





PROJET DE FIN D'ÉTUDES 2019 ECOLE NATIONALE DE L'AVIATION CIVILE, TOULOUSE,  
FRANCE. VERSION OF TUESDAY 27<sup>TH</sup> AUGUST, 2019 AT 16:49

VERSION OF TUESDAY 27<sup>TH</sup> AUGUST, 2019 AT 16:49

<https://github.com/toulbar2/toulbar2/tree/kad>

STAGE DU 18 MARS 2019 AU 15 SEPTEMBRE 2019 SUPERVISÉ PAR LES DR. SIMON DE GIVRY ET DAVID ALLOUCHE, CHERCHEURS À L'INRA, UNITÉ DE MATHÉMATIQUES ET INFORMATIQUE APPLIQUÉES DE TOULOUSE (MIAT), EQUIPE STATISTICS AND ALGORITHMICS APPLIED TO BIOLOGY (SAB), 31326 CASTANET TOLOSAN CEDEX FRANCE



## Preface

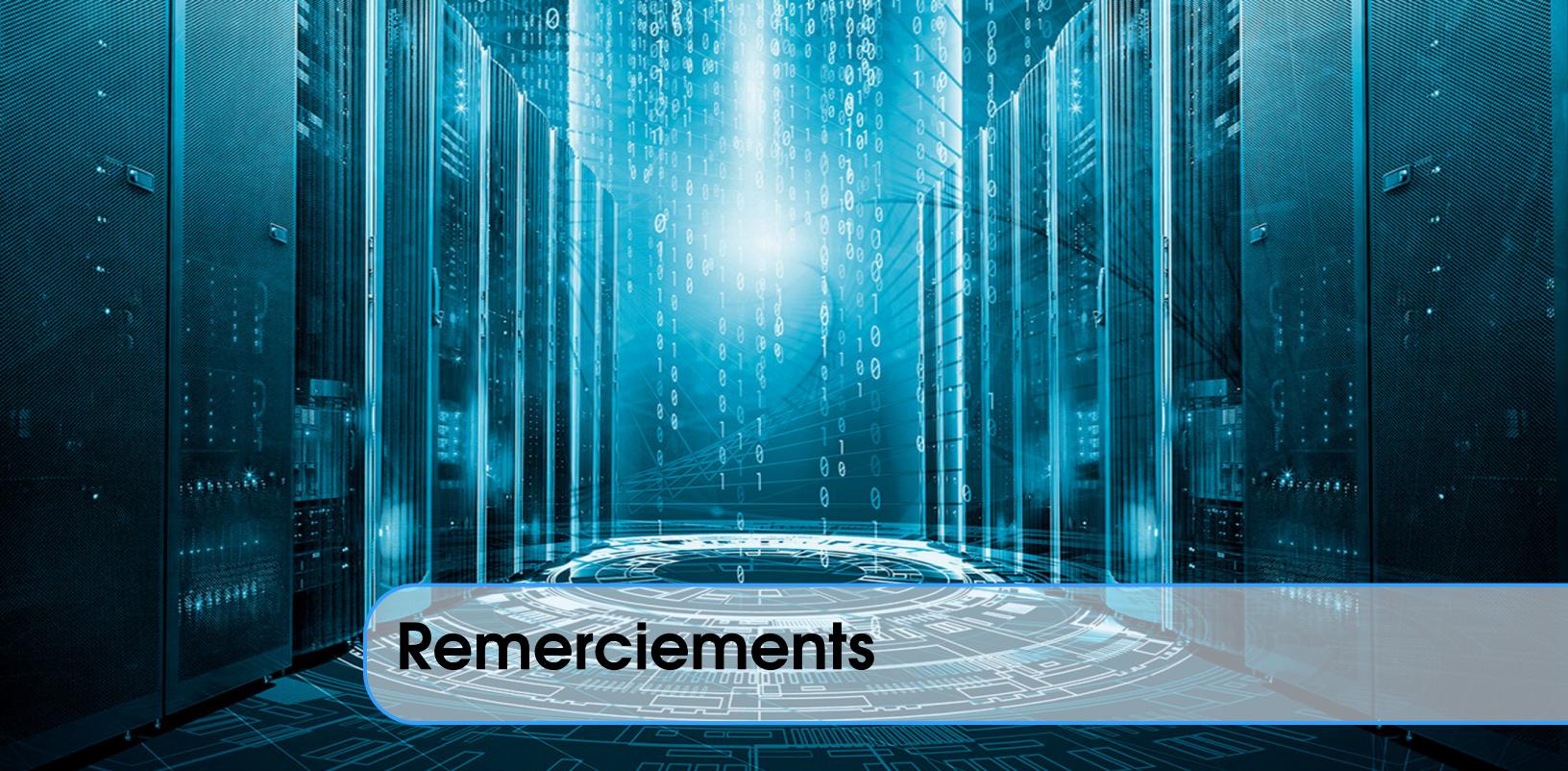
*"If you build it, they will come."*

And so we built them. Multiprocessor workstations, massively parallel supercomputers, a cluster in every department ... and they haven't come. Programmers haven't come to program these wonderful machines. Oh, a few programmers in love with the challenge have shown that most types of problems can be forcefit onto parallel computers, but general programmers, especially professional programmers who "have lives", ignore parallel computers.

And they do so at their own peril. Parallel computers are going mainstream. Multithreaded microprocessors, multicore CPUs, multiprocessor PCs, clusters, parallel game consoles... parallel computers are taking over the world of computing. The computer industry is ready to flood the market with hardware that will only run at full speed with parallel programs. But who will write these programs?

*Timothy Mattson , Beverly Sanders , Berna Massingill **Patterns for parallel programming**, Addison-Wesley Professional, 2004*





# Remerciements

A l'heure d'entamer la rédaction de ce rapport, ce vendredi 2 août 2019, je souhaiterais remercier le centre de recherche de l'INRA Toulouse et son **unité de Mathématiques et Informatique de Toulouse (MIAT)** pour m'avoir accueilli durant ce **Projet de Fin d'Etudes (PFE)** de 6 mois, entre le 18 mars et le 15 septembre 2019, et plus spécifiquement mes superviseurs les Dr. Simon de Givry et David Allouche pour avoir proposé ce sujet et l'aide précieuse apportée durant ce stage exigeant.

Je remercie également :

- Marie-Stéphane Trotard et Didier Laborie, pour le support sur le *cluster*<sup>1</sup> de la plateforme bio-informatique du réseau GenoToul<sup>2</sup>,
- Fabienne Ayrignac et Alain Perault pour le support administratif,
- Damien Berry et Mikaël Grialou pour le support informatique et réseaux
- et, last but not least, les développeurs de la librairie Boost<sup>3</sup> et plus particulièrement de Boost.MPI<sup>4</sup> qui m'auront évité d'utiliser l'interface C de Open MPI<sup>5</sup> dans un programme écrit en C++ ...

---

<sup>1</sup>Un **Cluster** est une grappe d'ordinateurs ou nœuds, relativement homogènes, connectés par un réseau local, propriété d'une entité déterminée et unique qui en assure la gestion. [https://fr.wikipedia.org/wiki/Grappe\\_de\\_serveurs](https://fr.wikipedia.org/wiki/Grappe_de_serveurs)

<sup>2</sup>GenoToul est la contraction de Génopole Toulouse qui est le réseau toulousain de plateformes de recherche en sciences du vivant qui mutualise les ressources : <https://www.genotoul.fr/vie-de-genotoul/>. Dans la suite, on utilisera les termes cluster bio-informatique ou simplement cluster pour désigner cet équipement

<sup>3</sup><https://www.boost.org/>

<sup>4</sup>[https://www.boost.org/doc/libs/1\\_70\\_0/doc/html/mpi.html](https://www.boost.org/doc/libs/1_70_0/doc/html/mpi.html)

<sup>5</sup><https://www.open-mpi.org/>





# Table des matières

<b>Résumés</b>	11
<b>Sujet</b>	19
<b>Introduction</b>	21
<b>1 Environnement de travail</b>	23
1.1 Présentation de la structure d'accueil	23
1.2 Rôle de l'équipe SAB	24
1.3 Ressources documentaires	25
1.3.1 Bases de données utilisées	25
1.3.2 Stratégie bibliographique associée	25
1.4 Ressources matérielles	25
1.5 Ressources logicielles	28
1.5.1 Systèmes d'exploitation	28
1.5.2 Gestion de version	28
1.5.3 Autres	29
1.5.4 Environnement de compilation/exécution	30
1.5.5 Environnement de développement	30

<b>2</b>	<b>Aspect gestion de projet .....</b>	<b>33</b>
2.1	Planning prévisionnel .....	33
2.1.1	Approche initiale pour traiter le sujet .....	34
2.2	Planning effectif du stage .....	34
<b>3</b>	<b>Présentation des wcsp .....</b>	<b>37</b>
3.1	Définitions .....	37
3.1.1	WCSP .....	37
3.1.2	Fonctions de coûts .....	38
3.1.3	Modèle initial .....	39
3.1.4	Fonction de coûts globale .....	39
3.1.5	Réseaux de fonctions de coûts .....	39
3.2	Formalisme des WCSP .....	41
3.2.1	Description succincte .....	41
3.2.2	Constance locale souple .....	41
3.2.3	Propagation de contraintes souples dans l'hypergraphe des fonctions de coûts .....	41
3.3	Exemple de modélisation .....	43
3.4	Exemple de projection .....	44
3.5	Exemple jouet : Le problème pondéré des 4 reines .....	46
<b>4</b>	<b>Toulbar2 .....</b>	<b>49</b>
4.1	Généralités .....	49
4.2	Caractéristiques de toulbar2 .....	50
4.3	Utilisation de toulbar2 .....	51
4.4	Elément synoptique du code de toulbar2 .....	53
4.4.1	Description succincte du code .....	53
4.4.2	Fonction général de toulbar2 .....	54
<b>5</b>	<b>Hybrid Best-First Search .....</b>	<b>55</b>
5.1	Definitions .....	55
5.1.1	Depth-First Search .....	55
5.1.2	Breadth-First Search .....	55
5.1.3	Best-First Search .....	56
5.1.4	nœud ouvert .....	56
5.1.5	Backtrack adaptatif .....	56

5.1.6	Hybrid Best-First Search .....	57
<b>6</b>	<b>Titre à préciser : Etat de l'art sur la parallélisation du Branch and Bound .....</b>	<b>61</b>
6.1	<b>Introduction</b>	<b>61</b>
6.2	<b>Algorithmes parallèles</b>	<b>62</b>
6.2.1	Définitions .....	62
6.2.2	Compromis .....	65
6.2.3	Load balancing .....	66
6.3	<b>Revue des approches choisies par certains solveurs</b>	<b>67</b>
6.4	<b>Comment mesurer les performances d'un solveur</b>	<b>67</b>
6.5	<b>conclusion</b>	<b>67</b>
<b>7</b>	<b>Embarrassingly Parallel Search .....</b>	<b>69</b>
7.1	<b>Définition</b>	<b>69</b>
7.2	<b>Benchmarks</b>	<b>69</b>
7.3	<b>Généralités sur la Embarrassingly Parallel Search (EPS)</b>	<b>69</b>
7.4	<b>Génération a priori des sous problèmes</b>	<b>70</b>
7.5	<b>Génération des sous problèmes via hbfs</b>	<b>71</b>
7.6	<b>Eléments descriptifs des résultats obtenus</b>	<b>74</b>
7.6.1	Performances .....	74
7.6.2	Nombre de nœuds totals traités vs nombre de nœuds ouverts .....	75
7.6.3	Optimisation de la génération .....	75
7.6.4	Distribution en nombre de variables non assignées .....	76
7.6.5	Distribution en complexité des sous problèmes .....	78
7.6.6	Temps individuel de résolution des sous problèmes .....	80
7.6.7	Performances globales de la parallélisation : speed-up .....	82
7.7	<b>Conclusion</b>	<b>84</b>
<b>8</b>	<b>Paradigme Master-Worker .....</b>	<b>87</b>
8.0.1	Master-workers .....	87
<b>9</b>	<b>Conclusion .....</b>	<b>89</b>
<b>10</b>	<b>Perspectives .....</b>	<b>91</b>
10.0.1	Paradigme Supervisor-workers .....	91

<b>11 Annexes .....</b>	<b>93</b>
<b>11.1 Annexe A : Génération a priori des sous problèmes</b>	<b>93</b>
<b>11.2 Annexe B : Utilisation de toulbar2 avec Eclipse</b>	<b>98</b>
<b>11.3 Annexe C : Compilation sur le cluster genotoul</b>	<b>102</b>
<b>Acronymes .....</b>	<b>115</b>
<b>Glossaire .....</b>	<b>117</b>
<b>Bibliography .....</b>	<b>119</b>
<b>Articles</b>	<b>119</b>
<b>Books</b>	<b>120</b>
<b>Index .....</b>	<b>123</b>



# Résumés

## Abstract

The algorithm [Hybrid Best-First Search \(HBFS\)](#) combines a recursive Depth First Search (DFS) Branch and Bound (B&B) with a Best-node First Search in order to find a complete assignment of the minimization optimization combinatorial problem variables. The problem can be seen as an hyper-graph, or a [Cost Function Network : Le problème est un hyper-graphe dont les nœuds sont les variables et les arêtes les contraintes \(CFN\)](#), where the nodes represent the variables and the edges the soft constraints, according to the [Weighted Constraint Satisfaction Problems \(WCSP\)](#) formalism.

