algorithme le plus efficace pour la résolution du sac-à-dos non borné.

EDUK est le premier algorithme publié et mis librement à disposition qui tire profit de la propriété dite de *périodicité* d'UKP(établie par Gomory dans [79]). Cette propriété signifie qu'au-delà d'un certain niveau y^* pour la capacité c, niveau dépendant uniquement de $\mathbf{w}$ et $\mathbf{p}$, toute valeur optimale de $f(w, p, c)$ peut être obtenue en utilisant un unique type d'objet pour compléter un sac-à-dos de capacité inférieure à y^*. Ce qui exprime le fait qu'au delà d'un certain niveau seul un objet reste non dominé (pour la relation de dominance de seuil).

Ce travail, collaboration avec Rumen Andonov et Sanjay Rajopadhye alors chercheur à l'IRISA, Rennes, a donné lieu à plusieurs communications dans des conférences internationales [27, 7, 8] et à l'article [2] dans *European Journal of Operational Research*. Dans cet article, outre ce nouvel algorithme et le résultat théorique sur la relation de dominance de seuil, nous présentons une comparaison entre les implantations de EDUK (écrit en ocaml) et MTU2 [86] qui implante un algorithme de séparation-évaluation (et est écrit en fortran). Cette comparaison est réalisée par la résolution de nombreux problèmes de différentes familles. Pour la première fois dans cet article, une étude de la sensibilité d'un algorithme aux variations de la valeur de la capacité du sac-à-dos est présentée.

Algorithme hybride, nouvelles bornes supérieures et SAW UKP. Suite aux comparaisons effectuées entre EDUK et MTU2, il apparaît clairement que les approches "programmation dynamique" (pour EDUK) et de "recherche arborescente par séparation-évaluation" (pour MTU2) n'ont pas leurs meilleures performances pour les mêmes instances, ce qui était attendu. La programmation dynamique creuse permet d'obtenir des performances prévisibles et stables, bornées par la complexité en $O(nc)$ classique alors qu'une recherche par séparation-évaluation a des performances en dents de scie avec des cas les pires en complexité exponentielle mais des meilleurs cas avec des résolutions instantanées. Nous avons donc entamé la conception d'un algorithme hybride pour résoudre UKP. En parallèle avec la conception de ce nouvel algorithme, nous avons pris en compte le fait que l'efficacité des algorithmes de séparation-évaluation est très dépendante de la qualité des bornes (supérieures pour UKP) utilisées. Les bornes préalablement publiées sont :

U_3 **(Martello et Toth [86])** Ici on suppose que les objets d'indice 1, 2 et 3 sont ceux avec les plus grands rapports profit/poids.

$$\begin{aligned} U_3 &= max\{U^0, \bar{U}^1\} \qquad (1.3)\\ \text{where : } z' &= \left\lfloor \frac{c}{w_1} \right\rfloor p_1 + \left\lfloor \frac{\bar{c}}{w_2} \right\rfloor p_2;\ \bar{c} = c \bmod w_1;\ c' = \bar{c} \bmod w_2 \\ U^0 &= z' + \left\lfloor \frac{c' p_3}{w_3} \right\rfloor \\ \bar{U}^1 &= z' + \left\lfloor \left(c' + \left\lceil \frac{w_2 - c'}{w_1} \right\rceil w_1 \right) \frac{p_2}{w_2} - \left\lceil \frac{w_2 - c'}{w_1} \right\rceil p_1 \right\rfloor \end{aligned}$$

U_s **(Caccetta et Kulanoot[70])** . $U_s = c + \left\lfloor \frac{c}{w_1} \right\rfloor \alpha$. Cette borne est meilleure que U_3 pour la famille des problèmes fortement corrélés (**SC-UKP**) définis par $p_i = w_i + \alpha$ avec $\alpha > 0$ (c'est à dire les problèmes ou le vecteur de profit $\mathbf{p}$ est obtenu par une translation du vecteur de poids $\mathbf{w}$). Ici, l'objet d'indice 1 est supposé être le plus léger.

Nous avons mis en évidence l'existence de bornes encore ignorées. La dernière borne exhibée est établie par le théorème suivant, démontré page PYAsUKP-3. Sans perte de généralité nous supposons aussi ici que l'objet d'indice 1 est l'objet le plus léger parmi ceux vérifiant $(p_i - w_i) \geq 0$ (i.e. $\forall\, i > 1,\ w_1 \leq w_i$ ou $p_i \leq w_i$) et $p_1 \geq w_1 + 1$.

Nous adoptons les notations suivantes :

pour tout $i \neq 1, q_i = \frac{p_i - p_1 \left\lfloor \frac{w_i}{w_1} \right\rfloor}{w_i \bmod w_1}$ si $w_i \bmod w_1 \neq 0$, $q_i = +\infty$ si $w_i \bmod w_1 = 0$

$q^* = max\,\{q_i, i \neq 1\}$, $\tau^* = min\,\{1, q^*\}$, $\beta(\tau) = max\left\{ \frac{(p_i - \tau w_i)}{\lfloor \frac{w_i}{w_1} \rfloor}, i \in N \right\}$, $\beta^* = \beta(\tau^*)$.

Théorème 1 *[U_{τ^*}]* *pour tout $UKP^c_{w,p}$, $f(\mathbf{w}, \mathbf{p}, c) \leq U_{\tau^*} = \tau^* c + \beta^* \left\lfloor \frac{c}{w_1} \right\rfloor$*

U^*_τ est une amélioration de la borne nommée U_v présentée la première fois dans [32] (définition rappelée page : PYAsUKP-3).

Nous définissons une famille de problèmes, que nous nommons SAW-UKP (initialement à cause de la forme en dents de scie de la frontière du domaine de cette famille Figure 1.4).

Définition 1 *La famille **SAW-UKP** est l'ensemble des instances d'$UKP^c_{w,p}$ telles que $q^* \leq 1$.*

Un problème de la famille SAW-UKP vérifie donc : pour tout i, $(p_i - w_i) \leq (p_1 - w_1) \left\lfloor \frac{w_i}{w_1} \right\rfloor$.

La famille SAW-UKP contient strictement la famille SC-UKP des problèmes fortement corrélés. Le **Theorem** 2 (page PYAsUKP-5) établi que U_{τ^*} est meilleure que les bornes connues pour la famille des SAW-UKP. Nous montrons que U_{τ^*} est toujours meilleure que U_v et que par-contre U_{τ^*} et U_3 (la borne établie par Martello et Toth dans [86]) sont incomparables.

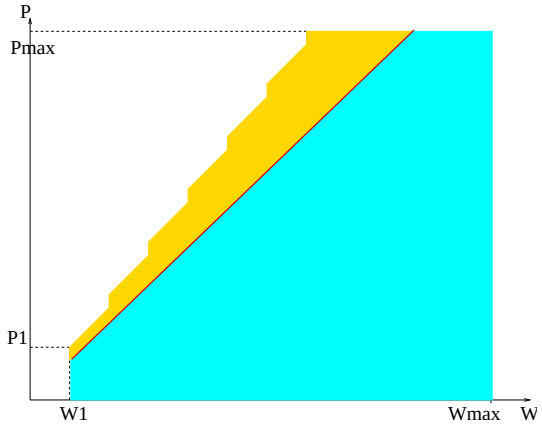


FIG. 1.4 – Domaine de la famille SAW-UKP

L'algorithme qui a été dérivé : (**EDUK2**) est intégré au logiciel ***P****YAsUKP is* ***Y****et* ***A****nother* ***s****olver for the* ***U****nbounded* ***K****napsack* ***P****roblem* (**Poirriez, Yanev and Andonov** [31]), un jeu d'instances tests est aussi mis à disposition. Cet algorithme combine donc plusieurs techniques classiques de l'optimisation combinatoire :

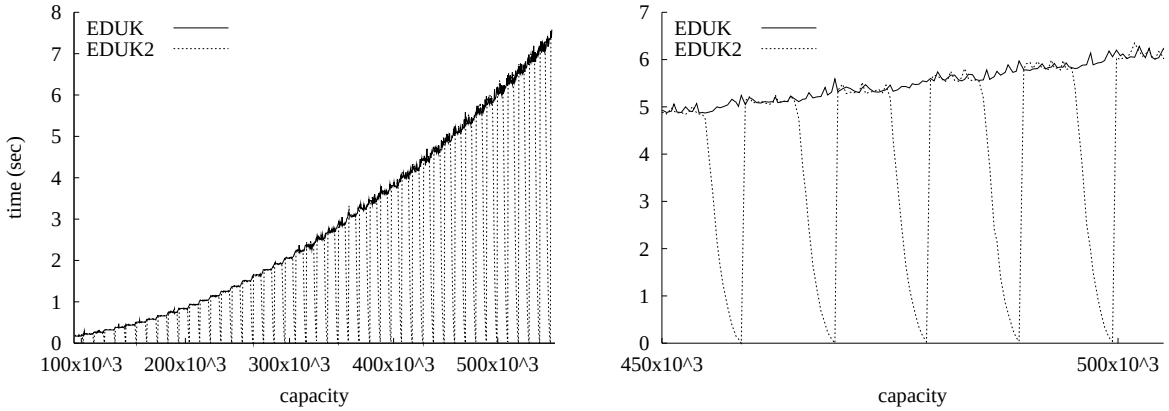
- la programmation dynamique ;
- les techniques de séparation et évaluation (*branch and bound*) ;
- il utilise nos nouveaux résultats sur les bornes.

Je ne reprends pas ici la description d'**EDUK2** qui se trouve page PYAsUKP-5.

J'ai présenté les préliminaires de ce travail à la conférence Combinatorial Optimization 2002 [28] et il a donné lieu à un rapport de recherche du LAMIH/ROI [32] dont une version améliorée, reprise dans ce document en tant qu'**Article 2**, est actuellement soumise à un journal. Nous y présentons des résultats d'évaluations mettant en concurrence les différents algorithmes connus et disponibles sur de très nombreuses instances de familles de problèmes différentes. Ces résultats démontrent l'efficacité atteinte par EDUK2 qui tire bénéfice des bonnes caractéristiques de chacune des méthodes de résolution combinées. Ceci peut être illustré par la figure **Fig. 1.5** où l'on peut observer que le temps de résolution mis par `EDUK2` est majoré par le temps de résolution via l'algorithme de séparation-évaluation et par le temps de complexité pseudo polynomial de l'algorithme en programmation dynamique creuse.

Perspectives

Ces algorithmes se sont avérés si performants en séquentiel qu'il s'avère inutile, malgré la problématique initiale, d'envisager de paralléliser la résolution de UKP. Pour nous ce travail sur le sac-à-dos non borné sera achevé en ce qui nous concerne avec la publication de l'article actuellement soumis. Toutefois, il y a des pistes à explorer pour lesquels nos résultats peuvent contribuer à des avancées. Je pense en particulier aux problèmes du sac-à-dos non-borné à plusieurs contraintes (le cas du sac-à-dos à deux contraintes semble un bon candidat à une première généralisation de nos résultats de dominance) et à un passage au sac-à-dos borné. La complexité des sac-à-dos à plusieurs contraintes réintroduit vraisemblablement de l'intérêt pour la conception d'un algorithme parallèle pour leur résolution. Une autre piste qu'il serait intéressant d'explorer consiste à dégager un schéma général d'algorithme coopératif combinant programmation dynamique et résolution arborescente pour résoudre des problèmes de décision.



Exemple connu comme difficile pour l'approche séparation évaluation, construit avec la formule (Chung *et al.* [72]) :

$$w_i = w_{\min} + i - 1 \quad \text{et} \quad p_i = w_i + \alpha \qquad \text{avec } w_{\min} \text{ et } \alpha \text{ fixés.} \tag{1.4}$$

avec les valeurs $n = 100$, $w_{\min} = n(n+1)+1$, $\alpha = -3$ et c pris de façon aléatoire uniforme dans l'intervalle $[90\,000, 560\,000]$. La figure de gauche est le tracé des temps de résolution pour la totalité de l'intervalle. Celui de droite se concentre sur le sous-intervalle $[450\,000, 500\,000]$. En moyenne, `EDUK2` est plus de 25% plus rapide que `EDUK`. `MTU2` nécessite systématiquement plus de 1200 sec., excepté pour les 5% de points pour lesquels il résout l'instance en moins de 12 sec. Ce sont les valeurs de c pour lesquels `EDUK2` trouve la solution dans les phases d'élimination de variable ou par la recherche par séparation-évaluation.

FIG. 1.5 – Sensibilité aux variations de la capacité des algorithmes `EDUK2` et `EDUK`

1.2.2 Pour la Bio-Informatique

Durant l'année 2000, nous avons réorienté nos recherches pour une application des compétences de l'équipe en algorithmique et parallélisme pour résoudre des problèmes émergeant dans le domaine de la bio-informatique. Nous collaborons avec plusieurs équipes sur cette thématique.

Structures 3D de protéïnes

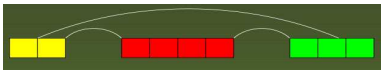
Avec l'équipe MIG de l'INRA de Jouy en Josas, nous travaillons autour de l'un des problèmes reconnus comme les plus difficiles de l'algorithmique pour la bio-informatique : le problème de prédiction du repliement des protéïnes par comparaison (*protein threading*)[69, 80, 81, 84, 91]. Il s'agit d'étudier la relation entre la séquence en acides aminés (structure linéaire ou chaîne sur un alphabet de 20 caractères) et la structure tridimensionnelle adoptée par la protéïne. La connaissance de la structure 3D est nécessaire pour expliquer la fonction d'une protéïne. Les éléments qui font qu'une approche de prédiction par comparaison est à la fois réaliste et nécessaire peuvent être synthétisés ainsi :

- Il y a un flux continu soutenu de nouvelles entrées de séquences de protéïnes dans les bases de données telles que la Protein Data Bank(PDB).
- Il y a relativement très peu de structures 3D différentes connues (de l'ordre de 1000). D'après les statistiques de la PDB toutes les nouvelles séquences soumises en 2004 appartenaient à des familles connues[4].

[4]statistics from `http ://www.rcsb.org/pdb/`

Fragment de la structure primaire : séquence de caractères.

Structure 3D avec hélices et feuillets colorés



Carte de contact de la structure secondaire, les séquences de liaison sont oubliées, les liaisons énergétiques entre les blocs de la structure sont représentés.

FIG. 1.6 – Structures d'une protéïne

- La détermination exacte (physique) de la structure 3D d'une protéïne est possible mais nécessite des moyens encore relativement lents et coûteux : radiocristallographie(rayons X), spectroscopie par Résonance Magnétique Nucléaire ;
- Il est possible de déterminer *empiriquement* les probabilités de présence des différents types d'acides aminés dans différents types de blocs structuraux, ce qui permet de définir des fonctions de scores pour mesurer l'adéquation d'un alignement entre une séquence et une structure 3D.

En se basant sur ces faits, les méthodes de reconnaissance de repliements basée sur le *threading* tentent d'aligner une séquence avec un ensemble de structures 3D pour discerner avec quelle(s) structure(s) la séquence est compatible, déterminant ainsi une (ou éventuellement plusieurs) structure vraisemblable. Les composants de ces méthodes peuvent être décrits par :

1. une base de données de structures 3D connues (appelées modèles, cœurs ou templates) ;
2. une fonction d'évaluation d'alignement entre une séquence et une structure modèle ;
3. une méthode pour trouver le meilleur (d'après la fonction d'évaluation) alignement possible ;
4. une analyse statistique pour ne conserver que les alignements pertinents.

Le troisième point de cette décomposition concerne le problème de trouver l'alignement optimal entre une séquence et une structure. C'est ce problème qui est connu sous le vocable *protein threading problem*(PTP). Pour un informaticien, c'est le composant le plus interpellant du problème de repliement. En effet, il a été montré que dans le cas général ce problème est NP difficile [82] et même MAX-SNP-hard [63] ce qui signifie qu'il n'existe pas d'algorithme d'approximation polynomiale arbitrairement précise, à moins que P=NP. Les résultats des différents algorithmes actuellement publiés [83, 94, 65, 92, 93, 67] sont donc remarquables et montrent que le problème PTP est plus facile à résoudre en pratique (avec des séquences de vraies protéïnes) qu'en théorie. Il reste cependant des problèmes concrets très difficiles, ainsi les auteurs de [92] ont trouvé 30 structures modèles pour lesquelles il a fallu plus de 15 heures pour calculer le repliement d'une séquence cible sur ces structures (la machine utilisée disposait de 20GO de RAM et d'un processeur MIPS R12000 à 40400 MHz).

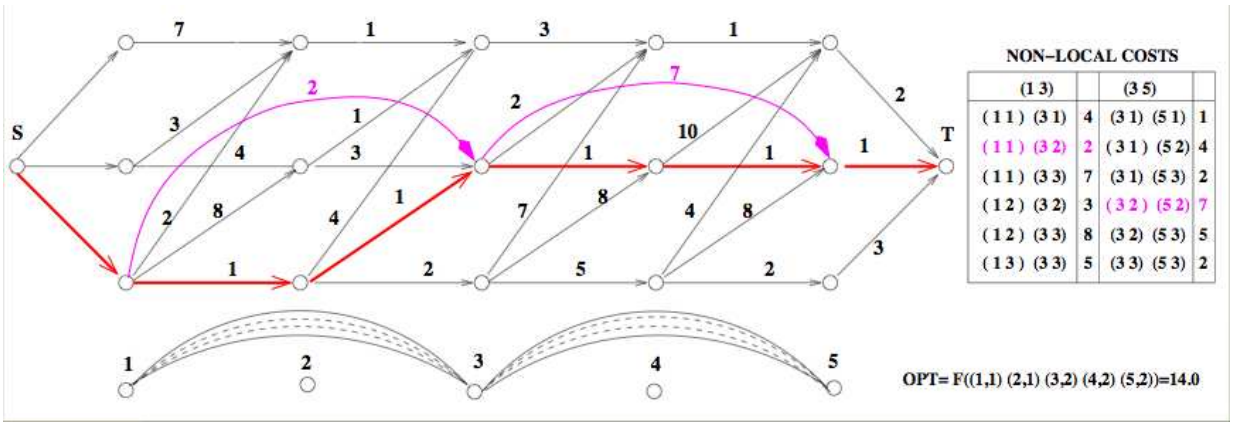
Description du *protein threading problem* Schématiquement, les biologistes identifient dans une séquence protéique des sous-séquences significatives qui constituent la structure secondaire de la protéïne. Ces sous-séquences sont reliées par des caractères "peu" significatifs. Ce sont les relations (liaisons énergétiques) entre les éléments de la structure secondaire qui déterminent la structure tridimensionnelle. Ces liaisons sont déterminées par les biologistes et fournies sous la forme de matrices, ceci constitue un des paramètres du problème. La figure 1.6 présente cette modélisation. Le problème PTP consiste à trouver l'alignement optimal entre une séquence (dont on veut trouver la structure) et une structure 3D modélisée par une structure secondaire et les liaisons entre blocs. C'est la présence de liaisons qui rend le problème difficile. Pour donner des ordres de grandeur sur les dimensions des espaces de recherche, nommons `L` la longueur de la séquence inconnue, `m` le

nombre de blocs de la structure secondaire, le bloc i ayant pour longueur l_i et $\mathtt{n}$ le degré de liberté de l'alignement, $\mathtt{n}$ est défini par :

$$n = L + 1 - \sum_{i=1}^{m} l_i$$

Avec ces notations, le nombre d'alignements possibles est $\frac{(n-1+m)!}{m!(n-1)!}$ et le tableau suivant donne la taille de cet espace de recherche pour quelques couples : (séquence requête, modèle) pris dans la base de donnée de FROST(cf ci-dessous).

Nom requête	**Nom modèle**	`Nombre alignements`
1bdo_0	1paub0	2.03e+05
2cyp_0	1thea0	1.8e+30
1gal_0	1ad3a0	1.3e+39



Graphe pour l'alignement d'un modèle de **m=5** blocs (les colonnes) avec un degré de liberté **n=3** (les lignes). Les arcs étiquetés rectilignes sont les liaisons locales. Les arcs étiquetés courbes sont les liaisons non locales, seuls deux sont dessinés, ils sont donnés par le tableau à droite. Le plus court chemin a un poids de 14, il est noté en bas à droite.

FIG. 1.7 – PTP sous forme de graphe

Yanev et Andonov ont modélisé ce problème sous la forme d'un problème de graphe de flux [94]. Il s'agit alors de déterminer le plus court chemin dans un tel graphe. Il faut noter que s'il n'y avait pas les liaisons non locales entre blocs, le problème serait un problème bien connu résolu facilement par une approche de programmation dynamique par exemple. La figure 1.7 représente un tel graphe de flux et le chemin optimal.

Collaborations, réalisations et contributions personnelles Le début de notre travail sur ce sujet a été financé par une action IHPERF98, ce qui a permis de financer un stage d'étudiant de maîtrise informatique (Julien Pley) que j'ai co-encadré. Nous travaillons à la mise à disposition de la communauté des biologistes d'une application : *Fold Recognition Oriented Search Tool* **FROST** [85, 77], parallèle et distribuée s'attaquant à ce problème. Cette application est conçue initialement en séquentiel par Antoine Marin et Jean-François Gibrat de l'équipe Mathématique Informatique et Génome de l'INRA[5]. Les chercheurs de l'équipe MIG participent en utilisant FROST aux réunions Critical Assessment of Structure Prediction (CASP) [71] dans lesquelles FROST tient une place honorable parmi les logiciels de prédiction de structure 3D de protéïnes. L'idée maîtresse de l'architecture de FROST est d'être une succession de filtres qui sélectionnent les structures 3D potentiellement coupables avec la séquence cible. Les filtres sont organisés par ordre croissant de

[5]Ce travail fait partie de l'action incitative GENOGRID dont je suis acteur.

complexité. Lorsqu'une requête (une séquence) est présentée en entrée au logiciel pour tenter de prédire sa structure 3D, elle est comparée aux structures présentes dans la base de données. Actuellement un pré-filtre sur la longueur (on ne compare que des séquences de longueurs proches dans la version courante de FROST) est suivi par deux filtres. Un premier filtre assez imprécis mais peu coûteux qui permet de sélectionner une dizaine de structures candidates puis un filtre qui réalise l'alignement en prenant en compte les liaisons non locales. C'est ce deuxième filtre qui est NP difficile. Il est très important de valider chacun des scores des alignements obtenus par une étude statistique. Cela nécessite pour chaque modèle dans la base le calcul d'environ 200 alignements pour disposer d'une distribution représentative.

Mes contributions principales peuvent être décrites ainsi :

Modularisation Initialement le calcul des distributions était effectué à la demande. Lorsqu'une requête est présentée si le modèle auquel elle est confrontée ne dispose pas de distribution alors le calcul de cette distribution est enclenché. La motivation de cette conception est la motivation habituelle de l'évaluation paresseuse : ne pas faire de calcul coûteux sans nécessité. La conséquence est une durée aléatoire, parfois très longue, du traitement d'une requête. Nous avons repensé l'architecture globale de FROST avec Antoine Marin. Le calcul des distributions peut être fait dans une phase de pré-calcul indépendante des requêtes. De plus comme différentes implantations des filtres peuvent être conçues, le logiciel a été rendu modulaire, il est possible de rajouter ou remplacer des fonctions-filtres. En particulier il est possible d'inter-changer les fonctions implantant les différents algorithmes mis au point pour trouver l'alignement optimal d'une séquence avec une structure.

Parallélisation/Distribution Le pré-calcul des distributions est une phase coûteuse en calcul. Or cette phase consiste en de très nombreux calculs d'alignements qui peuvent être considérés chacun comme une tâche élémentaire. Utilisant le programme rendu modulaire et les approches classiques de distribution maître-esclaves, nous avons obtenu une parallélisation quasi parfaite de cette phase de calcul des distributions. Ainsi le calcul des distributions de scores des repliements pour la totalité base de donnée de structures modèles, qui nécessiterait plus de 40 jours sur un processeur, est effectué en 3 jours sur un cluster de 12 ordinateurs mono-processeur. Ce calcul est d'une importance sensible pour les biologistes qui devraient l'effectuer chaque fois qu'ils changent leur fonction de calcul de score par exemple. Or avant notre parallélisation, le calcul entier n'avait jamais été réalisé.

La phase d'évaluation d'une requête peut elle même tirer partie d'une parallélisation puisque le premier filtre est constitué de quelque centaines d'alignements qui peuvent être mis en paquets et distribués. Les résultats sont récupérés par le processus maître qui fait le traitement statistique et distribue ensuite le calcul de la dizaine d'alignements du deuxième filtre.

Il est important de prendre en compte que pour une requête donnée, le nombre des tâches à réaliser et la complexité de chacune de ces tâches n'est pas prévisible. D'où le choix que nous avons fait d'utiliser une approche maître esclaves avec équilibrage dynamique de charge.

Conception et implantation d'un nouvel algorithme Durant un séjour d'un mois à l'Université de Sofia(Bulgarie), j'ai collaboré avec Nicola Yanev (Sofia), Philippe Veber et Rumen Andonov (IRISA) à la conception d'un nouvel algorithme pour résoudre le PTP. Cet algorithme utilise l'approche de séparation de coûts dans un graphe [87] et la relaxation Lagrangienne. Nous avons obtenu un algorithme et une implantation qui s'avère être plus efficaces que ceux décrits par Stephan Balev dans [67]. Au détour de la conception de ce nouvel algorithme, nous avons proposé une amélioration de la phase préliminaire à l'étude statistique (point 4 de la méthode). Il est en effet nécessaire d'effectuer un traitement statistique afin de déterminer si l'alignement obtenu est plausible. Ce traitement repose sur le calcul préalable d'un grand nombre de repliements pour chaque structure modèle contenue dans la base de données afin de constituer une distribution de scores. Ainsi pour FROST cela représente environ

1 200 000 calculs d'alignements. Nous avons observé grâce aux très nombreux calculs effectués que les valeurs approchées obtenues dans les phases initiales des algorithmes considérés sont pertinentes (suffisamment précises) pour constituer la base de données qui sera utilisée dans le traitement statistique. Or ces valeurs approchées sont obtenues en temps polynomial. Cela réduit considérablement le temps nécessaire à la constitution des distributions. C'est actuellement la version par défaut utilisée dans FROST.

Extensions possibles Comme je l'ai précisé ci-dessus, actuellement FROST effectue un alignement entre des séquences de tailles comparables. C'est ce qui est appelé un alignement global. Or la réalité biologique est qu'une protéïne peut être composée de plusieurs modules dont la structure 3D de chacun possède une pertinence. Une structure 3D est donc divisée en sous-structures appelées domaines. Il est donc intéressant de réaliser un alignement *semi-global* : aligner un domaine modèle avec une partie de la séquence requête. Des travaux de recherche dans cette direction sont en cours à l'Irisa[74]. D'autre part, le temps global de la phase de distribution ou de l'évaluation d'une requête est borné inférieurement par le temps maximum d'une tâche élémentaire. Or nous avons vu que ce temps pouvait être conséquent. Il serait donc intéressant de pouvoir prédire le temps nécessaire pour résoudre un PTP donné et ainsi de pouvoir utiliser pour les alignements que l'on prévoit être difficiles les algorithmes parallèles conçus [65] pour résoudre le PTP.

Cette activité a donné lieu à une communication au congrès Journées Ouvertes de Biologie, Informatique et Mathématiques en 2002 [24], à une publication à la conférence HiComb'05 à Denver [9] (reprise en tant qu'**Article 3** de la partie **Recueil de publications**) et à l'atelier NP-Par'05 [25] en 2005. Les résultats concernant notre nouvel algorithme ont été publiés en 2005 dans les conférences : [10] (reprise en tant qu'**Article 4**) et [29].

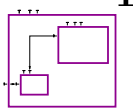
Autres travaux : structure secondaire ARN

Avec une équipe du laboratoire informatique de l'université de La Laguna (Espagne) nous nous sommes intéressés au problème de l'obtention d'une parallélisation efficace (voir optimale) d'un algorithme de recherche de structure secondaire des molécules d'ARN. L'algorithme séquentiel publié par Zucker [96] met en oeuvre les principes de la programmation dynamique. Nous avons obtenu une formule analytique permettant de calculer statiquement la taille optimale des tuiles. Nous avons ainsi put dériver et implanter un algorithme parallèle efficace. Ce travail, débuté en 2001 lors d'une visite de Francesco Almeida et Casiano Rodriguez à Valenciennes, a donné lieu à deux communications [5, 6] et à une publication dans un journal [1]. Dans le cadre de cette collaboration, j'ai été invité une semaine à l'université de La Laguna.

Perspectives

En septembre 2005 Rumen Andonov a obtenu sa mutation pour l'université de Rennes, je continue à travailler en lien avec lui sur cette thématique, en particulier sur les problèmes liés au repliement de protéïnes. Il est cependant clair qu'en ce qui me concerne, cette activité est un peu en sommeil, essentiellement à cause du nouvel isolement dans lequel je me trouve sur cette thématique au LAMIH.

1.3 Axe Spécifications et modularité, intégré à SID



En 1997, en réponse à une sollicitation de Georges Mariano, chargé de recherche à l'INRETS / ESTAS, j'ai initié le thème de recherche *spécifications et modularité* au LAMIH. Cet axe est construit autour d'une collaboration forte avec l'équipe ESTAS. Une des cibles visées était la conception d'une plate-forme ouverte de développement et de recherche pour la méthode B. Cette collaboration s'appuie sur des stages d'étudiants de maîtrise de l'ISTV, du DEA de Paris VII et sur des thèses cofinancées par l'INRETS et la région Nord-pas-de-Calais.

Ce travail a été possible grâce à la confiance que m'a accordée Arnaud Freville en acceptant que j'encadre des travaux de thèses forts éloignés de sa thématique de recherche (la recherche opérationnelle). Pour ces encadrements, je me suis appuyé sur les compétences acquises durant mes propres travaux de thèse (problématiques de la définition de langages de programmation).

1.3.1 Contexte initial

La méthode B [34] a été définie par Jean-Raymond Abrial à partir de la fin des années 1980 [33] dans la continuité des travaux de définition de la méthode Z [37]. Ce n'est pas le lieu ici de faire une présentation complète de la méthode B, je renvoie le lecteur intéressé au livre d'Abrial [34] et aux premiers chapitres des thèses de Dorian Petit [54] et Samuel Colin [39]. Je n'évoquerai que quelques éléments. Dans la méthode B, le langage B est conçu pour permettre de spécifier un logiciel et ses propriétés à un haut niveau d'abstraction, puis de raffiner ces spécifications pour enfin aboutir à un programme. Conçue pour produire du code à partir de spécification formelle par un processus de raffinements, de preuves assistées et de génération de code, la méthode B a trouvé rapidement un domaine d'applications privilégié : les logiciels critiques, et parmi ceux-ci particulièrement les logiciels pour les transports guidés. Basée sur une théorie assez simple à comprendre (la logique ensembliste, logique des prédicats du premier ordre) elle est accessible à des développeurs qui n'ont pas obligatoirement un bagage de théoriciens des logiques d'ordre supérieur. Dès le départ, et à la différence de Z, l'objectif était de fournir une méthode outillée avec une assistance à la preuve. Ce qui fut réalisé avec deux outils commerciaux : l'atelierB [38] de la société ClearSy et B-Toolkit de BCore [35]. La caractéristique *outils propriétaires et fermés* de ces deux outils logiciels ne permettait pas d'expérimenter facilement de nouvelles approches et idées pour faire évoluer la méthode. En particulier, si le BBook [34] spécifie complètement le passage de la spécification formelle à un langage appelé B0, rien n'y est dit sur la compilation du B0 en C ou en ADA, les langages cibles initiaux de la méthode B. Cette observation plus le fait que la compréhension de la modularité du langage B n'était pas aisée constituent les motivations initiales de nos travaux. Nous avions comme objectifs,

- au plan logiciel, de mettre à disposition de la communauté une plate-forme d'expérimentations et de développement ;
- d'implanter un processus de génération de code qui ne soit pas un tunnel opaque mais repose sur des résultats connus et validés en théorie de la compilation ;
- d'apporter un éclairage au mécanisme de modularité de la méthode.

Neuf ans après nos premiers travaux, où en sommes nous ?

1.3.2 Bilan d'étape

Nous pouvons mettre à disposition de la communauté scientifique une plate-forme ouverte, maintenue dynamiquement disponible en ligne sur le site :
`http://savannah.nongnu.org/projects/brillant/`. Ce projet est ouvert à d'autres acteurs, actuellement le LIFL(Lille), l'HEUDIASYC(Compiègne) et l'IRIT(Toulouse) y participent.

Bien que la version disponible actuellement n'offre pas une chaîne intégrée, complète, de développement de la méthode B, elle fournit un support d'expérimentations pour toutes les phases de cette chaîne. L'ensemble de ce travail est décrit dans un article proposé au journal **Software and Systems Modeling** repris dans l'**Article 5** de la partie **Recueil de publications**[6]. D'autres projets que le notre ont éclos ces dernières années en vue de mettre à disposition des plates-formes ouvertes de développement pour des méthodes formelles. Dans le paragraphe de comparaison page SoSyM06-21 les projets `Rodin` [57] (pour $B\#$), `Overture` [52] (pour VDM++) et les Community Z Tools [40] (`CZT`) notamment sont mis en perspectives.

D'un point de vue pratique, les travaux ont été réalisés par deux doctorants (cf. le paragraphe 1.3.3) aidés par des stagiaires de master 1 de l'ISTV ou de l'école d'ingénieur IIE d'Evry, et un post-doctorant (Raphaël Marcano) tous accueillis par Georges Mariano au sein de l'équipe ESTAS à Villeneuve d'Ascq. Ma contribution à ce travail réside dans l'orientation et le guidage des travaux de ces chercheurs, les réalisations effectives leur appartenant.

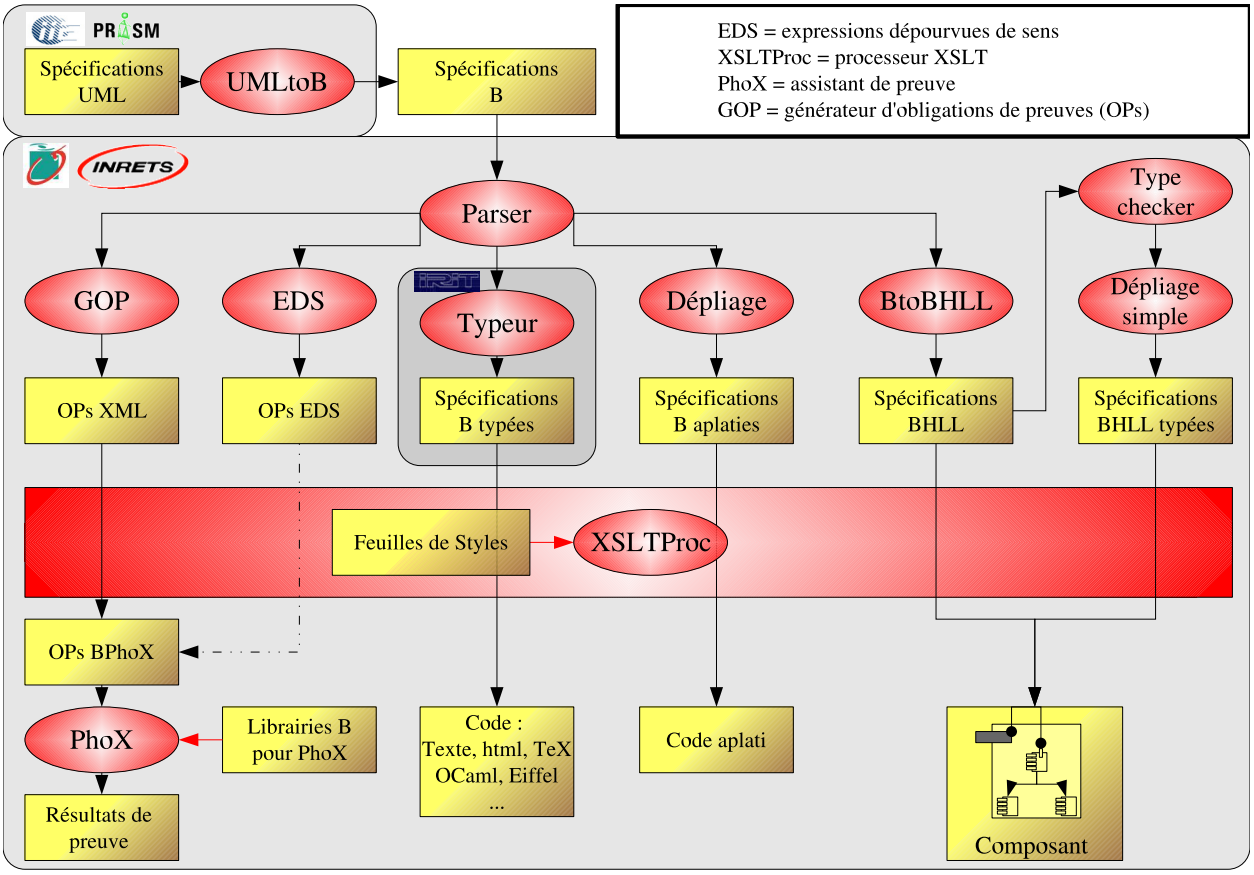


FIG. 1.8 – Architecture de la plate-forme brillant

La figure 1.8 décrit l'architecture de la plate-forme. Nous avons d'abord réalisé les premiers composants : analyseur syntaxique et outils de manipulation de l'arbre de syntaxe. Ces composants furent rapidement utilisés par l'équipe de Jean-Paul Bodeveix de l'IRIT pour implanter leur proposition de typage du langage B. Au fil des travaux, et souvent avec l'établissement de coopération avec d'autres équipes, la plate-forme a été enrichie de différents composants implantant suivant les cas

des outils nécessaires à l'utilisation de la méthode B : *générateur d'obligations de preuves,*

[6]Il s'agit d'une version étendue et améliorée de la publication faite à la conférence **Software Engineering and Formal Methods**(SEFM'05) à Koblenz [12].

prouveurs, expressions dépourvues de sens. Je pointe ici particulièrement la conception du prouveur **bphox** qui nous permet de prouver les obligations de preuves, il a été écrit en collaboration avec Christophe Raffalli de l'Université de Savoie, concepteur de PHOX [56] ;

des approches alternatives n'étendant pas la méthode B : implantation de l'algorithme de mise à plat de Salimeh Behnia [36], utilisation des modules *à la ML* pour implanter la modularité du langage ;

des études d'extension de la méthode : implantation du B typé par Mamoun Filali et Jean-Paul Bodeveix de l'IRIT, passage de UML à B, y compris avec la traduction d'expressions OCL par Raphaël Marcano en collaboration avec le laboratoire PRISM, plugout de modélisation de systèmes d'informations par Régine Laleau et Amel Mammar [46] du CEDRIC/CNAM.

Le choix d'XML comme base des formats d'échanges nous permet d'insérer facilement des éléments développés dans différents langages.

1.3.3 Principaux artisans

Les deux doctorants/docteurs encadrés sont les chevilles ouvrières de ce projet. Leurs travaux sont décrits dans les paragraphes ci-dessous.

Le premier artisan de ce travail est Dorian Petit qui a réalisé au cours de son stage de Maîtrise Informatique les premières briques nécessaires à cette plate-forme. Il a ensuite suivit le DEA Programmation : Sémantiques, Preuves et Langages de l'Université de Paris 7 en 1999. Son stage de DEA portait sur la spécification des caractéristiques du langage B par des grammaires attribuées [53].

Le deuxième artisan en est Samuel Colin qui, lui, a conçu et implanté *le générateur d'obligations de preuves* pour une spécification B au cours de son stage de maîtrise puis s'est intéressé principalement à l'introduction de caractéristiques temporelles dans les spécifications B. Samuel Colin a suivi le même parcours que Dorian Petit, un stage de maîtrise dans l'équipe ESTAS, puis le DEA de Paris 7 : Programmation : Sémantiques, Preuves et Langages, son stage portant sur la logique modale temporelle nommée : *Calcul des Durées.*

Modularité et contrats pour et avec B

Au cours de sa thèse, Dorian Petit a étudié la modularité du langage en utilisant les modules à la *Harper-Lillibridge-Leroy*[49, 43] des langages ML. Nous avons ainsi montré que foncteurs, structures et signatures de modules permettent de structurer la modularité de B [15, 21]. Plus précisément, Dorian a exprimé les règles de visibilité entre machines B à l'aide des catégories syntaxiques, ce qui a permis d'unifier et simplifier leur présentation. Puis les relations de dépendances entre machines B sont exprimées à l'aide des modules ce qui lui a permis d'implanter ce langage de module pour B. L'apport de cette implantation est de faciliter l'expression et de permettre la validation des règles de dépendances. Chaque module B-HLL est équipé de quatre signatures, chacune correspondant à un des modes de visibilité spécifiés par Abrial. Il a donc obtenu un processus de génération qui, vu en tant que boîte noire, se comporte comme les processus de génération de code des autres outils. Cependant, comme il s'appuie sur une description dans un modèle bien fondé : le langage des modules HLL, le respect des règles de visibilité notamment ne nécessite pas de processus ad-hoc plus ou moins opaque.

Développant une autre intuition, nous avons montré qu'il est possible de conserver sous forme d'assertions jusque dans le code produit des propriétés logiques présentes dans la spécification du logiciel. Nous obtenons ainsi par une démarche de développement formel du code logiciel qui s'intègre naturellement dans une démarche de développement par contrats telle qu'initiée par Bertrand Meyer [51].

Ces travaux ont donné lieu à publication dans une revue en 2004 :*Journal of Integrated Design and Process* [3] et des conférences internationales en

- 2002 : *Journées Francophones des Langages Applicatifs*[21] et *Component Based Software Development* [22]
- 2003 *Symposium on Formal Methods for Railway Operations and Control Systems* [13] et *Integrated Design and Process Technology* [14]
- 2004 *Forum on Specification & Design Language*[15]
- 2005 *3rd International Conference on Software Engineering and Formal Methods - SEFM 2005*[12]

et une conférence nationale en 2003 : *Approches Formelles dans l'Assistance au Développement de Logiciels* [23]

Temporalité et B

La préoccupation de la prise en compte de contraintes temps réel est essentielle pour assurer la sûreté des systèmes de contrôle où les contraintes de temps de réaction, face à des situations critiques, sont fortes. Il est donc intéressant de pouvoir spécifier ces contraintes assez tôt dans les modèles, éventuellement de raisonner sur elles, et de s'assurer que les implémentations (soit en tant que modèles, soit en tant que réalisations) les satisfont.

C'est le point de départ du travail de thèse de Samuel Colin qui a montré qu'il est possible d'intégrer proprement dans une spécification B une logique temporelle continue d'intervalles avec répétition : *le calcul des durées* (DC^*). L'objectif est là encore d'obtenir une méthode outillée de développement logiciel permettant de certifier le développement du point de vue des propriétés fonctionnelles *et* des propriétés temporelles prenant en compte des contraintes temps réel. Le travail de Samuel constitue une étape de fondation théorique vers cet objectif.

Une première partie du travail a donc consisté à implanter le calcul des durées (DC) dans un assistant de preuves utilisant la logique typée d'ordre supérieur. L'assistant choisi est COQ pour la puissance qu'il offre et le dynamisme de la communauté qui le développe et l'utilise. L'implantation réalisée de DC en COQ s'inspire de travaux similaires [44] utilisant Isabelle/HOL. Il y a eu quelques choix cruciaux à faire pour autoriser l'utilisation "gratuite" des bibliothèques de COQ. Je renvoie au chapitre 5 de la thèse de Samuel Colin pour les discussions de présentation et justification des choix qu'il a faits.

Ce travail a donné lieu à publications dans des conférences internationales en
- 2003 *4th International Workshop on the Implementation of Logics* [20]
- 2004 *Journées Francophones des Langages Applicatifs* [26], et *International Colloquium on Theoretical Aspects of Computing (ICTAC 2004)* [11].

L'autre partie du travail a consisté à étendre B pour permettre d'exprimer des contraintes temporelles et à donner une sémantique temporelle aux substitutions du langage B. Samuel a montré qu'une légère extension syntaxique de B avec des expressions de contraintes temps réel : `delay ; await ; TIMING` est possible. De plus, il a donné au langage étendu une sémantique précise à l'aide de la logique WDC^* [45], variante du calcul des durées. Il a montré que cette sémantique préserve la sémantique classique des substitutions de B. Ses résultats permettent notamment d'envisager de prouver des propriétés de concurrences spécifiées en B+DC. Pour cela il faudra engendrer les obligations de preuve temporelle d'une machine B+DC et les prouver avec l'aide de l'implantation de DC en COQ déjà réalisée.