The local soft constraints are combined to compute a global cost function. [Equivalent Preserving Transformations \(EPT\)](#) are then applied to the latter in order to get a *good* lower Bound in polynomial time which is used in the B&B in the same way, a relaxation gives a local lower bound in [Integer Linear Programming \(ILP\)](#), used to prune the decision tree. In HBFS, a global lower bound (LB) is computed together with the Upper Bound (UB) which is the current best value, or *incumbent* value associated with the best complete assignment or best solution found so far.

Part of the C++ solver [toulbar2](#), HBFS is still a sequential algorithm. As such, it does not get the best out of nowadays parallel architectures. To tackle this issue, we describe the phases that lead us to a parallelized release of HBFS.

First we present, a simplified overview of the underlying theory. Second, we examine two trails: the [Embarrassingly Parallel Search \(EPS\)](#), also known as natural parallelization, and the [Master-Worker](#) paradigm, popular among the [High Performance Computing \(HPC\)](#) community. The tests are performed on desktops, servers and/or the cluster's data center [GenoToul](#) of the French Agronomic Research Institute of Toulouse (INRA).

**Keywords:** tree search, toulbar2, combinatorial optimization, branch and bound, hypergraph, local soft consistency, depth first search, weighted constraint satisfaction prob-

lem, cost function networks, best node first search, incumbent value, incumbent solution, clusters, data center, high performance computing, parallelism, embarrassingly parallel search, Master-Worker paradigm.

## Résumé

L'algorithme hybride de recherche par meilleur nœud d'abord, ou [Hybrid Best-First Search \(HBFS\)](#), combine un algorithme par séparation et évaluation parcourant récursivement en profondeur l'arbre de recherche avec une sélection des nœuds ouverts, représentant un sous-problème non exploré, les plus prometteurs classés dans une file de priorité. L'objectif de cette recherche arborescente est de trouver une affectation complète des variables, représentées comme les nœuds d'un hypergraphe, ou réseau de fonctions de coûts, qui minimise la fonction de coût global du problème. Cette dernière combine des contraintes locales, dures et souples, selon le formalisme des problèmes de contraintes souples pondérées. Cette fonction de coût globale subit des transformations en temps polynomiale qui préservent le problème afin d'exhiber le meilleur, i.e. le plus grand, minorant possible (notion de cohérence locale souple). Cette borne inférieure locale est alors utilisée dans l'algorithme d'évaluation et séparation pour élaguer l'arbre de recherche et calculer un minorant global LB de manière à avoir, presque à tout instant, un encadrement  $[LB, UB]$  de la valeur optimale à déterminer de la fonction de coût globale où UB désigne la meilleure valeur courante associée à la meilleure affectation complète des variables dite solution *titulaire*.

L'implémentation séquentielle, déjà bien optimisée, de HBFS dans le solver C++ [toulbar2](#) n'exploite pas les architectures parallèles actuelles et ne bénéficie donc pas de ce gisement de gain en performances. Pour y remédier, autant que faire ce peut, ce rapport présente donc les phases qui ont conduit à proposer une parallélisation de HBFS.

On présente d'abord un état de l'art simplifié, puis l'exploration de deux pistes : la parallélisation naturelle, connue sous le terme pittoresque de [Embarrassingly Parallel Search \(EPS\)](#) et l'utilisation du paradigme [Master-Worker](#), populaire dans le domaine du calcul haute performance, accompagnées des tests et résultats sur PC, serveurs et/ou cluster en data center de l'INRA Toulouse.

**Mots-clés :** optimisation combinatoire, séparation et évaluation, parcours en profondeur d'abord, parcours au meilleur nœud d'abord, hypergraphe, réseau de fonctions de coûts, satisfaction de contraintes pondérées, cohérence locale souple, valeur titulaire, parallélisation embarrassante, paradigme Master-Worker, solution titulaire, toulbar2, recherche arborescente, parallélisme, calcul haute performance, clusters, data center.





## Table des figures

1.1	Vue arrière d'une des séries de baies du data center de l'INRA Toulouse. Consommation électrique annuelle : 100 000€. Chaque baie accueille des racks contenant plusieurs nœuds. Chaque nœud possède sa propre RAM et tourne sous sa propre instance de système d'exploitation en l'occurrence CentOS 7. Ils partagent un même système de fichier. . . . .	27
1.2	Vue de face d'une baie : chaque nœud du nouveau cluster SLURM genologin est constitué de 32 cœurs (64 threads) répartis sur deux puces de 16 cœurs montées sur une carte-mère demi-largeur équipée de 256GB de RAM. . . . .	28
3.1	Hypergraphe du problème des 4 reines : 4wqueens.wcsp. Le problème est représenté par un simple graphe car l'arité maximale des contraintes est égale à 2. Nous avons donc d'une part, le graphe représentant le problème, et d'autre part, l'arbre de recherche, ou arbre de décision, développé progressivement pour explorer l'espace d'état ici de taille $4^4 = 256$ , valeur qui correspond au cardinal du produit cartésien des domaines des variables. Ici quatre variables : V1 à V4 de domaine identique $D = \{1, 2, 3, 4\}$ . V4 = 2 signifie qu'on place une reine en colonne 4 et ligne 2. Dans ce type de graphe, la micro-structure n'apparaît pas ; les domaines ne sont pas représentés dans chaque nœud donc les fonctions de coûts unaires n'apparaissent pas non plus. . . . .	40
3.2	Hypergraphe du problème 404.wcsp. Problème-type utilisé lors des développements de hybridSolvePara(). On note qu'il présente clairement une structure en clusters. . . . .	40
3.3	Hypergraphe du problème scen06.wcsp . . . . .	40

3.4	Exemple de graphe avec micro-structure d'un problème simple à deux variables de même domaine $\{a, b\}$ . Les contraintes de coût nul ne sont pas représentées. Ainsi la contrainte unaire $w_1$ sur $x_1$ est telle que $w_1(b) = 0$ , la contrainte binaire $w_{12}$ est telle que $w_{12}(a, a) = 1$ et $w_{12}(a, b) = 0$ . On indique par la cas qu'on chercher à éviter, si possible, le cas (a,a) si $k > 1$ , ou qu'on l'interdit si $k = 1$ .	41
3.5	Trois types de transformations de base. Initialement, figure de gauche, on a $w_\emptyset = C_\emptyset = 0$ . La variable $z$ peut prendre la valeur $a$ dont le coût est 3 ou la valeur $b$ dont le coût est 2. Ce la donne une représentation "tabulaire" On peut donc opérer une projection unaire en transférant un coût égale à 2 des fonctions de la fonction de coûts	42
3.6	Micro-structure d'un réseaux de fonctions de coûts.	44
3.7	Nouveau réseau obtenu par projection unaire à partir de la 3.6.c. La contrainte constante est désormais $w_\emptyset = 1$ . Ce qui constitue un minorant de l'optimum de la fonction de coût globale.	45
3.8	Le problème des 4 reines.	46
5.1	Arbre binaire de décision partiellement exploré par un DFS avec une limite de backtrack, $Z = 3$ . Les nœuds verts sont les feuilles de l'arbre donnant lieu à une affectation complète des variables. Les nœuds jaunes non cerclés sur les branches droites de l'arbre sont les nœuds ouverts i.e. le sous arbre associé n'a pas encore été exploré. Ces nœuds sont placés dans la file <i>open</i> par le DFS lorsqu'il atteint la limite $Z = 3$ . En rouge, ce sont les nœuds fermés. Les nombres indiquent l'ordre de parcours DFS de l'arbre.	56
5.2	Illustration de l'algorithme : un nœud est retiré de la file open, donné au DFS qui retourne des nœuds ouverts dans la file open, qui les classe par clb croissants. Un vecteur de décision, i.e. de points de choix, optimise l'espace mémoire en évitant les redondances. En effet, deux nœuds peuvent avoir en commun un même chemin dans l'arbre de décisions. Il est plus économique en mémoire d'utiliser deux index, first et last, qui pointent sur le chemin du nœud que de mémoriser dans chaque nœud le chemin complet.	59
7.1	a minimum for the elapse time is observed near 200 sub problems	72
7.2	Minimum clockwall time is reached for 220 to 250 subproblems	73
7.3	Maximum speedup observé pour des valeurs de 28 à 31 voire 33 sous problème par cœur	74
7.4	Valeur de l'optimum trouvé par les 208 SP. Seule une minorité trouve l'optimum 114.	76
7.5	Taille des sous problèmes en nombre de variables	76
7.6	Taille des sous problèmes en nombre de variables	77
7.7	Taille des sous problèmes en nombre de variables	78
7.8	Distibution du log10 des complexités	79
7.9	time versus complexity of sub problems	80

7.10 courbe des temps de résolution en fonction des numéros des 208 SP du problème 404 sans utilisation de la borne ub calculée lors de la génération des SP . . . . .	81
7.11 courbe des temps de résolution en fonction des numéros des 208 SP du problème 404 avec -ub=114 . . . . .	82
7.12 Benchmarks sur architecture 4-8 cœurs . . . . .	83
7.13 Benchmarks sur architecture 24 cœurs . . . . .	84





## Sujet

### Contexte

Le cadre générique des réseaux de fonctions de coûts (Cooper, 2010) permet de résoudre des problèmes d'optimisation combinatoire variés. Il s'appuie sur des travaux menés en Intelligence Artificielle dans la communauté de la programmation par contraintes. Depuis une dizaine d'année, la vitesse des processeurs n'augmentant plus, plusieurs méthodes exactes d'optimisation ont exploité les nouvelles architectures multi-coeurs permettant une parallélisation à mémoire partagée et offrant des gains de performance importants. C'est notamment le cas des outils de la programmation linéaire en nombres entiers tels que IBM ILOG cplex et FICO Xpress. L'équipe SaAB mène des travaux en optimisation combinatoire dans les sciences du vivant et développe un outil C++ d'optimisation qui a remporté plusieurs compétitions sur les modèles graphiques probabilistes (toulbar2 <http://www.inra.fr/mia/T/toulbar2>).

### Sujet

L'objectif du stage est d'étudier la parallélisation d'une méthode de recherche arborescente hybride développée dans l'équipe [2] et récemment étendue dans [11]. Plusieurs stratégies de parallélisation seront considérées et analysées avant de faire des choix de parallélisation et d'implémentation ([14], [1], [20], [12], [17], [6]). Une comparaison à un travail similaire dans le cadre des modèles graphiques ([Kishimoto2015]) sera menée, ainsi que sur des problèmes issus de la recherche opérationnelle. Les benchmarks sont disponibles à <https://forgemia.inra.fr/thomas.schiex/cost-function-library>.





# Introduction

Le projet a consisté à paralléliser un algorithme séquentiel de recherche arborescente implémenté en C++ dans [toulbar2](#), un solveur de réseaux de fonctions de coûts ou [CFN](#) baptisé [HBFS](#).

Ce dernier combine une recherche dans un arbre binaire de décisions de type Best-node First Search ([BNFS](#)) avec un algorithme récursif qu'on peut nommer Adaptative Bounded Depth First Search Branch and Bound ([ABDFBB](#)).

La borne inférieure associée à chaque noeud utilisée dans le Branch and Bound est calculée en utilisant le formalisme [Weighted Constraint Satisfaction Problems \(WCSP\)](#) comme il sera présenté ultérieurement. La *philosophie globale* de toulbar2, notamment de l'algorithme [HBFS](#), contrairement aux algorithmes implémentant des méta-heuristiques : tabu search, recuit, génétiques, à population, etc., est de fournir, quand elle existe, une solution exacte et sa preuve d'optimalité.

Compte tenu des outils à disposition, la parallélisation demandée cible des plateformes matérielles équipées d'une soixantaine de cœurs. De ce fait, l'amélioration des performances, ou speedup, d'un programme parallélisé par rapport au programme séquentiel sous-jacent comparé au nombre de cœurs disponibles est le critère principal tandis que l'[échelonnabilité](#), qui vise à une augmentation linéaire des performances avec le nombre de cœurs restera relativement secondaire.

La finalité pratique de ce stage est de pouvoir résoudre des instances plus difficiles d'un même problème ou de résoudre des problèmes non résolus dans un temps raisonnable par la version séquentielle de toulbar2.

Ainsi, sur un seul noeud équipé de 64 cœurs du cluster [GenoToul](#), si une instance nécessite 3 mois pour être résolue par le programme séquentiel, avec un speedup de seulement 30, serait résolue en 3 jours par le programme parallèle.





# 1. Environnement de travail

## Résumé

Ce Projet de Fin d'études s'est déroulé dans l'[équipe Statistiques et Algorithmique pour la Biologie \(SAB\)](#) au sein de l'[unité de Mathématiques et Informatique de Toulouse \(MIAT\)](#), sur le site de l'[Institut National de Recherche Agronomique \(INRA\) Toulouse](#), située à Castanet-Tolosan.

### 1.1 Présentation de la structure d'accueil

Au même titre que le CNRS ou encore l'INRIA, l'[Institut National de Recherche Agronomique \(INRA\)](#) possède le statut d'[Etablissement Public à Caractère Scientifique et Technologique \(EPST\)](#) ce qui le range dans la catégorie des Etablissement Public à Caractère Administratif (EPA) lui permettant ainsi d'équilibrer son budget via des ressources non étatiques ; un EPA est, en effet, habilité à facturer des services à ses clients. C'est notamment le cas des prestations fournies par la plateforme bio-informatique, GenoToul.

En 2018, les ressources de l'INRA se montaient à 905 M€ dont 77% de subventions de l'Etat. De fait, le Ministère de tutelle de ce type d'établissement assure, via un contrat d'objectifs et de performance (COP), le suivi des orientations stratégiques de ces derniers en veillant à ce que leurs actions s'inscrivent dans les politiques publiques auxquelles ils participent. Généralement, un COP est établi par chaque établissement pour une période de 3 ans. Pour davantage d'information, on pourra télécharger ici le [rapport d'activité 2018](#).

L'INRA comprend 13 départements scientifiques parmi lesquels le [département de Mathématiques et Informatique \(MIA\)](#)<sup>1</sup>. Ce dernier se décline en plusieurs unités dont celle de Toulouse, l'[unité de Mathématiques et Informatique de Toulouse \(MIAT\)](#)<sup>2</sup>, anciennement

<sup>1</sup><http://www.mia.inra.fr/>

<sup>2</sup><https://mia.toulouse.inra.fr/Accueil>

dénommée Station de Biométrie et Intelligence Artificielle, située sur le site de Castanet-Tolosan/Auzeville-Tolosane.

Le centre Inra Occitanie-Toulouse<sup>3</sup>, présidé par Michèle Marin, est partie prenante du projet ANITI (Artificial and Natural Intelligence Toulouse Institute) sur l'intelligence artificielle hybride initié suite au rapport<sup>4</sup> Villani (2018).

L'**unité de Mathématiques et Informatique de Toulouse (MIAT)**, dirigée par Sylvain Jasson, est organisée en équipes parmi lesquelles, on trouve l'**équipe Statistiques et Algorithmique pour la Biologie (SAB)** dans laquelle s'est déroulé ce projet de fin d'études.

### Fusion INRA-IRSTEA en 2020

Au 1er janvier 2020, suite à sa fusion avec l'**Institut National de Recherche en Sciences et Technologies pour l'Environnement et l'Agriculture (IRSTEA)**, ex-CEMAGREF, l'**INRA** deviendra l'**Institut National de Recherche pour l'Agriculture, l'Alimentation et l'Environnement (INRAE)**. Le **département de Mathématiques et Informatique (MIA)** sera inclus dans une structure qui aura pour domaine la science des données, l'intelligence artificielle, les technologies robotiques et capteurs, la modélisation de systèmes complexes.

## 1.2 Rôle de l'équipe SAB

L'**équipe Statistiques et Algorithmique pour la Biologie (SAB)**, animée par Simon de Givry<sup>5</sup>, a pour objectif de développer et de mettre à disposition des biologistes des méthodes mathématiques, statistiques et informatiques permettant de contribuer à la compréhension du vivant. Elle s'intéresse à la localisation et à l'identification d'éléments fonctionnels dans les génomes des bactéries, plantes et animaux, et aux interactions qui existent entre ces différents éléments. Pour traiter ces problèmes, l'équipe mobilise et développe des méthodes en mathématiques, statistiques, probabilités (modélisation, inférence, modèles de mélanges de lois, régression pénalisée, modèles graphiques stochastiques, processus) et en informatique (modélisation, optimisation combinatoire, réseaux de contraintes, modèles graphiques déterministes, algorithmique) en vue de valoriser les méthodes développées dans des outils logiciels directement utilisables par les partenaires biologistes et rendant compte le mieux possible de la complexité et de la variété des données utilisables tout en capitalisant les développements méthodologiques dans des logiciels génériques, éventuellement déclinés ensuite sur différentes applications.

L'équipe développe en particulier des méthodes originales dans le domaine de l'optimisation combinatoire, en s'appuyant sur les réseaux de contraintes pondérées, aussi appelés "réseaux de fonctions de coût", un modèle graphique dédié à l'optimisation et généralisant les réseaux de contraintes utilisés en programmation par contraintes et proches des Champs de Markov. Ces techniques, implémentées dans l'outil toulbar2, développé dans l'équipe et très bien placé dans différentes compétitions internationales, sont ensuite mises en œuvre sur

---

<sup>3</sup><http://www.toulouse.inra.fr/Toutes-les-actualites/ANITI>

<sup>4</sup>[https://www.aiforhumanity.fr/pdfs/9782111457089\\_Rapport\\_Villani\\_accessible.pdf](https://www.aiforhumanity.fr/pdfs/9782111457089_Rapport_Villani_accessible.pdf)

<sup>5</sup><http://www7.inra.fr/mia/T/degivry/>

des problèmes issus de la bio-informatique (conception de protéines, localisation d'ARNs de familles connues, diagnostics de pedigrees complexes de grande taille...).

## 1.3 Ressources documentaires

### 1.3.1 Bases de données utilisées

1. bibliothèque ENAC
2. bibliothèque Université Paul Sabatier Toulouse III
3. [https://doc.lagout.org/science/0\\_Computer%20Science/5\\_Parallel%20and%20Distributed/](https://doc.lagout.org/science/0_Computer%20Science/5_Parallel%20and%20Distributed/)
4. <https://login.proxy.lib.enac.fr/login>. Ce site d'accès aux ressources de la bibliothèque de l'ENAC rassemble un ensemble de bases de données et permet de télécharger des articles parfois payant en dehors de cet abonnement académique d'où la nécessité d'un dépôt git documentaire privé sur [GitLab](#)
5. <https://link-springer-com.proxy.lib.enac.fr/>,
6. <https://dl-acm-org.proxy.lib.enac.fr/>
7. <https://www.google.fr/>
8. <https://scholar.google.fr/>
9. <https://www.lilo.org/fr/>

### 1.3.2 Stratégie bibliographique associée

La bibliographie relative aux [Weighted Constraint Satisfaction Problems \(WCSP\)](#) est conséquente. En outre, d'autres formalismes existent dont l'idée est de prendre en compte un réalité floue ou non déterministe. En la matière, l'exhaustivité aurait été contre productive vis à vis de l'objectif du stage, en l'occurrence améliorer les performances de l'algorithme séquentiel [Hybrid Best-First Search \(HBFS\)](#) de toulbar2 pour tenter de résoudre des problèmes non résolus ou des instances plus difficiles de problèmes connus. La bibliographie sur les WCSP et les techniques de parallélisation est basée sur un sous-ensemble d'articles sélectionnés par les encadrants du stage complétée par des recherches personnelles d'articles et de manuels sur internet ou en bibliothèque. Certains articles ont été présentés et discutés lors de réunions organisées régulièrement à l'INRA Toulouse.

## 1.4 Ressources matérielles

Sur le plan des ressources matérielles, un PC de bureau performant à 4 processeurs réels, multithreadés à 8, équipé de 32 Go de RAM a permis de tester localement toulbar2 tout en permettant un développement confortable sous l'environnement de développement Eclipse.

Par ailleurs, 4 serveurs équipés de 16 ou 24 cœurs, ont servi à diversifier les tests même s'ils étaient souvent monopolisés par d'autres utilisateurs.

Enfin, le data center "GenoToul", à finalité bioinformatique du site avec ses 3054 cœurs, 6218 en prenant en compte l'hyperthreading, connectés par réseau à haut débit et

faible latence [InfiniBand](#), pourra le cas échéant être mis à contribution sous réserve de disponibilité<sup>6</sup>.

La nouvelle tranche du cluster [genologin](#) comprend, entre autres, 48 nœuds<sup>7</sup>, 32 coeurs et 256 GB de RAM par nœud. Le Workload manager SGE a été remplacé par [Simple Linux Utility for Resource Management \(SLURM\)](#)<sup>8</sup>. Total : 1584 coeurs / 3168 threads / 51 TFlops de stockage. Les processeurs ont une microarchitecture broadwell<sup>9</sup>, gravure 14 nanomètres. SLURM est limité à quelques 20000 processus lancés en parallèle. Ainsi, pour exécuter la version parallèle de toulbar2, en mode non batch<sup>10</sup>, sur des cpu à micro-architecture broadwell, on pourra réserver 3 nœuds pour exécuter 96 processus avec la commande suivante :

```
srun -N 3 --ntasks-per-node=32 -n 96 --time=1 --exclusive  
--constraint=broadwell ./toulbar2 404.wcsp -para
```

nb : --time=1 indique qu'au bout de 1 minute le processus sera détruit. La résolution est la minute : si on indique des secondes, le système fera un arrondi à la minute supérieure. Cette fonctionnalité, outre le fait de permettre une utilisation parcimonieuse des ressources, permet de constituer des benchmarks pertinents sur la base d'un time-out prédéfini. Ainsi, le nombre de problèmes résolus, résolus partiellement avec une solution suboptimale et non résolus en moins d'une heure autorise la comparaison entre solvers.

Pour utiliser l'ancienne micro-architecture, on remplacera broadwell par ivy. Les performances sur l'architecture ivy, censée être moins performante, a montré au contraire un léger gain en performances.

En mode batch, on peut créer un script essai.sh

```
#!/bin/bash  
#SBATCH -J mpi_job  
#SBATCH --ntasks=64  
#SBATCH --cpus-per-task=2  
#SBATCH --mem-per-cpu=8G  
#SBATCH --exclusive  
#SBATCH --time=1  
  
mpirun -n $SLURM_NTASKS ./toulbar2 404.wcsp -para
```

puis utiliser la commande slurm : sbatch essai.sh pour soumettre le job au cluster.