1.3.4 Perspectives

Les travaux autour de la méthode B et la plate-forme BRILLANT continuent. Il y a à faire un travail d'intégration logicielle pour faciliter l'utilisation de la plate-forme. De même implanter les propositions émises dans la thèse de Samuel Colin pour B+DC reste à accomplir maintenant que ses résultats montrent qu'une telle extension est fondée théoriquement. Il s'agit plus de travaux d'ingénierie que de travaux de recherche, mais au stade atteint, c'est devenu un seuil nécessaire à

franchir pour pouvoir continuer sans perdre le bénéfice des travaux déjà réalisés. Par ailleurs, la maturation du B événementiel [50] et l'implantation en cours de la plate-forme ouverte Rodin [57] ouvrent de nouvelles perspectives pour les spécifications de composants temps-réels sûrs.

Le thème *spécifications et modularités* est maintenant intégré au groupe **S**ystèmes d'**I**nformations **D**istribués. Nous travaillons à unifier nos thématiques de recherche au sein du groupe. Il s'agit de trouver des synergies avec les thématiques de recherche de Sylvain Lecomte (professeur au LAMIH-ROI et animateur de SID). Ses axes de recherche s'articulent autour du déploiement de composants logiciels dans le cadre de l'informatique ubiquitaire et de l'adaptation de services aux évolutions du contexte d'évaluation. Une autre thématique de l'équipe avec laquelle nous voulons converger, portée par Marie Thilliez et Thierry Delot (tous les deux Maîtres de Conférences), s'intéresse aux composants logiciels de positionnement géographique et aux requêtes dans des bases de données dépendantes du positionnement.

De plus, nos thématiques doivent nécessairement se développer dans le cadre des orientations prioritaires du LAMIH et de la région Nord-Pas-de-Calais. En particulier dans le contexte de la mise en place du pôle de compétitivité à vocation mondiale iTrans [102], centré sur les transports. Nous confluons donc vers les thèmes suivants qui sont le cadre de travail pour Hung Ledang, chercheur vietnamien, qui vient d'intégrer l'équipe (octobre 2006) avec un contrat de post-doctorat financé dans le cadre du thème 6 du Pôle sciences et technologies pour la sécurité dans les transports(ST2), thème centré sur les exigences de sécurité, et aussi du master deux recherche d'Estelle His que nous co-encadrons avec Sylvain Lecomte.

Si aujourd'hui un certain nombre de méthodes existent pour produire des composants logiciels dignes de confiance, deux points ouverts de recherche peuvent être ainsi formulés :

- Comment spécifier des propriétés non-fonctionnelles (propriétés de concurrence, de temps de réponse, ...) pour pouvoir valider des composants vérifiant ces propriétés. Parmi les travaux en cours on peut citer :
  - Les travaux autour du modèle de composants Fractal [42] qui est aujourd'hui le modèle de composants le plus proche de ce que peut-être une définition théorique de composants. En particulier, les travaux autour du projet THINK[58] de composants pour logiciels embarqués.
  - le projet **Qua**[55] : *(Quality of Service Aware Component Architecture)* qui s'intéresse au développement de composants intégrants des contraintes de qualité de service pour développer des applications temps réel avec des plates-formes à composants.
- *Comment produire un assemblage digne de confiance ?* Les logiciels utilisés dans les systèmes de transport (ferroviaire, transports guidés, véhicules automatiques, ...) ne peuvent pas se contenter de la fiabilité de chaque composant. C'est l'assemblage global qui doit vérifier des propriétés de sûreté et de respect des exigences de sécurités. En faisant l'hypothèse que les composants utilisés sont fiables, quelles informations doivent être disponibles et quels algorithmes de composition peuvent être utilisés pour permettre de garantir la fiabilité de l'assemblage ? Dans les domaines d'application des transports, nécessairement les contraintes temporelles sont à prendre en compte. Il s'agit donc de trouver comment modéliser la composition des propriétés fonctionnelles et temporelles d'une manière qui permette de raisonner sur les assemblages produits (faire des preuves, générer des scénarios de test, de la validation par model-checking ...).

S'intéresser à ces points ouverts a plusieurs intérêts pour nous.

- Tout d'abord cela nous permet de mobiliser les diverses compétences de l'équipe autour d'un projet de spécification d'un assemblage de composants impliqué dans le positionnement géographique de véhicules. C'est un assemblage qui doit être fiable, qui met en jeu des contraintes temps réel et qui doit être intégré dans un véhicule et donc avec un contexte d'évaluation évolutif.
- D'autre part, alors que nous réfléchissions à cette convergence, nous avons eu l'opportunité de

monter un projet sur cette thématique dans le cadre d'appels d'offres de sécurité informatique de l'ANR (SETIN'06). Le porteur de ce projet est l'équipe DEDALE du LORIA, les autres partenaires sont l'équipe TFC du LIFC, LAS du LACL et MAIA du LORIA. Suite à des sollicitations de la région Lorraine il s'agit en particulier d'étudier des processus de validation de logiciels mis en œuvre dans les véhicules automatiques (idéalement) non guidés appelés Cycab. Dans ce projet TACOS *Trustworthy Assembling of Components : frOm Requirements to Specifications* (évolution de ACROBATYC cf. le paragraphe 3.2), nous nous intéresserons aux approches utilisables pour exprimer un assemblage de composants et aux méthodes à mettre en œuvre pour permettre de fournir une information sur le niveau de fiabilité obtenu.

Chapitre 2

Activités d'enseignement

L'activité *scientifique* d'un enseignant-chercheur comporte une partie *enseignement* ...
Mon recrutement à l'UVHC s'inscrivait dans le contexte de la création d'une Licence (puis d'une maîtrise) d'informatique à l'ISTV. J'effectue mes enseignements essentiellement dans la filière informatique de l'institut[1]. Recruté en 1992 en même temps que Sylvain Piechowiak et Hafid Bourzoufi[2], nous avons, avec les collègues déjà présents, mis en place un second cycle d'informatique qui a montré sa pertinence par rapport au bassin de recrutement de l'UVHC. En 2005 plus de 70 étudiants étaient inscrits au niveau Bac + 4 dans cette filière. Ce second cycle a été poursuivi par la mise en place d'un DESS Technologies Nouvelles des Systèmes d'Informations (TNSI) à l'initiative principale d'Hafid Bourzoufi.

J'ai pris ma part dans ce travail de définition de cursus de formation d'informaticiens. Je décris dans le paragraphe 2.1 mes implications dans la mise en place ou le pilotage d'évolution de cursus, dans le paragraphe 2.2 sont exposés succinctement les enseignements que j'ai portés.

2.1 Implications fortes dans l'évolution des cursus informatiques

C'est une activité passionnante (et prenante) de participer à la spécification et à l'implantation de nouvelles formations destinées à des étudiants de différents niveaux et d'objectifs différents. Avant la création d'un second cycle informatique, l'informatique à l'ISTV n'existait que dans le cadre des autres filières scientifiques de l'Institut : premiers cycles et formations en automatique et mécanique essentiellement.

Formations créées avec une implication du département informatique

J'ai participé à la définition et la mise en œuvre de

La Licence d'Informatique en 1992, avec la création d'un module d'enseignement

La Maîtrise d'Informatique en 1993, sans prise en charge d'un enseignement en responsabilité.

Le DESS Traitement du signal (électronique) Définition du programme informatique. Prise en charge d'un module d'enseignement l'année de création avec René Mandiau (alors MC en informatique), à la demande des électroniciens.

Le DESS Technologies Nouvelles des Systèmes d'Information en 1996, avec Hafid Bourzoufi qui en était le maître d'œuvre principal et Didier Donsez, création et prise en charge d'un module d'enseignement.

[1] Depuis l'année 2005-2006 j'assure un enseignement d'informatique fondamentale pour les étudiants de deuxième année du département informatique de l'IUT de Valenciennes, antenne de Maubeuge. Ceci pour soulager les collègues informaticiens en postes à l'IUT.

[2] Décédé en janvier 2000

Le DESS CCI en 1998, porté par Christophe Kolski. Création d'un module d'enseignement.

L'IUP GMI en 2002.

Dans la suite de ce paragraphe, je m'attache à présenter les deux expériences d'évolutions de formations dont je fus le coordonnateur. Avant de détailler plus avant, je retiens principalement de cette activité de coordination le grand intérêt que j'ai pris à parvenir à définir collectivement un contenu et des parcours de formation. Se poser, avec les collègues, des questions du type :

- comment articuler différents enseignements ?
- comment ne pas rebuter les étudiants tout en maintenant une exigence de qualité ?
- quelle progression adopter ?
- quel enseignement est indispensable à ce niveau de formation ?
- comment prendre en compte les étudiants arrivant d'IUT ?
- quel contenu demander aux disciplines voisines (mathématiques, électronique, automatique, ...) pour à la fois permettre des changements d'orientation (les fameuses passerelles), compléter utilement une formation d'informaticien, ouvrir des horizons ...

est une activité stimulante, à composants scientifiques et humains, enrichissante par la confrontation des idées. Je ne voudrais cependant pas en faire une description trop idyllique. Ces deux expériences très prenantes m'ont permis de découvrir bien des facettes cachées du monde universitaire. Si des aspects conflictuels ressortent de ma description ci-dessous, c'est, à la relecture, assez naturel. Les intérêts impliqués ne sont pas nécessairement convergeants. Cependant je pense être arrivé dans ces deux coordinations à des résultats satisfaisants du point de vue de la formation des étudiants tout en respectant les personnes impliquées tout au long du processus.

Transformation du DEUG A en DEUG MIAS En 1994/1995 Arnaud Freville, alors Directeur de l'ISTV, m'a confié la mission d'animer la réflexion des collègues intervenant en DEUG pour réaliser cette transformation. *Ce travail m'a permis de comprendre les enjeux humains pluridisciplinaires impliqués* dans une formation scientifique généraliste. C'est une expérience qui laisse des traces, à cette époque j'étais jeune maître de conférences naïf et plein d'illusions. L'informatique trouvait enfin sa place dans cette nouvelle définition nationale du DEUG. Ce qui me semblait alors une évolution cohérente de l'offre de formation de l'Université :

- cohérente avec la demande des jeunes étudiants,
- cohérente avec les besoins de formations,
- cohérente avec la place prise par la science informatique dans la société,

est vite apparue comme la remise en cause de beaucoup de situations.

Ce fut une première expérience riche en leçons :

- la mise en place de ce type de réforme dépend très fortement des disciplines scientifiques d'appartenance des personnes aux instances de pouvoirs.
- La partie difficile du travail fut d'obtenir un consensus d'acceptation.
- L'existence d'un canevas national constitue un garde fou(s) appréciable. Le fait que les grandes lignes d'un programme de Deug MIAS soient à respecter avec une dénomination nationale constitue une grande aide aux débats.
- Les arbitrages entre conflits d'intérêts sont aussi à faire entre collègues informaticiens. J'ai en particulier mesuré que le couple Mathématiques-Informatique de l'intitulé, et donc du contenu, pose problèmes pour quelques collègues plus sensibles à une informatique plus technologique que fondamentale (je précise que c'est sans connotation péjorative pour moi, la caractéristique technologique attirant par ailleurs un bon nombre d'étudiants).

Transformation en IUP GMI de la filière Informatique de l'ISTV et passage au LMD En 2000 le Département Informatique de l'UVHC, m'a confié la responsabilité d'animer la réflexion pour transformer la formation en informatique de l'UVHC en IUP Génie Mathématiques et Informatique. De 2000 à 2002 j'ai donc piloté et porté (ce qui inclus notamment la défense devant les experts du ministère et devant la commission nationale des IUP ; présen-

tation et défense du projet devant les différentes instances concernées de l'UVHC) le projet de transformation des licence et maîtrise d'informatique de l'UVHC en IUP GMI.

Nous avions déjà des relations fortes avec les entreprises, essentiellement par le truchement des stages de fin d'année que nous avons inclus dès l'origine (en 1993) dans le cursus de la maîtrise informatique, et par le DESS TNSI. Les objectifs de cette mutation étaient d'approfondir ces relations, d'être plus attractifs pour les étudiants, en particulier ceux provenant d'un IUT et de faire bénéficier les étudiants d'un titre reconnu et valorisant.

Ce travail de refonte complète, *réalisé avec les collègues* a été compliqué par la prise en compte des nouvelles contraintes issues du passage au LMD de l'UVHC en tête de cordée (première université française à effectuer ce passage) passage décidé courant 2002 et effectué en septembre 2002. Nous avons obtenu cette habilitation pour ouvrir en IUP en septembre 2002, ce de façon concomitante avec le passage au LMD !

2.2 Description succincte des enseignements mis en place

Je considère enrichissant, pour l'enseignant-chercheur que je suis et pour les étudiants, d'avoir des interventions dans les trois cycles. La répartition de mes interventions entre les cycles a varié au cours des années mais j'ai essayé de maintenir ce principe qui me permet d'avoir une vision globale de nos formations.

Je ne décris que les enseignements dont j'ai eu la responsabilité entière (Cours TD TP) et dont j'assurai le Cours en totalité ou partie. Les intitulés et les contenus ont évolué, dans un souci de clarté, pour chaque enseignement, je mentionne un intitulé, la formation hôte, l'année de création, la durée de prise en charge, l'effectif étudiant et un contenu succinct.

Parmi ces enseignements, celui qui m'a apporté le plus et à travers lequel je pense avoir le plus contribué à la formation des étudiants en informatique de l'UVHC est le cours actuellement nommé *Méthodologies de programmation*. Je m'attache à faire évoluer ce cours au fil des années, il me permet d'ouvrir les horizons des étudiants en présentant différents styles de programmation par le moyen d'un langage qu'ils découvrent. Les difficultés principales que j'y rencontre systématiquement sont relatives à la difficulté du processus d'abstraction. Il est plus facile de manipuler des indices de tableau que de décomposer un problème et s'attacher à *ce que l'on veut obtenir* plus qu'au *comment l'obtenir*.

Méthodologies de programmation(créé) Licence d'Informatique (L3 maintenant) ;1992 ; responsable depuis 1992 ; 40 à 80 étudiants par an. Entre 40 et 50 heures/étudiant. S'adresse à des étudiants ayant une connaissance de base de la programmation impérative classique. Introduction de la programmation fonctionnelle, du polymorphisme, insistance sur les *bonnes pratiques* : utilisation des exceptions, de la modularité ..., mise en perspective des approches fonctionnelles et impératives. Langage support CAML-LIGHT puis Objective Caml.

Préparation aux concours des écoles d'ingénieurs(créé) Deug 2 MIAS,1996, pendant 4 ans, environ 12 heures/étudiant ; 5 à 10 étudiants par an. Objectif, préparation aux épreuves d'informatique des concours ouverts aux étudiants de DEUG deuxième année.

Bases de la compilation (créé) Deug 2 MIAS,1996 à 2005, 25 heures/étudiant. autour de 100 étudiants/an. Raisonnement par induction, langages rationnels et automates finis.

Programmation orienté objet(créé) DESS TNSI ; 1996, Assuré pendant deux périodes de 2 ans ; 25 étudiants/an.

Logique (créé) Deug 2 MIAS,1998, assuré pendant 2 ans ; 25 heures/étudiant. autour de 80 étudiants/an. Logique des prédicats, réalisation d'un vérificateur de tautologies.

Langages et grammaires Licence(3) Informatique ; 1998 puis à partir de 2005.

Programmation Fonctionnelle(créé) DESS CCI, 1998, 6 ans, 20 étudiants, $\approx$ 15 heures/étudiant. Introduction de la programmation fonctionnelle.

Sûreté de développement(créé) Master 1, ex Maîtrise ;2000 ; 30 heures/étudiants ; présentation de méthodes formelles de développements, Z ; B ; ... Ce cours est actuellement assuré par Georges Mariano.

Spécification formelle(créé) Master 2 Recherche ; depuis la co-habilitation en Informatique et Automatique du DEA (2001) ; 6 heures/étudiant.

Programmation Système Licence(3) d'Informatique ; 2000 ; 2 ans ; 50 étudiants. Suite au décès d'Hafid Bourzoufi, j'ai assuré une partie de cet enseignement avec Didier Donsez.

Introduction à la programmation fonctionnelle et à la programmation objet(créé) 2002 à 2004 ; Licence 2 ; environ 30 heures/étudiants ; 80 étudiants / an.

Induction, langages et automates(créé) IUT Informatique ;2005 à maintenant ; 12 h cours étudiants. Deuxième année au département informatique de l'IUT antenne de Maubeuge.

Encadrements

Outre les deux thèses mentionnées, j'ai encadré les DEA ou Master2 Recherche suivants.

Les étudiants du DEA Sémantique Preuve et Programmes (ou Programmation) de Paris VI et Paris VII dont j'ai encadré le stage sont Dorian PETIT (1999) David DUBOIS (2000) et Samuel COLIN (2001). Parmi les étudiants issus de la maîtrise de Valenciennes que j'ai recommandés à ce DEA, ce sont ceux qui ont souhaité faire un stage dans le domaine spécification et modularité sous ma responsabilité. Ils ont obtenu leur DEA de manière honorable (mention B ou AB) dans un contexte relevé. Deux ont poursuivi en thèse sous ma responsabilité, David Dubois a fait le choix d'un salaire plus élevé dans une PME locale après une année de thèse financée aussi par l'INRETS et la région.

Les étudiants du DEA Automatique Informatique des Systèmes Industriels et Humains(AISIH) de l'UVHC que j'ai encadrés avant 2004 (Ingrid JACQUEMIN et Nicolas CORTOT) ont travaillé dans la thématique optimisation combinatoire pour la bio-informatique. Bien que ces deux étudiants aient obtenu leur diplôme à un rang respectable, nous n'avons pu trouver pour eux un financement de thèse à Valenciennes. Nous avons recommandé Ingrid Jacquemin pour une thèse (soutenue en 2005) dans l'équipe Symbiose de l'IRISA. Nicolas Cortot a abandonné la perspective de faire un travail de thèse.

Tous les ans j'encadre de nombreux étudiants des niveaux Licence 3(ex licence), Master 1(ex maîtrise), et Master 2 pro(ex DESS), dans le cadre des modules *Conception et Mise en œuvre* où ils ont à réaliser un projet durant un ou deux semestres. J'assure aussi régulièrement le suivi de stages en entreprise.

Tutorat de moniteur Je suis tuteur d'Arnaud Doniec, moniteur en section 27 depuis 2004 à l'UVHC.

Chapitre 3

Visibilité et Responsabilités collectives

3.1 Responsabilités Collectives

Exergue *«D'accord. Cent candidats tous sur-diplômés pour un poste. Vingt pinpins convoqués, courant pour arriver à l'heure avant d'aller tenter leur chance, toujours à leurs frais évidemment, à l'autre bout de la France. A peine le temps de dire trois mots par tête de pipe. Et tu sais quoi ?*

- *Non, dit Simon, qui avait déjà souvent entendu cette complainte universitaire quand ce n'était pas lui-même qui la psalmodiait.*
- *Deux cent dix kilos de papier à lire pour les rapporteurs de la commission.*
- *Comment tu sais ça, toi ? s'étonna Simon.*
- *J'ai fait peser les colis des candidats à l'accueil courrier. Plus de deux quintaux de paperasses et moins de cinq minutes pour en parler avec chaque gars, c'est dingue.»*

Pierre Christin, Petits crimes contre les humanités[98].

3.1.1 Mandats électifs

La citation en exergue de ce paragraphe est tirée d'un roman policier écrit par un universitaire, ancien directeur d'une école de journalisme (Bordeaux), ancien membre élu du Conseil National des Universités (CNU), très connu pour son activité d'auteur de bandes dessinées. J'ai revécu dans cette citation mon expérience lors de la campagne de recrutement à laquelle j'ai participé en tant que candidat en 1992. Ce livre m'a été offert suite à une présentation radiophonique où il en était dit : *C'est un roman noir qui se situe dans un univers que l'auteur connaît par cœur : le tout petit monde universitaire. Nous sommes donc sur le campus de la fac de Nevers, et un honorable professeur en histoire de l'art est frappé par une crise cardiaque, après avoir été victime d'un vilain chantage qui révèle un lourd passé. Panique sur le campus, émoi du Président, enquête policière, et une seule question : que vont devenir la bibliothèque et la collection d'art du grand érudit ? Si des abréviations telles que CAMIF, ATER, AMN, CNU vous sont familières, vous rirez beaucoup ; [...].*

Conscient de l'importance pour la communauté d'un travail d'administration et d'animation, j'ai sollicité et obtenu les mandats électifs décrits ci-après. Notre travail d'universitaire comporte une grande part d'évaluation, après tout, c'est notre métier de noter. La perspective est cependant très largement différente quand il s'agit non pas d'évaluer le travail d'étudiants mais bien des personnes : collègues ou candidats futurs collègues. J'ai bien écrit *évaluer des personnes* ce qui inclus l'évaluation du travail mais est bien plus large.

C'est ce travail que j'ai rencontré dans les différentes instances collectives (heureusement) auxquelles j'ai souhaité participer. Le livre de Pierre Christin s'ouvre sur une séance d'audition de CSE (Commission de Spécialistes) où sont croqués (c'est un roman) de façon acerbe les excès de localisme et de népotisme. Heureusement je n'ai pas rencontré de situation aussi extrême. Il reste

que c'est une responsabilité humaine vis-à-vis de l'individu concerné et de la collectivité universitaire, responsabilité parfois non exempte d'ambiguïté que celle d'évaluateur-cooptant. Sous un mode parfois caricatural et avec humour, dans *Petits crimes contre les humanités* Pierre Christin dépeint quelques uns des travers qui nous guettent.

Membre de la Commission de Spécialistes de l'Etablissement(CSE) section 27(Informatique) de l'UVHC depuis 1998 (et de son bureau de 1998 à 2001 et 2004 à 2007), j'ai participé aux décisions de recrutements de collègues sur des postes de maître de conférences et d'Attaché Temporaire d'Enseignement et de Recherche(ATER). C'est un des lieux où j'ai appris à mieux connaître mes collègues enseignant-chercheurs en informatique à l'UVHC. Lieu de décisions importantes pour les équipes : c'est un(e) collègue enseignant et chercheur que nous recrutons, cela occasionne des débats et des échanges parfois vifs, des négociations (pas toujours ouvertes), il ne faut pas oublier que c'est aussi une décision essentielle pour les candidat(e)s. Comme tout lieu de pouvoir, il gagne à être lisible et à éviter l'opacité, une publication de nos décisions est un élément de contrepoids aux tentations de localisme[1]. Quelle politique voulons nous privilégier : une politique recherche en renforçant uniquement les axes déjà présents ? Une couverture des enseignements à assurer dans les formations dont la responsabilité nous incombe ? Une valorisation du travail effectué par les docteurs que nous avons formé, en les recrutant, après tout ils n'ont pas démérité ? Un apport d'idées neuves par le recrutement de candidats extérieurs ? Quels critères utiliser pour départager les candidats : le nombre de revues (les pratiques des sous-disciplines n'étant pas identiques un critère de reconnaissance par la communauté des chercheurs du domaine est plus appropriée mais assez subjective à définir), la durée de thèse, l'implication en enseignement, la personnalité, la volonté manifestée de s'insérer régionalement ?

Ce sont des questions importantes, je pense qu'il nous faut éviter les excès et savoir garder un équilibre entre ces différentes approches. Le travail de membre d'une CSE est un travail qui a des aspects et des impacts scientifiques, mais c'est aussi très largement un travail de relations humaines avec des impacts durables sur ces relations.

Membre élu de la section 27 (informatique) du CNU depuis 2003. C'est une instance importante pour notre communauté universitaire. Composée de 2/3 d'élus et d'1/3 de nommés, ses décisions contribuent à définir les frontières de la discipline informatique. Je citerai ici les propos d'un ancien vice-président du CNU 27, tenus dans un échange privé lors des débats précédant l'élection du bureau de la section 27 du CNU en 2003 :

- Entre la 24ème section que nous étions et la 27ème section que nous sommes, il s'est écoulé trois mois pendant lesquels l'informatique avait disparu. C'était vers 1990. Nos gouvernants avaient repris un vieux fantasme : l'informatique n'est pas une science mais une simple technique. Il a fallu beaucoup d'énergie pour faire rectifier cette énormité. [...]. Les gouvernants ont changé mais gardent globalement la même opinion sur l'informatique, et c'est assez grave. Il faut rester très vigilants. - Parmi les problèmes très complexes que nous devons résoudre au CNU, il y a les problèmes de frontières, principalement avec la 25ème, la 26ème, la 61ème et la 70ème (linguistique). Si nous n'y prenons pas garde, l'informatique deviendra vite à peu près n'importe quoi. [...]

Ce point d'histoire est important, il situe effectivement quels sont les enjeux pour la communauté. À la différence d'un travail de CSE, le travail de qualification du CNU [100] ne consiste pas à recruter un futur collègue, il s'agit de *labelliser* une personne comme étant apte à exercer le métier d'enseignant-chercheur dans ses deux composantes essentielles, et cela du point de vue de la recherche et de l'enseignement en (et de) l'Informatique. J'apprécie beaucoup ce travail, c'est l'occasion d'avoir une vision relativement exhaustive de la recherche actuelle en France. C'est aussi l'occasion de rencontrer des collègues d'horizons divers, de sous-disciplines diverses, avec lesquels

[1] C'est dans cette optique que je mentionne ici l'opération postes mise en place par nos collègues mathématiciens et ouverte à l'informatique. En tant qu'assesseur de la CSE 27, je renseigne donc le site[103].

il s'avère agréable de travailler. Je relève particulièrement la qualité des relations qui ont pu être établies. Et pourtant il pourrait y avoir des querelles de chapelles ou de critères plus ou moins élitistes.

L'autre partie du travail d'une section du CNU est plus délicate, il s'agit de concours où nous devons classer des collègues pour un nombre de places forcément trop limitées. Que ce soit les demandes de congés pour recherche et/ou reconversion thématique ou les demandes de promotions, le travail consiste à choisir les heureux élus parmi les collègues. Là encore j'apprécie le sérieux et la qualité du travail accompli. Malgré les inévitables imperfections dues à l'examen sur dossiers, il me semble que la préoccupation essentielle partagée par les membres de la section 27 est d'éviter les injustices pérennes.

Membre élu du conseil d'administration de l'UVHC en 2000 et 2001, puis à partir d'avril 2006. Il s'agit ici d'influer sur la vie de l'établissement, c'est essentiellement un lieu où l'information est diffusée, il est donc important d'y être représenté. Les dernières années durant lesquelles la discipline informatique était quasiment absente des instances collectives de l'Université nous ont montré que sans représentant dans les différents conseils, nous étions très fragiles. Nous nous sommes donc organisés pour être présents, ce qui est le cas pour ce nouveau mandat.

3.1.2 Participation à la visibilité de l'informatique à l'UVHC

Co-fondateur du Département Informatique de l'UVHC En 1996, avec Arnaud Fréville (initiateur du projet) et René Mandiau, alors Professeur et Maître de Conférences à l'UVHC, nous avons créé le Département d'Informatique de l'UVHC, j'en ai formalisé les statuts [101] et les ai défendus devant le Conseil d'Administration(CA) de l'UVHC. Les objectifs de ce département de l'Université sont :

- d'offrir une structure de dialogue unique entre les enseignants-chercheurs en informatique et les différentes composantes de l'Université ;
- d'assurer une visibilité de la discipline tant interne, au sein de l'UVHC, qu'externe ;
- offrir quelques services aux enseignants-chercheurs en informatique.

L'informatique est, comme les mathématiques, une discipline dont l'enseignement est présent dans toutes les composantes de l'Université. Il nous a semblé important de donner une existence visible à cette réalité.

Relations internationales de la filière informatique de l'UVHC responsable des échanges SOCRATES/ERASMUS pour cette filière de 1995 à 2003. Avec la mise en place et la gestion d'échanges d'étudiants avec : l'université Anglia de Chemlsford (Angleterre) ; l'université de La Laguna (Espagne) ; l'université de Barcelone (Espagne) ; l'université de Sofia (Bulgarie).

3.2 Rayonnement, collaborations

3.2.1 Lecteur arbitre ou referee

2006 *IEEE International Conference on High Performance Computing* ; *DATE'07 : Design, Automation and Test in Europe*

2003-2005 *European Journal of Operational Research*

2004 *INFORMATION & SOFTWARE TECHNOLOGY* Elsevier.

3.2.2 Membre de jury de thèses

Dorian Petit le 18 décembre 2003, Valenciennes, LAMIH/ROI

Ninh Thuan Truong le 5 Mai 2006, Nancy, LORIA/Dedale

Samuel Colin le 3 octobre 2006, Valenciennes, LAMIH/ROI

Mohammed Al Achhab le 12 décembre 2006, Besançon, LIFC/TFC

3.2.3 Collaborations
axe spécifications et modularité

réseau régional avec l'INRETS/ESTAS, le LIFL, l'École des Mines de Douai, l'école centrale de Lille (LAGIS), dans le cadre de COLORS, MOSAIQUES, AS164

réseaux nationaux dans les cadres :

a) De l'Action Spécifique 164 du CNRS

b) Du Pôle sciences et technologies pour la sécurité dans les transports(ST2) regroupant le CNRS, l'INRETS, l'ONERA.

c) Du groupe B du GDR CNRS ALP pour lequel les doctorants que j'encadre ont à plusieurs reprises présenté nos travaux.

d) Du projet d'ARA ACROBATYC, classé deuxième sur la liste complémentaire des projets ARASSIA par l'ANR en 2005 (partenaires LORIA/DEDALE, LORIA/MAIA ; LIFC/TFC ; LACL ; LAMIH/ROI)

e) Du projet TACOS sélectionné le 8 novembre 2006 dans le cadre de l'appel d'offre SETIN06 de l'ANR, ce projet est une évolution du projet ACROBATYC avec les mêmes partenaires.

f) Du projet GENEVE soumis à l'appel d'offre du PREDIT en 2006 (partenaires INRETS/ESTAS, LIFC/TFC, LAGIS, LIFL, Alstom,Certifer,Heudiasyc,LAMIH/ROI), labellisé i-trans.

axe parallélisation, optimisation combinatoire pour la bio-informatique
nous collaborons au niveau national et international avec

a) des équipes françaises dans le cadre de l'ACI GENOGRID (Le Havre, équipe Symbiose de l'IRISA, équipe MIG de l'INRA de Jouy en Josas, Rouen)

b) L'équipe Proteome Informatic Group du Swiss Institut of Biology dans le cadre d'un PAI entre la Suisse et la France commencé en 2003, Robin Gras accueilli un mois en tant que professeur invité en 2004.

c) Une équipe de l'université de La Laguna (Tenerif Espagne), concrétisée par l'accueil de chercheurs de La Laguna en tant que professeurs invités (Casiano Rodriguez et Francisco Almeida)

d) Nicola YANEV, collègue de l'université de Sofia (Bulgarie) que nous recevons régulièrement en tant que professeur invité.

Séjours dans d'autres laboratoires

Sofia, Bulgarie Un mois en Novembre 2004 pour travailler avec le Professeur Nicola Yanev dans le cadre d'un projet européen PAI-RILA

Université de La Laguna, Espagne Une semaine en janvier 2001

Irisa, Rennes De nombreux séjours courts dans le cadre et la suite de l'ACI GENOGRID.

3.2.4 Organisations

Journées des doctorants 2005 Organisateur des troisièmes journées des doctorants de l'équipe ROI du LAMIH les 8 et 9 juin 2005 à Etréaupont.

Chapitre 4

Conclusions et perspectives

Me voilà arrivé, à tout ou à rien ? Certainement à la fin de cette rétrospective des années déjà passées en tant qu'enseignant-chercheur à l'UVHC. Logiquement, vient le temps de conclure et d'envisager la suite, quelles pistes pour mes prochaines années d'enseignant-chercheur ?

Ces quatorze années s'inscrivent dans un développement collectif. Mon recrutement coïncidait avec la création d'un second cycle informatique et nous étions sept enseignants-chercheurs actifs en recherche en informatique à l'UVHC. Quatorze ans plus tard, nous avons formé quelques cohortes d'étudiants au travers d'une licence-maîtrise devenue IUP Génie Mathématique et Informatique, d'un DESS devenu Master PRO et d'un Master recherche (cohabilité en informatique depuis 2001). Nous sommes, en septembre 2006, plus de vingt enseignants-chercheurs actifs en recherche en informatique au LAMIH. J'ai initié et animé une activité de recherche autour des thématiques spécification, modularité et sûreté de fonctionnement, deux thèses encadrées ont été soutenues, l'un des docteurs est maintenant Maître de Conférences à l'UVHC, l'autre qui vient de soutenir est en post-doctorat au LORIA(Nancy). J'ai aujourd'hui une activité de recherche sur deux thématiques disjointes : *optimisation combinatoire et parallélisme* et *spécification et modularité*, reconnue par la communauté. Je suis convaincu qu'un projet individuel prend du sens s'il s'inscrit dans un projet collectif, et pour cela il faut s'atteler à la vie *de la maison* recherche et université, au niveau local et si possible national ce que j'ai la chance de pourvoir vivre depuis 2003 en étant élu au Conseil National des Universités.

4.1 Apports de la diversité

Les diverses thématiques de mon activité scientifique se sont fécondées mutuellement.

Mon travail de recherche dans le domaine des spécifications formelles a déteint sur mes recherches en optimisation combinatoire. La tentation, peut-être trop fréquente, en optimisation combinatoire est d'avoir comme objectif uniquerem d'obtenir un code qui résout rapidement (le plus souvent) des problèmes difficiles (au sens de la complexité). Cela au moyen (parfois exclusif) d'ajustements empiriques de paramètres d'algorithmes et peut-être au prix de la réutilisabilité de la démarche. Cette approche a un intérêt quand il s'agit de résoudre un problème d'application concrète immédiate. Elle peut donc être pertinente dans le cadre d'une recherche appliquée transférée rapidement au monde non-académique. Adopter cette démarche m'aurait mis en porte-à-faux avec mes acquis dans le domaine de la spécification et avec les messages que je tente de faire passer aux étudiants dans mes cours de méthodologie de programmation. Je ne suis donc pas surpris, rétrospectivement, d'avoir pris plaisir à formaliser les propriétés du sac-à-dos non borné telles que la dominance de seuil ou la définition d'une nouvelle borne supérieure. Et il s'avère que l'effort de formalisation et de validation des développements effectués est productif puisque le résultat est reconnu comme améliorant les temps de résolution. De même spécifier, formaliser et se préoccuper de réutilisabilité a permis de modulariser le logiciel FROST et d'en obtenir aisément une version

distribuée efficace. La maîtrise de différents paradigmes de programmation est aussi un des acquis de mon activité dans le domaine des spécifications. Très peu de scientifiques utilisent des approches fonctionnelles pour résoudre des problèmes d'optimisation. Un ouvrage de référence pour cette approche est celui de *de Moor* [41], il est connu dans le monde de la programmation fonctionnelle mais très peu dans celui de l'optimisation combinatoire. C'est même bien souvent une incongruité que d'évoquer un langage fonctionnel, de plus fortement typé, dans ce domaine où seule l'efficacité du code produit compte. Et pour certains comment envisager un code efficace écrit dans un langage qui ne permet pas de gérer *à la main* la mémoire du processeur (par exemple) ? Là encore, que ce soit pour les algorithmes de résolution d'UKP ou pour les algorithmes de résolution du problème de repliement des protéïnes, nous avons montré que combiner des approches fonctionnelles de haut niveau, allant jusqu'à utiliser les *foncteurs*, avec des approches impératives classiques permettait d'implanter efficacement, de maintenir et de faire évoluer tout en obtenant un code efficace. L'illustration la plus parlante en est pour moi l'implantation de l'algorithme présenté dans [10] où avec Philippe Veber nous avons combiné du code écrit en C pour les boucles internes et du code modulaire, proche des spécifications, écrit en Objective Caml. Si cela est connu dans le monde de la conception de langage de programmation et des spécifications, cela l'est moins dans le monde de l'optimisation combinatoire.

Par ailleurs, je suis convaincu que les préoccupations d'efficacité du code produit issues de mon activité dans POC influencent mes enseignements. La préoccupation de la complexité des algorithmes et l'influence des représentations en mémoire induisent l'acquisition de réflexes lorsque l'on souhaite résoudre des problèmes d'optimisation.

Les allers et retours entre spécification attentive et formalisée d'une part et implantation raisonnée d'autre part, dans une démarche incrémentale, permettent d'obtenir des réalisations efficaces et bien fondées. Au final, il me semble qu'associer les préoccupations de spécification, d'efficacité, de réutilisabilité, de validation,... c'est *simplement* être *pleinement* informaticien.

4.2 Perspectives

L'investissement dans le maintien et l'évolution des formations restera un élément constitutif de mon travail d'enseignant chercheur.

Les perspectives de recherche quand à elles sont attrayantes et ouvertes.

- L'avenir des recherches autour du problème de sac-à-dos m'intéressent moins aujourd'hui. C'est une thématique qui tout en offrant un support d'illustrations pour un certain nombres d'idées, me semble très peu en phase avec le concret, autrement dit, très académique.
- La collaboration avec Rumen Andonov autour de la thématique optimisation combinatoire et parallélisme pour la bio-informatique est actuellement en phase peu active pour des raisons d'éloignement géographique et de priorité de thématiques de recherches du LAMIH. Je suis conscient qu'encadrer à court terme des chercheurs au LAMIH sur cette thématique est peu probable. C'est cependant un champs de recherches qui continue de m'intéresser. D'un point de vue philosophique, c'est stimulant de participer à des recherches en informatique qui peuvent permettre de mieux comprendre le vivant.
- Le travail au sein du thème SID est stimulant. C'est une évolution intéressante pour la recherche que j'anime depuis neuf ans autour de la thématique spécification et modularité. Les travaux des deux docteurs et des masters recherche que j'ai encadrés vont permettre d'enraciner la confrontation de nos approches de formalisation et validation avec les approches de l'informatique ubiquitaire et la problématique des composants de positionnement géographique. Cela ouvre sur un champ à défricher. La collaboration mise en place avec les équipes DEDALE du Loria, TFC du LIFC et le LACL dans le cadre de la conception du projet TACOS sera source d'idées nouvelles. Cela a déjà des effets concrets puisque nous accueillons Hung Ledang à l'automne 2006 sur un support de post-doctorat, Hung a réalisé ses travaux

de thèse au sein de l'équipe DEDALE du LORIA. Les collaborations régionales se poursuivent en s'inscrivant nécessairement dans le contexte du pôle de compétitivité i-trans avec des groupes de travail et des projets (en particulier GENEVE qui est labellisé par i-trans) montés et soumis avec différentes équipes régionales.

4.2.1 À venir...

Si je devais identifier deux défis qui me semblent à la fois constituer des vrais problèmes de recherche avec implications pratiques et à la résolution desquels je m'affronterai avec intérêt je citerai les deux problématiques suivantes.

- Comment les connaissances et les approches de spécification formelle peuvent elles permettre de mieux comprendre les phénomènes biologiques. Il y a des travaux en cours dans ce domaine depuis quelques années [76, 90, 68, 64, 62, 75, 73] et si l'opportunauté m'en est donné c'est avec plaisir que je réinvestirai et réunirai dans ce domaine mon expérience dans les problèmes de bio-informatique et dans les approches de formalisation.
- La programmation par composants est en phase de diffusion et des travaux ont été menés pour tenter de formaliser la notion de composants. Cependant, formaliser et fournir des méthodes pour raisonner-sur et valider des assemblages de composants en incluant des problématiques de qualité de services reste un domaine peu défriché. Quelques travaux très récents commencent à l'explorer [42, 58, 55, 59]. C'est une problématique qui intéresse particulièrement les logiciels mis en œuvre dans le cadre des transports où il y a nécessité de fiabilité maximale. Nous avons commencé à y travailler, et le projet TACOS validé en novembre par l'ANR nous conforte dans cette direction.

4.3 Pour conclure

J'ai la chance d'avoir un travail qui me plaît. Y sont notamment conjugués la créativité et la rigueur scientifique, la valorisation de la curiosité intellectuelle et de la persévérance, le bonheur de transmettre et les difficultés à surmonter pour intéresser, le travail très individuel et le travail en équipe, la joie de la découverte et l'exercice contraignant de la rédaction, l'accompagnement de talents qui s'épanouissent et l'enrichissement par les expériences partagées et la confrontation des idées, la mise en œuvre de projets collectifs, les rencontres diversifiées, l'adaptation au contexte qui évolue : collègues nouveaux, collègues qui partent, pré-acquis des étudiants, priorités de laboratoire....

Les résultats acquis ne sont jamais définitifs, cette conclusion ouvre une nouvelle phase de mon travail d'enseignant-chercheur. Il m'y faudra toujours savoir adapter dynamiquement les différents composants de mon activité scientifique aux changements du contexte d'évaluation et d'évolution, les exigences de fiabilité impliquant une attention particulière à la phase de spécification de ces composants et de leur assemblage.

Chapitre 5

Bibliographie

Dans ce chapitre, j'ai regroupé dans un premier temps (section 5.1) les publications dont je suis l'un des auteurs. Dans un deuxième temps (section 5.2) sont référencés les travaux des communautés des domaines de reherche concernés. L'objectif de la struturation de la première section est de hiérarchiser par ordre d'importance les travaux présentés. Si au deux extrémités la séparation est facile, à l'intérieur il n'y a pas vraiment de critères définitifs. Un atelier ou Workshop peut être plus important en terme de reconnaissance qu'une conférence. Une conférence francophone peut-être internationale...

5.1 Diffusion des connaissances. Travaux publiés

5.1.1 Dans des revues

[1] F. Almeida, R. Andonov, V. Poirriez, L.M. Moreno, C. Rodriguez, and M. Perez. On the parallel prediction of the rna secondary structure. *Parallel Computing*, 13 :525–532, 2004. this volume is not available due to an editor problem.

[2] R. Andonov, V. Poirriez, and S. Rajopadhye. Unbounded knapsack problem : dynamic programming revisited. *European Journal of Operational Research*, 123(2) :168–181, 2000. *Improved version of the IRISA RR 1152, year 1997.*

[3] D. Petit, V. Poirriez, and G. Mariano. The B method and the component-based approach. *Journal of Integrated Design & Process Science*, 8(1) :65–76, March 2004.

[4] V. Poirriez. MLOG : a strongly typed confluent functional language with logical variables. *Theoretical Computer Science*, 122 :201–223, 1994.

5.1.2 Dans des conférences internationales

Domaine parallélisme et optimisation combinatoire

[5] F. Almeida, R. Andonov, D. Gonzalez, L.M. Moreno, V. Poirriez, and C. Rodriguez. Optimal tiling for the rna base pairing problem. In *Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 173–182. ACM press, august 2002. ACM ISBN : 1-58113-529-7.

[6] F. Almeida, R. Andonov, V. Poirriez, L.M. Moreno, C. Rodriguez, and M. Perez. On the parallel prediction of the rna secondary structure. In *ParCo2003*, Parallel Computing. Elsevier, september 2003.

[7] R. Andonov, V. Poirriez, and S. Rajopadhye. The unbounded knapsack problem revisited. In Petar S. Kenderov and J.P. Revalski, editors, *Pliska studia mathematica Bulgarica : pro-*

ceedings of the 4th Conference on Mathematical Methods in Operations Research, Sozopol. Bulgarian Academy Sciences, september 1997. ISSN : 0204-9805, Bulgaria.

[8] V. Poirriez and R. Andonov. Unbounded Knapsack Problem : New Results. In *Proceedings of the Workshop Algorithms and Experiments (ALEX98)*, pages 103–111, February 1998. available at : `http://rtm.science.unitn.it/alex98/proceedings.html`.

[9] V. Poirriez, A. Marin, R. Andonov, and J.F. Gibrat. Frost : Revisited and distributed. In *Proceedings of the 19th IEEE International Parallel & Distributed Processing Symposium.* Fourth IEEE International Workshop on High Performance Computational Biology(HICOMB'05), Denver(USA), IEEE, April 2005. ISBN : 0-7695-2312-9.

[10] P. Veber, N. Yanev, R. Andonov, and V. Poirriez. Optimal protein threading by cost-splitting. In G. Myers R. Casadio, editor, *Algorithms in Bioinformatics-WABI 2005(5th Workshop on Algorithms in Bioinformatics)*, volume 3692, pages 365–375. EATCS, the European Association for Theoretical Computer Science, and ISCB, the International Society for Computational Biology, Springer Verlag Lecture Notes in Bioinformatics, october 3-6, Mallorca, Spain 2005. ISBN 3-540-29008-7.