Ces outils, PC, serveurs et data center, sont accessibles via une connexion VPN assurant l'ubiquité nécessaire notamment au télétravail des personnels de l'INRA ou aux tests à distance.

---

<sup>6</sup><http://bioinfo.genotoul.fr/>

<sup>7</sup>hostnames du node101 à node148

<sup>8</sup><https://slurm.schedmd.com/>

<sup>9</sup>[https://en.wikipedia.org/wiki/Broadwell\\_\(microarchitecture\)](https://en.wikipedia.org/wiki/Broadwell_(microarchitecture))

<sup>10</sup><https://slurm.schedmd.com/sbatch.html>



FIGURE 1.1 : Vue arrière d'une des séries de baies du data center de l'INRA Toulouse. Consommation électrique annuelle : 100 000€. Chaque baie accueille des racks contenant plusieurs nœuds. Chaque nœud possède sa propre RAM et tourne sous sa propre instance de système d'exploitation en l'occurrence CentOS 7. Ils partagent un même système de fichier.



FIGURE 1.2 : Vue de face d'une baie : chaque nœud du nouveau cluster SLURM genologin est constitué de 32 cœurs (64 threads) répartis sur deux puces de 16 cœurs montées sur une carte-mère demi-largeur équipée de 256GB de RAM.

## 1.5 Ressources logicielles

Toutes les applications utilisées durant ce stage sont des [Free Open Source Software \(FOSS\)](#).

### 1.5.1 Systèmes d'exploitation

- Ubuntu 18 LTS sur PC de bureau avec droit administrateurs (sudo) permettant la configuration du système sans passer par les administrateurs,
- Ubuntu 19 sur portable,
- Debian 10 sur les serveurs,
- CentOS 7 sur le cluster.

### 1.5.2 Gestion de version

- Dépôt sur GitHub avec droit d'écriture une branche personnelle : <https://github.com/toulbar2/toulbar2/tree/kad>

- Fork de toulbar2 sur GitHub. Utilisé comme bac à sable en début de stage : <https://github.com/kad15/SandBoxToulbar2>
- Dépôt sur GitLab privé pour gérer les documents relatifs au stage : articles, rapport, résultats, etc. GitHub ne permet pas la création sans abonnement de dépôts privés et placer des documents parfois sous copyright dans un dépôt public est à éviter : <https://gitlab.com>.

Conseil :

1. cloner les dépôts GitHub et GitLab en ssh i.e. adresse de type git@github.com :toulbar2/toulbar2.git et non en https et copier sa clé ssh publique sur GitHub (cf. annexe 11.2). Cela permettra d'éviter la saisie systématique d'un user et lot de passe.
2. Utiliser un script-maître et des script secondaire pour mettre à jour les dépôts en un clic :

```
script maître :
#!/bin/bash
echo "pfe:"
git pull origin kad 1>/dev/null 2>&1
git add . 1>/dev/null 2>&1
git commit -m "divers" 1>/dev/null 2>&1
git push origin kad 1>/dev/null 2>&1
git push origin kad | grep "Everything up-to-date"
echo "toulbar2 branch kad:"
cd ../toulbar2 && ./kad_maj_git.sh

script secondaire :
#!/bin/bash
git pull origin kad 1>/dev/null 2>&1
git add . 1>/dev/null 2>&1
git commit -m "master-worker" 1>/dev/null 2>&1
git push origin kad 1>/dev/null 2>&1
git push origin kad 1>/dev/null 2>&1
git push origin kad | grep "Everything up-to-date"
sleep 4s
```

### 1.5.3 Autres

- Outils graphiques : GraphViz<sup>11</sup>, gnuPlot<sup>12</sup>, R/RStudio<sup>13</sup>, LibreOffice Draw,
- Bureautique : TeXstudio <https://www.texstudio.org/>, Foxit : <https://www.foxitsoftware.com/>,
- Saros - Distributed Collaborative Editing and Pair Programming Eclipse plugin : <https://www.saros-project.org/>. Outils pour effectuer des développements col-

---

<sup>11</sup><https://graphviz.org/>

<sup>12</sup><http://www.gnuplot.info/>

<sup>13</sup><https://www.rstudio.com/>

laboratifs en temps réel. Plugin eclipse, Saros installé mais non utilisé. Voir aussi <https://vimeo.com/1195398>

#### 1.5.4 Environnement de compilation/exécution

##### Sur PC et serveurs

Les autotools<sup>14</sup> bien connus sous GNU/Linux, ne sont pas utilisés pour compiler le code. A la place, toulbar2 utilise le système de build cmake<sup>15</sup>, avec son configurateur ccmake. En fonction de la plateforme cible ou lors d'ajout de bibliothèques ou options, les fichiers cmake doivent être modifiés, en particulier pour prendre en compte les bibliothèques Boost.serialization<sup>16</sup> et Boost.MPI<sup>17</sup> ou accéder à des bibliothèques qui ne se trouvent pas dans les répertoires standards.

La compilation d'un programme parallèle est effectuée via un *wrapper* : mpic++. De même l'exécution se fait via l'outil mpirun qui attribue les processus aux processeurs (cœurs).

Les langages de script bash, awk, les expressions régulières, voire Perl, sont utiles pour l'automatisation des tests.

##### Sur Cluster

Outre cmake, sur le cluster, la compilation et l'exécution, nécessite de charger, voire d'écrire des modules écrits en Tcl<sup>18</sup> qui fixent les variables d'environnement nécessaires à l'exécution/compilation. C'est l'outil *module*<sup>19</sup> qui a été choisi.

Enfin une maîtrise minimale du gestionnaire [Simple Linux Utility for Resource Management \(SLURM\)](#)<sup>20</sup><sup>21</sup> est nécessaire pour soumettre les jobs sur les nœuds du cluster.

#### 1.5.5 Environnement de développement

Le choix d'un Environnement de développement ou [Integrated Development Environment \(IDE\)](#) adapté est essentiel dans un projet informatique. Parmi, les outils possibles plusieurs candidats ont été testés :

- Code : :Blocks : rapide à prendre en main mais trop limité. A réservé pour les petits développements.
- Visual Studio Code pour Linux : <https://code.visualstudio.com/Download>
- CLion : <https://www.jetbrains.com/clion/>. Payant mais licence académique gratuite limitée dans le temps. Utilise cmake comme système de build.
- vim + grep récursif pour l'analyse de code.

---

<sup>14</sup>text[https://www.gnu.org/software/automake/manual/html\\_node/Autotools-Introduction.html](https://www.gnu.org/software/automake/manual/html_node/Autotools-Introduction.html)

<sup>15</sup><https://cmake.org/>

<sup>16</sup>Permet de mémoriser, archiver tout ou partie de l'état d'un objet i.e. ses attributs

<sup>17</sup>Surcouche à la bibliothèque Message Passing Interface avec interface C++

<sup>18</sup>Tool Command Language

<sup>19</sup><https://github.com/cea-hpc/modules/blob/master/doc/source/index.rst>

<sup>20</sup><https://slurm.schedmd.com/>

<sup>21</sup>[http://bioinfo.genotoul.fr/index.php/faq/job\\_submission\\_faq/](http://bioinfo.genotoul.fr/index.php/faq/job_submission_faq/)

- Eclipse for Scientific Computing pour les développement d'application parallèle en fortran, C et C++, comprend le plugin Parallel Tool Plateforme et un débogueur parallèle mais probablement ... bugués : <https://www.eclipse.org/downloads/packages/release/2019-06/r/eclipse-ide-scientific-computing>
- Débogueur parallèle utilisé, gdb via le script gdb\_run : ./gdb\_run 8 problem.wcsp

```
#!/bin/bash
# script to put e.g. in toolbar2/your_build_folder/bin/Linux to run toolbar2 in ...
echo "usage: $0 nproc file.wcsp [other options]"

mpirun -n \$1 xterm -e gdb -ex run --args ./toolbar2 -para \$*
```

Autres outils de débogages (source : Bull Atos Technology, internet) :

- Valgrind : <https://fr.wikipedia.org/wiki/Valgrind>
- TotalView : GUI-based dynamic source code defect analysis tool that allows you to control processes and thread execution and see program state and variable values while your application runs. Link with MemoryScape for advanced memory debugging. <https://www.roguewave.com/products-services/totalview>
- Allinea DDT (debugger) and MAP (profiler) for HPC applications <https://www.arm.com/products/development-tools/server-and-hpc/forge/ddt>
- HPCToolkit : an integrated suite of tools for measurement and analysis of program performance on computers
- TAU : profiling and tracing toolkit for performance analysis of parallel programs written in Fortran, C, C++, Java and Python
- Scalasca : software tool that supports the performance optimization of parallel programs by measuring and analyzing their runtime behavior

### **Environnement sélectionné**

Malgré les défauts d'Eclipse, c'est ce dernier qui a été choisi car il facilite grandement la lecture de code. Les plugins permettant l'utilisation de cmake avec Eclipse ne fonctionnent pas très bien. Le plugin Eclipse, cmake editor, peut être utile. La solution retenue a consisté à produire un fichier de configuration avec ccmake, puis de créer un Makefile avec cmake en ligne de commande, puis de créer un projet de type "Makefile project". On trouvera un mode opératoire en annexe 11.2.





## 2. Aspect gestion de projet

### Résumé

La première section de ce chapitre reprend les éléments du planning prévisionnel tel qu'il était envisagé fin avril 2019, la seconde décrit les étapes effectivement suivies lors de ce stage qui s'est déroulé du 18 mars au 15 septembre 2019. Ce projet de fin d'étude s'est déroulé en plusieurs itérations :

1. Bibliographie :  $\approx 1.5$  mois,
2. Exploration de la piste parallélisation naturelle :  $\approx 1$  mois,
3. Exploration de la piste Paradigme Master-Worker :  $\approx 2$  mois,
4. Autres : rédaction rapports et beamer, séminaire, mise en place environnement de développement, prise en main des outils, etc. :  $\approx 1.5$  mois.

### 2.1 Planning prévisionnel

Il est prévu de procéder par itérations successives. L'itération 1 visera à produire un code fonctionnel répondant au cahier des charges du projet concernant HBFS. Les itérations suivantes tenteront le cas échéant de trouver des solutions plus performantes pour le HBFS tirées des réflexions et de l'expérience acquise ou de l'analyse d'éventuels de goulets d'étranglement et de l'adapter au BTD-HBFS (Backtracking with Tree Decomposition-HBFS).

#### 1. mars-juin : itération 1

- Bibliographie ([14], [1], [20], [12], [17], [6])
  - wcsp
  - consistance d'arc souple
  - propagation de contraintes souples
  - parallélisation de recherches arborescentes
  - Architectures logiciel : MPI, OpenMPI, OpenMP, Pthread

- Architectures matériel du laboratoire : serveurs MIAT, cluster genotoul
- prise en main de l'environnement de travail et du code.
- Recherche et analyse de stratégies de parallélisation.
- codage/benchmarks/analyse des performances. Les benchmarks sont disponibles à <https://forgemia.inra.fr/thomas.schiex/cost-function-library>.

## 2. juillet-août : itérations suivantes et autres selon temps disponible

- Comparaison avec un travail similaire mené dans le cadre des modèles graphiques([12])
- Comparaison à des travaux issus de la recherche opérationnelle.
- Piste de recherche 1 : adaptation de la méthode à une variante proposée dans (Larrosa, 2016, [13]).
- Piste de recherche 2 : intégration dans une métaheuristique parallèle [17]
- rédaction rapport, beamer
- rédaction rapport (suite)
- Présentation beamer
- Fin codage/benchmarks. Les benchmarks sont disponibles à <https://forgemia.inra.fr/thomas.schiex/cost-function-library>.

### 2.1.1 Approche initiale pour traiter le sujet

Parmi les approches possibles, il sera exploré lors de la première itération l'approche Embarrassingly Parallel Search (EPS) décrite dans [20] puis le paradigme Master-Worker[19] lors de l'itération suivante.

## 2.2 Planning effectif du stage

### 1. Début de stage : 18 mars 2019 09 :00

#### 2. mars-avril : Etat de l'art

- Analyse et mise en place de l'environnement de développement
- Bibliographie : Formalisme Weighted Constraint Satisfaction Problems (WCSP)
- Bibliographie : Parallélisation
- Rapport premier mois

#### 3. mai : Familiarisation avec le code

- Analyse du code.
- Recherche et analyse de stratégies de parallélisation.
- Premier codage : production des sous-problèmes par toulbar2.

#### 4. juin : Itération 1

- Analyse de la piste Embarrassingly Parallel Search (EPS),
- Benchmarks sur PC et serveurs,
- synthèse écrite,
- Présentation mi-stage à l'ENAC,
- Présentation à l'INRA : Journée stagiaires.

#### 5. juillet : Itération 2

- Analyse du Paradigme Master-Worker,

- Mise au point algorithme Master-Worker,
- Analyse d'implémentation : choix d'utiliser Boost.MPI et intégration dans cmake,
- Analyse du code toulbar2 : choix d'expurger le code de la partie clusterisée **Backtracking with Tree Decomposition Hybrid Best-node First Search (BTD-HBFS)**,
- Codage du **Hybrid Best-First Search (HBFS)** parallèle,
- Premiers tests sur PC et serveurs, débogage.

#### 6. **août : Itération 3**

- Analyse environnement matériel : cluster GenoToul,
- Résolution des problèmes de compilation et d'exécution sur le cluster,
- Codages complémentaires : nombres de backtracks, partage de solutions,
- benchmarks,
- rédaction du présent rapport.

#### 7. **septembre : Soutenance**

- Rédaction présentation beamer
- **Soutenance le 10 septembre 2019 13 :30-15 :00**

#### 8. **Fin de stage : 15 septembre 2019**





### 3. Présentation des wcsp

#### Résumé

Ce chapitre présente un Etat de l'art élémentaire sur les **Weighted Constraint Satisfaction Problems (WCSP)**. Ce formalisme permet de calculer une borne inférieure  $w_\emptyset$  utilisée par la suite notamment pour initialiser l'algorithme **Hybrid Best-First Search (HBFS)**. Un problème combinatoire est représenté par un hyper-graphe dont les nœuds sont les variables du problème et les arêtes les contraintes. Résoudre un WCSP consiste à minimiser une fonction de coûts globale, combinaison de fonctions de coûts locales, les contraintes souples. Le formalisme WCSP généralise celui des CSP en permettant la combinaison de contraintes dures, comme souples. Cette fonction est transformée de manière à obtenir une borne inférieure notée  $w_\emptyset$ . Cette borne inférieure est utilisée dans un branch and bound qui permet d'élaguer l'arbre de recherche et de calculer une borne inférieure globale. Une borne supérieure est obtenue à chaque fois que l'on obtient une affectation de toutes les variables du problème c'est à dire lorsque la suite de décisions décrit un chemin qui se termine par une feuille dans l'arbre binaire de recherche.

Par construction, la valeur optimale se situe entre ces deux bornes dont l'une augmente et l'autre diminue durant l'exécution de l'algorithme. Si le problème possède au moins une solution, la recherche se termine lorsque les bornes inférieure et supérieure deviennent égales, ce qui fournit la preuve d'optimalité. L'affectation complète associée à la valeur optimale constitue la solution recherchée.

## 3.1 Définitions

### 3.1.1 WCSP

Les **Weighted Constraint Satisfaction Problems (WCSP)**, problème de satisfaction de contraintes pondérées, généralisent les problèmes de type *Constraint Satisfaction Problems* (CSP) en permettant d'exprimer des contraintes dures comme souples ; Les CSP étant

NP-difficiles en général, il en est de même pour les WCSP (démonstration par restriction). Les contraintes des CSP sont remplacées par des *fonctions de coûts*.

### 3.1.2 Fonctions de coûts

Les fonctions de coûts sont aussi connues sous les termes de *contraintes valuées* ou de *contraintes souples*. Une contrainte porte sur un ensemble de variables appelé support, ou scope, de la contrainte. Le cardinal du support constitue son *arité n* avec :

$$n = \text{card}(S) = |S|$$

Une Fonction de coûts de support S,  $w_S(t)$ , associe un coût<sup>1</sup> à toute combinaison<sup>2</sup> de valeurs affecté aux variables impliquées dans la contrainte. Elle peut être avoir une représentation tabulaire ou analytique.

#### Exemple de fonction de coûts unaires

Les contraintes valuées unaires portent sur une seule variable. Ces fonctions de coûts permettent de modéliser une préférence pour certaines des valeurs que peuvent prendre les variables du problème d'où le terme de contraintes souples. Par exemple, une fonction de coûts telle que

$$w_{\{x\}}(b) = 0, \quad \text{avec } x \in \{a, b\}$$

modélise le fait que la valeur b pour la variable x est autorisée.

A contrario, la valeur b est interdite si on décide que la contrainte souple

$$w_{\{x\}}(b) = k$$

où k est un entier positif maximal fixé plus ou moins arbitrairement en fonction du problème. k est noté  $+\infty$  dans le formalisme des **Valued Constraint Satisfaction Problem (VCSP)**.

Ainsi, les fonctions de coûts de valeurs 0 et k modélisent des contraintes dures : si une fonction de coûts est définie de manière tabulaire, et non analytique, on ne trouvera que ces valeurs 0 ou k.

On aura compris que les valeurs comprises entre 1 et k-1 permettront de graduer les préférences liées à la nature du problème à modéliser. Par exemple, pour k = 3, choisir  $w_x(a) = 1$  et  $w_x(b) = 2$  signifierait que l'on n'interdit pas la valeur b pour x mais qu'on lui préfère la valeur a.

#### Exemple de fonction de coûts binaires :

Une contrainte souple binaire est une fonction de coûts d'arité 2. Son support est de la forme  $S = \{x_i, x_j\}$ . La contrainte souple est une fonction qui reçoit un tuple en entrée et renvoie un coût :

$$w_S : D_i \times D_j \mapsto E = \{0, \dots, k\}$$

On note X, l'ensemble des variables du problème tel que  $S \subseteq X$ .

---

<sup>1</sup>coût, poids ou weight en anglais d'où le choix de la notation w.

<sup>2</sup>combinaison ou tuple de valeurs

Chaque variable dans le support  $S$  possède un domaine  $D_i$ . Le nombre de tuples possible correspond au cardinal du produit cartésien des domaines :

$$\prod_{i=1}^n |D_i|$$

Ainsi, si  $X = S$  et  $D_1 = D_2 = \{a, b\}$ , la taille de l'espace de recherche sera égale 4.

### 3.1.3 Structure de valuation pour les WCSP

Une structure de valuation  $V$  est un Quintuplet tel que :

1. Ensemble de coûts possibles :  $E = \{0, \dots, k\}$  avec  $k$  élément *absorbant*,
2. Opérateur d'agrégation  $+_k$  tel que  $\forall a, b \in E, a +_k b = \min(a + b, k)$ ,
3. relation d'ordre :  $<$ ,
4. Coût min :  $0 \rightarrow$  sert à imposer une affectation donnée aux variables impliquées dans la contrainte,
5. Coût max :  $k \rightarrow$  idem sauf qu'il sert à interdire un tuple de valeurs.

La structure de valuation exprime les règles qui permettent notamment de combiner les contraintes locales en une valuation ou fonction de coûts globale.

### 3.1.4 Fonction de coûts globale

Les fonctions de coûts locales, i.e. qui ne concernent qu'un sous-ensemble de variables, sont agrégées pour obtenir une *Valuation* globale du problème, c'est à dire, une fonction de coûts globale en les variables du problème. Il s'agit alors de trouver le, ou les tuples, de valeurs à associer à toutes les variables du problème, c'est à dire de trouver *une affectation complète*,  $x = (x_1, \dots, x_n) := (t_1, \dots, t_n) = t$  qui minimise la fonction de coûts globale ou valuation globale.

$$Val(t) = Val(t_1, \dots, t_n) = w_\emptyset + \sum_{i=1}^n w_i(t_i) + \sum_{w_{ij} \in C} w_{ij}(t_i, t_j)$$

Résoudre un WCSP revient donc à trouver une affectation de toutes les variables, un tuple complet, qui minimisent sa fonction de coûts globale.

### 3.1.5 Réseaux de fonctions de coûts

Un réseau de fonctions de coûts, ou Cost Function Network (**CFN**), est un quadruplet :

1.  $X$  : ensemble de variables
2.  $D$  : ensemble des domaines des variables de  $X$
3.  $W$  : ensemble de fonctions de coûts locales
4.  $V$  : structure de valuation

On peut le représenté sous la forme d'un hyper-graphe dont le nœud sont les variables et les contraintes les fonctions de coûts.

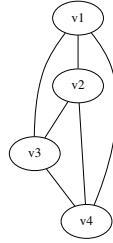


FIGURE 3.1 : Hypergraphe du problème des 4 reines : 4wqueens.wcsp. Le problème est représenté par un simple graphe car l’arité maximale des contraintes est égale à 2. Nous avons donc d’une part, le graphe représentant le problème, et d’autre part, l’arbre de recherche, ou arbre de décision, développé progressivement pour explorer l’espace d’état ici de taille  $4^4 = 256$ , valeur qui correspond au cardinal du produit cartésien des domaines des variables. Ici quatre variables : V1 à V4 de domaine identique  $D = \{1, 2, 3, 4\}$ .  $V4 = 2$  signifie qu’on place une reine en colonne 4 et ligne 2. Dans ce type de graphe, la micro-structure n’apparaît pas ; les domaines ne sont pas représentés dans chaque nœud donc les fonctions de coûts unaires n’apparaissent pas non plus.

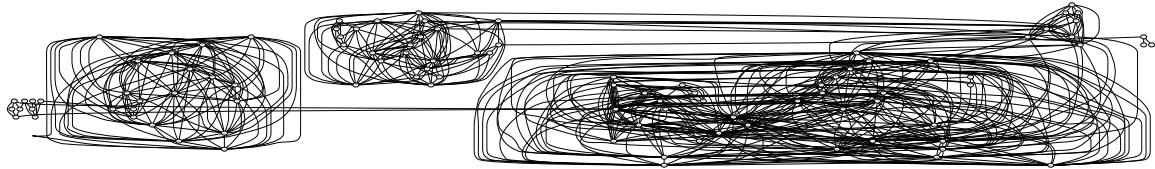


FIGURE 3.2 : Hypergraphe du problème 404.wcsp. Problème-type utilisé lors des développements de hybridSolvePara(). On note qu’il présente clairement une structure en clusters.

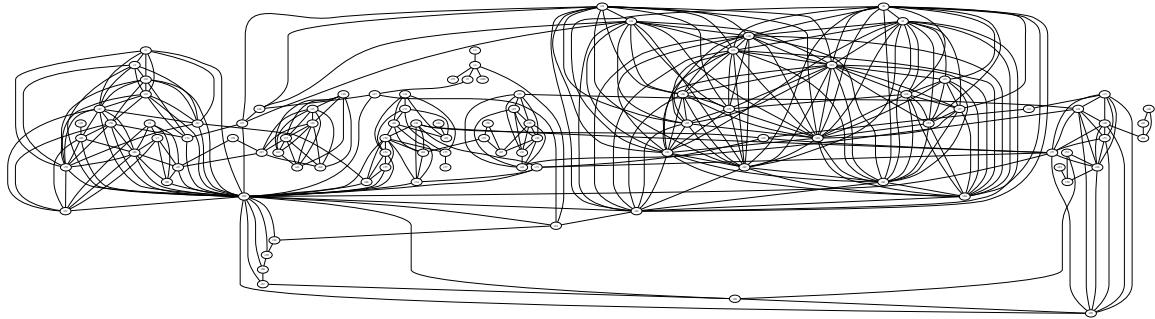


FIGURE 3.3 : Hypergraphe du problème scen06.wcsp

La micro-structure du réseau de fonctions de coûts fait apparaître, dans chaque nœud, les valeurs du domaine de la variable correspondante comme dans la figure 3.4.