Domaine spécifications et modularité

[11] S. Colin, G. Mariano, and V. Poirriez. Duration calculus : A real-time semantic for B. In Zhiming Liu and Keijiro Araki, editors, *International Colloquium on Theoretical Aspects of Computing (ICTAC 2004)*, volume 3407 of *Lecture Notes in Computer Science*, pages 431–446. Springer, September 2004. Guiyang, China. ISBN 3-540-25304-1.

[12] S. Colin, D. Petit, J. Rocheteau, R. Marcano, G. Mariano, and V. Poirriez. BRILLANT : An open source and XML-based platform for rigourous software development. In *SEFM (Software Engineering and Formal Methods)*, pages 373–382, Koblenz, Germany, september 2005. AGKI (Artificial Intelligence Research Koblenz, IEEE Computer Society Press. ISBN : 0-7695-2435-4.

[13] D. Petit, G. Mariano, V. Poirriez, and J.L. Boulanger. Automatic Annotated Code Generation from B Formal Specifications. In G. Tarnai and E. Schnieder, editors, *Symposium on Formal Methods for Railway Operation and Control Systems*, number ISBN 963 9457 45 0 in IEEE, pages 37–44. L'Harmattan, May 2003.

[14] D. Petit, V. Poirriez, and G. Mariano. The B method and the component-based approach. In *Formal Reasoning on Software Components and Component Based Software Architectures*, December 2003. Special topic session of the Seventh Conference on Integrated Design and Process Technology. ISSN 1090-9389.

[15] D. Petit, V. Poirriez, and G. Mariano. Reuse of ML module system for the B language. In Pierre Boulet, editor, *Forum on specification and Design Language*, September 2004. Lille, France.

Liés à mes travaux de thèse

[16] V. Poirriez. Fluo : an implementation of mlog. In Kari Systă, editor, *Nordic Workshop on Programming Environnement Research.* Tampere University of Technology Software Systems Laboratory, January 1992.

[17] V. Poirriez. Introduction à mlog : une extension réaliste de ml avec des variables logiques. *BIGRE*, 76-77 :110–129, 1992. Actes des Journées Francophones des Langages Applicatifs.

[18] V. Poirriez. Mlog a strongly typed confluent functional language with logical variables. In *Proceedings of FGCS'92,TOKYO*, 1992.

[19] V. Poirriez. Mlog : a pragmatic extension of ml with logical variables, unification and suspensions. In Andy Můck, editor, *Proceedings of the second international workshop on functional/logic programming*, 1993.

5.1.3 Dans des ateliers internationaux avec actes publiés ou conférences francophones

[20] S. Colin, V. Poirriez, and G. Mariano. Thougths about the implementation of the duration calculus with coq. In Boris Konev and Renate Schmidt, editors, *4th International Workshop on the Implementation of Logics in conjunction with the 10th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume ULCS-03-018, pages 33–45, Almaty, Kazakhstan, september 2003. University of Liverpool.

[21] D. Petit, G Mariano, and V Poirriez. Vers un système de modules à la harper-lillibridge-leroy pour les spécifications formelles *b*. In *Journées Francophones des Langages Applicatifs 2002*, pages 85–100. INRIA, January 2002. http ://pauillac.inria.fr/jfla/2002/actes/07-petit.ps.

[22] D. Petit, V. Poirriez, and G. Mariano. Development of Formal Components Using the B Method. In Manuel Carro, Claudio Vaucheret, and Kung-Kiu Lau, editors, *Proceedings of the First COLOGNET Joint Workshop on Component-based Software Development and Implementation Technology for Computational Logic Systems*, number CLIP4/02.0, pages 35–46, September 2002.

[23] Dorian Petit, Georges Mariano, and Vincent Poirriez. Génération de composants à partir de spécifications B. In *Actes de AFADL : Approches Formelles dans l'Assistance au Développement de Logiciels*, number ISBN 2-7261-1263-6, pages 103–119, Janvier 2003.

[24] J. Pley, R. Andonov, J.F. Gibrat, A. Marin, and V. Poirriez. Parallèlisation d'une méthode de reconnaissance de repliements de protéines(frost). In *Proceedings des Journées Ouvertes Biologie Informatique Mathématiques*, pages 287–288. INRIA, june 2002.

[25] Vincent Poirriez, Antoine Marin, Rumen Andonov, and Jean-François Gibrat. Frost(fold recognition-oriented search tool) : Re-conçu et distribué, Avril 2005. Workshop : Résolution Parallèle des Problèmes NP-Complets, NP-PAR'05.

[26] J. Rocheteau, S. Colin, G. Mariano, and V. Poirriez. Évaluation de l'extensibilité de Phox B/Phox un assistant de preuve pour B. In Valérie Ménissier-Morain, editor, *Actes des Journées Francophones des Langages Applicatifs2004*, pages 139–154. INRIA, January 2004. ISBN 2-7261-1272-2.

5.1.4 Dans des conférences internationales sans actes publiés

[27] Rumen Andonov, Vincent Poirriez, and Sanjay Rajopadhye. The unbounded knapsack problem revisited. European Chapter on Combinatorial Optimization, ECCO X, may 1997. Spain.

[28] V. Poirriez and R. Andonov. Dynamic programming with bounds for the ukp. In *CO2002, International Symposium on Combinatorial Optimization*, 2002.

[29] Philippe Veber, Nicola Yanev, Rumen Andonov, and Vincent Poirriez. Optimal protein threading by cost-splitting. 4th International Workshop on Efficient and Experimental Algorithms, May 2005. Poster Paper.

5.1.5 Présentés lors d'ateliers de travail ou dans des rapports de recherche

[30] Vincent Poirriez. *Intégration de fonctionnalités logiques dans un langage fortement typé : MLOG une extension de ML*. PhD thesis, Université de Paris 7, 1991.

[31] Vincent Poirriez, Nicola Yanev, and Rumen Andonov. Pyasukp home page. http ://gna.org/projects/pyasukp/.

[32] Vincent Poirriez, Nicola Yanev, and Rumen Andonov. Towards reduction of the class of intractable unbounded knapsak problem. Research Report 1, LAMIH/ROI UMR CNRS 8530, july 2002. http ://www.univ-valenciennes.fr/ROI/poirriez/RR-lamihroi-2002-01.ps.gz.

5.2 Références bibliographiques

5.2.1 Domaine conception de langage, spécification et modularité

[33] Jean-Raymond Abrial. The B tool (proof proving tool). In Bloomfield R. ; Marshall L. ; Jones R., editor, *VDM 88. VDM - The Way Ahead. 2nd VDM-Europe Symposium. Proceedings.* Springer-Verlag, Berlin, West Germany, 11-16 Sept. 1988.

[34] Jean-Raymond Abrial. *The B Book - Assigning Programs to Meanings.* Cambridge University Press, August 1996.

[35] B-core. The b-toolkit. `http ://www.b-core.com/ONLINEDOC/BToolkit.html`.

[36] Salimeh Behnia. *Test de modèles formels en B : cadre théorique et critères de couvertures.* Thèse de doctorat, Institut National Polytechnique de Toulouse, October 2000.

[37] S.M. Brien and J.E. Nicholls. Z base standard : Version 1.0. Technical Monograph PRG-107, Oxford University Computing Laboratory, 11 Keble Road, Oxford OX1 3QD, UK, November 1992.

[38] ClearSy. Présentation de l'atelierb.
`http ://www.atelierb.societe.com/ressources/BAtb3.pdf`.

[39] Samuel Colin. *Contribution à l'intégration de temporalité au formalisme B : Utilisation du calcul des durées en tant que sémantique temporelle pour B.* PhD thesis, Université de Valenciennes et du Hainaut Cambrésis, Octobre 2006.

[40] Comprehensive Z Tools. `http ://czt.sourceforge.net/`.

[41] O. de Moor. A Generic Program for Sequential Decision Processes. In *Proc. PLILP'95.* Springer Verlag, LNCS 982, 1995.

[42] Fractal project. `http ://fractal.objectweb.org/`.

[43] Robert Harper and Mark Lillibridge. A type-theoretic approach to higher-order modules with sharing. In *Conference record of POPL '94 : 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 123–137, Portland, OR, January 1994.

[44] Søren T. Heilmann. *Proof Support for Duration Calculus.* PhD thesis, Department of Information Technology, Technical University of Denmark, January 1999.

[45] Dang Van Hung and Paritosh K. Pandya. Duration calculus with weakly monotonic time. In Anders P. Ravn and Hans Rischel, editors, *5th symposium of Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98)*, volume LNCS 1486, pages 55–64, Lyngby, Denmark, september 1998. Springer-Verlag. Technical Report 122, UNU-IIST, P.O. Box 3058, Macau, September 1997.

[46] Regine Laleau and Amel Mammar. A generic process to refine a B specification into a relational database implementation. In ZB'2000 [61], pages 22–41.

[47] X. Leroy, D. Doligez, J. Garrigue, D. Rémy, and J. Vouillon. The objective caml system. Technical report, INRIA, 2005. Software and documentation available on the Web, `http://caml.inria.fr/`.

[48] Xavier Leroy. The ZINC experiment : an economical implementation of the ML language. Technical report 117, INRIA, 1990.

[49] Xavier Leroy. A modular module system. *Journal of Functional Programming*, 10(3) :269–303, 2000.

[50] Christophe Metayer, Jean-Raymond Abrial, and Laurent Voisin. Event-b language. RODIN Project Deliverable D7, May 2005.

[51] Bertrand Meyer. Contracts for Components. *Software Development Magazine*, July 2000. online magazine (http ://www.sdmagazine.com/).

[52] Overture (VDM++). http ://www.overturetool.org/.

[53] Dorian Petit. Spécification des caractéristiques du langage B par les grammaires attribuées. Technical report, INRETS, Septembre 1999. Rapport de stage de DEA.

[54] Dorian Petit. *Génération automatique de composants logiciels sûr à partir de spécifications formelles B.* PhD thesis, Université de Valenciennes et du Hainaut Cambrésis, Décembre 2003. Numéro d'ordre 03-34.

[55] Qua : Quality of service aware component architecture. http ://www.simula.no/departments/networks/projects/QuA/.

[56] Christophe Raffalli and Paul Rozière. Phox. In Wiedijk [60], pages 67–71.

[57] Rodin-B#. http ://rodin-b-sharp.sourceforge.net/.

[58] The think framework. http ://think.objectweb.org/.

[59] Jean-Charles Tournier. *Qinna, une architecture à base de composants pour la gestion de la qualité de service dans les systèmes embarqués mobiles.* PhD thesis, INSA de Lyon, 07 2005.

[60] Freek Wiedijk, editor. *The Seventeen Provers of the World, Foreword by Dana S. Scott*, volume 3600 of *Lecture Notes in Computer Science*. Springer, 2006.

[61] *ZB'2000 – International Conference of B and Z Users*, volume 1878 of *Lecture Notes in Computer Science (Springer-Verlag)*, Helsington, York, UK YO10 5DD, August 2000.

5.2.2 Domaine parallèlisme et optimisation combinatoire, Bioinformatique

[62] Jamil Ahmad, Adrien Richard, Gilles Bernot, Jean-Paul Comet, and Olivier F. Roux. Delays in biological regulatory networks (brn). In *International Conference on Computational Science (2)*, pages 887–894, 2006.

[63] T. Akutsu and S. Miyano. On the approximation of protein threading. *Theoretical Computer Science*, 210 :261–275, 1999.

[64] Vassil N. Alexandrov, G. Dick van Albada, Peter M. A. Sloot, and Jack Dongarra, editors. *Computational Science - ICCS 2006, 6th International Conference, Reading, UK, May 28-31, 2006, Proceedings, Part II*, volume 3992 of *Lecture Notes in Computer Science*. Springer, 2006.

[65] R. Andonov, S. Balev, and N. Yanev. Protein threading problem : From mathematical models to parallel implementations. *INFORMS Journal on Computing*, 16(4), 2004. Special Issue on Computational Molecular Biology/Bioinformatics.

[66] R. Andonov and S. Rajopadhye. A Sparse Knapsack Algo-tech-cuit and its Synthesis. In P. Cappello, R. Owens, E. Swartzlander, and B. Wah, editors, *International Conf. on Application Specific Array Processors ASAP'94*, pages 302–313. IEEE, 1994. San Francisco, California.

[67] Stefan Balev. Solving the protein threading problem by lagrangian relaxation. WABI 2004, 4th Workshop on Algorithms in Bioinformatics, September 14 - 17 2004. Bergen, Norway.

[68] G. Bernot, J-P. Comet, and J. Guespin. A fruitful application of formal methods to biological regulatory networks : Extending thomas' asynchronous logical approach with temporal logic. *Journal of Theorical Biology*, 229(3) :339–347, 2004.

[69] C. Branden and J. Tooze. *Introduction to protein structure.* Garland Publishing, 1999.

[70] L. Caccetta and A. Kulanoot. Computational Aspects of Hard Knapsack Problems. *Nonlinear Analysis*, 47 :5547–5558, 2001.

[71] Casp :critical assessment of structure prediction. http ://www.forcasp.org/.

[72] C-S. Chung, M. S. Hung, and W. O. Rom. A Hard Knapsack Problem. *Naval Research Logistics*, 35 :85–98, 1988.

[73] Gianfranco Ciardo and Philippe Darondeau, editors. *Applications and Theory of Petri Nets 2005, 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005, Proceedings*, volume 3536 of *Lecture Notes in Computer Science*. Springer, 2005.

[74] G. Collet, N. Yanev, A. Marin, R. Andonov, and J-F. Gibrat. A flexible model for protein fold recognition. In *Septièmes Journées Ouvertes de Biologie, Informatique et Mathématiques (JOBIM)*, 2006.

[75] Jean-Paul Comet, Hanna Klaudel, and Stéphane Liauzu. Modeling multi-valued genetic regulatory networks using high-level petri nets. In Ciardo and Darondeau [73], pages 208–227.

[76] Vincent Danos and Vincent Schächter, editors. *Computational Methods in Systems Biology, International Conference CMSB 2004, Paris, France, May 26-28, 2004, Revised Selected Papers*, volume 3082 of *Lecture Notes in Computer Science*. Springer, 2005.

[77] Frost web site. `http ://genome.jouy.inra.fr/frost/`.

[78] P.C. Gilmore and R.E Gomory. A Linear Programming Approach to the Cutting Stock Problem- part II. *Operations Research*, 11 :863–888, 1963.

[79] R.E. Gomory. Outline of an Algorithm for Integer Solutions to Linear Programs. *Bulletin of the American Mathematical Society*, 64 :275–278, 1958.

[80] H.J. Greenberg, W.E. Hart, and G. Lancia. Opportunities for combinatorial optimization in computational biology. *INFORMS Journal on Computing*, 16(3), 2004.

[81] T. Head-Gordon and J. C. Wooley. Computational challenges in structural and functional genomics. *IBM Systems Journal*, 40 :265–296, 2001.

[82] R. Lathrop. The protein threading problem with sequence amino acid interaction preferences is np-complete. *Protein Eng.*, 7 :1059–1068, 1994.

[83] R.H. Lathrop and T.F. Smith. Global optimum protein threading with gapped alignment and empirical pair potentials. *J. Mol. Biol.*, 255 :641–665, 1996.

[84] T. Lengauer. Computational biology at the beginning of the post-genomic era. In R. Wilhelm, editor, *Informatics : 10 Years Back - 10 Years Ahead*, volume 2000 of *LNCS*, pages 341–355. Springer Verlag, 2001.

[85] A. Marin, J. Pothier, K. Zimmermann, and J.F. Gibrat. Frost : A filter based recognition method. *Proteins*, 49(4) :493–509, 2002.

[86] S. Martello and P. Toth. *Knapsack Problems : Algorithms and Computer Implementation* . John Wiley and Sons, 1990.

[87] G. L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Willey & Sons, 1988.

[88] U. Pferschy, H. Kellerer, and D. Pisinger. *Knapsack Problems*. Springer Verlag, 2004. ISBN 3-540-40286-1.

[89] D. Pisinger. Dominance Relations in Unbounded Knapsack Problems. Technical Report 94/33, DIKU, University of Copenhagen,DK-2100 Copenhagen,Denmark, December 1994.

[90] Corrado Priami, editor. *Computational Methods in Systems Biology, International Conference CMSB 2006,Trento, Italy, October 18-19 , 2006*, Lecture Notes in Computer Science. Springer, 2006.

[91] J.C. Setubal and J. Meidanis. *Introduction to computational molecular biology*, chapter 8, pages 252–259. Brooks/Cole Publishing Company, 511 Forest Lodge Road, Pacific Grove, CA 93950, 1997.

[92] J. Xu, M. Li, G. Lin, D. Kim, and Y. Xu. RAPTOR : optimal protein threading by linear programming. *Journal of Bioinformatics and Computational Biology*, 1(1) :95–118, 2003.

[93] Y. Xu and D. Xu. Protein threading using PROSPECT : design and evaluation. *Proteins : Structure, Function, and Genetics*, 40 :343–354, 2000.

[94] N. Yanev and R. Andonov. Parallel divide and conquer approach for the protein threading problem. *Concurrency and Computation :Practice and Experience*, 16 :1–14, 2004.

[95] Nan Zhu and Kevin Broughan. On dominated terms in the general knapsack problem. *Operations Research Letters*, 21 :31–37, 1997.

[96] M. Zuker. Rna folding lectures. Web page. `http://bioinfo.math.rpi.edu/~zukern/BIO-5495/RNAfold.html`.

5.2.3 Vie universitaire

[97] Arrêté du 23 novembre 1988 modifié relatif à l'habilitation à diriger des recherches. `http ://www.dsi.cnrs.fr/rmlr/textesintegraux/volume1/1431-adu23-11-1988.htm`.

[98] Pierre Christin. *Petits Crimes contre les humanités*. Métailié, 2006.

[99] Circulaire no 89-004 du 5 janvier 1989 modifiée relative à l'application de l'arrêté du 23 novembre 1988 relatif à l'habilitation à diriger des recherches. `http ://www.dsi.cnrs.fr/rmlr/textesintegraux/volume1/1431-cir89-004.htm`.

[100] Site d'informations du cnu 27. `http ://cnu.ifsic.univ-rennes1.fr/`.

[101] Statuts du département informatique de l'uvhc. `http ://www.univ-valenciennes.fr/ROI/poirriez/statutsdeptinfo.ps.gz`.

[102] Site du pôle de compétitivité itrans. `http ://www.i-trans.org/`.

[103] Opérations postes de la smai. `http ://postes.smai.emath.fr/concours.php`.

Recueil de publications

Sont repris ici les articles publiés depuis 2000.

Références bibtex :

```
@article{APR_ejor,
   author = "Andonov, R. and Poirriez, V. and Rajopadhye, S. ",
   journal = {European Journal of Operational Research },
  volume  = {123},
   number = {2},
  pages =  {168-181},
   title = "Unbounded Knapsack Problem : dynamic programming revisited",
   year = {2000}
}
```

Unbounded knapsack problem : dynamic programming revisited

R.Andonov, V. Poirriez, S. Rajopadhye

Adresses

LIMAV, Université de Valenciennes
Le Mont Houy, B.P.311, 59304 Valenciennes Cedex, France
IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France

Abstract We present EDUK, an efficient dynamic programming algorithm for the unbounded knapsack problem. It is based primarily on a new and useful *dominance* relation, called *threshold dominance*, which is a strict generalization of all the previously known dominance relations. We show that combining it with a sparse representation of the iteration domain and the periodicity property leads to a drastic reduction of the solution space. We present computational experiments with various data instances to validate our ideas and demonstrate the efficiency of EDUK vis a vis the well known exact algorithm MTU2.

Keywords integer programming, dominances, dynamic programming, periodicity, combinatorial optimization

1 Introduction

The unbounded knapsack problem (UKP) is a classic NP-*hard*, combinatorial optimization problem with a wide range of applications [6, 10, 12, 13]. It may be formulated as follows : we are given a knapsack of capacity c, into which we may put n types of objects. Each object of type i has a *profit*, p_i, and a *weight*, w_i, (w_i, p_i, n and c are all positive integers, and we have an *unbounded* number of copies of each object type). Determine the number x_i, of i-th type objects that maximize total profit without exceeding capacity, i.e.,

$$\max \left\{ \sum_{i \in N} p_i x_i \ : \ \sum_{i \in N} w_i x_i \leq c, \ x_i \geq 0 \text{ integer}, \ i \in N = \{1 \dots n\} \right\} \tag{1.1}$$

The two classic approaches for solving this problem exactly are branch and bound (B&B) [12] and dynamic programming (DP) [6, 10, 13], the subject of this paper. DP takes $O(nc)$ time, and works in two phases : in the *forward* phase, we calculate the *optimal value* of the profit function by tabulation of a recurrence equation, and then we use this in the *backtracking* phase to determine an actual solution which has this optimal profit. Because the problem is NP-*hard*, a number of

schemes have been proposed for reducing the search space. The three important ones relevant to this paper are *dominance*, *periodicity* and *sparsity*.

In 1963, Gilmore and Gomory [8] proposed the notion of dominance (called *simple dominance* in this paper). Intuitively, if an object type has a larger (or no smaller) weight, and smaller (or no larger) profit than another, the former may never occur in an optimal solution. Martello and Toth [12] introduce a more powerful dominance (called *multiple dominance* in this paper). Many extensions have been proposed over the years [4, 5, 14, 17]. Of particular interest is when an object type has larger (or no smaller) weight and lower (or no larger) profit than a positive linear combination of (a subset of) the other object types (called ***collective dominance*** in this paper).

Another well known approach to reduce the search space is available for dynamic programming based algorithms. In 1966 Gilmore and Gomory [7] described a *periodicity* property which states that beyond a certain capacity, the only object type that contributes *further* to an optimal solution is the *best* one (the one with the highest profit-to-weight ratio).

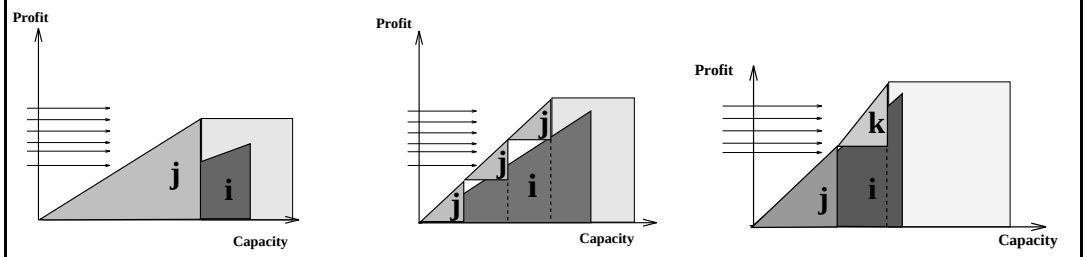
The third technique, again applicable for dynamic programming, was recently presented by Andonov and Rajopadhye [2]. It is based on the fact that the standard recurrence is monotonic. This allows a "sparse" representation of the computation (we need to tabulate only certain critical points, and not the entire recurrence), and provides for considerable reduction of the search space.

The goal of this paper is to show how to exploit *all* these properties together with efficient backtracking. Our main contributions are as follows.

- We introduce and formalize a new dominance called ***threshold dominance***. We show how threshold dominance includes collective dominance, which itself is a strict generalization of multiple dominance. We show that collective dominance is *maximal*, in terms of coefficient reduction.
- We show that threshold dominance provides an easy test for detecting when periodicity is attained.
- We design a dynamic programming algorithm, EDUK, which incorporates these properties and the sparse representation into a *slice wise* evaluation of a well known (but not commonly used) recurrence for the forward phase. EDUK needs no preprocessing, which is incorporated into the main algorithm itself.
- We present a backtracking algorithm with the same performance as Hu's well known idea [10] of computing an auxiliary recurrence, but *without any* auxiliary information. In addition, our technique enables us to determine *all* solutions.
- We validate our algorithm by numerous computational experiments. We also indicate some of the limitations in the currently proposed random data sets and also study the sensitivity of the running time to the capacity.

The remainder of this paper is organized as follows. In Section 2 we develop collective and threshold dominance, give their properties and their relation to periodicity. Section 3 describes how our algorithm by slices incorporates all the above features. It also presents the backtracking algorithm. Section 4 discusses the problem of test case generation for the UKP. Section 5 presents our experimental results, and we conclude in Section 6.

Notation : The n-vectors, $\mathbf{w}$ and $\mathbf{p}$ denote, respectively, the weights and profits for a given problem instance. An object type can be identified either by its index, i, or by the ordered pair (w_i, p_i). We denote by UKP_γ^S a knapsack (sub)-problem with capacity γ and whose object types are restricted to $S \subseteq N$; the original problem is thus UKP_c^N. $F(\gamma, S)$ is the optimal profit that can be achieved for knapsack UKP_γ^S. For any n-vector $\mathbf{x}$, in Z_+^n, we define its ***weight***, $W(\mathbf{x})$, and ***profit***, $P(\mathbf{x})$, respectively as $W(\mathbf{x}) = \sum x_i w_i$ and $P(\mathbf{x}) = \sum x_i p_i$. The pair $(W(\mathbf{x}), P(\mathbf{x}))$ is called the ***solution point*** of $\mathbf{x}$. $\mathrm{Opt}(\mathrm{UKP}_c^N)$ denotes the set of the optimal solutions of UKP_c^N, i.e., all feasible solutions $\mathbf{x}$ for UKP_c^N such that $P(\mathbf{x}) = F(c, N)$. The ***profitability*** of an object type is the ratio of its profit to weight, $\frac{p_i}{w_i}$. The ***profitability pre-order*** is the relation in N^2 defined as



Simple dominance, $i \ll_s j$ Multiple dominance, $i \ll_m j$ Collective dominance, $i \ll \{j,k\}$

An object type is shown as a triangle of width w and height p. Its "shadow" is the region to its right and below its height. Left : i is in the shadow of j. Middle : i is in the combined shadow of $3j$. Right : i is in the combined shadow of $j+k$.

FIG. 1.1 – Illustration of simple, multiple and collective dominance.

$j \sqsupseteq k$ iff $\frac{p_j}{w_j} \geq \frac{p_k}{w_k}$. This pre-order is often used in the literature, typically in a preprocessing phase either on all the object types [6, 12] or on pre-selected core problems [5, 12].

2 Dominance Relations

Definition 1 *A pair, (s',f') **dominates** another (illustrated in Fig. 1.1), (s,f) iff $s \geq s'$ and $f \leq f'$. An object type i is **simply dominated** by j, written as $i \ll_s j$, iff (w_i,p_i) is dominated by (w_j,p_j); i is **multiply dominated** by j, written as $i \ll_m j$, iff $\left\lfloor \frac{w_i}{w_j} \right\rfloor \geq \frac{p_i}{p_j}$.*

It is easy to verify that dominance between solution points (and hence between object types) is a partial order. Furthermore, for some $\mathbf{x}$ and $\mathbf{x}'$ feasible solutions of respectively UKP_c^T and $\mathrm{UKP}_{c'}^{T'}$, if the solution point of $\mathbf{x}'$ dominates that of $\mathbf{x}$, then either $\mathbf{x}$ cannot be an optimal solution for any subproblem which includes at least $T \cup T'$ (for *any* capacity), or $f = f'$ and so $\mathbf{x}'$ is also optimal, whenever $\mathbf{x}$ is.

Definition 2 *Let $J \subseteq N$ and $i \notin J$. The i-th object type is **threshold dominated** by J (we will not mention J explicitly when $J = N\backslash\{i\}$), written as $i \prec\!\!\ll J$ iff there exists $\alpha \in Z_+$ such that*

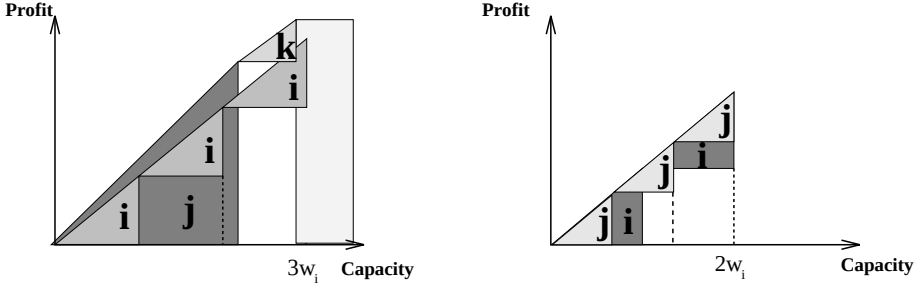
$$\sum_{j\in J} x_j w_j \leq \alpha w_i, \text{ and } \sum_{j\in J} x_j p_j \geq \alpha p_i \text{ for some } \boldsymbol{x} \in Z_+^n \tag{1.2}$$

*In other words, a positive linear combination of the object types in J yields a "better" (or at least, no worse) solution than α copies of i. The **threshold** of i **with respect to** J, denoted by $t(i,J)$, is defined as the smallest αw_i satisfying (1.2). If no such α exists, then $t(i,J) = \infty$. We let $t_i = t(i, N\backslash\{i\})$ and call it **the threshold** of i. The particular case when $\alpha = 1$ is called **collective dominance** and denoted by $i \ll J$.*

The idea of the threshold is illustrated in Fig. 1.2. Note that if j and i have equal profitability, then the lowest common multiple of their weights is a common threshold for both of them. This introduces a subtle implementation detail, and is discussed later. An alternative view of the threshold is given by the following remark, which leads to a natural criterion for threshold detection.

Remark 3 *If $i \prec\!\!\ll J$ then $t(i,J)$ is equal to the **largest** multiple αw_i of its weight such that for $0 \leq k < \alpha$, the optimal solution of each of $\mathrm{UKP}_{kw_i}^{J\cup\{i\}}$ is unique and given by $k\mathbf{e_i}$ (where $\mathbf{e_i}$ is the i-th unit vector).*

Criterion 4 *If i is threshold dominated, then $t_i = \alpha w_i$ where α is the smallest integer such that for some $J \subseteq N\backslash\{i\}$, and $J' = J \cup \{i\}$, either $F(\alpha w_i, J') > \alpha p_i$ or $F(\alpha w_i, J') = F(\alpha w_i, J) = \alpha p_i$*



$i \prec\!\!\!\ll \{j,k\}$ $\qquad\qquad$ $i \prec\!\!\!\ll \{j\}$ and $j \prec\!\!\!\ll \{i\}$

Left : $t_i = 3w_i$ since 3 copies of i are in the combined shadow of j and k.
Right : equiprofitable object : $2w_i = 3w_j = t_i = t_j$.

FIG. 1.2 – Illustration of the threshold of an object type.

Collective dominance is maximal in the following sense : if $i \ll J$, then i can be discarded from consideration without changing the optimal value for *any* capacity, c. Conversely, whenever i is not collectively dominated by all the others, it contributes to the optimal solution for some capacity (in particular, for $c = w_i$). The following proposition gives the motivation for introducing the notion of threshold.

Proposition 5 *If $t(i,J) = \alpha w_i$, then for* any capacity *c, there exists $\tilde{\mathbf{x}} \in \mathrm{Opt}(\mathrm{UKP}_c^{J\cup\{i\}})$ such that $\tilde{\mathbf{x}}_i < \alpha$.*

Proof : It suffices to consider $c \geq \alpha w_i$ (below this capacity, no feasible solution can have α copies of i). Let $\mathbf{x} \in \mathrm{Opt}(\mathrm{UKP}_c^{J\cup\{i\}})$. If $x_i < \alpha$ then $\mathbf{x}$ is the desired solution. Otherwise let $x_i = k\alpha + r$ for positive integers k, r such that $0 \leq r < \alpha$. By definition there exists $\mathbf{z} \in Z_+^n$ such that, $\sum_{j\in J} z_j w_j \leq \alpha w_i$ and $\sum_{j\in J} z_j p_j \geq \alpha p_i$. Now, $\mathbf{x} + k\sum_{j\in J} z_j \mathbf{e_j} - k\alpha \mathbf{e_i}$ is the desired optimal solution.

In other words, if $t_i = \alpha w_i$, then for any capacity $c = c' + (\alpha - 1)w_i$ with $c' \geq 0$, we have $F(c,N) = max_{0\leq k<\alpha}\, kp_i + F(c - kw_i, N\backslash\{i\})$. We thus need to efficiently detect the threshold. This can be done with the previous criterion 4 or with an alternative one given in the next section (criterion 10). The following properties can be readily verified.

1. $i \ll \{j\}$ iff $i \neq j$ and $i \ll_m j$
2. The object type with the smallest weight (if it is not unique, then the one with the best profitability among them) is never collectively dominated.
3. If $i \prec\!\!\!\ll J$, then there exists at least one object type $j \in J$ such that $i \sqsubseteq j$.

Note that the cost required to check for a dominance relation is increasing from $\ll_s$ to $\prec\!\!\!\ll$. In fact, $\ll$ and $\prec\!\!\!\ll$ require the solution of knapsack sub-problems. We extend $\ll_s$ and $\ll_m$ in the obvious way to relations between object types and subsets of N and we consider the four relations as subsets of $N \times P(N)$. The following result is now straightforward.

Theorem 6 *The four dominance relations are in strictly increasing order of generality :* $\ll_s \subset \ll_m \subset \ll \subset \prec\!\!\!\ll \subset \prec$.

2.1 Periodicity

We now describe the relation between threshold dominance and periodicity, a well known property of the UKP [6, 7]. Periodicity states that beyond a capacity, y, *only one object type contributes again* to the solution. This object type belongs to the set of object types with the best profitability, thus it is also noted below by b. Hence, knowing the solutions for each capacity below y is sufficient

to solve the problem for any capacity. Formally it can be formulated as follows. There exists a capacity y such that

$$\forall\, c \geq y, F(c,N) = \lceil (c-y)/w_b \rceil p_b + F(c - \lceil (c-y)/w_b \rceil w_b, N) \tag{1.3}$$

The following result relates the periodicity property with the threshold dominance and gives a simple test to check if the period level is reached. For any capacity y, we define $\mathcal{U}(y) = \{j \in N : t_j > y\}$ as the set of object types that are not threshold dominated above y.

Theorem 7 *The set $\mathcal{U}(y)$ is a singleton, iff the capacity y satisfies Eqn. 1.3.*

Proof
$\Rightarrow$ Obvious.
$\Leftarrow$ Let k be an object type different from b. We will show that the threshold of k is below y. Let t' be the smallest multiple of w_k such that $t' \geq y$. According to (1.3) there exists $\tilde{\mathbf{x}} \in \mathrm{Opt}(\mathrm{UKP}^N_{t'})$ such that $\tilde{x_b} > 0$ which implies $\tilde{x_k} < t'/w_k$. Hence from Remark 3 we obtain $t_k \leq t' \leq y$ and $k \notin \mathcal{U}(y)$. Therefore only the best object type has (eventually) a higher larger than y.

Corollary 8 *The* period level*, y^* is defined as the smallest y satisfying (1.3).*
$y^* = min\{y \mid \mathcal{U}(y) \mid = 1\}$

3 An Efficient Implementation

We now describe how the above properties are combined into an efficient algorithm, EDUK (**E**fficient **D**ynamic programming for the **U**nbounded **K**napsack problem). The forward phase is based on a standard, though not common recurrence, that can be easily derived from the principle of optimality. $F(c,N) = g(c,n)$ where,

$$g(j,k) = max(g(j,k-1), g(j-w_k, n) + p_k) \tag{1.4}$$

with boundary conditions $g(0,0) = 0$ and for $j < 0$, $g(j,n) = -\infty$. This is viewed as an $c \times n$ table, having n columns, one for each object type. The columns have height c and are "filled from top to bottom" (origin at the top-left). When we detect that an object type is (collectively) dominated by the other remaining object types, its column is deleted, thus achieving coefficient reduction. When we detect that the capacity has exceeded the threshold beyond which an object type does not contribute any more, the *remainder* of its column is deleted, thus achieving capacity reduction. And finally, we also exploit *sparsity* : in the (sub) columns where the recurrence is to be evaluated, we do not evaluate it *for all values of j*. Since the function $g(j,k)$ is monotonically increasing in j, we need to compute it *only when it changes value* (at the so called critical points).

Our challenges are to detect dominance (collective as well as threshold) as (i) cheaply and (ii) early as possible, and (iii) to integrate it with the sparse algorithm, i.e., the data required for this detection must be available only by inspecting the results at critical points.

The recurrence is evaluated by horizontal slices. This can be done easily if the last column of the table is retained up to the start of the slice (say capacity c_0), and the boundary conditions of Eqn. 1.4 are modified appropriately. For each slice, EDUK maintains $g_0 = F(c_0, N)$, the value of $\mathrm{Opt}(\mathrm{UKP}^N_{c_0})$ and the index of the current best object, b'. The main data structures are three lists :

- $\mathcal{R}$, a list of *residual* object types. It is initialized to N, and (at the start of any slice) contains the set of object types whose weights are larger than c_0 (hence we don't yet know whether they are collectively dominated or not), and which have not been discarded by some "easy" tests (described later).
- $\mathcal{I}$, a list of size p, which contains the object types *introduced* during this slice (p is a parameter—exactly p new object types are introduced in a slice).

– $\mathcal{U}$, the list of "currently non-dominated" object types (at the start of each slice). These are object types that (i) have weights smaller than c_0, (ii) are not collectively dominated by the other object types, and whose (iii) thresholds are not less than c_0.

Evaluation of a slice : The computation performed in evaluating a slice consists of the following.

1. **Pre-processing :** Make a single pass through $\mathcal{R}$ in order to determine the p lightest object types to introduce in the slice. The list $\mathcal{I}$ is maintained sorted by increasing weight. For each element from $\mathcal{R}$, EDUK performs two simple tests : test if it is collectively dominated by g_0, and test whether it is multiply dominated by the best object type so far. It is a tradeoff whether this test, which involves a couple of arithmetic operations should be done here for *all* elements of $\mathcal{R}$, or only the p object types selected from $\mathcal{R}$. If so it is immediately discarded. If it passes these tests, insert it in $\mathcal{I}$, and while doing so, check that it is not simply dominated by the object types already there. Then too it is discarded (these are the "easy tests" mentioned above).
2. **Main :** For each i in $\mathcal{I}$, resolve the successive sub-problems of capacity w_i, using only object types in $\mathcal{U}$. This is done by evaluating the recurrence $g(j,k)$, using the sparse algorithm (see Sec. 3.1). At the end of each such evaluation, check if the object type i is collectively dominated by the others. If so, it is discarded, otherwise it is appended at the end of $\mathcal{U}$.
3. **Post-processing** Update the current best object type (using those elements of $\mathcal{I}$ that were added to $\mathcal{U}$ in step 2 above. Also update the current optimal value, $(w_p, F(w_p, N))$. Finally, if $\mathcal{U}$ is sorted in decreasing order of profitability, it is possible to detect threshold dominance using the criterion 10. When an item is threshold dominated, remove it from $\mathcal{U}$.

Standard Phase The slice-wise evaluation is performed repeatedly until $\mathcal{R}$ becomes empty. This concludes the *reduction phase*. Assuming that the capacity c and the period level have not yet been reached (otherwise we are done), we move on to the *standard phase*. All that remains is to continue to evaluate the recurrence, and this done with *constant-height* slices (but without the preprocessing above). As before, we detect threshold dominance at the end of each slice, and thus $\mathcal{U}$ may continue to decrease. Each time this occurs we test to see if $\mathcal{U}$ is a singleton. When this happens, we have attained periodicity. The solution is computed with Eqn. 1.3 and we stop.

Threshold detection We first introduce a function $l(i,y)$ as follows. Let us choose an order $\succeq$ compatible with the profitability pre-order $\sqsupseteq$.

Definition 9 *For any item i and any capacity y, the value $l(i,y)$ is given by :*

$$l(i,y) = \begin{cases} 0 & \text{if } y < w_i \\ l(i,y-1) & \text{if } y \geq w_i \quad \text{and } F(y,N) > F(y-w_i,N) + p_i \\ l(i,y-1) & \text{if } y \geq w_i \quad \text{and } F(y,N) = F(y-w_i,N) + p_i \\ & \qquad \text{and } \exists k \succeq i \neq k \text{ such that } l(k,y) = y \\ y & \text{if } y \geq w_i \quad \text{and } F(y,N) = F(y-w_i,N) + p_i \\ & \qquad \text{and } \forall k \succeq i \neq k \text{ we have } l(k,y) < y \end{cases}$$

The function l provides the following sufficient test to detect threshold.

Criterion 10 *If $i \prec\!\!\prec N \backslash \{i\}$, then $t_i \leq min\{y : \ l(i,y) \leq (y - w_i)\}$.*

We can easily observe that the best object type w.r.t the order $\succeq$ (denoted below by b), is the only object type for which the threshold is not detected using criterion 10. Note that if several object types have the same best profitability, the actual b depends of the chosen order $\succeq$. It is an open problem to determine which particular choice allows earliest detection.

3.1 A sparse representation

Yet another method of search space reduction is the so called ***sparse representation*** [2]. A complete discussion is beyond the scope of this paper. We simply give an overview here. It is based on the key observation that the function $g(j,k)$ is monotonically increasing in j. As a result, there is no need to compute the recurrence for *all* values of j, but only at those j indices where its value changes. If this is done, the value of $g(j,k)$ over the entire k-th column can be represented as a sequence of pairs (s,f), similar to the representation used to store and manipulate sparse matrices (hence the name). A similar property also holds for the 0/1 knapsack problem, and an algorithm using a sparse representation (LIFO lists) was proposed at least twenty years ago [9], but this had not previously been used for UKP. The sparse representation, which is fully exploited in our algorithm requires *lazy* data structures known in functional languages as lazy lists, (or streams). For this reason, EDUK is implemented a programming language in the functional style.

3.2 Backtracking phase

In our entire discussion so far, we have not mentioned the *backtracking* phase of the dynamic programming algorithm, which consists of determining the actual solution, $\mathbf{x}$, once the recurrence $g(j,k)$ has been evaluated. A common criticism of DP is that like its worst case running time, its *space complexity* is claimed to be $\Theta(nc)$, i.e., the entire table has to be saved in order to compute the solution. Often overlooked is the fact that, for the *unbounded* knapsack problem, we can compute an auxiliary recurrence during the forward phase, and then, backtracking can be performed by traversing only the *last* column of the table. This was initially presented in Hu's text in 1969 [10]. Another algorithm was presented by Garfinkel and Nemhauser [6]. Both algorithms use an auxiliary function during the forward phase and thus two lists of size c are kept in the memory. We achieve the same space and time complexity as these, but with *no* overhead in the forward phase. In addition, our technique enables us to determine *all* solutions, unlike the others.

We start with recursions and notations similar to these of Garfinkel and Nemhauser [6], but we use them in a slightly different manner and also take into account the notion of dominance. Our algorithm is based on the following.

Remark 11 *Let* $\mathcal{D}(y) = \{k : x_k > 0$ *in some* $\boldsymbol{x} \in Opt(UKP_y^N)\}$. *Then when* $F(y,N) \neq 0$ *and for some* $k \in N$ *such that* $y - w_k \geq 0$ *we have* $F(y,N) - p_k = F(y - w_k, N) \Leftrightarrow k \in \mathcal{D}(y)$.

Clearly, the remark is correct if N were replaced by the set of object types that are not collectively dominated (defined as $\Omega = \{i \in N : i \not\ll N \backslash \{i\}\}$. Because collective dominance is maximal, no solution is lost when using Ω, which is known at the end of the forward phase and is often very small. Furthermore, the choice of k allows us to systematically determine *all* solutions. A useful heuristic for rapidly obtaining one solution is to follow the order of the decreasing profitability of object types (i.e., the best object type to be considered first). Finally, the algorithm above can be directly adapted to exploit sparsity, and this is implemented in the working version of our algorithm.

4 On benchmarks for the UKP

4.1 Random Examples

Martello and Toth [12] describe three kinds of uniformly random data sets. In the *uncorrelated* case, the weights and profits are both chosen randomly, independent of each other. In the *strongly correlated* case, the weights are chosen randomly, but the profit is an affine function of the weight (i.e., $p_i = aw_i + b$ for some predetermined constants, a and b). In *weakly correlated* data sets the w_i's are random, and p_i is chosen randomly in the interval $aw_i \pm b$. The interval for weights used

A randomly selected object type may be viewed as a point (w, p), in the rectangle defined by $w_{\min}, w_{\max}, p_{\min}$ and $p_{\max}$. Now, any other object type in the shaded region either dominates or is dominated by the first point. Specifically, it multiply (respectively simply) dominates (w, p) in region (i) resp. ii) and it is simply (multiply) dominated *by* (w, p) in region (iii, resp. iv).