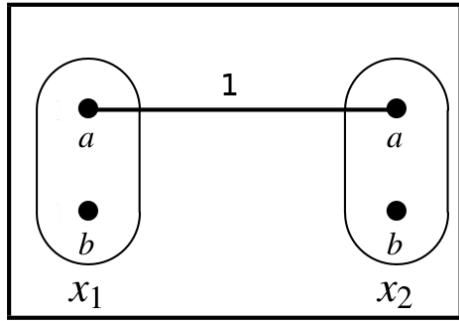


FIGURE 3.4 : Exemple de graphe avec micro-structure d'un problème simple à deux variables de même domaine  $\{a, b\}$ . Les contraintes de coût nul ne sont pas représentées. Ainsi la contrainte unaire  $w_1$  sur  $x_1$  est telle que  $w_1(b) = 0$ , la contrainte binaire  $w_{12}$  est telle que  $w_{12}(a, a) = 1$  et  $w_{12}(a, b) = 0$ . On indique par la cas qu'on chercher à éviter, si possible, le cas (a,a) si  $k > 1$ , ou qu'on l'interdit si  $k = 1$ .

Les définitions ci-dessus peuvent maintenant être mises à profit pour détailler un peu plus le formalisme de WCSP.

## 3.2 Formalisme des WCSP

### 3.2.1 Description succincte

Le problème de satisfaction de contraintes valuées (VCSP) est décrit dans [21],[4]. C'est un formalisme général qui permet, contrairement aux problèmes de satisfaction de contraintes *dures*, d'exprimer des préférences entre variables et donc entre solutions ou de traiter des problèmes sur-contraints pour lesquels certaines contraintes peuvent éventuellement être violées. Le problème de satisfaction de contraintes pondérées (*Weighted Constraint Satisfaction Problem* (WCSP)) constitue un cas particulier de VCSP. Les réseaux de contraintes valuées ou réseaux de fonctions de coûts (CFN : Cost Function Network) définissent une classe de modèles graphiques. En effet, l'ensemble X des variables, l'ensemble D de leurs domaines respectifs, l'ensemble F des fonctions de coûts et d'une structure de valuation V peuvent être représentés sous forme d'un hypergraphe plus ou moins détaillé selon qu'on veuille capturer la structure globale du problème ou sa micro-structure qui intègre l'information sur les domaines des variables et les fonctions de coûts [4]. Un exemple de graphe décrivant la micro-structure d'un problème est donné dans [22].

### 3.2.2 Consistance locale souple

Cf. : <https://www.sciencedirect.com/topics/computer-science/arc-consistency>

### 3.2.3 Propagation de contraintes souples dans l'hypergraphe des fonctions de coûts

Des algorithmes de propagation de contraintes souples ont été développés par l'équipe SAB, en particulier **Existential Directional Arc Consistency** (EDAC) et implémentés dans toulbar2

qui se basent sur des opérations d'extensions et de projections , dites **Equivalent Preserving Transformations (EPT)**, qui conservent la fonction de coûts globale, et qui permettent in fine d'obtenir une borne inférieure de cette fonction de coûts globale [8].

L'idée est de concentrer les coûts sur une seule variable ce qui peut déclencher une augmentation de  $w_\emptyset$  via une projection unaire.  $w_\emptyset$  est une fonction de coûts constante positive, i.e. d'arité nulle donc indépendante de toute variable.  $w_\emptyset$  est une borne inférieure de l'optimum car la fonction de coûts globale s'écrit comme la somme de  $w_\emptyset$  et d'une quantité positive  $Q$  :  $Val = w_\emptyset + Q$ . Il s'agit de trouver la bonne heuristique qui donnera la bonne séquence de projections et extensions pour maximiser  $w_\emptyset$  et donc se rapprocher de l'optimum. Parfois, la propagation suffit à trouver cette valeur optimale. Les valeurs  $w_\emptyset$  sont calculées pour chaque nœud de l'arbre de recherche et recalculées pour les nœuds prélevés dans la file de priorité open dont il sera question dans le chapitre **Hybrid Best-First Search (HBFS)**. Dans [22] est présenté la notion de propagation de contraintes souples à l'aide de 3 types de transformations : projection, extension et projection unaire comme en figure 3.5.

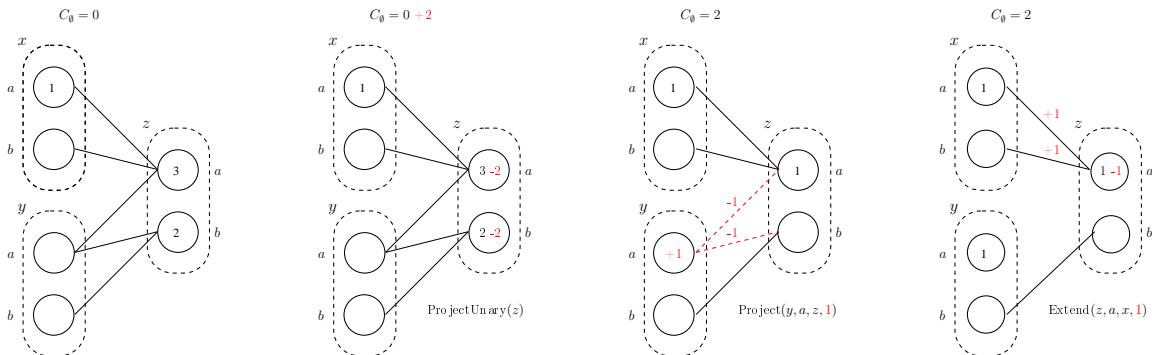


FIGURE 3.5 : Trois types de transformations de base. Initialement, figure de gauche, on a  $w_\emptyset = C_\emptyset = 0$ . La variable  $z$  peut prendre la valeur  $a$  dont le coût est 3 ou la valeur  $b$  dont le coût est 2. Ce la donne une représentation "tabulaire" On peut donc opérer une projection unaire en transférant un coût égale à 2 des fonctions de coûts

### Utilisation du minorant obtenu par propagation dans un branch and bound

Cette propagation par extension et projection dans l'hypergraphe fournit une borne inférieure  $w_\emptyset$  de l'optimum en chaque nœud de l'arbre de recherche du B&B [2].  $w_\emptyset$  est un minorant de l'optimum du sous-problème associé à un nœud.

Par ailleurs dès qu'on obtient une affectation complète des variables du problème, i.e on arrive à une feuille de l'arbre de recherche, on peut calculer une valeur  $V$  pour la fonction de coûts globale.

La borne supérieure  $UB$  du problème global qui constitue la meilleure valeur courante<sup>3</sup>, est mise à jour avec cette valeur  $V$  si  $UB \geq V$ . Dès que  $w_\emptyset \geq UB$ , on pourra élaguer l'arbre de recherche, ie supprimer le nœud en question qui ne peut donner de solution et tous ses

<sup>3</sup>valeur courante ou valeur titulaire, ou incumbent value, associée à la solution fournie sous la forme d'un tuple complet

descendants non plus car les  $w_\emptyset$  des descendants sont tous supérieurs ou égaux au  $w_\emptyset$  du nœud parent. Ainsi, l'algorithme HBFS, détaillé ultérieurement dans le chapitre éponyme, extrait le nœud le plus prometteur de la file de priorité et recalcule ce  $w_\emptyset$  et, si  $w_\emptyset \geq UB$ , il sera inutile de développer l'*arbre de décision* à partir de ce nœud. En ce sens, la recherche est partielle. On constate aussi que l'heuristique de calcul de  $w_\emptyset$  devra être la plus efficace possible.

Finalement, on peut résumer les étapes d'évaluation de la façon suivantes :

- Par projections, extensions et projections unaires, un minorant  $w_\emptyset$  de la valuation est calculé.
- Heuristique pour trouver le plus grand minorant en un temps raisonnable.
- Minorant qui sera utilisé pour élaguer l'arbre de recherche lors du Branch and Bound.
- HBFS initialisé avec  $w_\emptyset$
- En chaque nœud de l'arbre de recherche, une borne inférieure  $lb$  est calculée par propagation
- Si pour un nœud donné  $lb \geq UB$  où  $UB$  est la meilleure solution trouvée à ce stade de la recherche, il est inutile de développer l'arbre de recherche correspondant car il ne pourra améliorer la solution courante  $UB$ .

### 3.3 Exemple de modélisation

#### Problème

Soit le problème à modéliser dont l'ensemble de variables  $X = \{x_1, x_2, x_3\}$  de domaines identiques :

$$D_1 = D_2 = D_3 = \{a, b\} \quad a, b \in \mathbb{N}$$

,

La valeur  $a$  est préférée à la valeur  $b$  pour les 3 variables et nous avons la contrainte "dure" :  $x_1 \neq x_2$

#### Modélisation

La valeur  $a$  préférée à  $b$  peut être modélisée par 3 contraintes souples unaires identiques définies par :

$$w_i(a) = 0, \quad w_i(b) = 1, \quad i \in \{1, 2, 3\} \quad E = \{0, \dots, k\} \quad \text{avec} \quad k > 1$$

La contrainte dure binaire  $x_1 \neq x_2$  peut être modélisée par la fonction de coût binaire

$$w_{\{x_1, x_2\}}(t1, t2)$$

notée aussi

$$w_{12}(t1, t2)$$

On a  $\prod_{i=1}^n |D_i| = 2 \times 2 = 4$  coûts possibles pour chaque tuple. Ainsi  $w_{12}$  est représentable par la table 3.1.

$w_{12}$	a	b
a	k	0
b	0	k

TABLE 3.1 : Représentation tabulaire de la contrainte dure binaire  $w_{12}$

### 3.4 Exemple de projection

Cette section présente une exemple didactique qui développe celui de l'article [4] afin d'illustrer sur un cas concret les **Equivalent Preserving Transformations (EPT)** permettant de transférer les coûts vers une contrainte constante donc d'arité nulle  $w_\emptyset$  qui sera utilisée dans l'algorithme **Hybrid Best-First Search (HBFS)**.

Soient deux variables  $x_1, x_2$  avec les mêmes domaines :

$$D_1 = D_2 = \{a, b\}$$

et un ensemble de coûts :

$$E = \{0, \dots, k\}, \quad \text{avec} \quad k > 1$$

On décide qu'on préfère la valeur  $b$  à la valeur  $a$ , ce qui peut se traduire par :

$$w_1(a) = w_2(a) = 1$$

$$w_1(b) = w_2(b) = 0$$

et qu'on préfère éviter que nos deux variables prennent la même valeur  $b$ .

$$w_{12}(b, b) = 1$$

$$w_{12}(a, a) = w_{12}(a, b) = w_{12}(b, a) = 0$$

On représente la micro-structure de ce réseau de fonctions de coûts en figure 3.6.a avec la convention que les coûts unaires nuls et les arêtes de coût nul ne sont pas représentés.

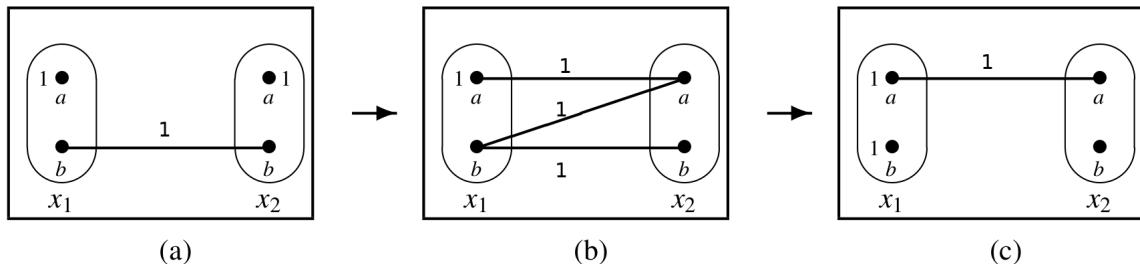


FIGURE 3.6 : Micro-structure d'un réseaux de fonctions de coûts.