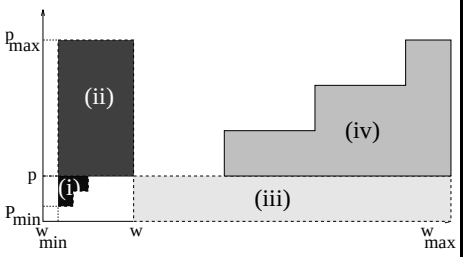


FIG. 1.3 – Why completely uncorrelated data sets are unrealistic for the UKP

by Martello and Toth is $[10, 10^3]$ and for weakly and strongly correlated data sets, the constants are $a = 1$ and $b = 100$.

Johnson and Khan [11] prove that for uncorrelated data sets, there are very few object types that are not multiply dominated, and that the expected value of the number of (multiply) non-dominated object types is as low as 1.6 ! An intuitive explanation is given in Fig 1.3. As a result, algorithms that exploit (at least) multiple dominance such as MTU2 and EDUK are bound to have excellent performance on such data sets. Indeed, it is almost inconceivable that in real life, a heavier object has smaller profit. We define ***realistically random*** data sets are those that preclude simple dominance (by fiat). Such data sets were introduced by Sinha and Zoltners [15] for the multiple choice knapsack problem. They may be generated as follows : first generate n distinct values of weights, randomly in the interval $[w_{\min}, w_{\max}]$, and n values of profits randomly in the interval $[p_{\min}, p_{\max}]$. Then (i) sort these two sequences, (ii) pair up the weights and profits, and (iii) permute this list randomly (or return it to the original order in which the weights (or profits) were generated). Although such data sets preclude simple dominance, the other three dominance relations may be present.

A special case of the strongly correlated data set deserves to be considered, namely the so-called *subset sum problem* (SSP) when weights are equal to profits. It has been observed [14, 17] that for (SSP), the number of *multiply* non dominated object types grows rapidly with n. However, it is easy to check that the number of *collectively* non dominated object types tends to $w_{\min}$ when n grows.

4.2 Hard Problems

Hard problems may be built with formulæ that ensure that no object type is collectively dominated by the others. Note that if the profit monotonically increases with the weight, then there is no simply dominated object type. Furthermore, if $w_{\max} < 2w_{\min}$, no object type is collectively dominated. Three such data sets can be constructed as follows : choose weights randomly in the interval $[w_{\min}, 2w_{\min}[$, and sort them in increasing order, set the profits with any one of Eqns. (1.5–1.7) below, and finally, permute the weight-profit pairs randomly. It is easy to verify that in all three cases, no object type is collectively dominated. These formulæ are distinguished by the fact that the *profitability* of the object types is, respectively, constant, decreases or increases with the weight.

$$p_i = w_i \qquad \text{this is actually SSP} \tag{1.5}$$

$$p_i = max\left(1 + p_{i-1}, \left\lfloor \frac{w_i p_{i-1}}{w_{i-1}} \right\rfloor\right) \quad \text{with } p_1 \text{ chosen randomly} \tag{1.6}$$

$$p_i = \left\lfloor \frac{w_i p_{i-1}}{w_{i-1}} \right\rfloor + i - 1 \quad \text{with } p_1 \text{ chosen randomly} \tag{1.7}$$

5 Experimental Results

An experimental study should ideally vary all parameters (n, c, $w_{\min}$, $w_{\max}$, $p_{\min}$, $p_{\max}$, and the correlation between the weights and profits and the capacity). Pisinger recently observed [14] that increasing the value of $w_{\min}$, augments the number of multiply non dominated object types. However, we have not found *any* experimentation in the literature with respect to the capacity parameter. Our experiments confirm that this is a very important parameter. We performed the following kinds of computational experiments :

- standard random data sets [5, 12].
- similar sets but with significantly larger range of weights (in order to preclude the "false" advantages due to simple dominance).
- ***realistically random*** data sets.
- hard instances that we discovered during our other experiments, and also those generated by Eqns. (1.5-1.7).
- data drawn from real life cutting stock problems.

Complete details of these experiments are reported elsewhere [1], and all experimental data is available on the web. In the interests of brevity, we only present some highlights here. The experiments were run on a 64 bit DEC/Alpha 2100 5/250 machine. Our code was written in the `objective caml` language[1]. All reported times are in CPU-seconds. For comparison purposes, we also give the results of the running time (on the same machine) with the standard branch and bound algorithm, MTU2 due to Martello and Toth [12], written in `fortran 77`, available in the public domain. It is recognized as the *de facto* standard program for the UKP.

5.1 Random examples with large weight range

Despite the limitations mentioned in Section 4, such data sets are widely used, and are *de rigeur* in comparing algorithms for the knapsack and related problem(s). Hence we first present a series of experiments using these data sets, but with an effort to reduce the effects of duplicates. Our goal was to show a progression of increasing running times and compare the performance of the two algorithms. Based on these observations, we chose to fix n at a reasonably large value and allowed $w_{\min}$ to vary. In particular, we used the following parameters : $n = 10^5$, $w_{\max} = 10^5$, and $w_i \in [w_{\min}, w_{\max}]$, with $w_{\min}$ varying between 1 and 10^4. The capacity was fixed to a large value greater than 2×10^8. We generated the three standard data sets (uncorrelated, weakly and strongly correlated).

EDUK and MTU2 are both understandably very fast for uncorrelated data sets, since the number of undominated items is low (no more than 9 for $n = 10^5$). The results for weakly and strongly correlated data are summarized in Fig. 1.4. The column UD gives the number of (collectively) non-dominated object types. Note that the number of multiply non-dominated object types determined by MTU2 is not available, but is no larger than this value. As can be seen, when $w_{\min} > 300$, EDUK consistently outperforms MTU2.

[1] A language in the ML family (see `http ://pauillac.inria.fr/caml`)

w_{min}	UD	eduk	mtu2
1	1	0.150	0.053
3	2	0.158	0.053
10	6	0.158	0.059
30	18	0.161	0.058
100	64	0.168	0.062
300	187	0.313	$10 > 2000$
1000	645	2.638	$7 > 2000$
3000	1932	32.315	$8 > 2000$
5000	6026	108.516	$10 > 2000$
7000	10642	242.971	$10 > 2000$
9000	16639	444.568	$10 > 2000$
10000	23355	564.393	$7 > 2000$

Strongly correlated, $p_i = w_i + 100$

w_{min}	UD	eduk	mtu2
1	1	0.161	0.065
3	1	0.171	0.135
10	3	0.165	0.074
30	5	0.161	0.056
100	10	0.151	0.059
300	30	0.183	0.056
1000	87	0.281	0.118
3000	260	1.293	1.191
5000	471	2.834	$1 > 2000$
7000	1084	5.123	$2 > 2000$
9000	1890	8.668	$4 > 2000$
10000	2776	10.960	$5 > 2000$

Weakly correlated
$p_i \in [w_i - 100, w_i + 100]$

Running time in CPU seconds and the number of (collectively) non dominated object types vs $w_{\min}$, for $n = 10^5$, $w_{\max} = 10^5$, $c > 2 \times 10^8$, for weakly correlated (right) and strongly correlated (left) data sets. Each table entry is the average of 10 runs (rounded down for the *UD* column). The entry $m > 2000$ indicates that m out of the 10 problem instances took more than 2000 seconds (timed out).

FIG. 1.4 – Comparison of EDUK and MTU2 for the standard random experiments.

5.2 Realistic Random Data Sets

We ran 480 such instances as follows : $w_i \in [w_{\min}, w_{\max}]$, $w_{\max} = 10^5$, $p_i \in [1, 2{\times}10^5]$, $c > 2{\times}10^8$. For each $n \in \{5000, 10000, 15000, 20000, 25000, 30000\}$ and $w_{\min} \in \{1, 3, 10, 30, 100, 300, 1000, 3000\}$ we generated 10 instances. These experiments demonstrate [1] that this family of problems is difficult for both EDUK and MTU2 although EDUK remains faster. In particular, 20% (respectively, 40%) of the instances required more than 1000 seconds to be solved by EDUK (respectively, MTU2). This confirms the better performance of EDUK and also validates the hypothesis that realistic random data sets are appropriate for experimentation.

5.3 Hard Problems

As mentioned before, hard problems may crop up during other experiments. We found one such instance while generating realistic random examples. The parameters were $n = 10^5$, $w_{\min} = 30$, $w_{\max} = 10^5$, $p_{max} = 10^6$ and $c = 10^6$. It turned out that this example had 22,077 (collectively) non dominated object types, and the period level was greater than c. EDUK took 3,430 seconds to solve it, and MTU2 failed to solve it in 20,000 seconds. We also built a number of sub-problems of this hard example, all of which turned out to be hard for both EDUK and MTU2 (although EDUK consistently outperformed MTU2). We also observed that these problems tend to be harder (for both algorithms) when the weights were distributed close to each other than when they are widely distributed.

The performance of EDUK and MTU2 on problems generated with the formulæ of Eqns. (1.5-1.7) is reported Fig. 1.5. It shows that the problem remains difficult for both algorithms. For (1.6) and (1.7) EDUK is far better than MTU2. However, for (1.5), i.e., the SSP with $w_{\max} < 2w_{\min}$, MTU2 has significantly better performance. On the other hand, we observed that MTU2 is very

Eqn. (1.5)			Eqn. (1.6)			Eqn. (1.7)		
n	eduk	mtu2	n	eduk	mtu2	n	eduk	mtu2
1000	22.63	0.02	1000	2.94	$4 > 10000$	1000	22.978	$7 > 10000$
2000	42.50	0.015	2000	10.80	$6 > 10000$	2000	43.57	$8 > 10000$
4000	103.52	0.02	4000	49.97	$10 > 10000$	4000	105.27	$10 > 10000$
5000	152.25	0.017	5000	86.85	$10 > 10000$	5000	154.15	$10 > 10000$
10000	580.55	0.016	10000	428.27	$10 > 10000$	10000	612.32	$10 > 10000$
20000	2570.07	0.025	20000	1974.34	$10 > 10000$	20000	615.85	$10 > 10000$

Each line is the average of 10 runs for different values of the capacity.

FIG. 1.5 – Hard problems generated by formulæ

sensitive to the initial order of the object types—if they are *sorted in increasing order of weights*, then MTU2 is unable to solve even a single problem in less than 10,000 seconds. It is important to note that the better performance of MTU2 is *not due to dominance* (otherwise EDUK would also have done better), but some inherent advantage of branch and bound and/or the heuristics used.

5.4 Other Examples

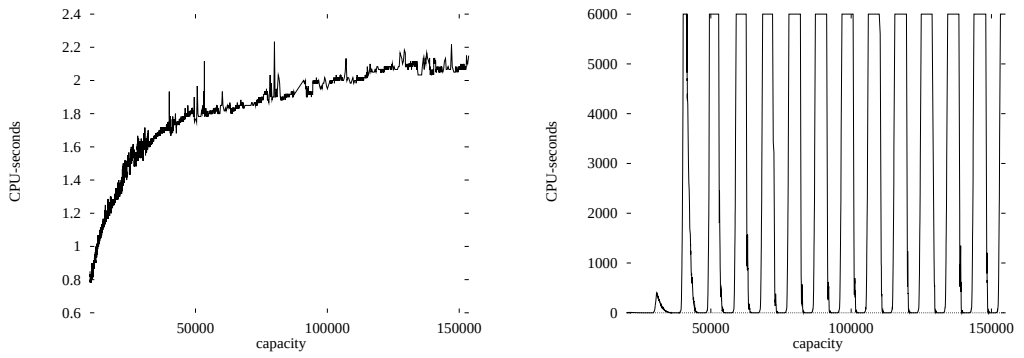
In practice, one often encounters problems with small capacity (the UKP often arises as a frequently called subproblem in other combinatorial optimization problems). For these problems, it seems that preprocessing may not be prohibitively expensive, and the overhead of computing by slices may be high. For this reason we wanted to compare the EDUK algorithm with a straightforward dynamic programming algorithm, with a preprocessing step removing object types according to simple dominance, $\ll_s$. We have done that comparison with a set of 265 problems generated with a generator that Valério de Carvalho and Roderiguez [16] have kindly allowed us to use. The values shown below are the average values of : the number of object types, the capacities, the time of EDUK, the time for MTU2, the time for SDP (standard dynamic programming plus sorting) and the number of non-dominated object types (using the two dominance relations). We give also the minimum and maximum time for EDUK and MTU2. They lead us to conclude that the EDUK algorithm is better than the standard DP with preprocessing to remove (simply) dominated object types.

n	c	EDUK	MTU2	SDP	$\ll$	$\ll_s$	$eduk_{min}$	$mtu2_{min}$	$eduk_{max}$	$mtu2_{max}$
1288	1035	0.033	0.023	0.039	124.61	125.17	0.	0.	0.066	0.438

5.5 Sensitivity to capacity

As mentioned before, there has been no previous study of the performance of UKP algorithms with respect to the capacity. Furthermore, dynamic programming approaches are (often unjustly) rejected out of hand for large capacities, and it is thus important to study how the capacity affects the running time. Our preliminary experiments indicated that there was significant difference in the performance of the two programs. The time required by EDUK was stable, the largest instance being twice as slow as the fastest one. On the same examples, the running times of MTU2 varied by a factor of 2500.

In order to try to illustrate the behavior more precisely, we conducted 30000 similar runs for a particular realistic random data set with parameters $n = 1000$, $w_{\min} = 10^3$, $w_{\max} = 10^4$, $p_{\min} = 1$



We set the experiment to time out beyond 6000 seconds; note that EDUK takes less than 3 seconds on *all* instances.

FIG. 1.6 – Capacity Sensitivity of EDUK (left) and MTU2 (right)

and $p_{\max} = 10^5$. The capacity was uniformly randomly chosen in the range $[w_{\max}, 1550000]$. Fig 1.6 shows the running time of EDUK and MTU2 as a function of the capacity, and the difference is startling. An interesting observation is that the running time of MTU2 is nearly periodic with a period equal to the weight of the best object type (here, about 10000) while that of EDUK is increasing from $w_{\max}$ to the period level (here : 146782) ; then it becomes constant. Let us note that in the considered example no single object is collectively dominated, i.e. EDUK takes advantages on the threshold dominance and the sparsity only.

We also observe that in the solution vectors, at most 10 of the 1000 x_i's are non-zero, and (apart from the best object type) they contribute only a few times. The faster instances of MTU2 correspond to cases when the solution is obtained essentially with the best object type (and a few others are selected once), while the poorer performance occurs when the other object types contribute a few more times (and the running time seems to be combinatorial with respect to these).

In addition to reducing the running time, exploiting *sparsity* also provides a memory savings. In general, the memory required grows linearly with capacity, until y^*, after which it remains constant. Our experiments showed [1] that using sparsity reduces the memory to about 60% of the dense case (the dense algorithm used 15,000 to 140,000 integers (1 Mb of memory) while the sparse one used 10,000 to 80,000).

6 Conclusions

We presented a fast and optimized dynamic programming algorithm, EDUK for the UKP. Its main theoretical foundation is a new dominance relation called *threshold dominance*, which occurs when a positive linear combination of the other object types collectively has a larger profit and lower weight than a *finite number*, say α, instances of a given object type. Hence no optimal solution will have more than α copies of this object type, and we can discard this object type from consideration beyond a capacity of α times its weight. Being a strict generalization of all previously known dominances, it prunes out more object types, at the expense of resolving of a knapsack subproblem.

In addition, we showed how to completely and efficiently exploit the other optimizations (sparsity, periodicity and efficient backtracking). EDUK is very insensitive to capacity and has the ad-

vantages inherent to dynamic programming : knowledge of the solution for *any* capacity lower than the given capacity ; and ability of *reuse* the known solutions for solving *larger* capacity problems.

A number of authors have studied dominance as a preprocessing step for other algorithms, usually branch and bound [12, 5, 14, 4, 17]. Their computational experiments show that even very large randomly generated instances have few undominated items and can be reduced to small core problems, *provided* $w_{\min}$ *is small*. However, they usually simply stop with a study of dominance. For example, though dominance relations of Pisinger [14] and Zhu and Broughan [17] reduce more object types than multiple dominance [12] we do not know what gains they bring to the performance of MTU2 on the UKP. In this paper, we treated dominance not as an end in itself, but rather as a means to an end, namely resolution of the UKP.

The following aspects differentiate our work from previous approaches : (i) threshold dominance ; (ii) sparse representation ; (iii) incorporation of dominance detection in the algorithm itself ; (iv) determining *all* solutions.

We have identified a number of non-trivial data sets, and a number of examples are available at our web site[2]. It would be interesting to compare other exact algorithms on such data sets. One candidate is an algorithm that reduces the UKP to testing the solvability of a single linear diophantine equation [3].

Acknowledgment : The authors thank Nicola Yanev for many discussions which contributed significantly for the current version of the paper.

7 References

[1] R. Andonov, V. Poirriez, and S. Rajopadhye. Unbounded knapsack problem : dynamic programming revisited. Internal Report 1152, IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France, 1997. This report superseeds RR-96-07, October, 1996, LIMAV, Université de Valenciennes("Efficient Dynamic Programming for the Unbouded Knapsack Problem").

[2] R. Andonov and S. Rajopadhye. A Sparse Knapsack Algo-tech-cuit and its Synthesis. In P. Cappello, R. Owens, E. Swartzlander, and B. Wah, editors, *International Conf. on Application Specific Array Processors ASAP'94*, pages 302–313. IEEE, 1994. San Francisco, California.

[3] D. Babaev, F. Glover, and J. Ryan. A new knapsack solution approach by integer equivalent aggregation and consistency determination. *INFORM Journal on Computing*, 9(1) :43–50, 1997.

[4] D. Babaev and S. Mardanov. Reducing the number of variables in integer and linear programming problems. *Computational Optimization and Applications*, 3 :99–109, 1994.

[5] K. Dudzinski. A note on dominance relation in unbounded knapsack problem. *Operations Research Letters*, 10 :417–419, 1991.

[6] R. Garfinkel and G. Nemhauser. *Integer Programming*. John Wiley and Sons, 1972.

[7] P. C. Gilmore and R. E. Gomory. The Theory and Computation of Knapsack Functions. *Operations Research*, 14 :1045–1074, 1966.

[8] P.C. Gilmore and R.E Gomory. A Linear Programming Approach to the Cutting Stock Problem- part II. *Operations Research*, 11 :863–888, 1963.

[9] E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, Maryland, USA, 1978.

[10] T. C. Hu. *Integer Programming and Network Flows*. Addison-Wesley, 1969.

[2] `http ://www.univ-valenciennes.fr/limav/knapsacks/ukp/`

[11] R. Johnson and L. Khan. A note on dominance in unbounded knapsack problems. *Asia-Pacific Journal of Operational Research*, 12 :145–160, 1995.

[12] S. Martello and P. Toth. *Knapsack Problems : Algorithms and Computer Implementation* . John Wiley and Sons, 1990.

[13] G. L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Willey & Sons, 1988.

[14] D. Pisinger. Dominance relations in unbounded knapsack problems. Technical Report 94/33, DIKU, University of Copenhagen, DK-2100 Copenhagen, Denmark, December 1994.

[15] A. Sinha and A. A. Zoltners. The multiple-choice knapsack problem. *Operations Research*, 27 :503–515, 1979.

[16] J.M. Valério de Carvalho and A.J. Rodrigues. An LP-based Approach to a Two-Stage Cutting Stock Problem. *European Journal of Operational Research*, 84 :580–589, 1995.

[17] N. Zhu and K. Broughan. On dominated terms in the general knapsack problem. *Operations Research Letters*, 21 :31–37, 1997.

Références bibtex :

```
@article{EDUK2,
   author = "Poirriez, V. and Yanev, N., andn Andonov, R.",
   title = "EDUK2: A Powerful Tool for solving UKP",
   year = {2006}
}
```

EDUK2 : A Powerful Tool for solving UKP

V. Poirriez, N. Yanev, R.Andonov

Adresses

LIMAV, Université de Valenciennes
Le Mont Houy, B.P.311, 59304 Valenciennes Cedex, France
Faculty of Mathematics and Informatics, University of Sofia, Bulgaria
IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France

Abstract This paper presents a new approach for solving the Unbounded Knapsack Problem (UKP) exactly and proposes a new bound that has proved to dominate the previous bounds on a special class of UKP instances. Integrating bounds within the framework of sparse dynamic programming led to the creation of an efficient and robust hybrid algorithm, called `EDUK2`. This algorithm takes advantage of the majority of the known properties of UKP, particularly the diverse dominance relations and the important periodicity property. Extensive computational results show that `EDUK2` significantly outperforms both MTU2 and EDUK, the currently available UKP solvers in all but a very few cases. These experimental results demonstrate that the class of hard instances needs to be redefined, and the authors offer their insights into the creation of such instances.

Keywords Algorithm Engineering, Computational Optimization, Integer Programming, Knapsack problem, Branch and Bound, Dynamic programming.

1 Introduction

The knapsack problem is one of the most popular combinatorial optimization problems. Its unbounded version, UKP (also called the integer knapsack), is formulated as follows : there is a knapsack of a *capacity* $c > 0$ and n types of items. Each item of type $i \in N = \{1 \dots n\}$ has a *profit*, $p_i > 0$, and a *weight*, $w_i > 0$. The problem, $\mathbf{UKP}^c_{w,p}$, is to fill the knapsack in an optimal way, which is done by solving

$$f(\mathbf{w}, \mathbf{p}, c) = \max \{\mathbf{px} \text{ subject to } \mathbf{wx} \leq c, \mathbf{x} \text{natural integers vector}\} \tag{2.1}$$

where $\mathbf{w}, \mathbf{p}$ and $\mathbf{x}$ denote vectors of size n.

Many of this problem's properties have been discovered over the last three decades : [1, 3, 5, 8, 7, 10], but no existing solver has yet been developed that benefits from all of them. (For a more comprehensive state-of-the art discussion see the very recent monograph [9].) In this paper we introduce *a new upper bound* and determine a UKP family for which this bound is the tightest one known. We present a new algorithm that combines dynamic programming and branch-and-bound methods to solve UKP[1]. To the best of our knowledge this is the first time that such an approach has been used for the UKP. Our extensive computational experiments demonstrate the effectiveness

[1]The U_v bound results has already been presented in a research report [11] and partial results have been used by others in [9].

of embedding a branch-and-bound algorithm into a dynamic programming.[2] Our results also shed light on the case of really hard UKP instances.

The paper is organized as follows. Section 2 briefly summarizes the basic properties of the problem. Section 3 presents the new upper bound and its corresponding class. Section 4 is dedicated to the description of the new algorithm, which takes advantage of all known dominance relations and successfully combines them with a variety of bounds, and this new algorithm is compared to other UKP solvers in Section 5. In Section 6 we conclude.

2 A summary of known dominance relations and bounds

The dominance relations between items and bounds allow the size of the search space to be significantly reduced. All the dominance relations, enumerated below, could be derived by the following inequalities :

$$\sum_{j\in J} x_j w_j \leq \alpha w_i, \text{ and } \sum_{j\in J} x_j p_j \geq \alpha p_i \text{ for some } \mathbf{x} \in Z_+^n \tag{2.2}$$

where $\alpha \in Z_+$, $J \subseteq N$ and $i \notin J$.

1. *Dominances*
   (a) *Collective Dominance* [1, 12]. The i-th item is **collectively dominated** by J, written as $i \ll J$ iff (2.2) hold when $\alpha = 1$. The verification of this dominance is computationally hard, so it can be used in a dynamic programming approach only. To the best of our knowledge `EDUK` (Efficient Dynamic programming for UKP) [1] is the only one that makes practical use of this property.
   (b) *Threshold Dominance* [1]. The i-th item is **threshold dominated** by J, written as $i \prec\!\!\ll J$ iff (2.2) hold when $\alpha \geq 1$. This is an obvious generalization of the previous dominance by using instead of single item i a compound one, say α times item i. The smallest such α defines the **threshold** of the item i, written t_i, as $t_i = (\alpha - 1)w_i$.
   The lightest item of those with the greatest profit/weight ratio is called *best item*, written as b. One can trivially show that $t_i \leq w_b w_i$ or even sharper inequality $t_i \leq lcm(w_b, w_i)$ where $lcm(w_b, w_i)$ is the least common multiple of w_i and w_b.
   (c) *Multiple Dominance* [7]. Item i is **multiply dominated** by j, written as $i \ll_m j$, iff for $J = \{j\}, \alpha = 1$, $x_j = \lfloor \frac{w_i}{w_j} \rfloor$ the inequations (2.2) hold.
   This dominance could be efficiently used in a preprocessing because it can be detected relatively easily.
   (d) *Modular Dominance* [12]. Item i is **modularly dominated** by j, written as $i \ll_{\equiv} j$ iff for $J = \{b, j\}, \alpha = 1, w_j = w_i + tw_b, t \leq 0$, $x_b = -t, x_j = 1$ the inequalities (2.2) hold.
2. *Bounds*
   U_3 **[7]** It is assumed here that the first three items are of the largest profit/weight ratio.

$$\begin{aligned} U_3 &= max\{U^0, \bar{U}^1\} \\ \text{where : } z' &= \left\lfloor \frac{c}{w_1} \right\rfloor p_1 + \left\lfloor \frac{\bar{c}}{w_2} \right\rfloor p_2;\ \bar{c} = c \bmod w_1;\ c' = \bar{c} \bmod w_2 \\ U^0 &= z' + \left\lfloor \frac{c' p_3}{w_3} \right\rfloor \\ \bar{U}^1 &= z' + \left\lfloor \left(c' + \left\lceil \frac{w_2 - c'}{w_1} \right\rceil w_1 \right) \frac{p_2}{w_2} - \left\lceil \frac{w_2 - c'}{w_1} \right\rceil p_1 \right\rfloor \end{aligned} \tag{2.3}$$

[2] `EDUK2` is free open-source software available. at :
`http ://download.gna.org/pyasukp/` where it is denoted by PYAsUKP.

U_s **[3]** . $U_s = c + \left\lfloor \frac{c}{w_1} \right\rfloor \alpha$. This bound is stronger than U_3 for the class of strongly correlated UKP (**SC-UKP**) defined as $p_i = w_i + \alpha$ where $\alpha > 0$. Here item 1 is supposed to be the lightest one.

U_v **[11]** . $U_v = c + max \left\{ \frac{(p_i - w_i)}{\lfloor \frac{w_i}{w_1} \rfloor}, i \in N \right\} \left\lfloor \frac{c}{w_1} \right\rfloor$. Here again item 1 is supposed to be the lightest one. This bound is stronger than U_3 for a special class of UKP (namely **SAW-UKP** see def. 1 below)

3 A new general upper bound for UKP

In following paragraphs, we introduce a new upper bound for the UKP and show that it improves U_v and is not comparable to U_3 in the general case. For the special UKP family, the *SAW-UKP*, which includes the **SC-UKP** class (with $\alpha \geq 0$), this new bound is tighter than the previously known bounds.

Without losing generality it is assumed in this section that : 1 is the lightest item within the set of items with $(p_i - w_i) \geq 0$ (i.e. $\forall\, i > 1,\ w_1 \leq w_i$ or $p_i \leq w_i$) and $p_1 \geq w_1 + 1$. (If all $p_i - w_i \leq 0$ then assume 1 is the item with the best ratio and by changing p to $\psi p, \psi = \left\lceil \frac{w_1+1}{p_1} \right\rceil$ we will achieve the goal. If such an equivalent transformation is done, the bound should be divided by ψ). It is also assumed that no item is multiply dominated. Let us define the following terms :
for all $i \neq 1, q_i = \frac{p_i - p_1 \left\lfloor \frac{w_i}{w_1} \right\rfloor}{w_i \bmod w_1}$ if $w_i \bmod w_1 \neq 0$, $q_i = +\infty$ if $w_i \bmod w_1 = 0$
$q^* = max_{i \neq 1} \{q_i\}$, $\tau^* = min\{1, q^*\}$, $\beta(\tau) = max_{i \in N} \left\{ \frac{(p_i - \tau w_i)}{\lfloor \frac{w_i}{w_1} \rfloor} \right\}$, $\beta^* = \beta(\tau^*)$.

Theorem 1 *[U_{τ^*}]* *for all* $UKP^c_{w,p}$, $f(\mathbf{w}, \mathbf{p}, c) \leq U_{\tau^*} = \tau^* c + \beta^* \left\lfloor \frac{c}{w_1} \right\rfloor \leq U_v$

Proof

First, for a fixed τ, $0 \leq \tau \leq 1$,

$$max\{\tau \mathbf{wx} + (\mathbf{p} - \tau \mathbf{w})\mathbf{x}, \mathbf{wx} \leq c\} \leq \tau c + max\{(\mathbf{p} - \tau \mathbf{w})\mathbf{x}, \mathbf{wx} \leq c\} \quad (2.4)$$

The above inequality was suggested by the classical *cost- splitting* optimization technique [8]. It leads to solve the $\mathrm{UKP}^{c}_{\mathbf{w},(\mathbf{p}-\tau\mathbf{w})}$.

Case $\tau^* = max\{q_i, i \neq 1\} \leq \tau \leq 1$ **:**
in this case, in $\mathrm{UKP}^{c}_{\mathbf{w},(\mathbf{p}-\tau\mathbf{w})}$ all items i are multiply dominated by the item 1. Then $\beta(\tau) = p_1 - \tau w_1$. Thus, $(\mathbf{p} - \tau \mathbf{w})\mathbf{x} \leq \beta(\tau)\Sigma_{i \in N} \lfloor \frac{w_i}{w_1} \rfloor x_i$ and $max\{(\mathbf{p} - \tau \mathbf{w})\mathbf{x}, \mathbf{wx} \leq c\} \leq \beta(\tau) \left\lfloor \frac{c}{w_1} \right\rfloor$.
The function $u(\tau) = \tau c + (p_1 - \tau w_1) \left\lfloor \frac{c}{w_1} \right\rfloor$ is an increasing function, and its minimum is reached for $\tau = \tau^*$. This prove both inequalities of the theorem as $U_v = u(1)$ and $U_{\tau^*} = u(\tau^*)$. is an increasing function, and its minimum is reached for $\tau = \tau^*$. This prove both inequalities of the theorem as $U_v = u(1)$ and $U_{\tau^*} = u(\tau^*)$.

Case $max\{q_i, i \neq 1\} > 1 = \tau^*$ **:** in this case,

$$\begin{aligned} \Sigma_{i=1}^{n}(p_i - w_i)x_i &\leq (p_1 - w_1)x_1 + \beta^* \Sigma_{i=2}^{n} \left\lfloor \frac{w_i}{w_1} \right\rfloor x_i \\ &\leq \beta^* \Sigma_{i=1}^{n} \left\lfloor \frac{w_i}{w_1} \right\rfloor x_i \leq \beta^* \left\lfloor \Sigma_{i=1}^{n} \frac{w_i x_i}{w_1} \right\rfloor \leq \beta^* \left\lfloor \frac{c}{w_1} \right\rfloor \end{aligned}$$

and $U_{\tau^*} = U_v = c + \beta^* \left\lfloor \frac{c}{w_1} \right\rfloor$.

Remark 1 *When $q^* > 1$, then almost in all cases the (classical) upper-bound $U = \left\lfloor \frac{p_b c}{w_b} \right\rfloor$ is better than U_{τ^*}. It can be obtained from (2.4) by taking $\tau = max\ \frac{p_i}{w_i} = \frac{p_b}{w_b}$.*

We recall the definition of SAW-UKP first given in [11]

Definition 1 *All $UKP^{c}_{w,p}$ instances in which $q^* \leq 1$ are called **SAW-UKP**.*

Thus a SAW-UKP verifies : $(p_i - w_i) \leq (p_1 - w_1) \left\lfloor \frac{w_i}{w_1} \right\rfloor$.

The following condition is a necessary condition for $\mathrm{UKP}^{c}_{w,p}$ to be a SAW-UKP.

Lemma 1 *If $UKP^{c}_{w,p}$ is a SAW-UKP, then the item 1 is the best one.*

Proof

$\mathrm{UKP}^{c}_{w,p}$ is a SAW-UKP means that $q^* \leq 1$, i.e. for all $i \in N, q_i = \frac{p_i - p_1 \left\lfloor \frac{w_i}{w_1} \right\rfloor}{w_i \bmod w_1} \leq 1$. Then we can derive for all $i \in N$:
$\frac{p_i - p_1 \left\lfloor \frac{w_i}{w_1} \right\rfloor}{w_i - w_1 \left\lfloor \frac{w_i}{w_1} \right\rfloor} \leq 1 \Leftrightarrow (p_i - w_i) \leq (p_1 - w_1) \left\lfloor \frac{w_i}{w_1} \right\rfloor$ which implies
$(p_i - w_i) \leq (p_1 - w_1)\frac{w_i}{w_1} \Leftrightarrow \frac{p_i}{w_i} \leq \frac{p_1}{w_1}$

Thus, it can now be established that U_v is tighter than U_3 for this family of UKP.

Theorem 2 *If $UKP^c_{w,p}$ is a SAW-UKP, then $U_{\tau^*} \leq U_v \leq U_3$*

Proof

It is assumed that the first three items are of the largest ratio, and also that $\frac{p_3}{w_3} \geq 1$ (as above, if it is not the case, changing p to $\psi p, \psi = \lceil \frac{w_3+1}{p_3} \rceil$ achieve the goal).
Since $U_3 = max\{U^0, \bar{U^1}\}$ it is enough to prove $U_v \leq U^0$. Since $w_2 \geq w_1$, $\left\lfloor \frac{c \bmod w_1}{w_2} \right\rfloor = 0$. Thus $z' = \left\lfloor \frac{c}{w_1} \right\rfloor p_1$ and $c' = \bar{c} = c \bmod w_1$.

$$\begin{aligned} U^0 &= \left\lfloor \frac{c}{w_1} \right\rfloor p_1 + \left\lfloor c' \frac{p_3}{w_3} \right\rfloor = \left\lfloor \frac{c}{w_1} \right\rfloor p_1 + \left\lfloor (c \bmod w_1) \frac{p_3}{w_3} \right\rfloor \\ &\geq \left\lfloor \frac{c}{w_1} \right\rfloor p_1 + (c \bmod w_1) = \left\lfloor \frac{c}{w_1} \right\rfloor p_1 + c - \left\lfloor \frac{c}{w_1} \right\rfloor w_1 = \left\lfloor \frac{c}{w_1} \right\rfloor (p_1 - w_1) + c \\ &\geq U_v \end{aligned}$$

Example 1 (A Saw UKP) *n=7; c=2900;N={1;...;7};*
p=*[300;580;301;601;605;322;310];* **w**=*[120;245;130;260;310;194;190].*
We can compute that **q**= [_; -4.; 0.1; 0.05; 0.0714285; 0.297297; 0.142857] (remember that q_1 is not define). Hence $q^* \approx 0.297$ and *Example 1* is therefore a SAW-UKP. The bounds are : $U_{\tau^*} = 7205 < U_v = 7220 < U_3 = 7246$. The optimal value is 7202.

Remark 2 *For a non-SAW-UKP, it is possible for U_v to be stronger than U_3 and vice versa : (c.f. examples 2 and 3)*

Example 2 (A non-SAW-UKP with $U_3 < U_v$) *case where **Remark 1** applies.*
n=6; c=1 619 881; **p** *=[1 001; 1 002; 10 025; 10 026; 11 248; 11 249].*
w=*[1 000; 1 001; 10 023; 10 024; 11 233; 11 234]. Here we obtain :*
$\mathbf{q} = [\_; 1.; 0.6521739; 0.6667; 1.0171673; 1.0170940]$ *and* $q^* = 1.017$.
The bounds are $U_3 = 1\,622\,044 < 1\,622\,088 = U_v = U_{\tau^*}$. *The optimal profit is* 1 622 032 *obtainable with* $x_1 = 8$; $x_2 = 28$; $x_3 = x_4 = x_6 = 0$; $x_5 = 141$.

Example 3 (A non-SAW-UKP with $U_3 > U_v = U_{\tau^*}$) *n=2; c=2 900;*
p=*[297;309];* **w**=*[120;131]. Here,* **q**=*[_;1.0909] hence* $q^* \approx 1.09$ *; and the bounds are* $U_{\tau^*} = U_v = 7172 < U_3 = 7175$. *The optimal profit is* 7140 *obtainable with* $x_1 = 23$; $x_2 = 1$.

4 The new algorithm EDUK2

The algorithm described below is based on a convenient combination of two basic approaches used in UKP solvers, namely dynamic programming **(DP)** and branch and bound ***(B&B)*** methods.

Dynamic programming (DP)
One of the basic recursions used for solving UKP is

$$f(N, y) = max_{j \in J_y} \{f(N, y - w_j) + p_j\} \text{ for } J_y \subseteq N \tag{2.5}$$

The eligible set J_y is supposed to contain at least one item i s.t. $x_i > 0$ in some optimal solution to UKP^y_N. The cardinality of this set is crucially important for the efficiency of any algorithm based on the above formula(2.5). To the best of our knowledge EDUK is the only solver that uses this recursion with obvious efficiency. The main components of its implementation are the computation

of formula (2.5) by slice, a sparse representation of the iteration space, and the use of threshold dominance. Slices are defined as intervals of y, and the sparse representation is based on the stepwise form of the function f. In the following presentation, the couples $(y, f(N, y))$ in which the function value changes will be called *optimal states*.

The *periodicity* property has been described by Gilmore and Gomory [6] as the capacity y^*, called the *periodicity level*, such that for each $y > y^*$, there is an optimal solution with $x_b > 0$. It is well known that, for each UKP_N^∞ such a y^* exists, but its value is not easily detectable. So, although the periodicity property can drastically reduce the search space, it can only be detected in a DP framework, using the following formula :
$y^* < y^+ = min\{y \mid \forall y' > y - w_{\max}, \text{there is an optimal solution with } x_b > 0\}$.

Finally, the fact that **DP** algorithms compute optimal solutions for all values of y below the capacity c allows the recursion to be stopped when the capacity :
$min\{max\{\frac{c}{2}, w_{\max}\}, y^+\}$ is reached.

Branch-and-bound (B&B)
Unlike DP, *B&B* algorithms compute an optimal solution *only* for a given capacity, and are dependent on the quality of the computed upper bounds. The `MTU2` algorithm proposed by Martello and Toth [7] uses the upper bound U_3 and the now well known variable reduction scheme : let z be the objective function value of a known feasible solution, and let $\mathcal{U}$ be an upper bound of $f(N, c - w_j) + p_j$; if $\mathcal{U} \leq z$, then either z is optimal or x_j can be set to zero.

Hybridization of DP and B&B
There are several complementary ways to integrate the knowledge of bounds into the DP process.

1. The first approach is to use the variable reduction scheme in a pre-processing stage to reduce the set N.
2. The second approach consists of computing, for each *optimal state* $(y, f(N, y))$, an upper bound $U(c - y)$ for a knapsack with $c - y$ capacity. If $U(c - y) + f(N, y) \leq z$, then the state can be discarded, or in other words, can be *"fathomed by bounds in the B&B context"*. In this way, if a sparse representation is chosen, fewer states must be computed. This designs a *DP with bounds algorithm.*
3. The third approach consists of solving an UKP_{core}^c using a *B&B* algorithm in which the *core* set is a subset of the items with the best ratios. If $f(core, c) = U(c)$ then the problem is solved. Otherwise, $f(core, c)$ is used during the *DP with bounds algorithm* described above.

4.1 The `EDUK2` algorithm outline

The algorithm `EDUK2` given below is an hybridization of `EDUK` with an arbitrary (but efficient) *B&B* algorithm, following the above description. The basic steps of `EDUK2` are :

step 1 Compute in $O(n)$ time an upper bound U and an initial feasible solution with value z. Discard from N all items multiply dominated by b. They are detected in linear time.

step 2 For the reduced set of items N, apply the variable reduction scheme in $O(\mid N \mid)$ times. Then, select a size C core subset containing the items with the best ratios.

step 3 To improve the lower bound, run a *B&B* algorithm on the core, limiting the algorithm to a maximum of B explored nodes.

step 4 Run the *DP with bounds fathoming states* algorithm (the second integration approach described above).

In the current implementation of EDUK2, a *B&B* similar to the one used by Martello and Toth in MTU2 [7] with the ability to choose the computed upper bound (currently U_v, U_τ or U_3), is used in step 3. An enhanced version of EDUK, which eliminates the fathomed states, operates in step 4.

The EDUK2 parameters, B and C, were experimentally tuned and in the current implementation of the algorithm, their values are C= min{n,max{100, n/100}} and $B = 10000$.

5 Computational results

Computational experiments were run in order to : (i) test the efficiency of the *B&B*/DP pairing and the state discriminating capacity of the new bounds U_{τ^*} and U_v ; (ii) exhibit some actual hard instances. Unfortunately, very few real-life instances of **UKP** have been reported in the literature. For this reason we concentrated our efforts on a set of benchmark tests using : (a) random profit and/or weight generation with some correlation formulae; (b) hard data sets that were specially designed for the *B&B* approach [4].

Very few other UKP solvers are available for comparison with EDUK2. Though Babayev *et al.* [2] have proposed an integer equivalent aggregation and consistency approach (CA) that appears to be an improvement over MTU2, this approach applies neither threshold dominance nor the new tight upper-bound. Caccetta & Kulanoot [3] have recently described two specialized algorithms for solving two particular classes of UKP : CKU1 for Strongly Correlated UKP (SC-UKP) and CKU2 for Subset Sum Problem(SS-UKP). However, these algorithms are not applicable to the general UKP. Thus, we chose to compare EDUK2 with the only two publicly available solvers : EDUK [1], which is considered to be the most efficient DP algorithm [9], and MTU2, a *B&B* solver [7]. All instances are free from simply dominated pairs $(p_i, w_i), (p_j, w_j)$, s.t. $p_i \geq p_j$ and $w_j \leq w_j$, and weights and profits are positive integers.

We start by a comparison of the behaviors of MTU2, EDUK and EDUK2 on classic data sets, then we focus on comparing EDUK with EDUK2 on new hard instances not solvable by MTU2. Finally, in case of SAW UKP, we study the impact on the resolution time when using the new bound U_{τ^*} instead of U_v.

5.1 Classic data sets

A complete study of the classic UKP benchmarks, where the behaviors of EDUK and MTU2 have been compared, can be found in [1]. Most of these UKP appear to be easy to solve by EDUK2, which is why we report only the most interesting subset of the data from our computational results.

EDUK2 and EDUK were written in objective caml 3.08. The respective codes were all run on a Pentium 4, 3.4GHZ with 4GB of RAM, and the time limit for each run was set to 300 sec. MTU2 was executed on the same machine and compiled with g77-3.2. The impact of the bounds was tested by simply substituting the bound U_v in EDUK2 with U_3 in a version called edu$_{U_3}$.

Known "hard" instances

First, we focus on the data sets found to be difficult for MTU2 or EDUK [1].
(A) The SS-UKP instances ($w = p$) are known to be difficult for EDUK.
We built such instances by generating 10 instances for each possible combination of $w_{\min} \in \{100, 500, 1000, 5000, 10000\}$, $w_{\max} \in \{0.5 \times 10^5; 10^5\}$ and $n \in \{1000; 2000; 5000; 10000\}$ with c randomly generated within $[5 \times 10^5, 10^6]$. We obtain in this manner 400 distinct instances. The average cpu time for the different algorithms was :

EDUK2 : $0.045s$;	edu$_{U_3}$: $0.045s$;	EDUK : 0.474 ;	MTU2 : $0.136s$.

According to these results, *EDUK2 is 10 (resp. 3) times faster than EDUK (resp. MTU2).*

We also tested the sensitivity of the algorithms with respect to $w_{\min}$, and the results showed that EDUK2 is much less sensitive to $w_{\min}$ than EDUK. The average time for EDUK increased about 80 times when $w_{\min}$ passed from 100 to 10000, while the EDUK2's average time increased 40 times only.

	EDUK2	EDUK	MTU2
$w_{\min} = 100$	0.005s.	0.025s.	0.042s.
$w_{\min} = 10000$	0.2s.	1.82s.	0.25s.

(B) A set of instances of a *Special SC-UKP* was built according to the formula

$$w_i = w_{\min} + i - 1 \quad \text{and} \quad p_i = w_i + \alpha \qquad \text{with } w_{\min} \text{ and } \alpha \text{ given.} \tag{2.6}$$

Chung *et al.* [4] have shown that solving this problem is difficult for *B&B* . We set $w_{\min} = 1+n(n+1)$ and $n \in \{50; 100; 200; 300; 500\}$, and used both a negative and a positive value for α. For each set, we generated 30 instances with a capacity taken randomly from the interval $[10^6, 10^7]$.

$\alpha > 0$ The average time needed to solve the 150 instances was :

EDUK2 : 3.32s, $\texttt{edu}_{U_3}$: 3.37s ; EDUK : 4.29s.

MTU2 was able to solve only 9 of the 60 instances with $n \in \{50; 100\}$ and none for $n > 100$. NB : *these problems are SAW-UKP.*

$\alpha < 0$ The average time needed to solve the 150 instances was :

EDUK2 : 6.01s ; $\texttt{edu}_{U_3}$:5.93s ; EDUK :8.65s.

MTU2 was able to solve only 10 of the 60 instances with $n \in \{50; 100\}$ and none for $n > 100$. NB : *these problems are not SAW-UKP.*

From these results, it appears that *EDUK2 is 1.3 (resp. 1.45) times faster than EDUK* when $\alpha > 0$ (resp. <0). As expected, these instances were hard for MTU2.

Sensitivity to variations in the capacity : a comparison with EDUK

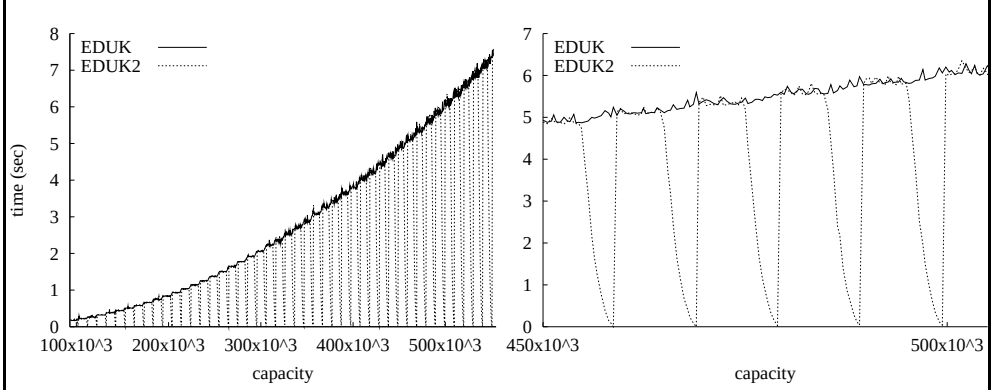
The *B&B* algorithms are known to be very sensitive to variations in the capacity. DP algorithms, on the other hand, are known to be robust, but with computational time that increase linearly with the capacity value. Our computational experiments show that EDUK2 inherits the good properties of both *B&B* and DP. EDUK2's overall computational time is less than the minimum time for the pseudo-polynomial DP time and the *B&B* time, as is shown in **Fig. 2.1**. (Formula (2.6) was used to generate the data for these runs as a *Special SC-UKP.*) As **Fig. 2.1** illustrates, EDUK2 has lost the stable behavior typical of EDUK, but this is not a disadvantage since the time ratio $\frac{\texttt{EDUK}(i)}{\texttt{EDUK2}\ (i)} \geq 1$ is valid for any instance i, and reaches a value of 2.5 for more than 12% of the c values.

SAW-UKP instances

This class contains 880 SAW-UKP instances according to **definition 1**, generated using the following parameters : $c = \frac{1}{10}\sum w$, $w_{\min} \in \{100; 200; 500; 1000\}$, $w_{\max} \in \{10000; 100000; 1000000\}$ and $n \in \{1000; 2000; 5000; 10000\}$. For each of the 44 possible parameter combinations[3], we randomly generated 20 instances, for which we obtained the following average times :

EDUK2 : 0.129s, $\texttt{edu}_{U_3}$: 0.252s ; EDUK : 0.610s.

[3]The combination $n = w_{\max} = 10000$ is not possible due to simple dominance.



Formula (2.6) where $n = 100$, $w_{\min} = n(n+1)+1$, $\alpha = -3$ and c is randomly and uniformly generated between $[90\,000, 560\,000]$. The whole figure is depicted on the left. On the right, a zoom on the sub-interval $[450\,000, 500\,000]$ is shown. On average, `EDUK2` is more than 25% faster than `EDUK`. `MTU2` always requires more than 1200 sec., except for 5% of the points where it requires less than 12 seconds. These are the points where `EDUK2` finds the solution with the *B&B* .

FIG. 2.1 – Capacity sensitivity of `EDUK2` and `EDUK`

We therefore observe that for this family ***EDUK2 is about 5 times faster than EDUK***, and using U_v instead of U_3 accelerates `EDUK2` by a factor of 2. A comparison of efficiency between U_v and U_τ is reported in the section 5.3.

Due to arithmetic overflow `MTU2` was run only on only 200 instances with $w_{\max} = 1000$. For 95 of these instances, it reached the time limit of 300 seconds.

5.2 Do hard UKP instances really exist ?

Based on these results, one is inclined to conclude –wrongly– that **UKP** are easy to solve. It is important to remind that, in the above experiments, instances of moderate size were only considered. A real-life problem of the same size would indeed be easy to solve. However, real problems may have large coefficients, which makes necessary testing the solvers' behavior on such data sets.

New hard UKP instances

In order to construct difficult instances, we considered data sets with large coefficients and/or large number of items. Because `MTU2` cannot be used for such instances because of arithmetic overflow, we restricted our comparisons to `EDUK`, $\texttt{edu}_{U_3}$ and `EDUK2`. For such data sets `EDUK2` and $\texttt{edu}_{U_3}$ benefit of the `num` ocaml library, which provides exact unlimited integer arithmetic to compute the bounds. All the runs were done on a Pentium IV Xeon , 2.8GHZ with 3GB of RAM. CPU time was limited to one hour per instance. If this time limit was reached, we reported 3600 sec. in order to compute the average[4]. We use the notation $x\overline{n}$ to denote $x \times 10^{u+1} + n$, where $0 < \lfloor \frac{n}{10^u} \rfloor < 10$ (e.g. $n = 213, 4\overline{n} = 4213$).

In order to measure the improvement of `EDUK2` in respect to `EDUK` and the influence of the new bound, we use the following metrics :

for each of the three algorithms :

nmd : number of non-multiply dominated items (step 1 of `EDUK2`) ;

ncd : number of non-collectively dominated items (as computed by `EDUK`) ;

[4]The notation $t(k)$ means that the average time is t sec., with k instances reaching the time limit.

cpu : running CPU time in seconds ;

rp : denotes the ratio $\frac{y^+}{c}$ where y^+ is the capacity level where the algorithm detects that the periodicity level y^* is reached.

for `EDUK2` **and** $\texttt{edu}_{U_3}$:

vrs : number of items eliminated in the variable reduction step [5] ;

wdp : number of instances for which the optimal solution was found without using DP (steps 1 to 3) ;

rst : ratio of the number of states in the DP phase (step 4 of `EDUK2`) with respect to the number of states for `EDUK`.

In the tables below, the reported value in the **nmd**, **ncd**, **cpu** columns is the average for the number of instances ; the value in the **wdp** columns refers to the total number of instances ; the value in the **vrs**, **rp** and **rst** columns, reports the average for the number of instances for which the algorithm enters the DP phase.

Instances known to be difficult for *B&B*

We generated large data sets using the formula (2.6). It is easy to see that for such a data set, ncd $= \min(n, w_{\min})$. For a given n, the formula determines n pairs (w_i, p_i), and we generated 20 different values for c (Fig. 2.2).

20 instances per line			`EDUK2`						$\texttt{edu}_{U_3}$					`EDUK`	
α	**n**	$\mathbf{w}_{\min}$	**nmd**	**ncd**	**cpu**	**vrs**	**wdp**	**rst**	**rp**	**cpu**	**vrs**	**wdp**	**rst**	**rp**	**cpu**	**rp**
5	5	10	**n**	**n**	21.77	0(13)	13	0.29	0.047	37.81	642(3)	3	0.38	0.069	80.06	0.108
		15	**n**	**n**	46.57	0(8)	8	0.34	0.099	52.29	83(7)	7	0.56	0.141	111.28	0.188
		50	**n**	**n**	154.19	0(2)	2	0.55	0.470	156.63	0(2)	2	0.68	0.555	261.29	0.661
5	10	10	**n**	**n**	0.03	0(20)	20	-	-	135.22	2420(3)	3	0.54	0.007	336.70	0.008
		50	**n**	**n**	344.12	0(6)	6	0.26	0.037	367.94	0(6)	6	0.41	0.052	915.11	0.079
		110	**n**	**n**	771.53	0(2)	2	0.20	0.112	816.90	0(2)	2	0.26	0.139	2808.50	0.300
-5	5	10	**n**	**n**	64.82	44(6)	6	0.78	0.091	65.14	44(6)	6	0.78	0.091	113.67	0.108
		15	**n**	**n**	104.89	11(2)	2	0.61	0.091	104.62	11(2)	2	0.61	0.091	183.31	0.188
		50	**n**	**n**	232.26	0(8)	8	0.86	0.650	231.13	0(8)	8	0.86	0.650	447.40	0.660
-5	10	10	**n**	**n**	167.26	1317(4)	4	0.67	0.009	170.34	1317(4)	4	0.67	0.009	317.01	0.009
		50	**n**	**n**	508.37	0(6)	6	0.45	0.058	511.36	0(6)	6	0.45	0.058	1539.74	0.079
		110	**n**	**n**	1401.(3)	0(4)	4	-	0.124	1394.(3)	0(4)	4	-	0.124	(20)	-

c is randomly generated between $[20\overline{n}; 100\overline{n}]$.

FIG. 2.2 – Large hard data sets created using formula (2.6). Data from **n** and $\mathbf{w}_{\min}$ columns should be multiplied by 10^3 to get the real value.

`EDUK` had some trouble in solving these sets and was unable to solve the 20 problems with $\alpha = -5, n = 10^4$, and $w_{\min} = 11 \times 10^4$ in less than one hour. In one special case, where $\alpha = 5$ and $n = w_{\min} = 10000$, the solution was always found immediately in the initial variable reduction step, using the bound U_v. Excluding these two special sets, `EDUK2` is on average from 1.7 to 3.7 times faster than `EDUK`. Note that for all these instances, the optimal solution was found by `EDUK2` and $\texttt{edu}_{U_3}$ either in the variable reduction step, either in the DP phase but never in the *B&B* step. Note that `EDUK2` was 1.01 to 1.7 times faster than $\texttt{edu}_{U_3}$when $\alpha > 0$ (these instances belong to the SAW-UKP family). However, in the case $\alpha < 0$ `EDUK2` and $\texttt{edu}_{U_3}$ behave very similarly.

Data sets without simply dominated items

For the data in (Fig. 2.3), w_i were randomly generated between $[w_{\min}; w_{\max}]$, and p_i values were generated using $p_1 \in [w_1; w_1 + 500]$, $p_i \in [p_{(i-1)} + 1; p_{(i-1)} + 125]$. c was randomly generated between $[w_{\max}; 2 \times 10^6]$. Clearly, for these instances, the number of non-collectively dominated

[5] The notation $x(y)$ in this column means that for y instances the optimal value was founded in this step and x is the average of the number of reduced variables in the *other* instances.

200 instances per line					EDUK2					$\texttt{edu}_{U_3}$					EDUK	
n	$\mathbf{w}_{\min}$	$\mathbf{w}_{\max}$	**nmd**	**ncd**	**cpu**	**vrs**	**wdp**	**rst**	**rp**	**cpu**	**vrs**	**wdp**	**rst**	**rp**	**cpu**	**rp**
20	5	$1\overline{n}$	20000	19999	317.27	10989	3	0.49	0.703	317.47	10989	3	0.49	0.703	713.23	0.676
50	5	$1\overline{n}$	46569	10052	38.66	44106	6	0.54	0.441	38.61	44106	6	0.54	0.441	208.48	0.485
20	20	$10\overline{n}$	19985	16851	118.65	11121	2	0.25	0.989	120.47	11121	2	0.25	0.989	344.81	0.994
50	20	$10\overline{n}$	50000	49999	1026.(1)	28881	0	0.22	1.00	1015.(1)	28881	0	0.22	1.00	2959.(8)	1.00
20	50	$10\overline{n}$	19999	19924	126.(2)	9955	0	0.23	1.	210.(1)	9997	0	0.23	1	504.	1
50	50	$10\overline{n}$	50000	49999	1553.(1)	22827	0	0.32	1.00	1555.(1)	26981	0	0.32	1.00	3289.(51)	1.00

FIG. 2.3 – Data sets without simply dominated items. The data from **n** and $\mathbf{w}_{\min}$ columns should be multiplied by 10^3 to get the real value.

items determines the efficiency of the algorithms. With this kind of data generation, where $c < 2 \times w_{\max}$ and n is large enough, the periodicity property does not help ($rp \approx 1$). EDUK2 outperforms significantly EDUK and slightly $\texttt{edu}_{U_3}$.

SAW data sets

SAW-UKP instances (according to **definition 1**) were generated with the following parameters : $w_{\max} = 1\overline{n}$, $p_{\max} = 2\overline{n}$ and $c \in [w_{\max}; 10\overline{n}]$. For each pair $(n, w_{\min})$, we generated ***nbi*** *distinct instances* (Fig. 2.4). The tight and computationally cheap upper-bound for these sets gives a clear

					EDUK2					$\texttt{edu}_{U_3}$					EDUK	
n	$\mathbf{w}_{\min}$	**nbi**	**nmd**	**ncd**	**cpu**	**vrs**	**wdp**	**rst**	**rp**	**cpu**	**vrs**	**wdp**	**rst**	**rp**	**cpu**	**rp**
10	10	200	9975	1965	8.03	8015	14	0.40	0.597	11.12	5323	2	0.47	0.630	29.06	0.636
50	5	500	49925	5568	70.78	41289(1)	17	0.05	0.51	108.97	25287(1)	11	0.53	0.517	294.30(1)	0.521
50	10	200	49955	8983	71.02	39779(3)	6	0.40	0.49	122.66	26510(3)	3	0.49	0.492	416.88	0.496
100	10	200	99809	6592	264.12	90436	1	0.32	0.510	387.03	65289	1	0.45	0.519	1268.45	0.523

FIG. 2.4 – SAW data sets. Data from **n** and $\mathbf{w}_{\min}$ columns should be multiplied by 10^3.

advantage to EDUK2 compared to EDUK and $\texttt{edu}_{U_3}$. This bound has an impact on the number of instances solved in the variable reduction step or by the initial *B&B* (column wdp), the number of reduced variable (column vrs), and the number of states (column rst).

Increasing ratio sets

In order to create difficult instances for DP, we generated items in such a way that the ratio $\frac{p}{w}$ is an increasing function of the weight. It is easy to see that $cd = n$ in this case. w values were uniformly and randomly generated within the interval $[w_{\min}..w_{\max}]$ (without duplicates) and were sorted in an increasing order. Then p was generated using

$$\begin{aligned} p_1 &= p_{\min} + k_1 \quad \text{and} \\ p_i &= \lfloor w_i \times (0.01 + \frac{p_{i-1}}{w_{i-1}}) \rfloor \quad + \quad k_i \text{ with } k_i \text{ randomly generated } \leq 10 \end{aligned} \tag{2.7}$$

We set $w_{\min} = p_{\min} = n$, $w_{\max} = 10\overline{n}$, and c was randomly generated within $[w_{\max}..1000\overline{n}]$. We did

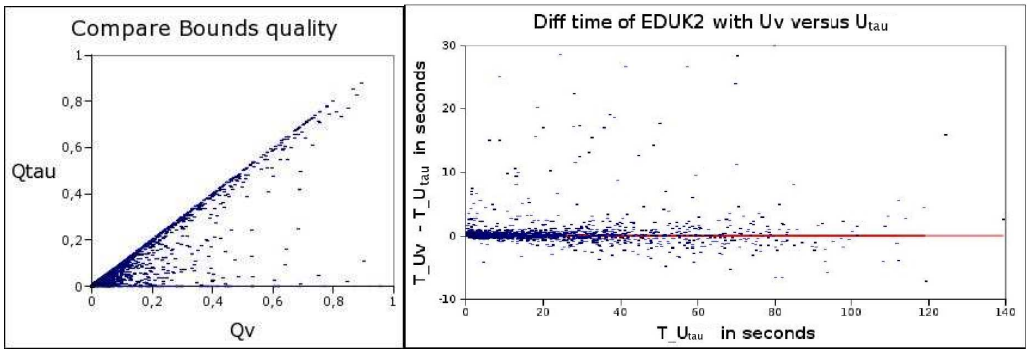
500 instances per line				EDUK2					$\texttt{edu}_{U_3}$					EDUK	
n	$\mathbf{w}_{\min}$	**nmd**	**ncd**	**cpu**	**vrs**	**wdp**	**rst**	**rp**	**cpu**	**vrs**	**wdp**	**rst**	**rp**	**cpu**	**rp**
5	**n**	**n**	**n**	7.93	3101	23	0.40	0.827	7.84	3101	23	0.40	0.827	29.05	0.816
10	**n**	**n**	**n**	36.84	5660(1)	13	0.43	0.745	36.73	5660(1)	13	0.43	0.745	147.76	0.759
20	**n**	**n**	**n**	184.55	12010	3	0.38	0.791	184.18	12010	3	0.38	0.791	735.24	0.783
50	**n**	**n**	**n**	808.26	25499	2	0.46	1	805.24	25499	2	0.46	1	2764.59	1

FIG. 2.5 – Increasing ratio data sets generated with formula (2.7). Data from **n** column should be multiplied by 10^3.

not observe any significant difference between EDUK2 and $\texttt{edu}_{U_3}$, though both were about 4 times faster than EDUK.

5.3 U_{τ^*} versus U_v for the SAW-UKP family

Theorem 2 states that the new bound U_{τ^*} is the best known upper-bound for the SAW-UKP family. In order to illustrate the improvement on the quality of the bound, we have run EDUK2 with U_{τ^*} and U_v on a set of 14.000 SAW-UKP instances. The plot on the left side of **Fig. 2.6** draws the quality improvement obtained by using U_{τ^*}. The quality of a bound U is computed by the gap : $Q_U = (U - opt)/opt$ where opt is the optimal value of the UKP and U the value of the upper-bound. On these 14.000 instances, we observe that the average values are $Q_{U_{\tau^*}} = 0.016$, $Q_{U_v} = 0.023$ and the maximum difference of the quality equals 0.956. However, as illustrated by the plot on the right side, the impact on the required time to solve the instance is not always in favor of using U_{τ^*}. While the interval of solution time is $[0..140s]$ with both bounds, the difference $(T_{U_v} - T_{U_{\tau^*}})$, which measures the gap of time when using U_v instead of U_{τ^*}, stands in the interval $[-7.25s..29.9s]$. The average of this gap equals $0.05s$ in favor of U_{τ^*}.



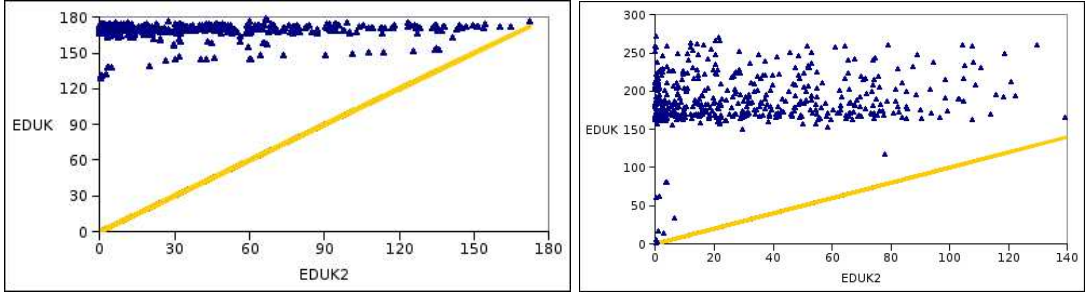
On these plots, each point corresponds to one instance of a SAW-UKP. On the left, each point has coordinates (Q_v, Q_τ) where Q_v (resp. Q_τ) is the gap (U-opt)/opt between the optimum value opt and the upper-bound U_v (resp. U_{τ^*}). On the right, we compare the time taken by EDUK2 when it uses U_v versus U_{τ^*}. Each point has coordinates $(T(U_{\tau^*}), (T(U_v) - T(U_{\tau^*})))$, the line is the equal line time.

FIG. 2.6 – U_{τ^*} versus U_v on 14.000 instances of SAW-UKP

Summary

EDUK2 consistently and significantly outperformed EDUK on all data sets. On average, EDUK2 was between 1.7 and 6 times faster than EDUK; for many instances, EDUK2 yielded the solution immediately while EDUK required several minutes (sometimes more than 1 hour). The plots of two complete sets of instances **(Fig. 2.7)** show the stable behavior of EDUK compared to the irregular behavior of EDUK2. The efficiency of EDUK2 is obtained by the cumulative effect of the different ways that *B&B* and DP are integrated. Taking into account all the new hard instances (except those generated with formula 2.6), the reduction variable step reduces the number of items to be considered by an average varying from 55% to 95%. Integrating the bounds during the DP phase further reduces the number of states from 46% to 95%. The impact of our new bound is important for all SAW-UKP instances and it affects all the steps of the algorithm. For the UKP instances that are not in the SAW family, no significant difference was observed between computations with U_{τ^*}, U_v or U_3.

There are still hard instances with large values for n and $w_{\min}$, notably those generated with formula (2.6), where $\alpha < 0, w_{\min} = 110000, n = 10000$. These were solved by EDUK2 on average of 25 to 30 minutes. For all these difficult instances, the number of items that are not collectively dominated is very large. Thus, it appears that for such cases, the DP algorithm must explore a huge iteration space if the *B&B* does not find the solution.



The running times (in seconds) of `EDUK2` (resp. `EDUK`) are on the horizontal(resp. vert.) axis. Each point corresponds to one instance. The line is the equal-time line. The increasing ratio data set (where $n = 2 \times 10^4$) is on the left, the SAW data set (where $n = 5 \times 10^4$) is on the right.

FIG. 2.7 – Plots of two complete set of instances

6 Conclusion

We have shown that a hybrid approach combining several known techniques for solving UKP performs significantly better than any one of these techniques used separately. The effectiveness of the approach is demonstrated on a rich set of instances with very large inputs. The combined algorithm inherits the best timing characteristics of the parents (DP with bounds and *B&B*) and performs significantly better on almost all of the instances. We also proposed a new upper bound for the UKP and demonstrated that this bound is the tightest one known for a specific family of UKP. Our `EDUK2` algorithm takes advantages of most of the known UKP properties and is able to solve all but the very special hard problems in a very short time. It appears that instances previously known to be difficult are now solvable in less than a few minutes.

7 References

[1] R. Andonov, V. Poirriez, and S. Rajopadhye. Unbounded knapsack problem : dynamic programming revisited. *European Journal of Operational Research*, 123(2) :168–181, 2000. *Improved version of the IRISA RR 1152, year 1997.*

[2] D. Babayev, F. Glover, and J. Ryan. A new knapsack solution approach by integer equivalent aggregation and consistency determination. *INFORMS Journal on Computing*, 9(1) :43–50, 1997.

[3] L. Caccetta and A. Kulanoot. Computational Aspects of Hard Knapsack Problems. *Nonlinear Analysis*, 47 :5547–5558, 2001.

[4] C-S. Chung, M. S. Hung, and W. O. Rom. A Hard Knapsack Problem. *Naval Research Logistics*, 35 :85–98, 1988.

[5] R. Garfinkel and G. Nemhauser. *Integer Programming.* John Wiley and Sons, 1972.

[6] P. C. Gilmore and R. E. Gomory. The Theory and Computation of Knapsack Functions. *Operations Research*, 14 :1045–1074, 1966.

[7] S. Martello and P. Toth. *Knapsack Problems : Algorithms and Computer Implementation* . John Wiley and Sons, 1990.

[8] G. L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization.* John Willey & Sons, 1988.

[9] U. Pferschy, H. Kellerer, and D. Pisinger. *Knapsack Problems.* Springer Verlag, 2004. ISBN 3-540-40286-1.

[10] V. Poirriez and R. Andonov. Unbounded Knapsack Problem : New Results. In *Proceedings of the Workshop Algorithms and Experiments (ALEX98)*, pages 103–111, February 1998. available at : `http://rtm.science.unitn.it/alex98/proceedings.html`.

[11] Vincent Poirriez, Nicola Yanev, and Rumen Andonov. Towards reduction of the class of intractable unbounded knapsak problem. Research Report 1, LAMIH/ROI UMR CNRS 8530, july 2002. `http ://www.univ-valenciennes.fr/ROI/poirriez/RR-lamihroi-2002-01.ps.gz`.

[12] Nan Zhu and Kevin Broughan. On dominated terms in the general knapsack problem. *Operations Research Letters*, 21 :31–37, 1997.

Références bibtex :

```
@InProceedings{dfrost2005,
  author = {Vincent Poirriez and Antoine Marin and Rumen Andonov and Jean-François Gibrat},
  title =  {FROST: Revisited and Distributed},
  booktitle = {Proceedings of the 19th IEEE International Parallel
               \& Distributed Processing Symposium},
  year =  {2005},
  month = {April},
  organization = {Fourth IEEE International Workshop on High Performance
Computational Biology(HICOMB'05), Denver(USA)},
  url = "http://www.hicomb.org/papers/HICOMB2005-05.pdf",
  publisher = {IEEE},
  annote = {select: \~{ } 31\%},
  note =  {ISBN: 0-7695-2312-9}
}
```

FROST : Revisited and Distributed

V. Poirriez, A. Marin, R. Andonov, JF. Gibrat

Adresses

LAMIH-ROI, Université de Valenciennes
Le Mont Houy, B.P.311, 59304 Valenciennes Cedex, France
IRISA,Campus de Beaulieu,35042 Rennes Cedex, France
Mathématique, Informatique et Génome,INRA
78352 Jouy-en-Josas, France

Abstract FROST (Fold Recognition-Oriented Search Tool) [12] is a software whose purpose is to assign a 3D structure to a protein sequence. It is based on a series of filters and uses a database of about 1200 known 3D structures, each one associated with empirically determined score distributions. FROST uses these distributions to normalize the score obtained when a protein sequence is aligned with a particular 3D structure. Computing these distributions is extremely time consuming ; it requires solving about $1,200,000$ hard combinatorial optimization problems and takes about 40 days on a 2.4 GHz computer. This paper describes how FROST has been successfully redesigned and structured in modules and independent tasks. The new package organization allows these tasks to be distributed and executed in parallel using a centralized dynamic load balancing strategy. On a cluster of 12 PCs, computing the score distributions takes now about 3 days which represents a parallelization efficiency of about 1.

Keywords protein threading, parallel algorithms, large scale problems.

1 Introduction

The protein folding problem can be simply stated in the following way : given a protein sequence, which is a string over the 20-letter amino acid alphabet, determine the positions of each amino acid atom when the protein assumes its 3D folded shape. Although simply stated, this problem is extremely difficult to solve and is widely recognized as one of the most important challenges in computational biology, today [5, 7, 8, 11, 15].

In case of remote homologs, one of the most promising approaches to the above problem is protein threading, i.e., one tries to align a query protein sequence with a set of 3D structures to check whether the sequence might be compatible with one of the structures. This method relies on three basic facts :

- 3D structures of homologous proteins are much better conserved than their amino acid sequences. Indeed, many cases of proteins with similar folds are known, although having less than 15% sequence identity.
- There is a limited, relatively small number of protein structural families (figures vary between 1,000 and 10,000 according to different estimations [6, 13]).
- Different types of amino acids have different preferences for occupying a particular structural environment. These preferences are at the basis of the empirically calculated score functions that measure the fitness of a given sequence for a 3D structure.

Fold recognition methods based on threading are complex and time consuming computational techniques consisting of the following components :

1. a database of 3D structural templates ;
2. an objective function which evaluates any alignment of a sequence to a template structure ;
3. a method for finding the best (with respect to the score function) possible sequence-structure alignment ;
4. a statistical analysis of the raw scores allowing the detection of the significant sequence-structure alignments.

The third point above is related to the problem of finding the optimal sequence-to-structure alignment and is referred as protein threading problem (PTP). From a computer scientist's viewpoint this is the most challenging part of the threading methods. Until recently, it was the main obstacle to the development of efficient and reliable fold recognition methods. In the general case, when variable-length alignment gaps are allowed and pairwise amino acid interactions are considered in the score function, PTP is NP-hard [9]. Moreover, it is MAX-SNP-hard [2], which means that there is no arbitrary close polynomial approximation algorithm, unless P = NP. In this context the progress done by the computational biology community in solving PTP during the last few years is really remarkable [10, 21, 3, 17, 18, 20, 19, 4]. However the empirical results clearly show that the problem is easier in practice than in theory and that it is possible to solve real-life (biological) instances in a reasonable amount of time. One of the most promising approaches in solving this problem is using advanced mathematical programming (Mixed Integer Programming, MIP) models for PTP [21, 3, 17, 18, 16]. A further step in this direction of research is the development of special-purpose algorithms for solving MIP models instead of using general-purpose branch-and-bound algorithms based on linear programming (LP) relaxation. Impressive computational results reported in [4] show that MIP models can be successfully solved using Lagrangian relaxation. This approach is today the default algorithm for solving a PTP instance in FROST.

This paper focuses on the 4th point above. Despite the significant recent progress in solving PTP, this component is still an extremely time consuming part of the threading methods. The underlying score normalization procedure involves threading a large set of queries against each template and requires solving millions of PTP [12]. Accelerating computations involved in this component is crucial for the development of efficient fold recognition method.

The organization of the papers is as follows. In section 2.1 we present a computer scientist's vision of FROST, i.e. as a succession of functions and procedures, yielding dependent and/or independent tasks. This new vision permits to redesign FROST and to organize it in modules which presents numerous advantages. This is discussed in section 2.2. We also distinguish independent tasks and discuss how to distribute them on a cluster of PCs in order to obtain an efficient parallelization (section 3.1). In section 4, the performances of the proposed algorithm are experimentally validated on the entire FROST database and corresponding running times are given.

2 FROST : a computer science vision

2.1 Description of the original FROST

FROST (Fold Recognition-Oriented Search Tool) is intended to assess the reliability of fold assignments to a given protein sequence (hereafter called the query sequence or query for short) [1, 12]. This tool is based on a series of filters, each one possessing a specific scoring (or fitness) function used to measure the adequacy between the query sequence and template structures. There are currently about 1200 template structures. For the time being, FROST employs only two filters. The first filter is based on a fitness function whose parameters involve only a local (degenerate) description of the 3D structure : a given structural state is assigned to each amino acid in the sequence. On the other hand, parameters of the second filter fitness function require the knowledge of the interactions between residues in contact in the 3D structure, thus making use of spatial information. Hereafter, these filters will be called 1D and 3D filters, respectively.

When aligning a given query sequence to a set of 3D structures it is not possible to directly use the raw scores to rank the 3D structures since these scores strongly depend on the query and template lengths and also, in a complicated way, on the particular features of the 3D structures. In addition, the query sequence may correspond to none of the existing folds. Therefore one must have a mean to evaluate the significance of an alignment score. This is done by empirically calculating a distribution of scores for each core, using a set of sequences not related to it. We then compare the alignment score between the query and a core to the corresponding distribution. The more far away the score from the corresponding distribution, the more significant it is because the more far away the score from the bulk of not related sequences scores, the better the chance the sequence has to be related to the template structure.

Because we do not know the analytical form of the distribution we use the following scheme : the raw score (RS) is normalized (NS) using the first and third quartiles of the distribution (q_{25} and q_{75} respectively) according to : $NS = \frac{q_{75} - RS}{q_{75} - q_{25}}$.

As the scores are highly dependent on sequences lengths, for each template, we compute five distributions (for five different sequence lengths corresponding to -30%, -15%, 0%, +15% and +30% of the template length, as explained below). We then linearly interpolate the corresponding quartile values according to the actual query length. Thus, the whole threading procedure is composed of two phases, the first one is the computation of scores distributions (hereafter called phase D) and the second one is the alignment of the sequence of interest with the dataset of templates (hereafter called phase E for evaluation) making use of the previously calculated distributions. These two phases are repeated for each filter (1D and 3D) but some tricks have been developed to accelerate the whole procedure. First, as a template represents a fold, we search for a global match between a query and a template and we thus do not consider queries and templates when their sequences lengths differ from more than 30%. This strategy usually reduces the number of alignments from 1200 to about 300 for the 1D filter. The second is that after the 1D filter, the templates are ranked and only the 10 best are passed to the 3D filter which is more computationally expensive. The algorithm used by FROST in the 1D filter, denoted here by `Ali1D(Q,C)`, is based on dynamic programming and has a quadratic complexity for a fixed query Q and a template C. On the other hand, the algorithm for the 3D filter, here denoted by `Ali3D(Q,C)`, has to solve an alignment problem which is proven to be NP-complete [9]. Even using the fastest algorithm currently available for solving the underlying combinatorial optimization problem [4], computing the score distributions for all the templates takes more than a month when performed sequentially.

The whole procedure requires the following computations :

1. Phase D : align non homologous sequences in order to obtain the scores distributions for all templates and all filters (two for now). Since five distributions are associated to any template, and there are about 200 sequences for each distribution, this procedure needs solving about 1,200,000 quadratic problems `Ali1D` and the same amount of NP-complete problems `Ali3D`.

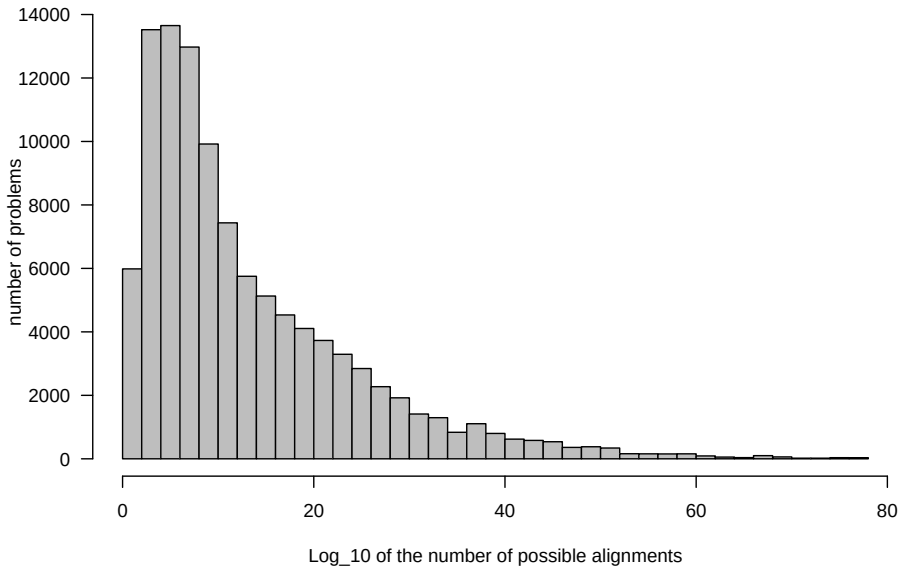


FIG. 3.1 – Populations of 3D problems solved during phase D as a function of the $\log_{10}$ of the number of possible alignments (the size of the search space).

2. Phase E : align the query with the dataset of templates which requires solving several hundreds of quadratic problems `Ali1D` and N NP-complete problems `Ali3D` (where N is usually ten).

To give an idea of the amount of computation required by the 3D filter, Figure 3.1 shows the distribution of the $\approx 10^6$ alignment problems solved during phase D w.r.t the number of possible alignments. The latter can be as large as $6.6\ 10^{77}$.

Figure 3.2 shows the plot of the mean cpu time required to solve the 3D problems involved in phase D as a function of the number of possible alignments.

The purpose of the procedure proposed in the next section is to distribute all these tasks.

Note that phase D needs to be repeated each time the fitness functions or the library of templates change, which is almost always the case when the program is used in a development phase.

2.2 Dividing up FROST into modules

The first improvement in the distributed version (DFROST) compared to the original FROST consists in clearly identifying the different stages and operations in order to make the entire procedure modular. The process of computing the scores distributions is dissociated from the alignment of the query versus the set of templates. We therefore split the two phases (D and E) which used to be interwoven in the original implementation. Such a decomposition presents several advantages. Some of them are :

- Phase D is completely independent from the query, it can be performed as *a preprocessing stage* when it is convenient for the program designer.
- The utilization of the program is *simplified*. Note that only the program designer is supposed to execute phase D, while phase E is executed by an "ordinary" user. From a user's standpoint DFROST is *significantly faster* than FROST, since only phase E is carried out at his request (phase D being performed as a preprocessing step).

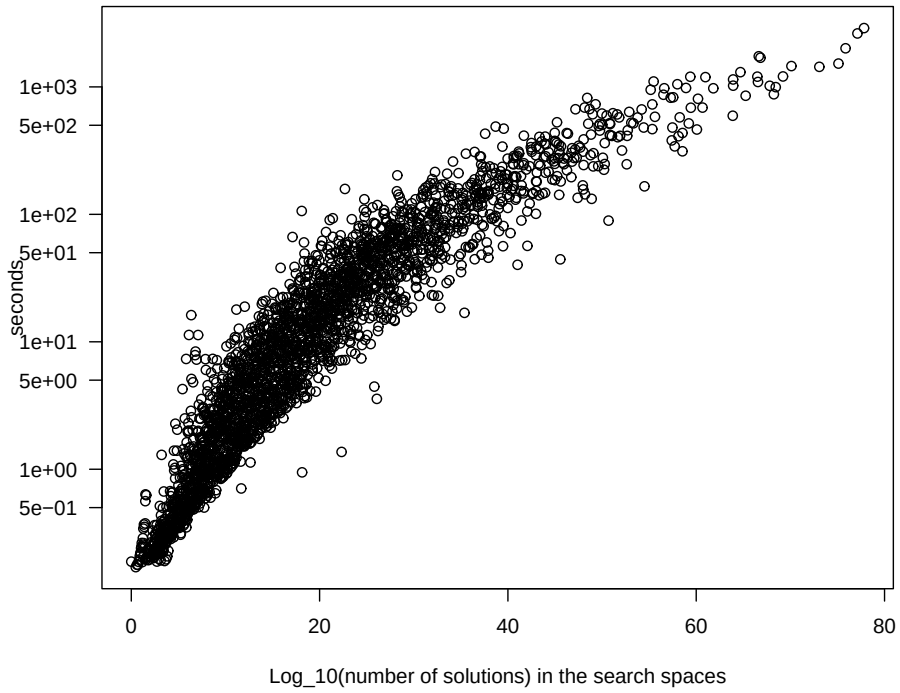


FIG. 3.2 – Mean CPU time required to solve the Phase D 3D problems, partitioned w.r.t. the $\log_{10}$ of the number of possible alignments (the size of the search space).

- The program designer can *easily carry out different operations* needed for further developments of the algorithm or for database updating such as : adding new filters, changing the fitness functions, adding a new template to the library, etc.
- This organization of DFROST in modules is very *suitable for its decomposition in independent tasks* which can be solved in parallel.

The latter point is discussed in details in the next section.

3 Parallelization

We distinguish two kinds of atomic independent tasks in DFROST : the first is related to solving an instance of a problem of type `Ali1D`, while the second is associated with solving an instance of an `Ali3D` problem[1].

Hence phase D consists in solving 1,200,000 independent tasks of type `Ali1D`, and of type `Ali3D`, while phase E consists in solving several hundreds of independent tasks `Ali1D` and ten independent tasks `Ali3D`. The final decision requires sorting and analysis of the ten best solutions of type `Ali1D` and the ten best solutions of type `Ali3D`.

In the next section we describe how these tasks can be distributed and executed in parallel on a cluster of processors.

[1] In reality this problem can be further decomposed in subtasks. Although non independent, these subtasks can be executed in parallel as show in [3, 21]. This parallelization could be easily integrated in DFROST if necessary.

3.1 Parallel Algorithm

There is a couple of important observations to keep in mind in order to obtain an efficient parallel implementation for DFROST. The first is that the exact number of tasks is not known in advance. Second, which is even more important, the tasks are irregular (especially tasks of type `Ali3D`) with unpredictable (for now) and largely varying execution time. In addition, small tasks need to be aggregated in macro-tasks in order to reduce data broadcasting overhead. Since the complexity of the two types of tasks is different, the granularity for macro-tasks `Ali1D` should be different from the granularity for macro-tasks `Ali3D`.

The parallel algorithm that we propose is based on *centralized dynamic load balancing* : macro-tasks are dispatched from a centralized location (pool) in a dynamic way. The work pool is managed by a "master" who gives work *on demand* to idle "slaves". Each slave executes the macro-tasks assigned to it by solving sequentially the corresponding subproblems (either `Ali1D` or `Ali3D`). Note that dynamic load balancing is the only reasonable task-allocation method when dealing with irregular tasks for which the amount of work is not known prior to execution.

In phase E the pool contains initially several hundreds of tasks of type `Ali1D`. The master increases the work granularity by grouping `gran1D` of them in macro-tasks. These macro-tasks are distributed on demand to the slaves that solve the corresponding problems. The solutions computed in this way are sent back to the master and sorted by it locally. The templates associated to the ten best scores yield ten problems of type `Ali3D`. The master groups them in batches of size `gran3D` and transmits them to the slaves where the associated problems are solved. The granularity `gran1D` is bigger than the `gran3D` granularity. Finally the slaves send back to the master the computed solutions.

The strategy in phase D is simpler. The master only aggregates tasks in macro-tasks of size either `gran1D` or `gran3D`, sends them on demand to idle slaves (where the corresponding problems are sequentially solved), and gathers finally the distributions that have been computed. The master processes the library of templates in a sequential manner. First, it aims at distributing all the tasks for a given template to the slaves. However, when the list of tasks for a given template becomes empty and there is at least one slave demanding work, the master continues to distribute tasks from the next template. This strategy allows to reduce globally the idle time of the processors.

4 Computational experiments

The numerical results presented in this section were obtained on a cluster of 12 Intel(R) Xeon(TM) CPU 2.4 GHz, 2 Gb Ram, RedHat 9 Linux, connected by 1 Gb ethernet network. The behavior of DFROST was tested by entirely computing the phase D of the package, i.e. all the distributions for 1125 templates for both filters.

In the case of 3D filter, solving 1,104,074 alignments required **3 days 3 hours 20 minutes** (wall time of the master). We were not in single-user mode but there were very few other users during this period. We then added the running times reported in the log files of the slaves and obtained a total sequential time equal to **37 days 5 hours 11 minutes**. Therefore, for this very representative set of instances, DFROST exhibits a **speedup of 11.9** with an efficiency close to one. Details from this execution are presented in table 3.1. The value of the parameter `gran3D` was experimentally fixed to 10.

In the case of 1D filter, solving 1,107,973 alignments required **31 minutes and 20 seconds** (wall time of the master). When we added the running times reported in the log files of the slaves we obtained a total sequential time equal to **4 hours 12 minutes and 55 seconds**. The total time to compute, sequentially, the distributions for the 2 filters was **37 days 9 hours and 24 minutes**, while computed by DFROST on twelve processors the same amount of work required **3 days 3 hours and 55 minutes**.

Template	DFROST	CPU tot	Cpu av	NAli
1BGLA0	15455	107569	113	945
1ALO_0	9565	96579	97	995
1CXSA0	5988	55808	58	960
1DIK_0	4506	46855	47	977
(...)				
1AYL_0	1807	18961	19	973
1EUT_0	1753	18883	18	995
1CTN_0	1535	16670	16	1000
1ECL_0	1439	15589	16	953
(...)				
1LYLA0	782	8335	8	990
1BIF_0	657	7129	7	948
1AD3A0	629	6669	6	1000
1DNPA0	776	6580	6	960

TAB. 3.1 – execution times **in seconds** for calculating the 3d score distributions. the templates for which the distributions are calculated are listed in the first column. the second column gives the parallel time (the execution time for the master) on a cluster of 12 processors. the third column shows the cpu sequential time (obtained by adding the cpu times from the slaves). the fourth column reports the average cpu time per threading and the last column shows the actual number of sequences that have been threaded to calculate the distributions. the value of the granularity was fixed to 10.

We can calculate an upper limit for the number of processors beyond which it is not possible any more to benefit from additional processors with the proposed algorithm. Maximum time for an alignment is 797.4 seconds, this time is the lower limit of the wall clock time for the complete computation of the distributions for `Ali3d`. Total time CPU necessary to calculate all `Ali3D` alignments is 3,215,460 seconds. Thus if one reaches **4032 processors** (3215460/797.4) to add processors will not allow to accelerate the total process. This gives a theoretical upper limit, it supposes that difficult computations are submitted first. That was not implemented in this work, it supposes to have criteria making it possible to estimate in advance the difficulty of a computation. The calculation of all the distributions confirms that a meaningful criterion is the possible number of alignments (see figure 3.2), it is also possible now to take into account the times used during the calculation of the distributions for the preceding versions of the function of fitness.