La fonction de coût globale a pour expression :

$$Val(t_1, t_2) = w_\emptyset + \sum_{i=1}^n w_i(t_i) + \sum_{w_{ij} \in C} w_{ij}(t_i, t_j)$$

On calcule, par exemple, sa valeur pour le tuple  $t = (a, a)$  dans le cas de la figure 3.6.a pour illustrer le fait que notre réseau de fonctions de coûts évolue mais la fonction de coût globale est préservée. On pourrait faire le même calcul avec les 3 autres tuples :  $(a, b), (b, a), (b, b)$  pour vérifier la conservation de la valuation du problème par projection, extension et projection unaire.

$$Val(a, a) = w_\emptyset + w_1(a) + w_2(a) + w_{12}(a, a) = 0 + 1 + 1 + 0 = 2$$

De la figure 3.6 (a) à la figure 3.6 (b), on effectue une extension de la contrainte unaire  $w_2$  vers la contrainte binaire  $w_{12}$  d'où le terme d'extension. Autrement dit, on transfert le coût  $w_2(a) = 1$ , qui devient alors nul, vers les coûts binaires tels que  $w_{12}(a, a) = 1$  et  $w_{12}(b, a) = 1$  de manière à préserver la fonction de coûts globale. Ainsi, on peut vérifier sur la figure 3.6.b que la valeur associée au tuple  $(a, a)$  est conservée :

$$Val(a, a) = w_\emptyset + w_1(a) + w_2(a) + w_{12}(a, a) = 0 + 1 + 0 + 1 = 2$$

De manière similaire, de 3.6.b à 3.6.c, une projection d'une contrainte binaire vers une contrainte unaire telle que sur  $w_1(b)$  prend la valeur 1 tandis que  $w_{12}(b, a) = 1$  et  $w_{12}(b, b) = 1$  deviennent nulle conserve la valuation globale. La valeur pour la figure 3.6.c est préservée mais le réseau est modifié :

$$Val(a, a) = w_\emptyset + w_1(a) + w_2(a) + w_{12}(a, a) = 0 + 1 + 0 + 1 = 2$$

En figure 3.6.c, on constate qu'il est possible d'effectuer une projection unaire de la contrainte unaire  $w_1$  vers la contrainte d'arité nulle  $w_\emptyset$ . On obtient le nouveau réseau de la figure 3.7 en soustrayant le min de  $w_1(a) = 1$   $w_1(b) = 1$ , qui égal à 1, de chacunes de ces valeurs, ce qui donne  $w_1(a) = 0$   $w_1(b) = 0$  et en le transférant sur la contrainte constante  $w_\emptyset$ . Nous avons donc notre minorant  $w_\emptyset = 1$  utilisable par exemple dans un branch and bound.

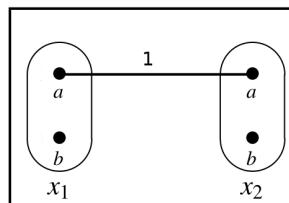


FIGURE 3.7 : Nouveau réseau obtenu par projection unaire à partir de la 3.6.c. La contrainte constante est désormais  $w_\emptyset = 1$ . Ce qui constitue un minorant de l'optimum de la fonction de coût globale.

Ce dernier réseau préserve également la valuation :

$$Val(t_1, t_2) = w_{\emptyset} + \sum_{i=1}^n w_i(t_i) + \sum_{w_{ij} \in C} w_{ij}(t_i, t_j)$$

$$Val(a, a) = w_{\emptyset} + w_1(a) + w_2(a) + w_{12}(a, a)$$

$$Val(a, a) == 1 + 0 + 0 + 1 = 2$$

### 3.5 Exemple jouet : Le problème pondéré des 4 reines

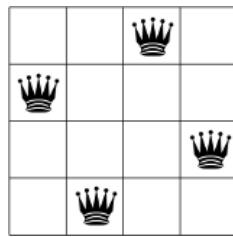


FIGURE 3.8 : Le problème des 4 reines.

Le problème des 4 reines, dont le graphe a été introduit figure 3.1, consiste à placer 4 reines sur un échiquier 4x4 telles qu'elles soient inoffensives les unes par rapport aux autres. On pourra consulter le site suivant : [https://acrogenesis.com/or-tools/documentation/user\\_manual/manual/introduction/4queens.html](https://acrogenesis.com/or-tools/documentation/user_manual/manual/introduction/4queens.html) La version pondérée consiste à exprimer des préférences parmi les solutions possibles comme décrit dans le fichier wcsp commenté en détail ci-après.

PROBLEME DES 4 REINES - VERSION PONDÉRÉE

nom du problème      nombre de variables      taille maxi du domaine      nombre de fonctions de coûts

Fichier : /home/toto/Bureau/4wqueens.wcsp

Page 1 sur 2

Arité de la fonction de coûts = contrainte binaire sur 2 variables.

4-WQUEENS-2-0 4 4 10      borne supérieure initiale UB du problème e.g = k  
 4 4 4 4      taille des domaines des 4 variables d'index implicite 0 à 3 : wcsp est un format positionnel  
 2 0 1 0 10      nombre de tuples avec un coût <> du coût par défaut = 10

0 0 5	Le coût par défaut est 0
0 1 5	
1 0 5	
1 1 5	
1 2 5	
2 1 5	
2 2 5	
2 3 5	
3 2 5	
3 3 5	
2 0 2 0 8	
0 0 5	
0 2 5	
1 1 5	
1 3 5	
2 0 5	
2 2 5	
3 1 5	
3 3 5	
2 0 3 0 6	
0 0 5	
0 3 5	
1 1 5	
2 2 5	
3 0 5	
3 3 5	
2 1 2 0 10	
0 0 5	
0 1 5	
1 0 5	
1 1 5	
1 2 5	
2 1 5	
2 2 5	
2 3 5	
3 2 5	
3 3 5	
2 1 3 0 8	
0 0 5	
0 2 5	
1 1 5	
1 3 5	
2 0 5	
2 2 5	
3 1 5	
3 3 5	
2 2 3 0 10	
0 0 5	
0 1 5	
1 0 5	
1 1 5	
1 2 5	
2 1 5	
2 2 5	
2 3 5	
3 2 5	
3 3 5	

1 0 0 2	1 : arité de la fonction de coûts. 0 : index de la 1ère variable donc la contrainte porte sur la variable X0 ; le support est le singleton {X0}. on la note w0. 0 : valeur du coût par défaut 2 : nombre de tuples, ici, singletons de coûts différents du coût par défaut.
1 1	
3 1	
1 1 0 2	
1 1	
2 1	

La contrainte binaire porte sur les variables X0 et X1  
 Le scope ou support de la contrainte est {X0,X1}

10 tuples de coût 5 : la contrainte concerne les deux variables X0 et X1, on la note w01, et leurs domaines sont de même taille 4. On a donc  $4 \times 4 = 16$  tuples possibles : 10 ont un coût = 5, et 6 un coût = 0. La fonction de coûts est donc parfaitement déterminée.

NB : la fonction de coût a pour valeurs soit 0, soit le coût max = k = 5. Cette fonction de coût exprime donc une contrainte DURE

Le problème des 4 reines ici modélisé utilise 4 variables X0, X1, X2, X3. Chaque index des variables correspond à une colonne dans le damier 4x4 sur lequel on doit placer 4 reines sans qu'elles puissent s'annexer mutuellement. Les valeurs des variables sont 0, 1, 2, 3. Elles correspondent aux lignes. Ainsi X0 = 1 et X1 = 0 signifie que la reine 1 est placée en case (0,0), la reine 2 en case (0,1). On interdit cette configuration en lui associant le coût max = 5 !

3 : index de la valeur prise par X0. Ici la taille du domaine est 4 ; les index des valeurs sont donc égaux aux valeurs.  
 1 : coût associé si X0 prend la valeur indexée par 3, i.e. la valeur 3 :  $w0(3)=1 \Leftrightarrow X0=3$  n'est pas interdite.  
 X0 peut prendre les valeurs : 0, 1, 2, 3. On a  $w0(0)=0$ ,  $w0(1)=1$ ,  $w0(2)=0$ ,  $w0(3)=1$  puisque la coût par défaut est 0. Ceci exprime que le modélisateur, pour des raisons qui lui sont propres, préfère que X0 prennent les valeurs 0 ou 2. Il préfère que la reine de la première colonne soit placée sur la première ou la troisième ligne mais ce n'est pas une obligation.  
 Ici, la fonction de coût exprime une CONTRAINTE SOUPLE !

```
1 2 0 2  
1 1  
2 1  
1 3 0 2  
0 1  
2 1
```

4 contraintes souples pour ce  
weighted 4 queens problem.

Résolution du problème avec toulbar2 :

```
..../toulbar2 4wqueens.wcsp -s
```

```
version : 1.0.1-440-gc96322c-kad (1566679382), copyright (c) 2006-2019, toulbar2 team
```

```
loading wcsp file: 4wqueens.wcspRead 4 variables,  
with 4 values at most, and 10 cost functions, with maximum arity 2.
```

```
Cost function decomposition time : 2e-06 seconds.
```

```
Preprocessing time: 0.000282 seconds.
```

```
4 unassigned variables, 16 values in all current domains (med. size:4, max size:4) and  
6 non-unary cost functions (med. degree:3)
```

```
Initial lower and upper bounds: [0, 5] 100.000%
```

```
SEQUENTIAL HBFS MODE!!! ADD -para OPTION FOR PARALLEL MODE
```

```
New solution: 0 (0 backtracks, 1 nodes, depth 2)  
2 0 3 1
```

```
Node redundancy during HBFS: 0.000 %
```

toulbar2 trouve une valeur optimale nulle (minimale) pour la fonction de coût globale.  
Cette valeur est associée à une affectation complète  $(X_0, X_1, X_2, X_3) = (2, 0, 3, 1)$   
ce qui correspond à la 1ère reine placée en 3ème ligne, la 2nd en 1ère ligne, etc.  
on vérifie que toutes les contraintes dures sont respectées. Les 4 contraintes souples  
le sont également mais cela aurait pu ne pas être le cas.

```
Optimum: 0 in 0 backtracks and 1 nodes ( 0 removals by DEE) and 0.000 seconds.
```

```
end.
```



## 4. Toulbar2

### Résumé

Ce chapitre présente succinctement toulbar2, un solveur de contraintes *souples* qui comprend plusieurs algorithmes dont l'[Hybrid Best-First Search \(HBFS\)](#). Cet exposé se place tant du point de vue de l'utilisateur que celui du programmeur appelé à contribuer aux développements et désireux d'acquérir une vue synoptique. On trouvera des informations plus précises en consultant sa [documentation](#) ou l'article [ToulBar2, an open source exact cost functionnetwork solver](#)

### 4.1 Généralités

toulbar2 est un solver de réseaux de fonctions de coûts dont la particularité est de pouvoir résoudre notamment des problèmes de satisfactions contraintes souples. D'autres informations générales sont intéressantes à connaître :

- Codé en C++,
- Développé à l'origine par les équipes INRA-MIAT à Toulouse et IIIA-CSIC de l'Université Polytechnique de Catalogne (UPC) à Barcelone,
- toulbar : Contraction de Toulouse et Barcelone,
- toulbar version 1 codée en C,
- version 2 codée en C++,
- toulbar2 a bénéficié d'autres contributions,
- Maintenu par [Simon de Givry](#),
- Libre(free), gratuit(free) et open source,
- Financé partiellement par l'agence Nationale de la Recherche (ANR).