These significant results, obtained on such a large data set, justify the work done to distribute FROST and prove the efficiency of the proposed parallel algorithm.

4.1 Statistical analysis

Using this parallel algorithm we were able to compute all distributions for the entire FROST templates library – something which was never done with the sequential version. As mentioned before, this required solving 1,104,074 alignments. We can see from Figure 3.3 that for large templates like `1BGLA0` , with 528 amino acids, the size of the solution search space is as large as 6.647E+77. The distribution curve of all alignments (each time is for one PTP instance), is characterized by the following quartiles : minimum : 0s, first quartile : 0.03s mean : 0.58s third quartile : 2.32s maximum : 795.64s. We observed that for 188 templates the computation of the distributions requires more than one hour CPU time. Statistical details concerning four of the most time consuming templates are presented in table 3.2. Remember, that one PTP instance (i.e. when the query and the 3D structure are fixed) is considered as an atomic independent task in the current parallel strategy. As shown in [3, 21], such an instance can be further decomposed in subtasks that can be executed in parallel. We investigated the necessity of implementing this parallelization for our application.

	Nb Sol	NAli	Min	Q_1	Med	Mean	Q_3	Max
1BGLA0	$5.4\ 10^{27}$	55	0.95	0.96	0.98	0.97	0.98	1.02
	$1.2\ 10^{35}$	56	0.95	0.96	0.97	0.97	0.98	1.01
	$3.5\ 10^{58}$	192	35.6	39.9	42.2	45.2	50.0	73.2
	$1.3\ 10^{70}$	199	102.4	116.3	131.0	145.7	164.6	510.0
	$6.6\ 10^{77}$	150	203.8	229.7	252.6	291.7	327.5	797.4
1QBA_0	$1.6\ 10^{3}$	58	1.82	1.83	1.83	1.84	1.84	1.89
	$8.3\ 10^{37}$	57	1.82	1.83	1.83	1.84	1.84	1.89
	$5.2\ 10^{57}$	197	27.1	30.2	32.5	36.3	39.8	76.6
	$2.8\ 10^{68}$	200	68.4	77.5	86.9	101.4	116.0	354.8
	$7.2\ 10^{75}$	200	130.1	154.7	178.3	207.0	239.8	789.8
1ALO_0	$3.1\ 10^{33}$	57	0.85	0.87	0.87	0.87	0.88	0.89
	$6.0\ 10^{33}$	57	0.85	0.86	0.87	0.87	0.87	0.89
	$2.5\ 10^{57}$	190	25.8	29.3	36.1	40.8	46.7	135.2
	$1.6\ 10^{69}$	200	67.4	86.3	113.2	123.2	134.8	397.6
	$1.3\ 10^{77}$	200	139.9	175.7	231.0	262.2	303.4	735.0
1YGE_0	$3.4\ 10^{23}$	61	0.39	0.40	0.41	0.41	0.41	0.43
	$2.8\ 10^{45}$	59	0.40	0.41	0.41	0.41	0.42	0.42
	$2.1\ 10^{55}$	192	34.8	39.9	43.1	47.5	48.9	139.8
	$6.5\ 10^{61}$	173	71.2	80.5	89.5	102.0	115.9	365.1
	$4.4\ 10^{66}$	199	120.2	138.5	158.3	178.2	208.9	443.7

TAB. 3.2 – Sequential times **in seconds** for computing the 3D score distributions of four templates selected for their "difficulty" (search space size). For a given template the 5 rows represent alignment of sets of non related sequences having length respectively equal to : -30%, -15%, 0%, +15%, +30% of the template length. **Nb Sol** is the number of possible alignments that can be generated with the sequences and the template. This gives an indication of the difficulty of the problem to solve. **NAli** is the number of alignments (sequences) in the corresponding set. The last six columns report diverse running time characteristics obtained when aligning the set of sequences with the corresponding 3D structure : **Min** is the minimum value, Q_1 is the time at the 1st quartile position, **Med.** is the time at the median position, **Mean** is the average time, Q_3 is the time at the 3rd quartile position and **Max** is the maximum value.

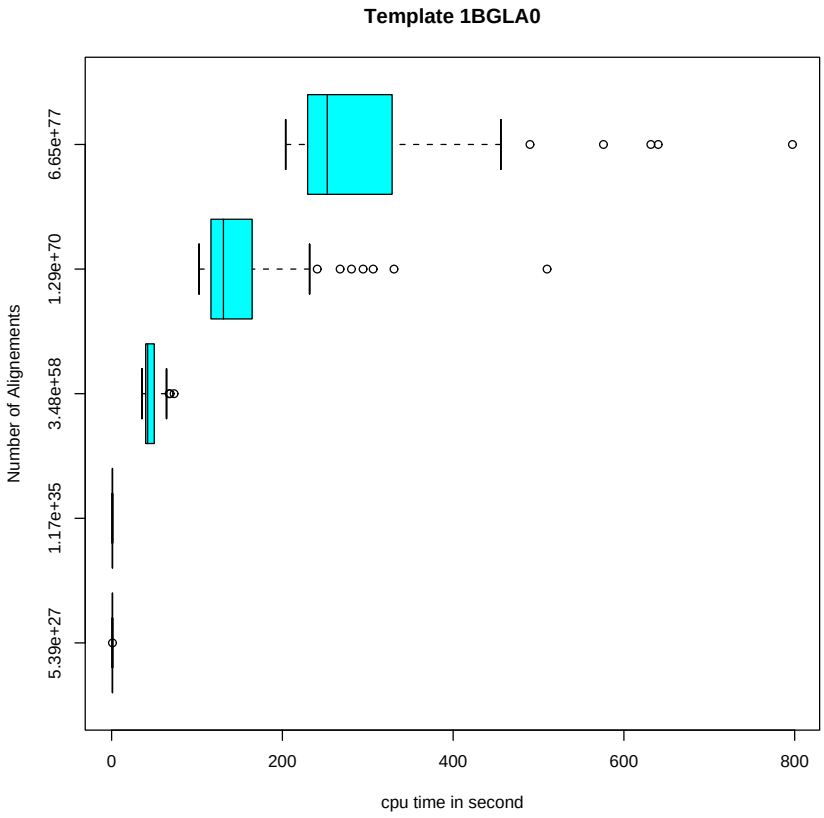


FIG. 3.3 – The two templates for which the distribution calculations are the most computer intensive, 1BGLA0 and 1QBA_0(cf fig. 3.4), are selected from table 3.2 and the corresponding boxplots of the running time distributions are plotted using the statistical package **R** [14]. The box contains the middle half of the data, i.e., the left and right ends of the box are at the lower and upper quartiles and the middle line corresponds to the median of the distribution. Vertical lines, usually called "whiskers", go left and right from the box to the extreme of the data (here defined as 1.5 times the inter-quartile range). Points outside the whisker lines are plotted by themselves. Note that the distribution is not symmetric and exhibits a heavy tail for longer CPU times.

However, in view of : i) the huge number of independent tasks when computing FROST distributions ; ii) the running time presented in tables 3.1 and 3.2, as well their statistical recapitulations in figure 3.3 and figure 3.4 showing clearly that really hard PTP instances are very rare ; iii) the very satisfactory speedup reported in section 4, we decided to stay for now with the current parallel algorithm.

5 Conclusion

Solving the protein threading problem remains time consuming even though we are able to achieve peak rates as high as 10^{74} equivalent threadings per second. For the purpose of testing new developments of the FROST algorithm we are often led to recompute the score distributions which involves carrying out millions of threading alignments. This is not easily tractable with a single computer. In this paper we have presented a simple, but very efficient, load balancing algorithm enabling us to use FROST on a cluster of computers. Using this implementation we were able to compute the score distributions for the whole set of templates in about 3 days on a cluster of

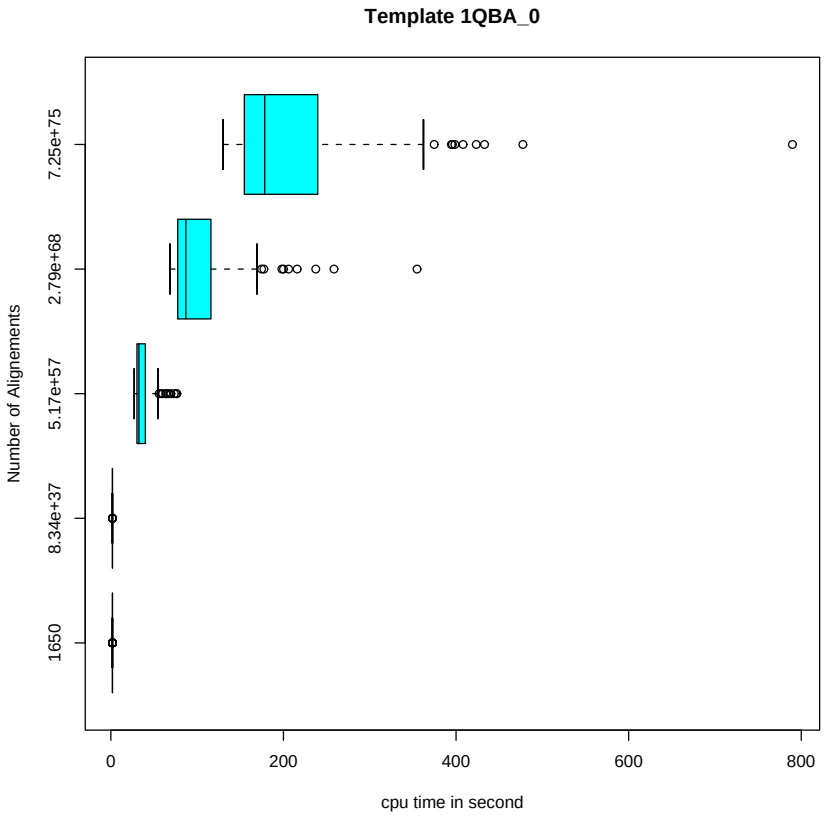


FIG. 3.4 – Same as (fig. 3.3), but for1 QBA_0

12 computers instead of more than 40 days on a single computer. This parallelization procedure achieved an efficiency of about 1.

6 References

[1] K. Zimmermann A. Marin, J.Pothier and J-F. Gibrat. Protein threading statistics : an attempt to assess the significance of a fold assignment to a sequence. Protein structure prediction : bioinformatic approach, International University Line : La Jolla, California, 2002.

[2] T. Akutsu and S. Miyano. On the approximation of protein threading. *Theoretical Computer Science*, 210 :261–275, 1999.

[3] R. Andonov, S. Balev, and N. Yanev. Protein threading problem : From mathematical models to parallel implementations. *INFORMS Journal on Computing*, 16(4), 2004. Special Issue on Computational Molecular Biology/Bioinformatics.

[4] Stefan Balev. Solving the protein threading problem by lagrangian relaxation. WABI 2004, 4th Workshop on Algorithms in Bioinformatics, September 14 - 17 2004. Bergen, Norway.

[5] C. Branden and J. Tooze. *Introduction to protein structure*. Garland Publishing, 1999.

[6] C. Chothia. Proteins. one thousand families for the molecular biologist. *Nature*, pages 543–544, 1992.

[7] H.J. Greenberg, W.E. Hart, and G. Lancia. Opportunities for combinatorial optimization in computational biology. *INFORMS Journal on Computing*, 16(3), 2004.

[8] T. Head-Gordon and J. C. Wooley. Computational challenges in structural and functional genomics. *IBM Systems Journal*, 40 :265–296, 2001.

[9] R. Lathrop. The protein threading problem with sequence amino acid interaction preferences is np-complete. *Protein Eng.*, 7 :1059–1068, 1994.

[10] R.H. Lathrop and T.F. Smith. Global optimum protein threading with gapped alignment and empirical pair potentials. *J. Mol. Biol.*, 255 :641–665, 1996.

[11] T. Lengauer. Computational biology at the beginning of the post-genomic era. In R. Wilhelm, editor, *Informatics : 10 Years Back - 10 Years Ahead*, volume 2000 of *LNCS*, pages 341–355. Springer Verlag, 2001.

[12] A. Marin, J. Pothier, K. Zimmermann, and J.F. Gibrat. Frost : A filter based recognition method. *Proteins*, 49(4) :493–509, 2002.

[13] C.A Orengo, T. D. Jones, and J. M. Thornton. Protein superfamilies and domain superfolds. *Nature*, 372 :631–634, 1994.

[14] R : A language and environment for statistical computing, 2004. `http ://www.R-project.org`.

[15] J.C. Setubal and J. Meidanis. *Introduction to computational molecular biology*, chapter 8, pages 252–259. Brooks/Cole Publishing Company, 511 Forest Lodge Road, Pacific Grove, CA 93950, 1997.

[16] J. Xu. Speedup LP approach to protein threading via graph reduction. volume 2812, pages 374–388. Proceedings of WABI 2003 : Third Workshop on Algorithms in Bioinformatics, Springer-Verlag, 2003.

[17] J Xu, M. Li, G. Lin, D. Kim, and Y. Xu. Protein structure prediction by linear programming. pages 264–275. Proceedings of The 7th Pacific Symposium on Biocomputing (PSB), 2003.

[18] J. Xu, M. Li, G. Lin, D. Kim, and Y. Xu. RAPTOR : optimal protein threading by linear programming. *Journal of Bioinformatics and Computational Biology*, 1(1) :95–118, 2003.

[19] Y. Xu and D. Xu. Protein threading using PROSPECT : design and evaluation. *Proteins : Structure, Function, and Genetics*, 40 :343–354, 2000.

[20] Y. Xu, D. Xu, and E.C. Uberbacher. An efficient computational method for globally optimal threading. *Journal of Computational Biology*, 5(3) :597–614, 1998.

[21] N. Yanev and R. Andonov. Parallel divide and conquer approach for the protein threading problem. *Concurrency and Computation :Practice and Experience*, 16 :1–14, 2004.

Références bibtex :

```
@InProceedings{veber05,
  author =   {Philippe Veber and Nicola Yanev and Rumen Andonov and Vincent Poirriez},
  title =   {Optimal protein threading by cost-splitting},
  booktitle = {Algorithms in Bioinformatics-WABI 2005
               (5th Workshop on Algorithms in Bioinformatics)},
  pages =   {365-375},
  year =   {2005},
  editor =   {R. Casadio, G. Myers},
  volume =   {3692},
  month =   {october 3-6,  Mallorca, Spain},
  organization = {EATCS, the European Association for Theoretical Computer Science,
                  and ISCB, the International Society for Computational Biology},
  publisher = {Springer Verlag Lecture Notes in Bioinformatics},
  note =   {ISBN 3-540-29008-7},
}
```

Optimal protein threading by cost-splitting

Philippe Veber and Nicola Yanev and Rumen Andonov and Vincent Poirriez

Adresses

LAMIH-ROI, Université de Valenciennes
Le Mont Houy, B.P.311, 59304 Valenciennes Cedex, France
IRISA,Campus de Beaulieu,35042 Rennes Cedex, France
Mathématique, Informatique et Génome,INRA
78352 Jouy-en-Josas, France
University of Sofia, 5, J. Bouchier str., 1126 Sofia, Bulgaria

Abstract In this paper, we use integer programming approach for solving a hard combinatorial optimization problem, namely protein threading. For this sequence-to-structure alignment problem we apply cost-splitting technique to derive a new Lagrangian dual formulation. The optimal solution of the dual is sought by an algorithm of polynomial complexity. For most of the instances the dual solution provides an optimal or near-optimal (with negligible duality gap) alignment. The speed-up with respect to the widely promoted approach for solving the same problem in [17] is from 100 to 250 on computationally interesting instances. Such a performance turns computing score distributions, the heaviest task when solving PTP, into a routine operation.

1 Introduction

Protein folding is one of the most extensively studied problems in computational biology. The problem can be simply stated as follows : given a protein sequence, which is a string over the 20-letter amino acid alphabet, determine the positions of each amino acid atom when the protein assumes its 3D folded shape. Although simply stated, this problem is extremely difficult to solve and is widely recognized as one of the most important challenges in computational biology today [10, 16, 6].

In case of remote homologs, one of the most promising approaches to the above problem is protein threading, i.e., one tries to align a query protein sequence with a set of 3D structures to check whether the sequence might be compatible with one of the structures. Fold recognition

methods based on threading are complex and time consuming computational techniques consisting of the following components :

1. a database of known 3D structural templates ;
2. an objective function which evaluates any alignment of a sequence to a template structure ;
3. a method for finding the best (with respect to the score function) possible sequence-3D structure alignment ;
4. a statistical analysis of the raw scores allowing the detection of the significant sequence-structure alignments.

The third point above is related to the problem of finding the optimal sequence-to-structure alignment and is referred to as protein threading problem (PTP). From a computer scientist's viewpoint this is the most challenging part of the threading methods. Until recently, it was the main obstacle to the development of efficient and reliable fold recognition methods. In the general case, when variable-length alignment gaps are allowed and pairwise amino acid interactions are considered in the score function, PTP is NP-hard [8]. Moreover, it is MAX-SNP-hard [1], which means that there is no arbitrary close polynomial approximation algorithm, unless P = NP. In this context the progress done by the computational biology community in solving PTP during the last few years is really remarkable [12, 20, 2, 17, 18, 3]. The empirical results clearly illustrate that PTP is easier in practice than in theory and that it is possible to solve real-life (biological) instances in a reasonable amount of time. These results also show that one of the most promising approaches in solving this problem is using advanced mathematical programming (Mixed Integer Programming, MIP) models for PTP [19, 20, 2, 17]. The most amazing observation is that for almost all (more than 95%) of the instances, the LP relaxation of the MIP models is integer-valued, thus providing optimal threading. This is true even for polytopes with more than 10^{46} vertices. Moreover, when the LP relaxation is not integer, its value is a relatively good approximation of the integer solution. However, to the best of our knowledge, this observation has not been practically used before the current paper. Other successful Integer Programming approaches for solving combinatorial optimization problems originated in molecular biology are discussed in the recent survey [11].

The main drawback of mathematical programming approaches is that the corresponding models are often very large (over 10^6 variables). Even the most advanced MIP solvers need prohibitively large running time for solving such instances. For example, the authors in [17] find out 30 templates for which it takes about 15 hours to thread one target onto them on a Silicon Graphics Origin 3800 system, which has 40400 MHz MIPS R12000 CPUs and 20 GB of RAM. Different divide-and-conquer methods and parallel algorithms can be used to overcome this drawback [18, 19, 20].

A further step in solving the huge MIP models is the development of special-purpose algorithms based on advanced combinatorial optimization techniques like Lagrangian relaxation. Such an algorithm has been recently designed by S. Balev in [3] and computationally compared with the B&B algorithm from [8] and a heuristic used in [9]. The computational results are very impressive and clearly show that the Lagrangian relaxation (LR) significantly outperforms both other algorithms. However, comparisons with MIP solver are not provided in [3].

In this paper we continue the same direction of research and propose a new dedicated algorithm for solving protein threading MIP models. It is as well based on Lagrangian relaxation. But, both our Lagrangian dual formulation and the optimization technique that we use for solving it (the so-called cost-splitting [13]), differentiate from those described in [3]. Extensive computational results prove that : (i) our algorithm is in most cases faster than the one in [3] ; (ii) both Lagrangian relaxation algorithms significantly outperform solving MIP models by LP relaxation. To the best of our knowledge the only other impressive application of LR to an alignment problem is discussed in [5].

Another contribution of the current paper concerns the 4th point above. When aligning a given query sequence to a set of 3D structures it is not possible to directly use the raw scores to rank the 3D structures. The reason is that these scores strongly depend on the query and template

lengths and also, in a complicated way, on the particular features of the 3D structures. In addition, the query sequence may correspond to none of the existing folds. Therefore one must have means to evaluate the significance of an alignment score. This can be done as a preprocessing stage, by empirically calculating a distribution of scores for each template, using a set of sequences not related to it[1]. The underlying score normalization procedure involves threading a large set of queries against each template and requires solving millions of PTP. For example the package FROST (Fold Recognition-Oriented Search Tool) [9], uses a database of about 1,200 known 3D structures, each one associated with empirically determined score distributions. Computing these distributions is extremely time consuming : it requires solving about 1,200,000 sequence-to-structure alignments and takes about 40 days on a 2.4 GHz computer and about 3 days on a cluster of 12 PCs [14]. Accelerating computations involved in this component is crucial for the development of efficient fold recognition methods.

Based on extensive comparisons we observe that the approximated solutions obtained by any one of the three algorithms considered in this paper can be successfully used when computing scores distributions. Since these approximated solutions are obtained by polynomial algorithms, we experimentally prove that this heavy stage can be polynomially computed.

The organization of the paper is as follows. In section 2 we introduce a formal presentation of PTP, and then study some special cases in the section 3. Section 4 presents the cost-splitting technique. Last section is dedicated to experimental results.

2 Protein threading problem

For the sake of brevity, in this paper we stick to the network optimization problem formulation proposed in [19, 2].

Let us introduce variables y_{ij}, and denote by Y the set of feasible threadings, defined by the following constraints :

$$\sum_{k=1}^{n} y_{ik} = 1 \qquad i = 1, \ldots, m \tag{4.1}$$

$$\sum_{l=1}^{k} y_{il} - \sum_{l=1}^{k} y_{i+1,l} \geq 0 \qquad i = 1, \ldots, m-1,\ k = 1, \ldots, n-1 \tag{4.2}$$

$$y_{ik} \in \{0, 1\} \qquad i = 1, \ldots, m,\ k = 1, \ldots, n \tag{4.3}$$

These constraints describe the set of feasible paths in a particular digraph (see figure 4.4 in Appendix 1), with vertex set $V = \{(i,j) \mid i = 1, \ldots, m,\ j = 1, \ldots, n\}$. The vertices $(i,j), j = 1, \ldots, n$ will be referred to as ith layer. Each layer corresponds to a structural element, and each vertex in a layer corresponds to a positioning of this element on a query protein. Let $L \subseteq \{(i,k) \mid 1 \leq i < k \leq m\}$ be a given set of inter-layers links. This is the so-called *contact graph* : a link between layers i and k means that the corresponding structural elements are in contact in the 3D structure.

Let A_{ik} be the $2n \times \frac{n(n+1)}{2}$ node-arc incidence matrix for the subgraph spanned by the layers i and k, $(i,k) \in L$. The submatrix A_i, the first n rows of A_{ik}, (resp. A_k, the last n rows) corresponds to the layer i (resp. k). To avoid added notation we will use vector notation for the variables $y_i = (y_{i1}, \ldots y_{in}) \in B^n$ where B^n is the set of n-dimensional binary vectors, with assigned costs $c_i = (c_{i1}, \ldots c_{in}) \in R^n$ and $z_{ik} = (z_{i1k1}, \ldots, z_{i1kn}, z_{i2k1}, \ldots, z_{inkn}) \in B^{\frac{n(n+1)}{2}}$ for $(i,k) \in L$ with assigned costs $d_{ik} = (d_{i1k1}, \ldots, d_{i1kn}, d_{i2k1}, \ldots, d_{inkn}) \in R^{\frac{n(n+1)}{2}}$. In the sections below the vector d_{ik} will be considered as a $n \times n$ upper triangular matrix, having arbitrarily large coefficient below

[1] More justifications for this phase the interested reader can find in [9].

the diagonal. This slight deviation from the standard definition of an upper triangular matrix is used only for formal definition of some matrix operations.

Now the protein threading problem $PTP(L)$ is defined as :

$$z_{ip}^{L} = v(PTP(L)) = min\{\sum_{i=1}^{m} c_i y_i + \sum_{(i,k)\in L} d_{ik} z_{ik}\} \tag{4.4}$$

$$\text{subject to : } \quad y = (y_1, \ldots, y_m) \in Y, \tag{4.5}$$

$$y_i = A_i z_{ik}, \;\; y_k = A_k z_{ik} \qquad (i,k) \in L \tag{4.6}$$

$$z_{ik} \in B^{\frac{n(n+1)}{2}} \qquad (i,k) \in L \tag{4.7}$$

The shortcut notation $v(.)$ will be used for the optimal objective function value of a subproblem obtained from $PTP(L)$ with some z variables fixed. Throughout the next section, vertex costs c_i are assumed to be zero. We study three sorts of contact graph that make PTP polynomially solvable.

3 Special cases

3.1 Contact graph contains no crossing edges

Two links (i_1, k_1) and (i_2, k_2) such that $i_1 < i_2$ are said to be crossing when k_1 is in the open interval (i_2, k_2). The case when the contact graph L contains no crossing edges has been mentioned to be polynomially solvable for the first time in [1]. Here we present a different sketch for $O(n^3)$ complexity of PTP in this case.

If L contains no crossing edges, then $PTP(L)$ can be recursively divided into independent subproblems. Each of them consists in computing all shortest paths between the vertices of two layers i and k, discarding links that are not included in (i,k). Thus the result of this computation is a distance matrix D_{ik} such that $D_{ik}(j,l)$ is the optimal length between vertices (i,j) and (k,l). Note that for $j > l$ as there is no path in the graph, $D_{ik}(j,l)$ is an arbitrarily large coefficient. Finally, the solution of $PTP(L)$ is the smallest entry of D_{1m}.

We say that an edge $(i,k), i < k$ is included in the interval $[a,b]$ when $[i,k] \subseteq [a,b]$. Let us denote by $L_{(ik)}$ the set of edges of L included in $[i,k]$. Then, an algorithm to compute D_{ik} can be sketched as follows :

1. if $L_{(ik)} = \{(i,k)\}$ then the distance matrix is given by

$$D_{ik} = \begin{cases} d_{ik} & \text{if } (i,k) \in L \\ \tilde{0} & \text{otherwise} \end{cases} \tag{4.8}$$

   where $\tilde{0}$ is an upper triangular matrix in the previously defined sense (arbitrary large coefficients below the main diagonal) and having only zeros in its upper part.

2. otherwise as $L_{(ik)}$ has no crossing edges, there exists some $s \in [i,k]$ such that any edge of $L_{(ik)}$ but (i,k) is included in $[i,s]$ or in $[s,k]$. Then

$$D_{ik} = \begin{cases} D_{is}.D_{sk} + d_{ik} & \text{if } (i,k) \in L \\ D_{is}.D_{sk} & \text{otherwise} \end{cases} \tag{4.9}$$

   where the matrix multiplication is computed by replacing $(+, \times)$ operations on reals by $(min, +)$.

Remark 12 *If the contact graph has m vertices, and contains no crossing edges, then the problem is decomposed into $O(m)$ subproblems. For each of them, the computation of the corresponding distance matrix is a $O(n^3)$ procedure (matrix multiplication with $(min,+)$ operations). Overall complexity is thus $O(mn^3)$. Typically, n is one or two orders of magnitude greater than m, and in practice, this special case is already expensive to solve.*

3.2 All edges have their left end tied to a common vertex

A set of edges $L = \{(i_1, k_1), \ldots, (i_r, k_r)\}, k_1 < k_2 < \ldots k_r$ is called a *star* if it has at least two elements and $i_t = i_1, t \leq r$. The arc costs corresponding to the link (i, k_s) are given by the upper triangular matrix d_{ik_s}. The following algebra is used to prove the $O(n^2)$ complexity of the corresponding PTP.

Definition 2 *Let A, B be two matrices of size $n \times n$. $M = A \otimes B$ is defined by $M(i,j) = min_{i \leq r \leq j} A(i,r) + B(i,j)$*

In order to compute $A \otimes B$, we use the following recursion : let M' be the matrix defined by $M'(i,j) = min_{i \leq r \leq j} A(i,r)$, then

$$M'(i,j) = min\{M'(i,j-1), A(i,j)\}, \text{ for all } j \geq i$$

Finally $A \otimes B = M' + B$. From this it is clear that $\otimes$ multiplication for $n \times n$ matrices is of complexity $O(n^2)$.

Theorem 3 *Let $L = \{(i,k_1), \ldots, (i,k_r)\}$ be a star.*
Then $D_{ik_r} = (\ldots (d_{ik_1} \otimes d_{ik_2}) \otimes \ldots) \otimes d_{ik_r}$

Proof

The proof follows the basic dynamic programming recursion for this particular case : for the star $L = \{(i,k_1), \ldots, (i,k_r)\} = L' \bigcup \{(i,k_r)\}$, we have $v(L : z_{ijk_r l} = 1) = d_{ijk_r l} + min_{j \leq s \leq l} v(L' : z_{ijk_{r-1} s} = 1)$

3.3 Sequence of independent subproblems

Given a contact graph $L = \{(i_1, k_1), \ldots, (i_r, k_r)\}$, $PTP(L)$ can be decomposed into two independent subproblems when there exists an integer $e \in (1, m)$ such that any edge of L is included either in $[1, e]$, either in $[e, m]$. Let $I = \{i_1, \ldots, i_s\}$ be an ordered set of indices, such that any element of I allows for a decomposition of $PTP(L)$ into two independent subproblems. Suppose additionally that for all $t \leq s-1$, one is able to compute $D_{i_t i_{t+1}}$. Then we have the following theorem :

Theorem 4 *Let $p = (p_1, p_2, \ldots, p_n)$ be obtained by the following matrix-vector multiplication $p = D_{i_1 i_2} D_{i_2 i_3} \ldots D_{i_{s-1} i_s} \overline{p}$, where $\overline{p} = (0, 0, \ldots, 0)$ and the scalar product in the matrix-vector multiplication is defined by changing "+" with "min" and "." with "+". Then for all i, $p_i = v(PTP(L : y_{1i} = 1)$, and $v(PTP(L)) = min\{p_i\}$.*

Proof

Each multiplication by $D_{i_k i_{k+1}}$ in the definition of p is an algebraic restatement of the main step of the algorithm for solving the shortest path problem in a graph without circuits.

Remark 13 *With the notations introduced above, the complexity of $PTP(L)$ for a sequence of such subproblems is $O(sn^2)$ plus the cost of computing matrices $D_{i_t i_{t+1}}$.*
From the last two special cases, it can be seen that if the contact graph can be decomposed into independent subsets, and if these subsets are single edges or stars, then there is a $O(srn^2)$ algorithm, where s is the cardinality of the decomposition, and r the maximal cardinality of each subset, that solves the corresponding PTP.

4 Cost splitting

In order to apply the results from the previous section, we need to find a suitable partition of L into $L^1 \bigcup L^2 ... \bigcup L^t$ where each L^s induces an easy solvable $PTP(L^s)$, and to use the s.c. cost-splitting variant of the Lagrangian duality. Now we can restate (4.4)-(4.7) equivalently as :

$$v_{ip}^L = min \left\{ \sum_{s=1}^{t} (\sum_{i=1}^{m} c_i^s y_i^s + \sum_{(i,k) \in L^s} d_{ik} z_{ik}) \right\} \tag{4.10}$$

$$\begin{aligned}
\text{subject to : } \quad & y_i^1 = y_i^s, && s = 2, t && (4.11)\\
& y^s = (y_1^s, .. y_m^s) \in Y, && s = 1, \ldots, t && (4.12)\\
& y_i^s = A_i z_{ik}, \;\; y_k^s = A_k z_{ik} && s = 1, \ldots, t \qquad (i,k) \in L^s && (4.13)\\
& z_{ik} \in B^{\frac{n(n+1)}{2}} && s = 1, \ldots, t \qquad (i,k) \in L^s && (4.14)
\end{aligned}$$

Taking (4.11) as the complicating constraints, we obtain the Lagrangian dual of $PTP(L)$:

$$v_{csd} = max_{(}\, \lambda)\, min_y \sum_{s=1}^{t} (\sum_{i=1}^{m} c_i^s(\lambda) y_i^s + \sum_{(i,k) \in L^s} d_{ik} z_{ik}) = max_{(}\, \lambda) \sum_{s=1}^{t} v_{ip}^{L^s}(\lambda) \tag{4.15}$$

subject to (4.12), (4.13) and (4.14).

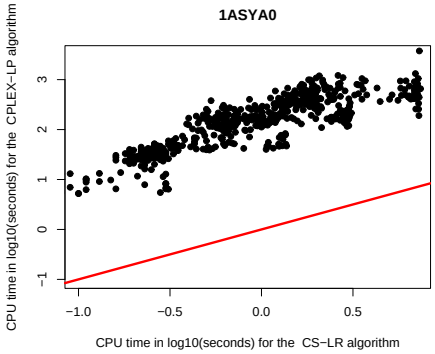
The Lagrangian multipliers λ^s are associated with the equations (4.11) and $c_i^1(\lambda) = c_i^1 + \sum_{s=2}^{t} \lambda^s$, $c_i^s(\lambda) = c_i^s - \lambda^s, s = 2, \ldots, t$. The coefficients c_i^s are arbitrary (but fixed) decomposition (cost-split) of the coefficients c_i, i.e. given by $c_i^s = p_s c_i$ with $\sum p_s = 1$. From the Lagrangian duality theory follows $v_{lp} \leq v_{csd} \leq v_{ip}$. This means that for each PTP instance s.t. $v_{lp} = v_{ip}$ holds $v_{csd} = v_{ip}$. By applying the subgradient optimization technique ([13]) in order to obtain v_{csd}, one need to solve t problems $v_{ip}^{L^s}(\lambda)$ (see the definition of $v_{ip}^{L^s}$) for each λ generated during the subgradient iterations. As usual, the most time consuming step is $PTP(L^s)$ solving, but we have demonstrated its $O(n^2)$ complexity in the case when L^s is a union of independent stars and single links. More details concerning the actual implementation are given in Appendix 2.

5 Experimental results

The numerical results presented in this section were obtained on an Intel(R) Xeon(TM) CPU 2.4 GHz, 2 GB RAM, RedHat 9 Linux. The behavior of the algorithm was tested by computing the same distributions as given in [3] (for the purpose of comparison), plus few extra-large instances based on real-life data generated by FROST (Fold Recognition Oriented Search Tool) software [9]. The MIP models were solved using CPLEX 7.1 solver [7].

In our first computational experiment we focus on computing score distributions phase and we study the quality of the approximated solutions given by three PTP algorithms. Five distributions are associated to any 3D template in the FROST database. They are computed by threading the template with sets of non related protein sequences having length respectively equal to : -30%, -15%, 0%, +15%, +30% of the template length. Any of these sets contains approximately 200 sequences.

Hence, computing a score distribution in the FROST database requires solving approximately 1000 sequence-to-template alignments. Only two values will be finally used : these are the score values obtained at the 1st and at the 3rd quartiles of the distribution (denoted respectively by q_{25} and q_{75}). FROST uses the following scheme : the raw score (RS) (i.e. the score obtained when a given query is aligned with the template) is normalized according to the formula $NS = \frac{q_{75} - RS}{q_{75} - q_{25}}$. Only the value NS (called normalized score) is used to evaluate the relevance of the computed raw score to the considered distribution.



Plot of time in seconds with CS-LR algorithm on the x-axis and the LP algorithm from [2] on the y-axis. Both algorithms compute approximated solutions for 962 threading instances associated to the template 1ASYA0 from the FROST database. The linear curve in the plot is the line $y = x$. What is observed is a significant performance gap between the algorithms. For example in a point $(x, y) = (0.5, 3)$ CS-LR is $10^{2.5}$ times faster than LP relaxation.

FIG. 4.1 – Cost-Splitting Lagrangian Relaxation versus LP Relaxation

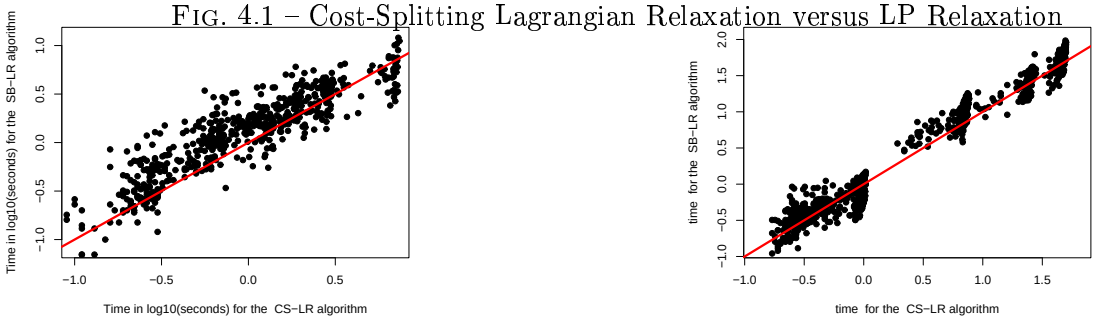
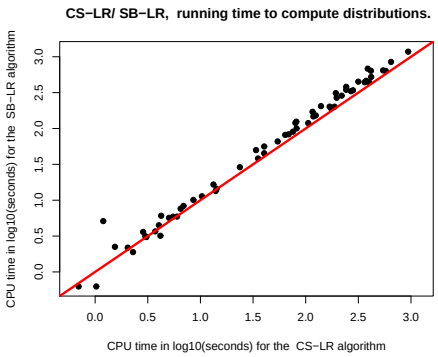


FIG. 4.2 – Plot of time in seconds with CS-LR (Cost-Splitting Lagrangian Relaxation) algorithm on the x-axis versus SB-LR (Stefan Balev's Lagrangian Relaxation) algorithm [3] on the y-axis concerning score distributions of two templates. Both the x-axis and y-axis are in logarithmic scales. The linear curve in the plot is the line $y = x$. **Left :** The template 1ASYA (the one referenced in [3]) has been threaded with 962 sequences. **Right :** 1ALO_0 is one of the templates yielding the biggest problem instances when aligned with the 704 sequences associated to it in the database. We observe that although CS-LR is often faster than SB-LR, in general the performance of both algorithms is very close.

We conducted the following experiment. For the purpose of this section we chose a set of 12 non-trivial templates. 60 distributions are associated to them. We first computed these distributions using an exact algorithm for solving the underlying PTP problem. The same distributions have been afterwords computed using the approximated solutions obtained by any of the three algorithms here considered. By approximated solution we mean respectively the following : i) for a MIP model this is the solution given by the LP relaxation ; ii) for SB-LR (Stefan Balev's Lagrangian Relaxation) algorithm this is the solution obtained for 500 iterations (the upper bound used in [3]). Any exit with less than 500 iterations is a sign that the exact value has been found ; iii) for the Cost-Splitting Lagrangian Relaxation algorithm (CS-LR) this is the solution obtained either for 300 iterations or when the relative error between upper and lower bound is less than 0.001.

We use the MYZ integer programming model introduced in [2]. It has been proved faster than the MIP model used in the package RAPTOR [17] which was well ranked among all non-meta servers in CAFASP3 (Third Critical Assessment of Fully Automated Structure Prediction) and in CASP6 (Sixth Critical Assessment of Structure Prediction). Because of time limit we present here the results from 10 distributions only.Concerning the 1st quartile the relative error between the exact and approximated solution is 3×10^{-3} in two cases and less than 10^{-6} for all other cases. Concerning the 3rd quartile, the relative error is 10^{-3} in two cases and less than 10^{-6} for all other cases.

All 12125 alignments for the set of 60 distributions have been computed by the other two algorithms. Concerning the 1st quartile, the exact and approximated solution are equal for all



Plot of time in seconds with CS-LR algorithm on the x-axis and the SB-LR algorithm on the y-axis. Each point corresponds to the total time needed to compute one distribution determined by approximately 200 alignments of the same size. 61 distributions have been computed which needed solving totally 12125 alignments. Both the x-axis and y-axis are in logarithmic scales. The linear curve in the plot is the line $y = x$. CS-LR is consistently faster than SB-LR algorithm.

FIG. 4.3 – CS-LR versus SB-LR : recapitulation plot concerning 12125 alignments.

cases for both (SB-LR and CS-LR) algorithms. Concerning the 3rd quartile and in case of SB-LR algorithm the exact solution equals the approximated one in all but two cases in which the relative error is respectively 10^{-3} and 10^{-5}. In the same quartile and in case of CS-LR algorithm the exact solution equals the approximated one in 12119 instances and the relative error is 7×10^{-4} in only 6 cases.

Obviously, this loss of precision (due to computing the distribution by not always taking the optimal solution) is negligible and does not degrade the quality of the prediction. We therefore conclude that the approximated solutions given by any of above mentioned algorithm can be successfully used in the score distributions phase.

Our second numerical experiment concerns running time comparisons for computing approximated solutions by LP, SB-LR and CS-LR algorithms. The obtained results are summarized on figures 4.1, 4.2 and 4.3. Figure 4.1 clearly shows that CS-LR algorithm significantly outperforms the LP relaxation. Figures 4.2 and 4.3 illustrate that CS-LR is mostly faster than SB-LR algorithm. Time sensitivity with respect to the size of the problem is given in Fig. 4.3.

6 Conclusion

The results in this paper confirm once more, that integer programming approach is well suited to solve protein threading problem. Here, we proposed a cost-splitting approach, and derived a new Lagrangian dual formulation for this problem. This approach compares favorably with the Lagrangian relaxation proposed in [3]. It allows to solve huge instances[2], with solution space of size up to 10^{77}, within a few minutes.

The results lead us to think that even better performance could be obtained by relaxing additional constraints, relying on the quality of LP bounds. In this manner, the relaxed problem will be easier to solve. This is the subject of our current work.

[2]Solution space size of 10^{40} corresponds to a MIP model with 4×10^4 constraints and 2×10^6 variables [20].

Bibliographie

[1] T. Akutsu and S. Miyano. On the approximation of protein threading. *Theoretical Computer Science*, 210 :261–275, 1999.

[2] R. Andonov, S. Balev and N. Yanev, Protein Threading Problem : From Mathematical Models to Parallel Implementations, INFORMS Journal on Computing, 2004, 16(4), pp. 393-405

[3] Stefan Balev, Solving the Protein Threading Problem by Lagrangian Relaxation, WABI 2004, 4th Workshop on Algorithms in Bioinformatics, Bergen, Norway, September 14 - 17, 2004

[4] D. Fischer, http ://www.cs.bgu.ac.il/ dfishcer/CAFASP3/, Dec. 2002

[5] A. Caprara, R. Carr, S. Israil, G. Lancia and B. Walenz, 1001 Optimal PDB Structure Alignments : Integer Programming Methods for Finding the Maximum Contact Map Overlap *Journal of Computational Biology*, 11(1), 2004, pp. 27-52

[6] H. J. Greenberg, W. E. Hart, and G. Lancia. Opportunities for combinatorial optimization in computational biology. *INFORMS Journal on Computing*, 16(3), 2004.

[7] Ilog cplex. http ://www.ilog.com/products/cplex

[8] R. Lathrop, The protein threading problem with sequence amino acid interaction preferences is NP-complete, Protein Eng., 1994 ; 7 : 1059-1068

[9] A. Marin, J.Pothier, K. Zimmermann, J-F. Gibrat, FROST : A Filter Based Recognition Method, Proteins, 2002 Dec 1 ; 49(4) : 493-509

[10] T. Lengauer. Computational biology at the beginning of the post-genomic era. In R. Wilhelm, editor, *Informatics : 10 Years Back - 10 Years Ahead*, volume 2000 of *Lecture Notes in Computer Science*, pages 341–355. Springer-Verlag, 2001.

[11] G. Lancia. Integer Programming Models for Computational Biology Problems. *J. Comput. Sci. & Technol.*, Jan. 2004, Vol. 19, No.1, pp.60-77

[12] R.H. Lathrop and T.F. Smith. Global optimum protein threading with gapped alignment and empirical pair potentials. *J. Mol. Biol.*, 255 :641–665, 1996.

[13] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1988.

[14] V. Poirriez, A. Marin, R. Andonov, J-F. Gibrat. FROST : Revisited and Distributed, HiCOMB 2005, Fourth IEEE International Workshop on High Performance Computational Biology

[15] R : A language and environment for statistical computing, R Foundation for Statistical Computing, Vienna, Austria, 2004, http ://www.R-project.org

[16] J.C. Setubal, J. Meidanis, Introduction to computational molecular biology, 1997, Chapter 8 : 252-259, Brooks/Cole Publishing Company, 511 Forest Lodge Road, Pacific Grove, CA 93950

[17] J. Xu, M. Li, G. Lin, D. Kim, and Y. Xu. RAPTOR : optimal protein threading by linear programming. *Journal of Bioinformatics and Computational Biology*, 1(1) :95–118, 2003.

[18] Y. Xu and D. Xu. Protein threading using PROSPECT : design and evaluation. *Proteins : Structure, Function, and Genetics*, 40 :343–354, 2000.

[19] N. Yanev and R. Andonov. Solving the protein threading problem in parallel. In *HiCOMB 2003 – Second IEEE International Workshop on High Performance Computational Biology*, 2003, Avril, Nice, France

[20] N. Yanev and R. Andonov, Parallel Divide and Conquer Approach for the Protein Threading Problem, Concurrency and Computation : Practice and Experience, 2004 ; 16 : 961-974

Appendix

1 Network flow model

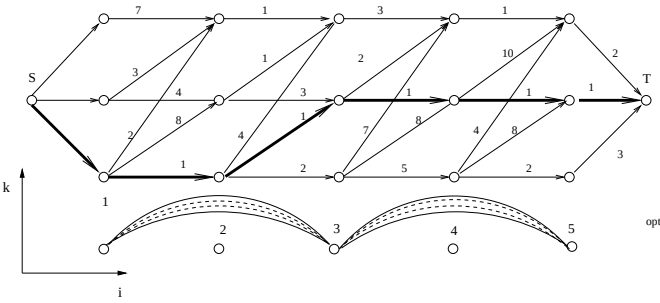


FIG. 4.4 – According to the model introduced in [2], any feasible threading corresponds to a path from S to T in this particular graph. The optimal threading is the augmented shortest path, i.e. the path charged with the cost induced by links of the contact graph (non-local costs). Here, the contact graph L is $\{(1,3),(3,5)\}$.

2 Implementation issues

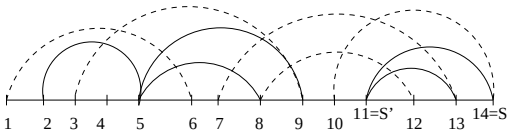
In order to apply the subgradient optimization technique to solve $PTP(L)$ one need i) to find a suitable partitioning of L into subgraphs with a special structure, and ii) to tune appropriately the parameters of the algorithm, used for finding v_{csd}.

2.1 Problem decomposition

The aim of cost-splitting techniques is to decompose a problem into subproblems of reasonable complexity. In our case, each subproblem should be a set of independent stars and single links. Several such decompositions exist, but the choice of a particular decomposition may impact a lot on performance : with a lot of subproblems, each one is simpler to solve, but it takes more subgradient iterations to reach a complete agreement between subproblems. Experimentally, the following decomposition appears to be suitable and is easy to obtain.

First all $(i, i+1)$ links ($i \in [1, m-1]$) are put in L^1 subproblem (if one of them, doesn't exist, it is added with a corresponding null cost matrix). All vertex costs c_i, $i \in [1, m]$ are affected to L^1 subproblem. Thus, all other subproblems have null vertex cost.

Then, given K a set of links we start constructing a subproblem by selecting the star with maximal left and right end (called right most star, rms), and delete it from K. Then we iterate this selection scheme, and each time add a rightmost star which is compatible with the last selected one (they are independent in the sense described above). When no star can be added to the subproblem, we start building a new subproblem following the same procedure until K is empty. Fig. 4.5 illustrates the construction of a subproblem on an example.



$$K^{14}_{rms} = \{(11,13),(11,14)\}$$
$$K^{11}_{rms} = \{(5,8),(5,9)\} \quad K^{5}_{rms} = \{(2,5)\}$$
subproblem : $K^{5}_{rms} \cup K^{11}_{rms} \cup K^{14}_{rms}$

FIG. 4.5 – Construction of a subproblem from a set of links K

2.2 Subgradient algorithm

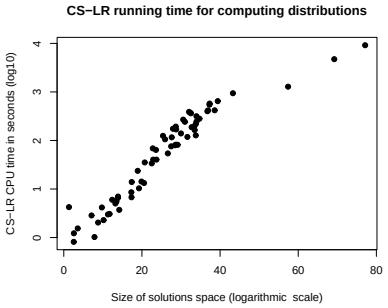
The subgradient ascend is an iterative search procedure that is used to maximize a concave function (for a comprehensive introduction, see [13]). In our case this is the function

$$v_{csd}(\lambda) = min_y\{\sum_{s=1}^{t}(\sum_{i=1}^{m} c_i^s(\lambda)y_i^s + \sum_{(i,k)\in L^s} d_{ik}z_{ik})\} \tag{4.16}$$

$$= min\{\sum_{i=1}^{m} c_i y_i + \sum_{(i,k)\in L} d_{ik}z_{ik} + \sum_{s=2}^{t}\sum_{i\in I^{1s}} \lambda^{is}(y_i^1 - y_i^s)\} \tag{4.17}$$

subject to maximize in (4.15). This function is obtained from (4.10) by relaxation of the complicated constraints (4.11). The sets I^{1s} are derived from the actual realization of the partition of L and by elimination from (4.11) all i not common for subproblems L^1 and L^s. If $I^s, s = 2, \ldots, t$ be the set of layers covered by L^s then $I^{1s} = I^1 \bigcap I^s$. Instead (4.11) to hold for each $i \in \{1, \ldots, m\}$ one can achieve the goals by taking $I^{1s}, s = 2, \ldots, t$ as inclusive sets for the respective i. Thus (4.15) is function of $\lambda = (\lambda^2, \ldots, \lambda^t)$ with $\lambda^s \in R^{n|I^{1s}|}$. From Lagrangian duality theory the added term $\sum_{s=2}^{t}\sum_{i\in I^{1s}} \lambda^{is}(y_i^1 - y_i^s)$ is used for approaching the optimal λ^* from the current one by taking a small step along the subgradient. In this case $(\bar{y}_i^1 - \bar{y}_i^s), i \in L^{1s}$(for each i this is a n-vector with only two non-zero coefficients equal to 1 and -1 resp.), with $\bar{y}^s$ being an optimal solution to $PTP(c^s(\lambda), L^s)$ is the part of the subgradient, corresponding to λ^s. At each iteration, $\bar{y}^1$ provides a feasible solution that is used to compute an upper bound u_b for the optimal value; a lower bound l_b is given by the highest found value of the Lagrangian. In our experiments the step length : θ_i in iteration i is controlled by $\theta_i = 1.4(u_b^i - l_b^i)\rho^i\rho_0/\text{nbm}$ where $\rho^{500} = 0.001$, ρ_0 is an initial guess for step length, and nbm is the number of violated relaxed constraints (the length of the subgradient).

3 Additional experimental results



Each point in this plot corresponds to the total time required by CS-LR algorithm to compute one distribution determined by approximately 200 alignments of the same size. About 60 distributions have been computed which needed solving about 12000 alignments totally. The size of the biggest instance is $O(10^{77})$.

FIG. 4.6 – Evolution in time as a function of the solutions space size.

Référence bibtex

```
@Article{BRILLANT06,
  author = {{Colin}, Samuel and {Petit}, Dorian and  {Mariano}, Georges and
  {Poirriez}, Vincent and {Rocheteau}, J\'er\^ome and {Marcano}, Rafa\"el},
  title = {Design and Implementation of a Cooperative Framework for B based
           Software Development:{BRILLANT}},
  journal =   {\texbf{SUBMITTED}Software and Systems Modeling},
  year =  {2006},
  publisher =    {Springer}
}
```

Design and Implementation of a Cooperative Framework for Bbased Software Development : *BRILLANT*

Dorian Petit[1], Samuel Colin[1], Georges Mariano[2], Vincent Poirriez[1], Jérome Rocheteau[2], Rafaël Marcano[2] *

LAMIH/ROI, UMR CNRS 8530

[1] Université de Valenciennes et du Hainaut Cambrésis
Le Mont Houy F-59313 Valenciennes Cedex 9, France
Firstname.Lastname@univ-valenciennes.fr

INRETS-ESTAS National Institute for Transport and Safety Research

[2] 20 rue Elisée Reclus - B.P. 317 F-59666 Villeneuve d'Ascq, France
Firstname.Lastname@inrets.fr

Abstract The need for the Bmethod first appeared in industry, and several commercial tools have been developed to support this formalism. However, few of these tools allow reasoning related to the formalism itself or on its possible extensions. This article presents an open-source platform, with a focus on the platform's core component, the BCamlproject. The tools presented here are used to show how very different approaches can be brought together around a central design to form a consistent toolbox that can be used to generate safe code, and to develop safe systems, from their specifications to their validation.

Keywords Bmethod, tool support, UML modelling, XML, proof tools, code generation

1 Introduction

During the last decade of the previous millennium, theoretical research produced the Bmethod, which is based on the same fundamentals as Z [11] and VDM [38]. This method reconciles the pragmatic constraints of industrial development of critical software with the strict theoretical requirements inherent to mathematical formalism. The Bmethod is one of the rare successful formal methods used in industry, and it supports multiple paradigms : *Substitutions* as a means for describing dynamic behaviour naturally ; *Formulas* in a simple yet efficient logical framework (set theory) ; *Composition mechanisms* that simplify development ; and *Refinements* that provide a safe and efficient way to obtain secure computer code from abstract specifications

*The present research work has been partly supported by Science and Technology for Safety in Transportation, the European Community, the Délégation Régionale à la Recherche et à la Technologie, the Ministère Délégué à l'Enseignement Supérieur et à la Recherche, the Région Nord Pas de Calais and the Centre National de la Recherche Scientifique : the authors gratefully acknowledge the support of these institutions

The Bmethod was used in the METEOR project [4], as well as in less well-known development projects [13, 18] and even non-critical development projects [51]. The software industry adopted Blargely because of the availability of software tools supporting all phases of the Bdevelopment process (semantics verification, refinement, proving, automatic code generation). Unlike most software tools, Bsupport tools resulted from prototypes developed by industry rather than by the academic community. In fact, from 1993 to 1999, the *Atelier B* development project was funded by the "Convention B", which was a collaborative effort of the RATP (Parisian Autonomous Transportation Company), SNCF (the French National Railway Society), INRETS (the National Institute for Transport and Safety Research) and Matra Transport (now Siemens Transportation Systems), among others.

A computer scientist in the field of formal methods will perceive certain paradoxes in the implementation of the Bmethod. For instance, the Bmethod uses programming languages, such as Bkernel, that are not well documented or specified and that are not particularly well-suited to B's high level of abstraction. In addition, some have observed that the Bmodule language is rather complex and hard to understand. G. N. Watson [53] goes as far as to call the proliferation of constructs in the Bmodule system, and the rules associated with them, somewhat *daunting*. Certainly, the semantics of the Bmodule system has been studied by several research teams. Bert *et al.* [6] have developed a component algebra to express the INCLUDES and USES clauses. Potet *et al.* [45] have studied the IMPORTS and SEES links, demonstrating that, in B, it is possible to build an erroneous project that appears to be correct ; they have also proposed criteria derived from the dependency graph, which must be verified to eliminate erroneous architectures. Dimitrakos *et al.* [19] have studied the different composition clauses, focusing on the conditions that must be verified to preserve the various component properties. Of course, it can be argued that Bmodularity is complex because there are different kinds of restrictions to ensure correctness. In any case, the inherent complexity of the Bmethod makes providing a synthetic way to understand these modular constructs important.

These apparent paradoxes can be explained by the industrial origins of the method. Nonetheless, the fact remains that the industrial tools currently available for Bare often inappropriate for scientific research, and thus do not provide effective support for efforts to extend the use of the Bmethod.

To remedy this problem, we have proposed the *BRILLANT*[12] framework, showing the feasibility of a safe software development system that ranges from semi-formal specification (UML) to contract-equipped code generation. This framework provides a central core into which different components can be plugged, including a central component, a UML plug-in, a proof plug-out, and a code generator plug-out. Because we agree with Bert *et al.* [6] that the modularity of the Blanguage can be separated from the base/core language, we have adopted the powerful Harper-Lillibridge-Leroy modular module (**HLL**) system [24, 31], which builds a module language from a given core language without modularity constructions. HLL has the advantage of separating the semantics of the core language from that of the module language. Moreover, the precise semantic provided by instanciating HLL in the Blanguage guarantees the preservation of the specified visibility rules at no extra cost to the system.

The ability to view Bmodularity through an HLL semantic is a theoretical result, established by Petit in his 2003 dissertation [39] ; it does not change the work of the Bspecifications writer, who should continue to specify Bmachines in the usual manner. However, it may shed light on the problem and thus facilitate the modular design of Bspecifications. In this way, the central core addresses the problem of obscurity in current Blanguage modular constructs.

This paper is a revised and enhanced version of the article presented in 2005 [15]. In this revised version, we use a specification of the level crossing problem inspired by Einer *et al.*. [21] as a case study, although some parts of the specifications (the traffic lights, for example) are specific to French systems. Our example concerns a single railroad line system. This system is composed of :

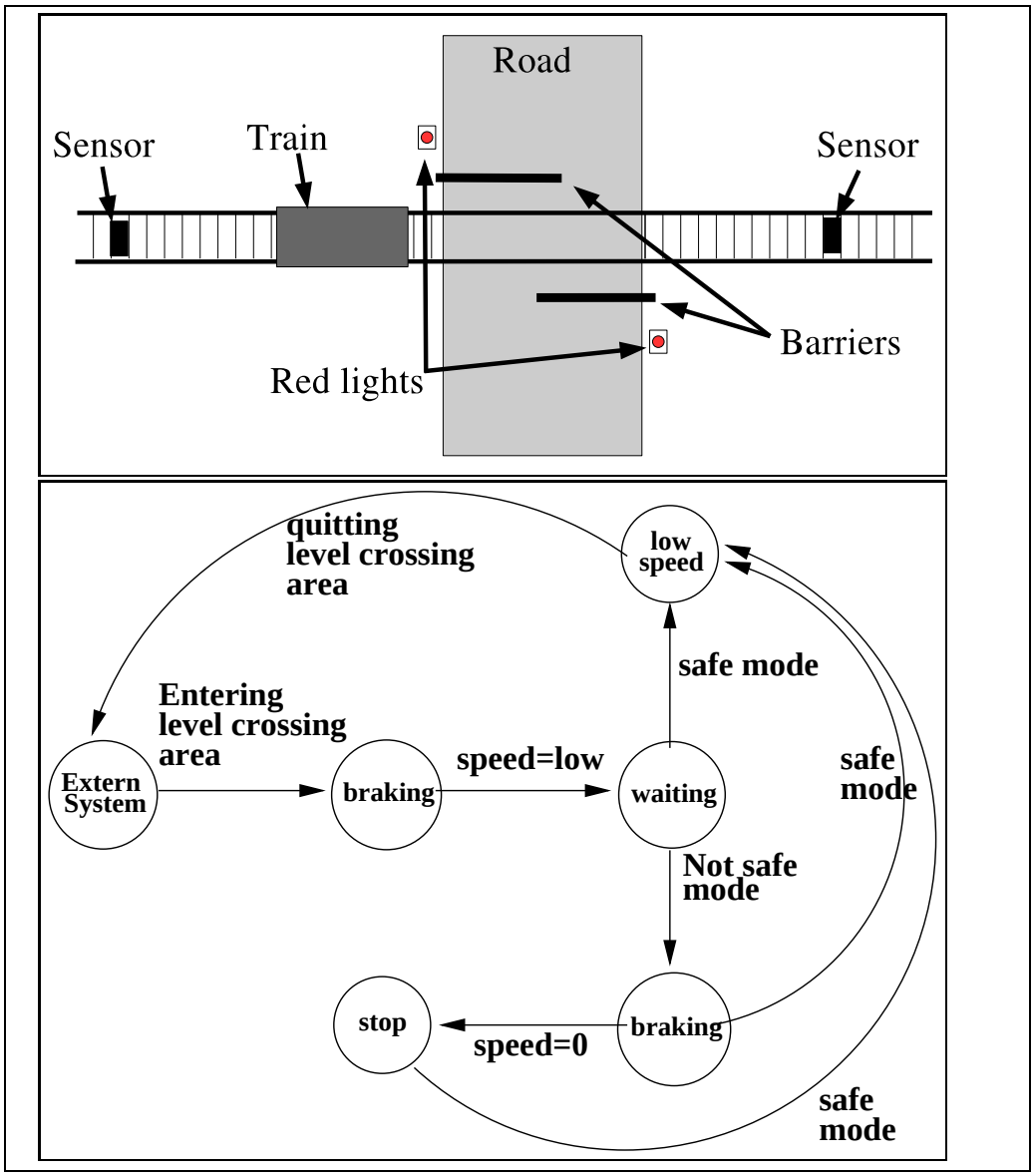


FIG. 5.1 – The level crossing problem

- sensors placed on the railroad line to detect the beginning or ending of a level-crossing protection procedure,
- barriers and road traffic lights to stop cars from passing,
- an onboard train system to activate certain train functions (e.g., braking), and
- a sequencer to specify the different states of the system and the transitions.

This problem is illustrated in figure 5.1. The top of the figure presents a schematic view of the overall problem, and the bottom shows a schematic representing the automata for the different system states. The state *External System* characterises the initial and final state of the automata.

Section 2 presents the central component of *BRILLANT*, which allows the components of a Bproject to be manipulated. The parsing of Bmachines is examined in section 2.1, highlighting the central role of XML as an exchange format. Abstract syntax tree manipulations, such as specification flattening, are examined in section 2.2, and proof obligation generation, in section 2.3. Section 3 presents the proof plug-out used to validate Bproof obligations, and section4 describes a code generator plug-out that supports several target languages and is able to embed contracts into the code. Section 5 describes the UML plug-in used to translate UML projects into Bso as to

validate them, as described by Marcano & Levy [34] and Laleau & Polack [29]. Finally, section 8 presents our conclusions based on the results of our experiments with the implementation and use of the *BRILLANT*platform.

2 The BCaml Kernel

In this Section, we describe the three component parts of OCaml : a parser (with an XML output library to connect BCamlwith the outside world), libraries to handle the modularity of Bprojects, and a proof obligation generator.

2.1 BCaml input and output

The kernel of the BCamlplatform is made up of the first bricks that were developed when the platform was created. These bricks specify the concrete grammar (section 2.1) that defines the Blanguage and the abstract syntax (section 2.1) that defines the type used to manipulate the specifications. This may seem obvious, but it is nonetheless important because it makes collaboration with other developers possible. One of these collaborative projects led to the development of another brick in the BCamlkernel, called the *Btyper*. This brick is not described in this paper, but more details can be found in Bodeveix & Filali [8].

A concrete grammar for B

Several Bgrammars have been introduced over the years, coming from a variety of sources : *Clearsy* (prev. *Steria*), which corresponds to the grammar used in *Atelier B* ; *B-Core*, which corresponds to the grammar used in *B-Toolkit* ; and Mariano's PhD dissertation [36], based on the *B-Core grammar*, which was introduced to do metrics on Bspecifications.

In order to build a tool that would be as useful as possible, we needed to define a grammar that would take into account the criticisms of the grammars presented above. Our BCamlchoices had to respect the following constraints :

- they had to be as compatible as possible with the machines that can be correctly parsed by the commercial tools mentioned above (*Atelier B*, *B-Toolkit*),
- they had to comply with the standard *Lex* and *Yacc* tools that allow *LALR* grammars to be defined, and
- they had to cause as few conflicts as possible.

We chose OCaml [30] as a support tool for our Bdevelopment for several reasons. Not only does it allow symbolic notations to be handled easily, but in addition, it also implements efficiently and comes bundled with tools that allow the parsing of *LALR* grammars. Using this tool, we defined the Blanguage in *LALR*. Some of the problems we encountered in doing so are described below.

Peculiarities in the parsing of Bmachines Certain Blanguage features made defining an adequate grammar for the Bmethod difficult, including :

- The **records**, whose **;** separator causes a conflict with the separator currently used to define the sequence of two substitutions. We decided to give priority to the correct parsing of substitution sequences, to the detriment of a correct parsing of records.
- The **definitions**, designed to lighten repetitive, cumbersome notations, which cause a partial conflict with the *LALR* grammars. We decided not to expand the definitions and to keep them in the abstract syntax trees used in the original text, thus restricting the definitions to a subset that is expressive enough to allow them to be implemented with the *LALR* parser without losing too much of their usefulness.

– Several so-called *reduce/reduce* and *shift/reduce* **conflicts**, which represent ambiguous notations. We decided to remove the *reduce/reduce* conflicts, but to defer our decision about the *shift/reduce* conflicts.

Abstract syntax and XML syntax—two isomorphic formats

We used our definition of abstract syntax to directly infer an XML representation for Bformal specifications. (Due to lack of space, this abstract syntax is not described here.) This XML encoding is called "B/XML" and is stored in an XML DTD file. Such abstract syntax is, as could be expected, more tolerant than concrete syntax, and contains elements that facilitate the handling of the syntax structure. For instance, the [*substitution*]*predicate* and [*variable_instanciation*]*substitution* constructions appear in this abstract syntax, which means that the structure can be manipulated to bring it closer to the matching mathematical definitions given in the B-Book [1].

We chose XML as our pivot format because of its flexibility and its ease-of-use with third-party tools. Using it makes our tools as independent of one another as possible, allowing a researcher to use our parser, but someone else's proof tool, for example. This flexibility is due to the XSL style sheets that formulate simple recursive treatments of the XML structure, mostly transformations into other structured formats (LaTeX, HTML, or PhoX, as mentioned in section 2.3).

2.2 Abstract syntax tree manipulation

The flattening algorithm

Overview Bspecifications are flattened by eliminating the refinement and composition links. The flattening algorithm uses one set of Bcomponents to build a single Bcomponent equivalent to the original set of Bcomponents. All the information for the different specifications are then grouped in one formal text. This notion of flattening exists implicitly in the BBook [1]. Though Potet and Rouzaud used the term "flattening" in their work [45], it was S. Behnia [5] who specified the algorithm entirely, and it is this specification that we used in our tool. The principle of the algorithm is to connect the specification from the leaves (where only the `IMPORTS` and `REFINES` links are taken into account) to the root machine of the project.

The enrichment mechanism Our flattening tool uses an enrichment mechanism that combines two specifications in one. This enrichment mechanism is described briefly below. (More details can be found in Petit [40].)

Refinement : In `REFINES` links, flattening consists of combining some clauses (`SETS`, `CONSTANTS PROPERTIES`) and using the more concrete parts of the specifications for other clauses. For example, let us consider an abstract machine M and its refinement R. Let *VarM* (respectively *VarR*) be the variables of the machine (the refinement) and *InvM* (*InvR*), the invariants of this machine (this refinement). The variables of the flattened component are the variables of the refinement, but these variables are renamed if the same variable name exists in the more abstract component. In the case of renaming, a gluing invariant is added, and the new variable name is propagated. Let *VarR_1* be this new variable clause, and *InvR_1*, the invariant in which the variables are renamed. Thus, the invariant of the flattened component is $\boxed{\exists\, VarM.(InvM \wedge InvR\_1)}$. Every specification property follows the same schema, in which the abstract variables are existentially quantified because they disappear in the flattened component.

Importation : In the `IMPORTS` links (as in the `INCLUDES` links), flattening consists of merging some clauses (`SETS`, `CONSTANTS PROPERTIES`, `INVARIANT`, `INITIALISATION`), instantiating the parameters in the imported machine, and then expanding the operation of the machine that is called in the implementation phase.

Implementation : The flattening tool was the first tool implemented after the BCamlkernel was designed (section 2). One of the aims of this implementation was to "evaluate" the kernel's usability and to add to the platform those tools/libraries that would be useful for manipulating Bspecifications. For implementation, two things had to be done. First, the specification dependency graph had to be made manipulatable, since it is necessary to navigate through the specifications in order to build the successive flattened components. Therefore, we developed a library called BGraph that implements the dependency graph type and the functions needed to manipulate that graph. Second, all the conditions that allow a set of Bcomponents to be flattened had to be verified. The total implementation of the flattening tool (condition verification + algorithm implementation) requires about 3000 lines of OCaml code.

The B-HLL module system

Overview The Harper-Lillibridge-Leroy module system (HLL) presented in Leroy [31] formalises the Standard ML-like modules. The HLL system provides a means for adding a module language to a module-less core language. This system also permits a formal semantic to be given to an existing module language, as is the case for the ML modules. Moreover, this powerful semantic is able to implement the module language with relative simplicity.

Once the HLL module system has been instantiated, it is possible to define structures (i.e. list of values, types, modules or sub-modules) and functors (module-to-module functions) in the obtained modular language. A more complete description of our work on B-HLL can be found in our article published in 2004 [43].

Instantiation Figure 5.2 sumarizes our use of the HLL system. Our efforts to instantiate the HLL module system were divided into two parts. The first part involved defining the abstract language of the the Bcore language under study, based on the abstract syntax defined during the development of the BCamlKernel. From this abstract syntax, we removed the part of the syntax dedicated to the modularity language, and then we developed a mapping function from the BCamlkernel abstract syntax to our new abstract syntax.

The second part of the instantiation involved defining the type checker. The types and the type-checking algorithm we used were adapted from the work of J.P. Bodeveix and M. Filali [8]. We added some type-checking rules to express the visibility rules described in the B-Book [1], and we also defined type checking rules that take into account Blanguage particularities, such as the semi-hiding principle and the prohibition of calling a given operation in the component where that operation is defined.

To translate the visibility rules, we had to divide the classes of things that can be defined in a Bspecification into several sub-classes. For example, the substitution constructions were divided into B0 substitution and non-B0 substitution. This sub-division allowed us to express certain rules, such as "an abstract variable can not be used in a B0 substitution, but can be used in the other substitutions".

The B_to_BHLL tool that translates the specification from the kernel's abstract syntax into our B-HLL syntax also generates four interfaces for each Bcomponent. Each of these interfaces is used to simulate the four composition links under study : `INCLUDES`, `IMPORTS`, `SEES` and `USES`. Generating these interfaces allows us to translate the visibility rules that make up the Bmodule language.

2.3 Generating proof obligations

In this section, we first describe the method used to implement the existing calculus for the weakest precondition. Then, we show how this calculus can be used to generate a Bproject's proof obligations. Last, we present the different options available for exporting these proof obligations to other formats and other tools.

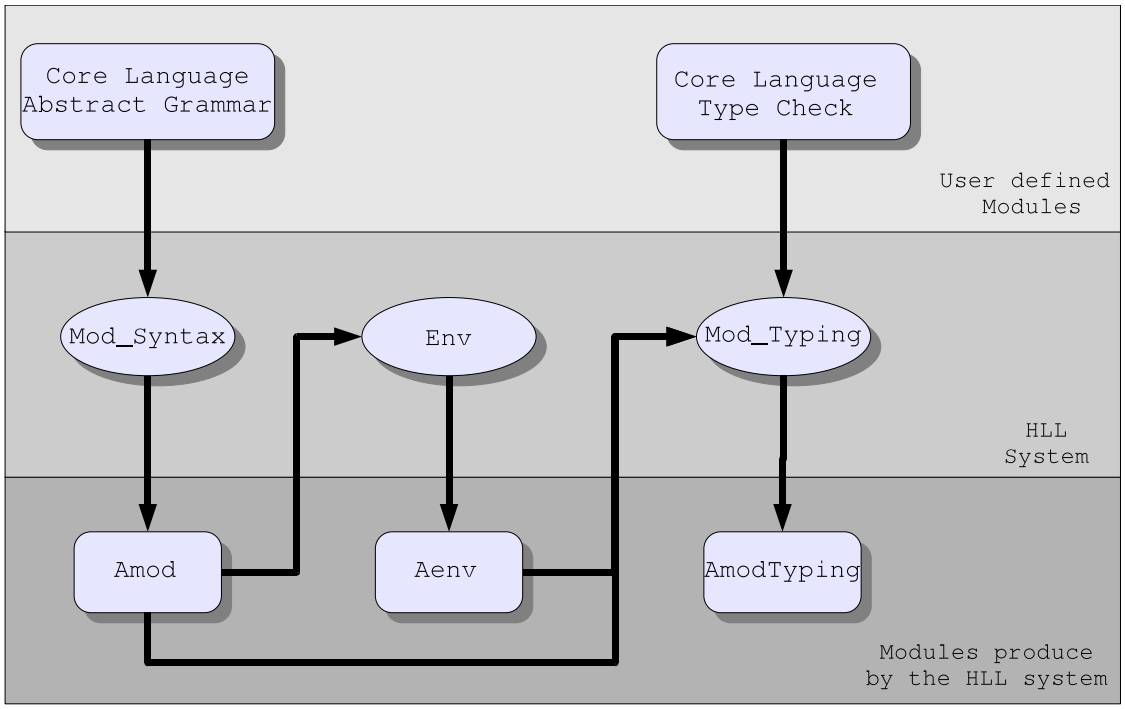


FIG. 5.2 – Using the HLL module system

Generalised substitution Language (GSL)

In order to generate proof obligations for Bmachines, we must be able to calculate the weakest preconditions of the substitutions. We chose to use the approach defined in the B-Book [1] : reducing Bsubstitutions to their smallest syntactic and semantic set (i.e. generalised substitutions). In the following paragraphs, we use *GSL* to designate both the syntactic set and the substitutions that compose it. Like the B-Book [1, B.3], we define the *GSL* in BCamlas an abstract data type, with the following notable exceptions :

- The assignment is defined as a multiple substitution ; it serves as a basic construct once the parallel substitutions have been reduced.
- The repetition substitution "$\hat{}$" does not appear ; instead, we use the *while substitution* since it does not exist in the loop proof rules [1, E.7].
- The instantiation of a substitution variable ([*variable* := *expression*]*substitution*) is reduced before transforming the substitution.

With the help of the abstract data type, proof obligations can be generated according to the rules described in the B-Book [1, appendix E]. The corresponding BCamlcode was written with readability in mind, making it easy to match the code with the rule from which it is derived.

Proof Obligation Generation

The main steps for generating proof obligations from a project can be divided into precise steps. These steps are described in more detail below :

Parsing First, the machine and all the machines it depends on are parsed. This parsing phase is followed by a scoping phase in which all unique identifiers that represent the same variable name, machine name or operation name are made equal. In fact, prior to the parsing phase, all identifiers are associated with a single stamp ; however, when the parsing is finished, all the identifiers have different stamps. The scoping phase acts to make the stamps for those identifiers that represent

the same variable, machine or operation name equal in terms of visibility.

Generation of formulas The formula generation step is based on the B-Book [1, appendix F], resulting in proof obligations with the following shape :
$[Instanciation]Hypothesis \Rightarrow [substitution]Goal.$
This generation method allows more handling flexibility later on, for instance during debugging, or when showing students how proof obligations are generated, or when the proof tool applies the substitution to the goal. Bodeveix [7] has shown how substitution calculus can be defined in Coq and PVS. It is possible to use this calculus to rewrite the uncalculated proof obligation to obtain one that is more concise (but less understandable). Figure 5.3 shows an example of an uncalculated proof obligation, derived from the Bproject presented in section 5.

$$
\begin{array}{l}
\quad (\mathrm{LCC} \in \mathbb{P}_1(\mathbb{INT}) \\
\wedge\ \mathrm{STATE} \in \mathbb{P}_1(\mathbb{INT}) \\
\wedge\ \mathrm{STATE} = \{\mathrm{Deactivated}, \\
\quad \mathrm{ShowingYlight,ShowingRlight}, \\
\quad \mathrm{ClosingB,OpeningB,ClosedB,Failure}\} \\
\wedge\ \ldots \\
\wedge\ \mathrm{Yellow.lState(yellowLight(obj))} = \mathrm{On}) \Rightarrow \\
\quad [\ \mathrm{state(obj)} := \mathrm{ShowingRlight} \\
\quad \|\ \mathrm{Yellow.switchOff(yellowLight(obj))} \\
\quad \|\ \mathrm{Red.switchOn(redLight(obj))}\ ] \\
\quad (\ \mathrm{lcc} \subseteq \mathrm{LCC} \\
\quad \wedge\ \mathrm{lcc\_barrier} \in \mathrm{lcc} \rightarrow \mathrm{barrier} \\
\quad \wedge\ \ldots \\
\quad \wedge\ \forall \mathrm{obj}\ .\ (\ \mathrm{obj} \in \mathrm{lcc} \\
\quad \wedge\ \mathrm{bStatus(lcc\_sensor(obj))} = \mathrm{Opened} \\
\quad \wedge\ \mathrm{bState(lcc\_barrier(obj))} = \mathrm{Closed} \Rightarrow \\
\qquad \mathrm{mode(obj)} = \mathrm{Unsafe})\ )
\end{array}
$$

FIG. 5.3 – Uncalculated proof obligation for the **timeOut_1_showRlight** operation

Optimisations Several additional optimizations, or processing procedures, can be applied to the generated formulas. For example, formulas can be calculated, resulting in predicates that contain no substitutions. It is also possible to split the goal, by splitting the formula into as many formulas as there are members of the conjunction in the goal :
$(H \Rightarrow G_1 \wedge \cdots \wedge G_n) \rightsquigarrow (H \Rightarrow G_1), \ldots, (H \Rightarrow G_n)$
Other possible optimisations that were not implemented include removing formulas when the goal is trivially true or appears in the hypotheses, or changing the shape of the formula to adapt it to a precise theorem prover. Certainly, it is sometimes easier to apply such transformations to the abstract syntax tree than to XML files using stylesheets.

Final files and trace information Once the formulas have been generated, some trace information is embedded into the resulting file. Trace information can be found in the absolute name of the file, which reflects the kind of proof obligation is in the file, and the machine from which it is generated. The XML information in the file contains not only the predicate itself, but also a

root tag named (for obvious reasons) `ProofObligation`. In addition, the file contains a tag that includes all the free variables of the formula because some theorem provers require that all variables be bound. This tag helps the stylesheet to generate a file for such theorem provers more easily.

Exporting to other tools

Once the proof obligations in the XML format are available, the XSL style sheets allow them to be exported to other tools. For instance, the proof obligations can be transformed into LaTeXfiles (figure 5.3 is an example of the results obtained) ; into text files, which are easily read by humans ; into HTML files, which improve the readability of the formulas ; or into BPhox files, which allow the proof obligations to be verified.

Figure 5.4 in section 3.2 presents an example of an XSL stylesheet application for the proof obligation shown in figure 5.3. First, the header of the file is inserted, which provokes the loading of the appropriate PhoX library and finetunes the power of the prover (the `flag` commands). Then, the free variables in the formula (the identifiers after the `/\` quantifier) are quantified. The formula itself is inserted into the BPhoX syntax by replacing the conjunctions of the hypotheses with implications, which facilitates the work of PhoX. Finally, the command that is given to the prover is added to start the proof (`Try intros ; ; auto.`). In the next step, the theorem prover is fed the generated proof obligations file (see section 3). All of these steps (including replacing the conjunctions in the hypotheses with implications) are done via the XSL stylesheet, demonstrating the *ad hoc* quality of this technology designed for simple treatments involving recursivity.

Now that the BCamlcore has been presented, the different plugins/plugouts revolving around it can be introduced, starting with a translator that transforms UML/OCL specifications into Bspecifications.

3 From Bproof obligations to correctness

The BCamlprovides the first two important types of Btools, presented in Abrial's $B\#$ [2, section 4]. The first includes the lexer, parser and typer ; the second, the proof obligation generator. The third and last important Btool is the automatic, interactive prover. We chose not to develop such a tool with BCamlfor a pragmatic reason : building a Bprover according to our specifications takes much more time than developing dedicated libraries for an already existing prover. Instead, we built a replaceable add-on. We included the PhoXproof checker [44] because 1) it can be extended to the Bmathematical foundations ; 2) its GPL licence permits distribution with BCaml ; 3) its developers were willing to work closely with us ; and 4) its highly intuitive syntax minimises library development time.

Our contributions to a PhoX-based Bprover include a process killer used to control the proof time, the Bextension of PhoX, the translation from Bto the PhoXextension and the B/PhoXGNU Make script that binds those tools together.

3.1 The *bphox* GNU Makescript

A Bprover is designed to verify whether each proof obligation is a theorem or not. In BCaml, every B/XML obligation has to be translated into the PhoXsyntax and has to be proved. PhoX- produces a po.pho file from a po.phx translated proof obligation when the proof is successful. The B/PhoXsession involves a one-step (.phx to .pho) or a two-step (.xml to .phx then .phx to .pho) transformation, depending on the file extension. This process corresponds exactly to the GNU Make transformation using a suffix schemata, and can be copied and configured to link BCamlto other theorem provers. As proved by Rocheteau *et al.* [49], the principal property that must be preserved throughout this process is that every sentence is a Btheorem, if and only if its translation is a B/PhoXtheorem.

```
add_path "/usr/share/brillant/bphox/".
Import Blib.
flag auto_lvl 2.
flag auto_type true.
theorem op
/\Activated,BARRIER,Closed,ClosedB,
  Closing,ClosingB,Deactivated,DownSpeed,
  ...,
  Yellow.lState,Yellow.light(
(LCC in (part1 Z)) ->
(STATE in (part1 Z)) ->
...
((Yellow.lState app (yellowLight app (obj))) = On)
  -> (
 /\obj ((
   ((obj in lcc) &
    ((state <+ \o (o = obj,ShowingRlight) app (obj))
       in Activated)) &
    ((bStatus app (lcc_sensor app (obj))) = Opened))
   ->
  ((mode app (obj)) = Unsafe)) ) )
 .
Try intros ;; auto.
save.
```

FIG. 5.4 – One of the exploded proof obligations for **timeOut_1_showRlight**, converted to B/Phox

3.2 The *bgop2phox* XSLstyle sheet

During the translation step, our XSL*bgop2phox* stylesheet is applied to the B/XMLproof obligationsusing a XSLTprocessor. The XSLtransformation schema allows recursive mapping. Thus, our translation is also defined recursively. A first-order language *à la B* is composed of different symbols for functions, relations, connectors and quantifiers. Figure 5.4 in section 2.3 shows a PhoXobligation generated from the proof obligation shown in figure 5.3, after it has been calculated and saved into an XMLfile.

A high-order language *à la PhoX* is a simply-typed lambda calculus with some typed constants. Our translation is based on associating every first-order Bsymbol S with a B/PhoXexpression $S^{\dagger}$, such that its extension to the first-order terms formulae is simply defined by an inductive commutation. Thus, using the PhoXsystem of simple types makes our translation sound. Moreover, every non-freeness rule and every substitution rule can be easily obtained through the λ binder properties.

3.3 The *blib* PhoXlibrary

The PhoX library for Breflects the first three chapters of the B-Book. The content of the library is outlined briefly here because the process for embedding Binto PhoXis based on it. (More details are available in Rocheteau *et al.* [49]). In fact, the library is a collection of successive libraries for predicate calculus with equality, the boolean domain, cartesian products, set operators, binary relations, functions, arithmetic theory and finite sequencing.

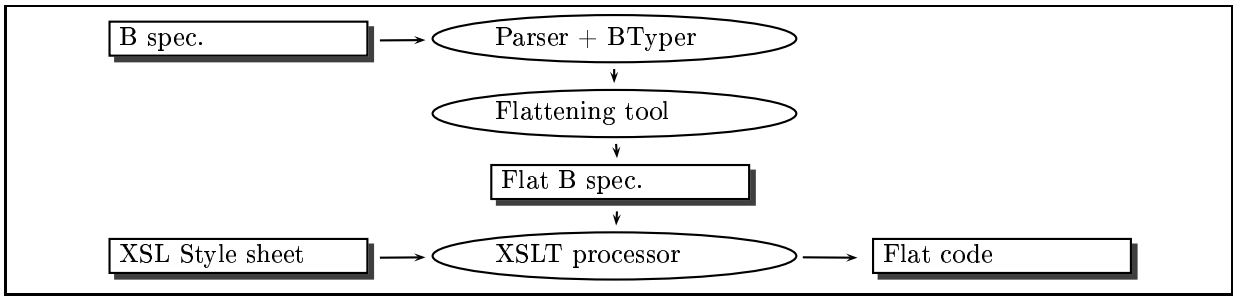


FIG. 5.5 – The code generation process to obtain flat code

3.4 The *chronos* process killer

The proof step consists of producing a po.pho file from a po.phx one. The existence of the po.pho file means that the required obligation holds. The absence of this file can mean that the proof obligation does not hold. It can also mean that the proof can not be completed due to lack of time or space, causing the proof session to loop endlessly. In order to deal with this problem, every PhoXcall is accompanied by a process killer named *chronos*.

Using the *chronos* process killer produces the following proof obligation classification : obligations with a successful proof, those with a failed proof, and those with a killed proof. The following table gives the results for several different time-out proof sessions for the famous boiler Bproject.

Time-out	1 s.	5 s.	60 s.
Generated proof obligations		2295	
Successful	1823	1955	1971
Failed	0	0	0
Killed	462	340	324
Proof Rate	79%	85%	85%

A successful proof obligation is built using a fixed-point application. Assuming an increasing function f on natural numbers, which means that the $n+1^{\text{th}}$ time-out is greater than the n^{th} time-out, the first session runs through the whole set of generated proof obligations, and the $n+1^{\text{th}}$ session runs only through the n^{th} session's killed proof obligations. However, since using PhoXas a "black box" does not allow the state of the proof to be saved at that moment it is killed, the next session (the $n+1^{\text{th}}$) replays the unkilled proof obligations from the beginning.

4 From Bspecifications to code

The code generation process is summarized in Figures 5.5 and 5.6. The first figure illustrates the generation process for producing flat code, and the second figure illustrates the process for producing component-oriented code. (More details on our approach to generating code can be found in [41] and in [42].) To generate flat code, the specifications have to be parsed, annotated with their type, and then flattened. Code can be easily generated from the flat Bspecifications by using a XSLTprocessor and the appropriate stylesheet. To generate component-oriented code, the specifications must be parsed, and then translated into B-HLL specifications ; these, in turn, are annotated with their type. Once the specifications have been typed, they are run through the part of the flattening algorithm dedicated to eliminating refinement links in order to produce B-HLL components. A stylesheet is then applied to the components thus obtained in order to generate the code.

Figure 5.7 presents a Bspecification of a bounded stack. The code presented in figure5.8 is generated from this specification. The package specifications use the generic Ada construction to

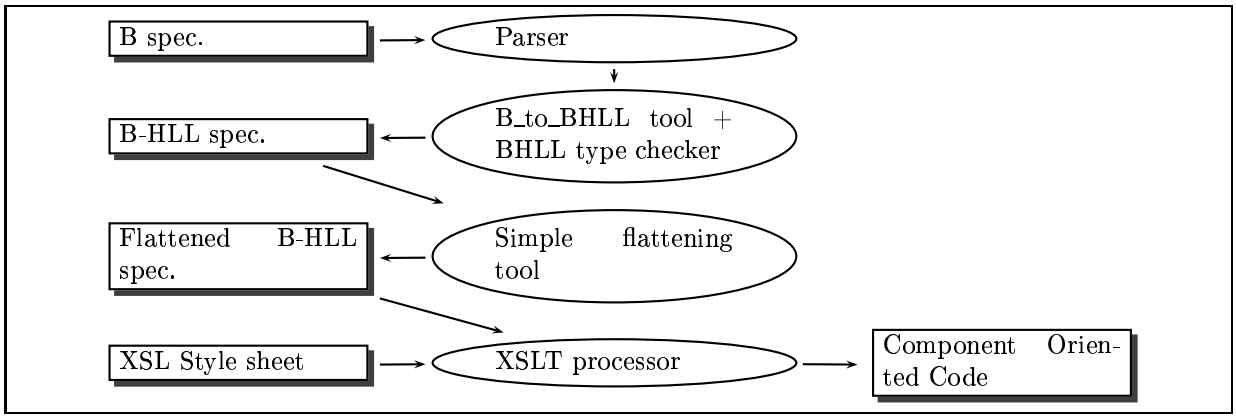


FIG. 5.6 – The code generation process to obtain component-oriented code

MACHINE
stack(stack_size)
CONSTRAINTS
stack_size : $\mathbb{N} \wedge$ stack_size ≥ 1
$\wedge$ stack_size $\leq$ MAXINT
VISIBLE_VARIABLES
the_stack, stack_top
INVARIANT
the_stack : (1..stack_size) $\rightarrow \mathbb{N}$
$\wedge$ stack_top : $\mathbb{N}$
$\wedge$ stack_top ≥ 0
$\wedge$ stack_top $\leq$ stack_size
INITIALISATION
the_stack : : (1..stack_size) $\rightarrow \mathbb{N}$
|| stack_top := 0
OPERATIONS
push(addval) =
PRE
stack_top < stack_size
$\wedge$ addval $\in \mathbb{N}$
THEN
stack_top := stack_top + 1
|| the_stack(stack_top + 1) := addval
END...
END

IMPLEMENTATION
stack_1(stack_size)
REFINES
stack
INITIALISATION
the_stack := (1..stack_size) * {0}
; stack_top := 0
OPERATIONS
push(addval) =
BEGIN
stack_top := stack_top + 1
;
the_stack(stack_top) := addval
END...

END

FIG. 5.7 – A Bspecification of a bounded stack

translate the parameters that specify the size of the stack. Our approach to code generation allows the properties that are expressed in the specifications to be put into the code. (Please note that the code generation step described above did not use the example introduced in the next section but a bounded stack specification. This toy exemple is more adapted to illustrate the code generation process.)

5 From UML/OCL models to Bspecifications

5.1 From Requirements to UML models

The "elicitation problem" (i.e., the creation of a valid initial description of the required system properties) is essential to ensure the correctness of the future system. The consistency of the system depends on the developer's ability to understand and incorporate key safety properties. Therefore, knowledge of the context in which the system operates plays an important role in elicitating system requirements. Our approach to requirement analysis is based on the approach taken by Marcano

```
generic
   stack_size :  natural  ;

package stack is
   function  is_empty return boolean ;
   procedure  push(addval : in natural );
   procedure  pop;
   function  top return natural ;
   function initialised return boolean;
end stack;
```

```
package body stack is
--# invariant stack_top >= 0
--# and stack_top <= stack_size

   the_stack : array (1..Stack_size) of natural ;
   stack_top : 0..Stack_size ;

   procedure push(addval : in  natural ) is
   begin
      --# pre stack_top <  stack_size
      stack_top:=stack_top +  1;
      the_stack(stack_top) := addval;
   end push;

   ...

begin --initialisation
  stack_top := 0
end stack;
```

FIG. 5.8 – The specification and the body of the bounded stack Ada package

et al. [35], in which both the static and dynamic properties of the system are taken into account through different UML diagrams.

Describing the entities involved and their invariants facilitates comprehension of the static properties. Describing the way that system actors will interact with each other and the system leads to a full comprehension of the dynamic properties of the future system. The Object Constraint Language (OCL) is used to express all the properties that cannot be expressed through diagrammatic notation alone (i.e., hypotheses and facts related to subsystems, classes, attributes and associations). OCL is also used to describe the system safety conditions. Thus, system modeling can be divided into two steps : first, UML sequence diagrams must be defined in order to describe both correct operating scenarios and failure scenarios. Second, the expected behaviour of the system must be described completely as a set of OCL pre- and post-operational conditions. UML state diagrams will be defined for the combined operations of the entire system in order to describe the overall interaction of the system with each actor in its environment.

5.2 A Railway Level-Crossing system

The traffic control system considered in this paper has been described in detail by Jansen and Schnieder [27]. They include knowledge about the railway domain as a basis for the formal specifications. The problem is to specify a radio-based Railway Level-Crossing (RLC) application developed for the German Railways [23]. This application is distributed over three sub-systems : a train-borne control system (on-board system), a level-crossing control system, and an operations center system. The level crossing is situated on a single-track railway line at a point where the line crosses a road on the same level. The intersection of the road and the railway line is considered the

danger zone, since train traffic and road traffic must not enter it at the same time. Note that this is the main safety constraint that must be taken into account when describing the system.

5.3 UML-based modelling

The whole system is composed of three sub-systems which communicate with each other, with the Level-Crossing Control subsystem (LCC) being central. The other sub-systems-the Train-borne Control system (TC) and the Operation Center (OC)- are cooperating actors that make use of the LCC.

First, it is necessary to identify the main entities that must be modelled to determine possible LCC system failure conditions. A primary cause of such failure conditions could be malfunctioning sensors or actuators. Defects leading to failures may be detected in the main physical structures or in the control systems themselves. In this case study, only a limited number of failures are considered : failures of the yellow or red traffic lights (which are considered separately), the barriers, the vehicle sensor, and the delay or loss of messages sent through the radio network. Given these failures, the following objects that interact with the LCC system (Fig.5.9) are examined : the lights, the barriers, the vehicle sensors, the train-borne control system, and the operations center.

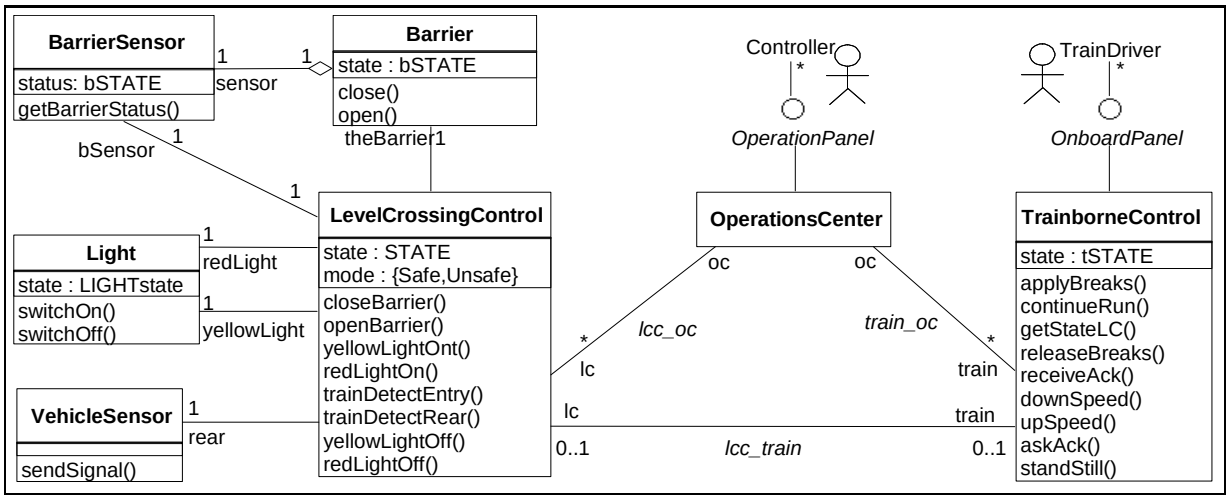


FIG. 5.9 – RLC system : Class diagram

The traffic lights and barriers at the level crossing are controlled by the LCC system. This system must be activated when a train approaches the level crossing. In the activated mode, the LCC performs a sequence of timed actions in order to safely close the crossing and to ensure that the danger zone is free of road traffic. First, the yellow traffic lights are switched on, and after 3 seconds, they are switched to red. After another 9 seconds, the barriers begin to lower. If the barriers have been completely lowered within a maximum time of 6 seconds, the LCC system signals the safe state of the level crossing, thus allowing the train to pass through the intersection.

The level crossing is opened to road traffic again once the train has passed completely through the crossing area, and the LLC system switches back into the deactivated mode.

Trains approaching the level crossing are detected via a process of continuous self-localisation on the part of the train and radio-based communication between the train and the LCC system. The vehicle sensor situated on the far side of the crossing triggers the re-opening of the barriers and tells the the traffic lights to switch off. During the activated mode, the LCC system can be in one of the following substates (Fig.5.10) : yellow light showing ; barrier closing ; barrier closed ; or barrier opening. Note that the time expirations occurring after the LCC is activated are denoted by the events `timeOut_1` (3 seconds later) and `timeOut_2` (9 seconds later).

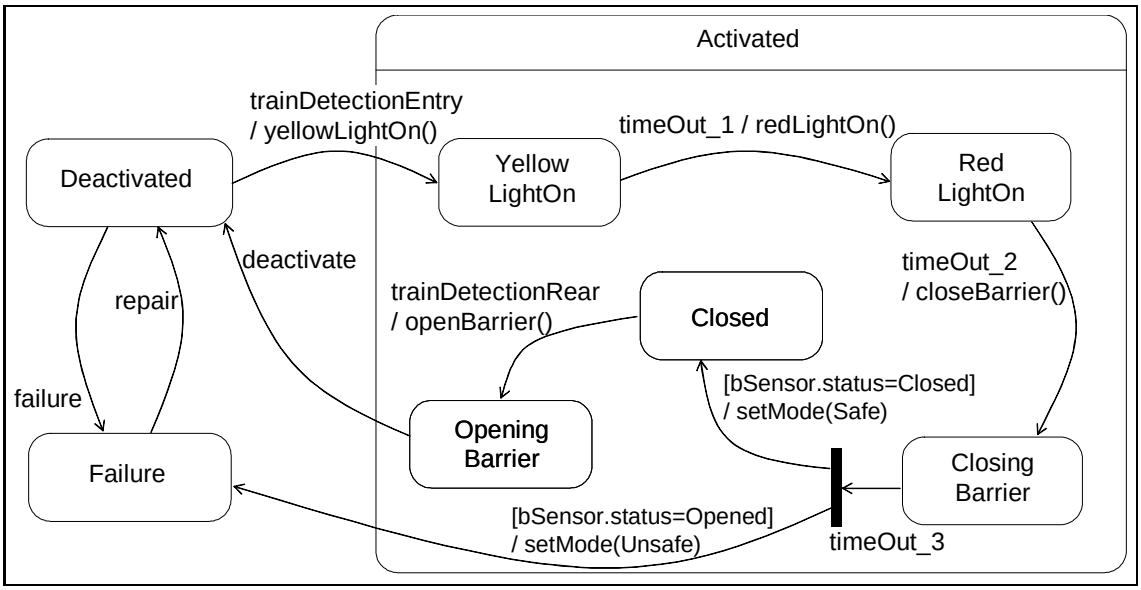


FIG. 5.10 – State diagram of the LCC system

5.4 Adding OCL constraints

Using a standard formal language for constraint specification is an important step towards formalising complex models, particularly in the context of critical safety systems. The purpose of OCL is to allow the constraints related to system objects to be formally specified, preserving the comprehensibility and readability of the UML models. OCL facilitates the statement of the properties and the invariants of the objects, as well as that of the pre/post-conditions for the operations. OCL also provides a navigation mechanism that allows attributes, operations and associations to be referenced in the context of a class or an object (a class variable), and query operators that permit a set of elements to be selected and/or modified. Each OCL expression has a specific type and belongs to a specific context. The context of an OCL expression determines its scope. Only the visible elements in the context of the expression can be referenced by means of navigation expressions.

Safety properties are included in the system invariants in order to propagate them from the abstract specification phase to the implementation phase. The main property of the LCC system is to prevent both road and rail traffic from entering the danger zone at the same time; to do so, the control specifications for the crossing area and its barrier, as well as any trains that may pass through the level crossing at any time, must be modelled at a high level of abstraction. For these reasons, the following OCL invariants are specified for these classes, as shown in Fig.5.11 :

1. **Req.***If there is a train crossing the danger zone then the barrier is closed* context CrossingArea
   inv :
   not(self.train->isEmpty()) implies self.barrier.state=Closed

2. **Req.***If the barrier of the crossing area is open, then no train is approaching the danger zone*
   context Barrier inv :
   self.state=Opened implies self.guards.train->isEmpty()

3. **Req.***The barrier of the crossing area is closed when it is being crossed by a train* context
   PhysicalTrain inv :
   self.crosses.barrier=Closed

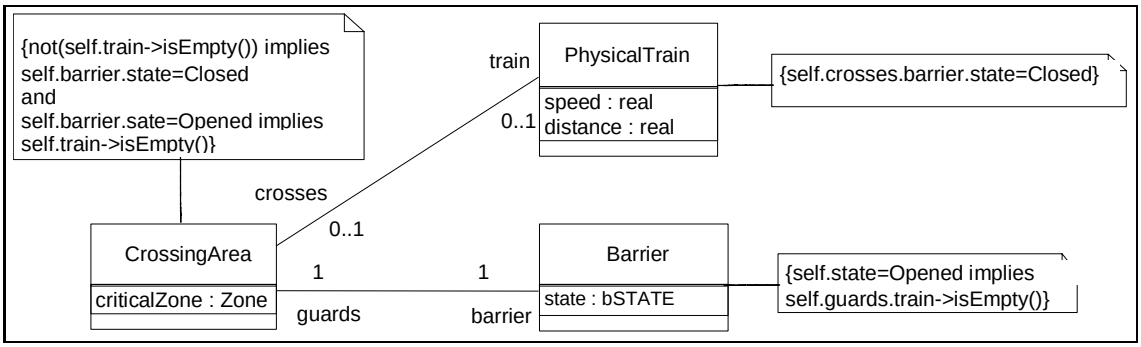


FIG. 5.11 – Constraints related to the danger zone

These constraints must hold true for a more detailed design once decisions have been made about the actual type of hardware to be used in an implementation. In this case study, the notion of "train passing through the crossing area" is connected to the activation of the railway level crossing. In order to accomplish this task, the front and the rear of the train must somehow be detected. We assume that the train can be detected directly through use of abstract vehicle sensors. The barrier state is detected by introducing a barrier sensor. In light of these assumptions, the previous OCL invariants can be refined by adding the following LCC system constraints for the class shown in Fig.5.9 :

1. **Req.***The red light is switched on whenever the barrier is closed, and the yellow light is switched on when the barrier is closing. If both the yellow and the red lights are switched off, then the barrier is open.*

   context LCC_System inv :
   self.theBarrier.state=Closed implies self.redLight.state=On
   and self.theBarrier.state=Closing implies self.yellowLight.state=On
   and self.yellowLight.state=Off and self.redLight.state=Off
   implies self.theBarrier.state=Opened

2. **Req.***If a train is in the danger zone, the level crossing is in an activated state composed of four substates (WaitingAck, Closing, Closed, Opening)*

   context LCC_System inv :
   not(self.train->isEmpty()) implies self.state=Activated and

   Set(Activated)=Set(WaitingAck->Union(Closing)->Union(Closed)->Union(Opening))

3. **Req.** *If the LCC system is in the activated state while the barrier is open, then the level crossing is in an unsafe mode*

   context LCC_System inv :
   self.state=Activated and self.bSensor.state=Opened implies self.mode=Unsafe

4. **Req.***If the registered state of the barrier is closed and the trigger sensor indicates that it is open, then the level crossing is in an unsafe mode. This is the case when the barrier is in the closing state (the LCC remains unsafe until the barrier is completely closed)*

   context LCC_System inv :
   self.bSensor.state=Opened and self.theBarrier.state=Closed) implies self.mode=Unsafe

The operations of the LCC class are specified with OCL pre- and post-conditions. OCL is also used in sequence diagrams to complete the preconditions and invariants related to operations. Although state diagrams are used to derive the initial specification of each operation (i.e., the

description of a state transition), OCL constraints are needed to add supplementary information that can not be retrieved from the state diagrams.

Consider the closing of the barrier raised by the event `timeOut_1`. The precondition of the operation `closeBarrier` ensures that the yellow light is switched on before sending the closeBarrier order, in addition to ensuring that the barrier has not yet been closed. The postcondition ensures that the state of the yellow light is off, the state of the red light is on, and the state of the barrier is closed. The operation is specified as follows :

context LCC_System : :closeBarrier
pre : self.yellowLight.state=On and self.theBarrier.state=Opened
post : self.yellowLight.state=Off and self.redLight.state=On and
self.theBarrier.state=Closed

5.5 Formalisation of classes and state diagrams

Our main goal is to extract an initial Bspecification (called the "abstract" specification) from the UML diagrams and to use it to check for inconsistencies. To do so, an abstract machine is associated to each class. Subsequently, the Bmethod is used to provide details about each component with regard to the behavior of class operations and the global invariants. At this point, the system developer must make several important decisions concerning unspecified properties and then introduce these properties into the UML diagrams using OCL constraints. The UML diagrams are then translated into B, and the resulting Bspecification is used to check the consistency of all the UML diagrams and OCL constraints.

Classes

Consider the class *Barrier* and its first Bspecification, presented in Fig. 5.5. Since a class includes both the static and dynamic properties of a set of objects, it seems natural to model it using one abstract machine. The resulting abstract machine Barrier describes the deferred set BARRIER of all the possible instances of the class *Barrier* . The set of existing instances is modelled using a variable barrier constrained to be a subset of BARRIER.

Each attribute (i.e., *bState*) is represented by a variable (i.e., bState) defined in the INVARIANT clause as a total function between the set barrier and its associated type (i.e., bSTATE). Each operation of the machine has at least one parameter obj representing the object on which the operation is called. It may have a list of typed arguments args, which will be completed in the later translation of the state diagrams and OCL constraints.

Formalisation of state diagrams

State diagrams are used to introduce the behavioral (i.e., dynamic) properties of the system into the Bspecification. The set of all possible states of a class is formalised using an abstract set that is defined in the corresponding Bmachine. An abstract variable is used to reference the current state of the class objects. This variable is defined as a total function, whose domain is the set of instances and whose range is the set of possible states. Each transition between two states is formalised by a Boperation, whose name is that of the incoming event. Whereas the precondition of the operation is deduced from the transition guard, the postcondition describes the transition to the new state. Please consider the state diagram of the `LCC_System` class shown in figure (Fig.5.10). The transition from the `showingYlight` state to the `closingB` state activated by the event `timeOut_1` is formalized as shown in Fig.5.13. Note that we have included some information obtained from the OCL definition of the operation closeBarrier, since this operation is activated by the event `timeOut_1`. (The OCL translation is described below.)

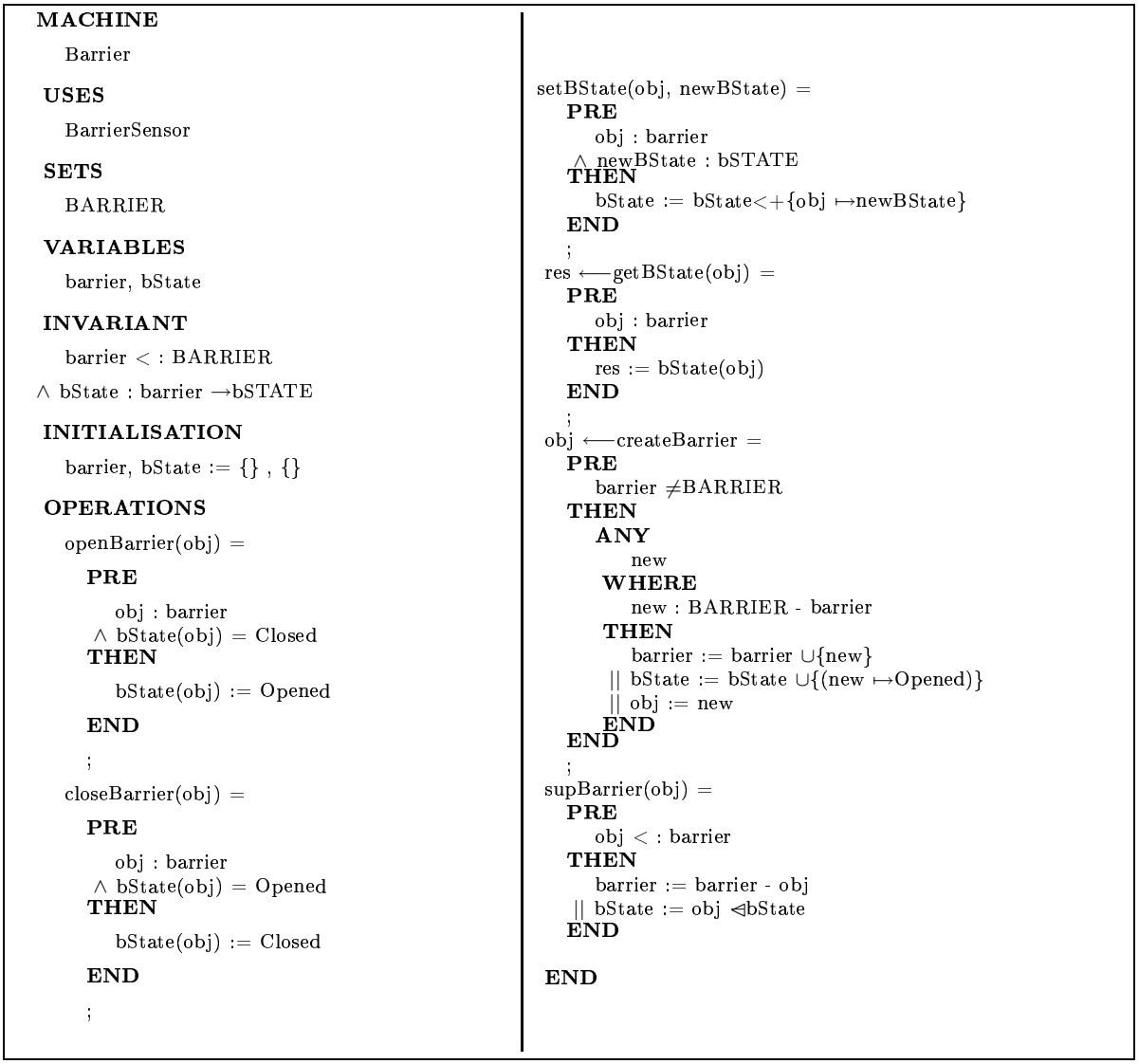


FIG. 5.12 – Formalisation of classes (machine Barrier)

When the same event can activate two different transitions depending on the guard condition, then both transitions are formalized by the same operation of the Bmachine. The non-deterministic construction SELECT is used to describe each transition, as illustrated in Fig.5.13 for the formalisation of the event `timeOut_2`.

```
MACHINE LCC_System ...
OPERATIONS ...
timeOut_1(obj) =
PRE
   obj ∈ lcc ∧
   state(obj)=YellowLightOn ∧
   bStatus(lcc_sensor(obj))=Opened ∧
   bState(lcc_barrier(obj))=Opened ∧
   Red.lState(redLight(obj))=Off ∧
   Yellow.lState(yellowLight(obj))=On
THEN
   state(obj) :=ClosingB ||
   closeBarrier(lcc_barrier(obj)) ||
   Yellow.switchOff(yellowLight(obj)) ||
   Red.switchOn(redLight(obj))
END;
```

```
timeOut_2(obj) =
PRE
   obj ∈ lcc ∧
   state(obj)=ClosingB ∧
   bState(lcc_barrier(obj))=Closed ∧
   Red.lState(redLight(obj))=On ∧
   Yellow.lState(yellowLight(obj))=Off
THEN
   SELECT bStatus(lcc_sensor(obj))=Closed THEN
      state(obj) :=ClosedB ||
      mode(obj) :=Safe
   WHEN bStatus(lcc_sensor(obj))=Opened THEN
      state(obj) :=Failure
   ELSE skip END
END;
```

FIG. 5.13 – Formalisation of state diagrams

Once the classes and state diagrams have been translated and integrated into the initial specification, OCL constraints are used to complete the Bmachine invariants and operations.

6 Formalisation of OCL constraints

In this section, we explain how OCL expressions are translated into Bexpressions. The Object Constraint Language has thus far been defined semi-formally using textual descriptions, a grammar that specifies the concrete syntax, and examples that illustrate the semantics of the expressions. Such a presentation style is adequate for illustrating OCL concepts, but it is not sufficient for providing precise semantics. This semi-formal nature of the OCL definition, which often leads users to interpret the UML models ambiguously, restricts its use in critical safety applications. This difficulty is increased by the lack of tools that support the analysis and proof of OCL expressions and the complete UML models.

Recent work proposing a precise semantic for OCL has been carried out by Richters and Gogolla [47]. In addition, both these authors [48] and Jackson *et al.* [26] have published research related to tools for verifying UML design. The first publication proposes an approach for validating UML models using simulation, and the second proposes an object model analyzer that uses Alloy, which is based on Z. Two of the authors of the present paper have also previously worked to formalize OCL with B, using a system of translation rules between the abstract syntaxes of both languages [33].

In *BRILLANT*, two types of OCL constraints are taken into account. The first type of constraint specifies an invariant of a class, and the second type specifies a precondition and/or a postcondition of an operation. In the first case, translating the OCL constraint consists of combining a new predicate with the invariant of the related Bmachine, whereas in the second case, it requires completing an operation of the machine. The formalisation of the LCC system's OCL invariant is shown in Fig.5.14.

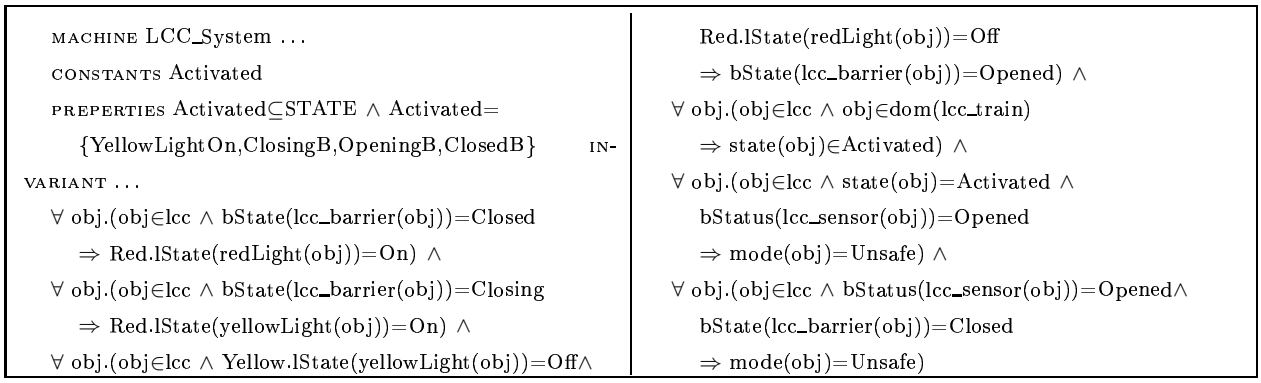


FIG. 5.14 – Formalisation of OCL invariants

OCL pre- and post-conditions are used to enrich Bmachine operations. In Fig.5.13, the pre-condition of the operation `timeOut_1` not only requires that the LCC system be in the `yellowLight` state (which is generated from the state diagram) but also that the red light be switched on and the barrier be closed. These three contraints together consitute the translation of the OCL predicate : self.yellowLight.state=On and self.theBarrier.state=Opened.
The post-condition of the operation initially includes only the substitution
`state(obj) := ClosingB`
that sets the new state of the LCC instance (obj). This post-condition is completed by translating the OCL post-condition
self.yellowLight.state=Off and self.redLight.state=On and self.theBarrier.state=Closed
into B, which generates the following parallel substitutions :
`closeBarrier (lcc_barrier(obj)) || Yellow.switchOff (yellowLight(obj)) || Red.switchOn (redLight(obj))`

7 Verification of the entire model

In order to automate the formalisation process, we implemented a prototype tool that derives Bspecifications from UML/OCL models. Once the whole Bformal specification has been generated from the UML/OCL model, it has to be type-checked and then verified through a proof process. A dedicated tool is then used to automatically generate and proof the proof obligations (POs). The POs guarantee that the Bmachine operations conform to its invariants. Each operation raises proof obligations related to its pre-condition and substitution parts. The non-proven POs are used to detect inconsistencies between the invariant and the preconditions, as well as to detect the incompleteness of a post-condition.

If a proof obligation cannot be proven using the theorem prover, then the developer must review the related OCL invariant or operation and make the necessary modifications to allow the obligation to be proved. Our approach is a one-way approach : it allows UML/OCL to be translated into B, but not the other way round. When the type-checker or the prover finds an error in the specification, the user must first understand the Bspecification and then search in the UML/OCL model to find the error. Please note that it is quite simple for the developer to find the UML element associated to a Bexpression because the names are roughly the same and each OCL expression is translated into a simple Bexpression. Thus, in order to facilitate the task, we have made it possible to create and maintain concrete links between the UML/OCL models and Bspecifications throughout the development process.

8 Discussion, conclusion and perspectives

8.1 Comparison with other works

Several other formal methods also employ the same kind of tools, and were developed similarly to use either open-source high-level languages or formats, or both. Many of them can be found in the www formal methods' virtual library [22], along with links to the formal methods they implement. In addition, freely available (but not open-source) tools exist for the Bmethod : B4Free [3] (distributed by *Clearsy*) and ProB [46]. The former is a parser, type-checker, proof obligation generator and prover programmed in the B0 language. The latter is an animator and model-checker programmed in Prolog, and uses the XMLfiles produced by the jBTools [28] as an input.

*BRILLANT*can also be compared to projects of a similar nature, that have similar ambitions and/or designs : Rodin [50] (for *B#*), Overture [37] (for VDM++) and the Community Z Tools [17] (*CZT*). These projects share several common points. For example, they all use XML-based interchange formats, and they all have similar architectures, in which the core tools (e.g., parsers, typecheckers, testers/validators, plugins and/or plugouts) are clearly separated. Some are driven by research needs (*BRILLANT*, *CZT*), some by industrial interests (Rodin), and some by both (Overture).

However, despite their similar architectures, the tools' underlying implementations are quite different. Rodin and Overture are based on the Eclipse IDE [20] and thus provide a very consistent development environment. Writing the mandatory parts of such a framework (e.g., parsing, compiling, graphic interfaces, test suites) is therefore more scalable and reusable, if not any easier. *CZT*and *BRILLANT*, on the other hand, are more or less a loosely connected set of tools : the first was developed using Java (their edition of Z specifications is even supported by jEdit), while most of the *BRILLANT*tools were developed in OCaml.

Because *CZT*and *BRILLANT*appear to have more points in common, it is appropriate to focus this comparison on them. *BRILLANT*was developed before the others, although the ideas behind *CZT*to appeared around the same time. The developers of *CZT*have provided information about the design decisions behind their tools. In the following paragraphs, these decisions are analyzed through a comparison with similar decisions that have been made at one time or another during the development of *BRILLANT*. (All citations come from [17].)

BCaml, the most important tool of *BRILLANT*, uses what in *CZT*terminology [17] is called an immutable type, which helps *BRILLANT*to avoid the problems described by the developers of *CZT*. In fact, they "initially implemented mutable", then tried "implementing [a] 'Both' [approach]" and in the end, gradually shifted "towards immutable." This means that, initially, the Abstract Syntax Tree (AST) (i.e., the type representing the Z specifications) could be modified during tool execution, which allowed more efficient algorithms to be used, but made the sharing of common subtrees (i.e. common pieces of specifications) difficult. In addition, given the "mutability" of the AST, the programmers had to ensure that it did not change during certain specific parts of the execution, which required a comprehensive knowledge of the overall tools. The 'Both' approach tried to combine the advantages of the mutable approach and the immutable approach, but only succeeded in increasing complexity. *CZT*has now moved towards the immutable approach, both because it is less error-prone and because it makes things simpler for the developer. Thus, by using BCaml, *BRILLANT*avoided the pitfalls of optimisation for the type of the AST. This made sense to us for 2 reasons : 1) the literature usually advocates using a static tree for representing abstract data structures, and 2) the BCamllanguage actually encourages such an implementation.

When developing *BRILLANT*, we wanted to make it possible to add B-HLL to BCamlat some later date, which required insuring identifier unicity. The *CZT*developers also faced this problem, which arises when an abstract structure defines binders, either as universal or existential quantifiers in predicate calculus, or as local vs. global variables in programming languages. In *CZT*, three common solutions to this problem were tried : renaming bound variables when necessary, which

is a traditional solution, but one that makes it "incredibly easy to make subtle errors") ; using De Bruijn indices, which is "easier to get right", but leads to unreadability and complexity ; or using unique names for all bound variables, which is safer, but requires unique names for all models of a Z project, and ends up making the output less readable. They finally chose the last solution for *CZT*, with the additional benefit that the unique attributes of the identifiers could be exported in the XMLAST so as to use the ID/IDREF (unique identifiers/symbolic link to unique identifiers) standard attributes. For *BRILLANT*, we chose a similar solution in BCaml : we decided to institute a scoping phase after the parsing phase in order to associate each identifier with a unique identity. Later on, the processing of new variables was facilitated not only by that unicity, but also by the recursive nature of the syntax trees, which allows all free variables to be retrieved. Indeed, several libraries that provide functions for high-level data structures (e.g., lists, sets, hash tables) exist in OCaml : and we could benefit from the use of these libraries immediately.

Another frequent problem is slow processing times. However, both teams managed to avoid this problem : the BCamltools in *BRILLANT*, like the tools in *CZT*, interact via the abstract syntax tree directly instead of XMLexchange files, which speeds up the processing.

All the above problems have been described in publications dealing with *Overture*[37], although the problem seems to have been more connected to their choice of compiler generator than to the compilation technique itself. Since the Rodin project is still in the development stage, we would encourage its developers to look at the projects discussed above, among others, in order to benefit from their experience.

From our prospective, *BRILLANT*is a more than adequate tool. Since most development projects are done by PhD students and young interns who are not yet full-time engineers, the simplicity of programming language chosen for *BRILLANT*(OCaml) is a definite plus. Its powerful, well-documented, high-level libraries, and its ability to blend different programming design paradigms, already well-known in the academic scientific community, make it a most appropriate choice. In addition, the by-the-book formalism implemented in *BRILLANT*make integrating extensions, such as the B-HLL module system,much easier than the other possibilities.

8.2 Conclusion

The *BRILLANT*platform design has two principal advantages : it uses open and standardised formats, and the source codes for its tools (OCaml and/or Java so far) are openly available. In addition, it can be used to test and/or validate B-related experiments, and in fact, we were the first users of many of the prototypes now available for the platform (e.g., bparser, bgop, btyper, bphox). We have been working to finetune the platform to help it meet the needs of other theoretical research projects, including but not limited to extending the Blanguage, improving the current tools, providing couplings with other provers (such as Coq, Harvey), and offering other validation formalisms (e.g. model-checking).

From the information presented in this article, it would appear that we have reached our goal of providing an open and standardised format (XML) for a platform that has become a testbed for several other fundamental research projects (UML/OCL/Bcoupling in section 5, proof in section 3, code generation in section 4). All the source code examples in the article (Bmachines, logical formulas, XML, PhoX) either come from *BRILLANT*, or derive from its use : The Bmachines are LaTeXfiles in the *BRILLANT*style, and the LaTeXfiles were obtained by applying an XSLstylesheet to XMLabstract machines, themselves obtained by parsing ASCII Bmachines.

8.3 Perspectives

In the future, we will work to integrate technologies that endorse the use of open formats into the *BRILLANT*platform. The following modifications are planned : the use of XMLschemas [52] instead of DTDs for validating XMLfiles ; the use of XMLflexibility to increase traceability between

UML models, Bmachines, proof obligations and other derived models (e.g., generated code, test cases) ; the representation of Bmodels as projects databases using XPath and XML-Query [16] ; and a distributed platform architecture using XML-RPC [54], which will allow the parser and prover to be represented as servers to which Bprojects can be sent for parsing or validation. We also plan to finalize the integration of ABTOOLS [10, 9] within BCaml. Now part of the *BRILLANT*, these tools were initially developped independantly. ABTOOLS provides an open environment based on ANTLR and JAVA and offers some facilities for designing and testing extensions of the Blanguage. Lastly, we hope to define an ergonomic interaction mode for the different platform tools, by proposing a graphic interface suitable for the underlying platform technologies. This interaction will, consequently, rely heavily on XML technologies.

Several other projects, these more related to the fundamental research currently under way, also offer interesting perspectives for the future, such as UML/OCL/B coupling [34], temporal extensions for B[14], and safe software component generation [42]. Much work remains to be done, and the platform developers are happy to provide their assistance to those who would like to try to use the tools in the context of their own research. All the necessary resources for building *BRILLANT* are available on a web site dedicated to collaborative free software development [12].

9 References

[1] Jean-Raymond Abrial. *The B Book - Assigning Programs to Meanings.* Cambridge University Press, August 1996.

[2] Jean-Raymond Abrial. $B^{\#}$: Toward a Synthesis between Z and B. In *ZB'2003 - Formal Specification and Development in Z and B*, pages 168–177, 2003.

[3] B4Free. .

[4] Patrick Behm, Paul Benoit, Alain Faivre, and Jean-Marc Meynadier. METEOR : A successful application of B in a large project. In *Proceedings of FM'99 : World Congress on Formal Methods*, pages 369–387, 1999.

[5] Salimeh Behnia. *Test de modèles formels en B : cadre théorique et critères de couvertures.* Thèse de doctorat, Institut National Polytechnique de Toulouse, October 2000.

[6] D. Bert, M.-L. Potet, and Y. Rouzaud. A study on components and assembly primitives in B. In *Proceedings of 1st Conference on the B method*, pages 47–62, 1996.

[7] J.-P. Bodeveix, Mamoun Filali, and C.A. Munoz. Formalisation de la méthode B en COQ et PVS. In *AFADL'2000*, pages 96–110, 2000.

[8] Jean-Paul Bodeveix and Mamoun Filali. Type synthesis in B and the translation of B to PVS. In *ZB'2002 – Formal Specification and Development in Z and B* [32], pages 350–369.

[9] Jean-Louis Boulanger. Abtools : Another B tool. In Ricardo J. Machado Johan Lilius, Felice Balarin, editor, *ACSD, Third International Conference on Application of Concurrency to System Design*, pages 231 – 232, Guimaraes, Portugal, 18-20 June 2003. IEEE. ABTools provides an open environnement based on ANTLR and JAVA and provides somes facilities for design and test an extension for the B language.

[10] Jean-Louis Boulanger. Abtools : A free tool for the B method. In *WMSCI 2005,*, Orlando, Florida, USA, July 10-13 2005.

[11] S.M. Brien and J.E. Nicholls. Z base standard : Version 1.0. Technical Monograph PRG-107, Oxford University Computing Laboratory, 11 Keble Road, Oxford OX1 3QD, UK, November 1992.

[12] BRILLANT.

[13] M. Carnot, C. DaSilva, B. Dehbonei, and F. Mejia. Error-free software development for critical systems using the B-methodology. *IEEE*, pages 274–281, 1992.

[14] Samuel Colin, Georges Mariano, and Vincent Poirriez. Duration calculus : A real-time semantic for B. In *First International Colloquium on Theoretical Aspects of Computing*. UNU-IIST, september 2004. Guiyang, China.

[15] Samuel Colin, Dorian Petit, Jérôme Rocheteau, Rafaël Marcano, Georges Mariano, and Vincent Poirriez. BRILLANT : An open source and XML-based platform for rigourous software development. In *SEFM (Software Engineering and Formal Methods)*, Koblenz, Germany, september 2005. AGKI (Artificial Intelligence Research Koblenz), IEEE Computer Society Press. selectivity : 40/120.

[16] World Wide Web Consortium. XQuery : the W3C query language for XML – W3C working draft. Available at http ://www.w3.org/TR/xquery/, 2001.

[17] Comprehensive Z Tools. .

[18] Babak Dehbonei and Fernando Mejia. Formal development of safety-critical software systems in railway signalling. In Hinchey and Bowen [25], pages 227–252.

[19] Theo Dimitrakos, Juan Bicarregui, Brian Matthews, Tom Maibaum, and Ken Robinson. Compositional structuring in the B method : A logical viewpoint of the static context. In ZB'2000 [55], pages 107–126.

[20] Eclipse. .

[21] S. Einer, H. Schrom, R. Slovák, and E. Schnieder. A Railway demonstrator model for experimental investigation of integrated specification techniques.

[22] The www formal methods' virtual library. .

[23] Betriebliches Lastenheft für FunkFahrBetrieb. Stand 1.10.1996, 1996.

[24] Robert Harper and Mark Lillibridge. A type-theoretic approach to higher-order modules with sharing. In *Conference record of POPL '94 : 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 123–137, Portland, OR, January 1994.

[25] M. G. Hinchey and J. P. Bowen, editors. *Applications of Formal Methods*. Series in Computer Science. Prentice Hall International, 1995.

[26] Daniel Jackson, Ian Schechter, and Ilya Shlyakhter. Alcoa : the Alloy costraint analyzer. In *International Conference on Software Engineering*, Limerick, Ireland, June 2000.

[27] L. Jansen and E. Schnieder. Traffic control system case study : Problem description and a note on domain-based software specification. technical report, 2000.

[28] jBTools. .

[29] Régine Laleau and Fiona Polack. Coming and going from UML to B : A proposal to support traceability in rigorous is development. In *ZB'2002 – Formal Specification and Development in Z and B* [32], pages 517–534.

[30] X. Leroy, D. Doligez, D. Garrigue, J.and Rémy, and J. Vouillon. The objective caml system. Technical report, INRIA, 2005. Software and documentation available on the Web, `http://caml.inria.fr/`.

[31] Xavier Leroy. A modular module system. *Journal of Functional Programming*, 10(3) :269–303, 2000.

[32] LSR-IMAG. *ZB'2002 – Formal Specification and Development in Z and B*, volume 2272 of *LNCS*, Grenoble, France, January 2002.

[33] Rafael. Marcano and Nicole Levy. Transformation rules of ocl constraints into B formal expressions. In *CSDUML'2002, Workshop on critical systems development with UML. 5th International Conference on the Unified Modeling Language*, Dresden, Germany, September 2002.

[34] Rafael. Marcano and Nicole Levy. Using B formal specifications for analysis and verification of UML/OCL models. In *Workshop on consistency problems in UML-based software development. 5th International Conference on the Unified Modeling Language*, Dresden, Germany, September 2002.

[35] Rafael Marcano, Georges Mariano, and Philippe Bon. UML-based design and formal analysis of railway traffic control systems. In *5th Symposium Formal Methods for Automation and Safety in Railway and Automotive Systems*, pages 173–182, 2004.

[36] Georges Mariano. *Évaluation de logiciels critiques développés par la méthode B : une approche quantitative.* Thèse de doctorat, Universitée de Valenciennes et du Hainaut-Cambrésis, Dec 1997.

[37] Overture (VDM++). .

[38] P. G. Larsen and B. S. Hansen and H. Brunn N. Plat and H. Toetenel and D. J. Andrews and J. Dawes and G. Parkin and others. Information technology — Programming languages, their environments and system software interfaces — Vienna Development Method — Specification Language — Part 1 : Base language, December 1996.

[39] Dorian Petit. *Génération automatique de composants logiciels sûr à partir de spécifications formelles B.* PhD thesis, Université de Valenciennes et du Hainaut Cambrésis, Décembre 2003. Numéro d'ordre 03-34.

[40] Dorian Petit, Georges Mariano, and Vincent Poirriez. Flattening B Components for Code Generation. Technical Report INRETS/RT-01-716-FR, INRETS, July 2001. http ://www.univ-valenciennes.fr/LAMIH/ROI/dpetit/Biblio/ Pub/RR/Flattening/RT-01-716-FR.ps.gz.

[41] Dorian Petit, Georges Mariano, Vincent Poirriez, and Jean-Louis Boulanger. Automatic Annotated Code Generation from B Formal Specifications. In G. Tarnai and E. Schnieder, editors, *Symposium on Formal Methods for Railway Operation and Control Systems*, pages 37–44. L'Harmattan, May 2003. ISBN 963 9457 45 0.

[42] Dorian Petit, Vincent Poirriez, and Georges Mariano. The B method and the component-based approach. *Journal of Design & Process Science : Transactions of the SDPS*, 8(1) :65–76, Mars 2004. ISSN 1092-0617.

[43] Dorian Petit, Vincent Poirriez, and Georges Mariano. Reuse of ML module system for the B language. In *Forum on specification and Design Languages*, September 2004.

[44] PhoX website. .

[45] Marie-Laure Potet and Yann Rouzaud. Composition and refinement in the B method. In *B'98 : The 2nd International B Conference*, pages 46–65, 1998.

[46] ProB. .

[47] M. Richters and M. Gogolla. On formalizing the UML object constraint language OCL. In *Proceedings of the 17t International conference on Conceptual Modelling, ER'98*, volume 1507 of *LNCS*. Springer-Verlag, 1998.

[48] M. Richters and M. Gogolla. Validating UML models and OCL constraints. In *Proceedings UML 2000*, 2000.

[49] Jérôme Rocheteau, Samuel Colin, Georges Mariano, and Vincent Poirriez. Évaluation de l'extensibilité de PhoX : B/PhoX un assistant de preuves pour B. In *JFLA*, pages 139–153, January 2004.

[50] Rodin-B#. .

[51] Bruno Tatibouët, Antoine Requet, Jean-Christophe Voisinet, and Ahmed Hammad. Java card code generation from B specifications. In In J.S. Dong and Eds. J. Woodcock, editors, *ICFEM*, volume 2885, pages 306–318. Formal Methods and Software Engineering, Springer-Verlag, 2003.

[52] H. S. Thompson, D. Beech, M. Maloney, and Mendelsohn Mendelsohn. "XML Schema Part 1 : Structures". W3C Recommendation, May 2001. http ://www.w3.org/TR/xmlschema-1/.

[53] Geoffrey Norman Watson. A comparison of modularity in B and Cogito. In S. Reeves L. Groves, editor, *Formal Methods Pacific*, pages 263–286, 1997.

[54] XML-RPC. Internet remote procedure call. *http ://www.xmlrpc.com/spec*, 1999.

[55] *ZB'2000 – International Conference of B and Z Users*, volume 1878 of *LNCS*, Helsington, York, UK YO10 5DD, August 2000.