## 4.2 Caractéristiques de toulbar2

Sur un plan technique, outre la documentation sur <https://github.com/toulbar2/toulbar2/tree/master/doc>, on pourra utilement consulter les quelques éléments ci-dessous :

- toulbar2 est compilable en tant qu'application ou en tant que bibliothèque<sup>1</sup>
- Après compilation l'exécutable se trouve dans le dossier toulbar2/release/bin/Linux si on suppose que l'utilisateur a nommé son dossier de build : "release", Mettre les problèmes à résoudre dans ce dossier et lancer la résolution avec ./toulbar2 problem.wcsp
- Le build system permet de produire un package \*.deb pour installation sur ubuntu : make package,
- Test de non regression : make test,
- Utilise le système de build cmake,
- Supporte les plateformes Linux, Mac et Windows,
- Comprend un ensemble d'algorithmes dont l'**Hybrid Best-First Search (HBFS)**,
- Implémente divers heuristiques pour la transformation des **Weighted Constraint Satisfaction Problems (WCSP)** qui conservent la fonction de coût globale tout en fournissant une borne inférieure correcte en un temps raisonnable : Virtual Arc Consistency for Weighted CSP[5], Existential Directional Arc Consistency(EDAC)[8] (consistance d'arc locale par défaut dans toulbar2)
- toulbar2 effectue des pré-traitements efficaces qui limitent la taille de l'intervalle dans lequel se trouve la valeur optimale ce qui accélère la phase de recherche de solution optimale, i.e. minimale.
- toulbar2 autorise de nombreuses options affichables par la commande ./toulbar2 sans options. quelques options utiles :
  - -v=3 : option qui augmente la verbosité des sorties de toulbar2. Associé à compilation en mode debug, elle permet de mieux superviser l'exécution du programme.
  - -s : affiche les solutions en plus des valeurs optimales trouvées,
  - -nopre : désactive les pré-traitements,
  - -A : utilise la consistance d'arc simple local VAC au lieu de EDAC,
  - si -option désigne une option activée par défaut -option suivie de ":" (-option :) désactive cette dernière. Exemple -solr est une option par défaut, -solr :; la désactive,
- L'ordre des options n'a pas d'importance : toulbar2 utilise SimpleOpt<sup>2</sup>

---

<sup>1</sup>Un exemple d'utilisation de la *librairie* toulbar2, libtb2.so, qui fournit une Application Programming Interface(API) se trouve dans toulbar2/src/toolbar2test.cpp

<sup>2</sup>SimpleOpt est utilisé pour lire les nombreuses options de toulbar2. Cf toulbar2/src/tb2main.cpp <https://github.com/brofield/simpleopt/>

## 4.3 Utilisation de toulbar2

L'utilisation de toulbar2, compilé en tant qu'application et non sous forme de bibliothèque, consiste à modéliser un problème et à le transcrire en un format supporté par le solveur puis à utiliser ce fichier : ./toulbar2 problem.wcsp -option-1 ... -option-n. Toulbar2 affiche alors en temps réel la progression de la recherche sous forme d'un gap d'optimalité qui désigne l'intervalle dans lequel se trouve la valeur optimale recherchée en l'occurrence [LB,UB], où LB est la borne inférieure globale et UB la meilleure solution courante à ce stade de la recherche.

La commande ci-dessous lance la version parallèle boot.MPI de HBFS, sur 4 cpus, pour résoudre le problème 404.wcsp.

L'option -s, \textit{ou --show}, visualise la solution optimale et pas seulement la valeur optimale.

Le gap d'optimalité indique la progression de la recherche.

```
mpirun -np 4 ./toulbar2 404.wcsp -s -para
```

Preprocessing time: 0.006 seconds.

88 unassigned variables, 226 values in all current domains  
(med. size:2, max size:4) and 594 non-unary cost functions  
(med. degree:13)

Initial lower and upper bounds: [66, 158] 58.228%

PARALLEL HBFS MODE!!!

New solution: 122 (0 backtracks, 17 nodes, depth 18)

```
0 0 2 1 0 1 1 3 3 1 1 0 1 1 1 1 3 1 1 0 1 1 1 1 0 1 2 0 1 1  
1 0 1 1 0 1 1 1 1 1 0 1 1 1 1 3 1 1 1 1 1 1 3 1 3 1 1 1 1  
1 1 1 3 1 0 1 0 1 1 0 1 1 0 0 3 3 0 2 1 1 3 1 1 3 1 0 3 1  
1 1 3 3 1
```

.....

New solution: 114 (170 backtracks, 368 nodes, depth 16)

```
0 0 2 1 1 1 0 3 3 1 1 1 1 1 0 1 3 1 1 1 0 1 1 0 1 1 3 1 0  
3 1 1 0 0 1 1 0 1 1 1 0 1 1 1 1 3 1 1 1 0 1 1 3 3 1 3 1 1  
1 0 1 1 1 1 1 0 1 0 1 1 0 1 1 0 0 1 2 1 0 3 1 3 1 1 3 1 0  
3 1 1 1 3 3 1
```

Optimality gap: [85, 115] 26.087 % (4624 backtracks, 9607 nodes)

Optimality gap: [85, 114] 25.439 % (6694 backtracks, 13777 nodes)

Optimality gap: [86, 114] 24.561 % (24270 backtracks, 49131 nodes)

Optimality gap: [87, 114] 23.684 % (27365 backtracks, 55355 nodes)

Optimality gap: [88, 114] 22.807 % (72805 backtracks, 146665 nodes)

Optimality gap: [89, 114] 21.930 % (123145 backtracks, 247592 nodes)

Optimality gap: [90, 114] 21.053 % (252452 backtracks, 506713 nodes)

Optimality gap: [91, 114] 20.175 % (360518 backtracks, 723280 nodes)

Optimality gap: [92, 114] 19.298 % (485927 backtracks, 974600 nodes)

```

Optimality gap: [93, 114] 18.421 % (587870 backtracks, 1178905 nodes)
Optimality gap: [94, 114] 17.544 % (696440 backtracks, 1396820 nodes)
Optimality gap: [95, 114] 16.667 % (784642 backtracks, 1574130 nodes)
Optimality gap: [96, 114] 15.789 % (903232 backtracks, 1812364 nodes)
Optimality gap: [97, 114] 14.912 % (958291 backtracks, 1923538 nodes)
Optimality gap: [98, 114] 14.035 % (1009840 backtracks, 2028010 nodes)
Optimality gap: [99, 114] 13.158 % (1050466 backtracks, 2110700 nodes)
Optimality gap: [100, 114] 12.281 % (1069395 backtracks, 2149633 nodes)
Optimality gap: [101, 114] 11.404 % (1081933 backtracks, 2175637 nodes)
Optimality gap: [102, 114] 10.526 % (1091363 backtracks, 2195620 nodes)
Optimality gap: [103, 114] 9.649 % (1096809 backtracks, 2207552 nodes)
Optimality gap: [104, 114] 8.772 % (1101985 backtracks, 2219221 nodes)
Optimality gap: [105, 114] 7.895 % (1103827 backtracks, 2223803 nodes)
Optimality gap: [106, 114] 7.018 % (1106727 backtracks, 2230612 nodes)
Optimality gap: [107, 114] 6.140 % (1108582 backtracks, 2235893 nodes)
Optimality gap: [108, 114] 5.263 % (1109520 backtracks, 2239138 nodes)
Optimality gap: [109, 114] 4.386 % (1110212 backtracks, 2241956 nodes)
Optimality gap: [110, 114] 3.509 % (1110684 backtracks, 2244500 nodes)
Optimality gap: [111, 114] 2.632 % (1111033 backtracks, 2246767 nodes)
Optimality gap: [112, 114] 1.754 % (1111180 backtracks, 2248350 nodes)
Optimality gap: [113, 114] 0.877 % (1111228 backtracks, 2249481 nodes)
Optimality gap: [114, 114] 0.000 % (1111245 backtracks, 2250258 nodes)
Optimum: 114 in 1111245 backtracks and 2250258 nodes
( 6 removals by DEE) and 8.915 seconds.

```

Dans le cadre de ce PFE, seuls le format *positionnel* \*.wcsp, et sa version compressée \*.wcsp.xz<sup>3</sup> pour les problèmes dont la taille se compte en Giga Octets, auront été utilisés.

On distinguera le problème **Weighted Constraint Satisfaction Problems (WCSP)** du format qui le représente. Une même modélisation d'un problème peut être représentée par différents formats. En effet, toulbar2 supporte une quinzaine de ces formats d'entrée dont un format de type json<sup>4</sup> \*.cfn<sup>5</sup>. Cf. documentation <https://github.com/toulbar2/toulbar2/blob/master/doc/CFNformat.pdf> et l'exemple ci-dessous :

```
{
problem: { name: "maximization_example", mustbe: ">-5.0" },
variables: { "v1": ["a", "b"], "v2": ["c", "d"] },
functions: {
"c0": {scope: [], costs: [-6.0]},
"c1": {scope: ["v1"], costs: [1.0, 0.5]},
```

---

<sup>3</sup>Compression gérée par la bibliothèque Boost

<sup>4</sup>Le format JavaScript Object Notation est utilisé, à l'origine, pour les communications entre navigateurs et serveurs. C'est un format texte qui enregistre les données sous forme de couples nom\_de\_la\_variable : valeur\_de\_la\_variable

<sup>5</sup>cost function network format

```

"v2": {"scope": ["v2"], "costs": [1.0, 0.5]},
"v1": {"scope": ["v1", "v2"], "costs": [-1.0, 0.5, -2.0, 5.5]}
}
}

```

Ainsi, un problème wcsp, i.e. un réseau de fonctions de coûts<sup>6</sup>, peut être représenté, au choix, par le format wcsp ou cfn. On peut exécuter toulbar2 sans options pour en obtenir une liste complète des formats supportés.

## 4.4 Elément synoptique du code de toulbar2

On décrit ici succinctement des choix de conceptions de toulbar2 utiles à la compréhension du code.

### 4.4.1 Description succincte du code

1. Fichier principal du programme : toulbar2/src/tb2main.cpp :
  - Lit les options passées à toulbar2 avec SimpleOpt,
  - Charge les fichiers d'entrées,
  - Crée l'objet solver,
  - Lance la méthode solve()
2. Fichiers essentiels du solver : toulbar2/src/search/tb2solve.cpp et tb2solve.hpp
3. La classe solver dérive de la classe WeightedCSPSolver qui représente un solver wcsp
4. Attributs principaux de la classe solver qui contient à la fois des types de base et d'autres classes ou objets :
  - Un pointeur vers un objet wcsp de type WeightedCSP : représente un problème wcsp. On accède en lecture ou écriture à la borne inférieure globale LB ou à la borne supérieure globale UB via cet objet wcsp : wcsp->getUb(). idem pour les méthodes : wcsp->setSolution(...), wcsp->updateUb(...),
  - Classe OpenNode : représente un nœud ouvert avec un coût de type Cost qui définit la borne inférieure locale. Cost est un type personnalisé dans toulbar2 correspondant à un *long long int*. D'autres types, telle que Value sont en fait des entiers.
  - Classe OpenList : c'est en fait une file de priorité qui contiendra les nœuds classés selon leur borne inférieure locale croissante.
  - Structure ChoicePoint : représente un point de choix, une décision dans l'arbre de recherche (branching)
  - Classe CPstore : vecteur de points de choix qui mémorise de manière économique les décisions prises pour pouvoir les "rejouer",
  - Classe Work : classe ajoutée pour passer des messages entre processus avec la bibliothèque boost.MPI. Utilisée dans la méthode hybridSolvePara().
5. Principales méthodes :

---

<sup>6</sup>Un réseau bayésien et un Markov Random Field (MRF) pourront être représentés par des fichiers au format uai.

- pair<Cost, Cost> Solver : :hybridSolve(Cluster \*cluster, Cost clb, Cost cub) : Effectue le HBFS séquentiel avec ou sans prise en compte de la structure du problème sous forme d'arbres de clusters : [Backtracking with Tree Decomposition Hybrid Best-node First Search \(BTD-HBFS\)](#),
- pair<Cost, Cost> Solver : :hybridSolvePara(Cost clb, Cost cub) : version parallélisée du HBFS,
- recursiveSolve(...) et binaryChoicePoint(...) : Effectuent récursivement le branch and bound en profondeur d'abord (DFBB),
- bool Solver : :solve() : initie la recherche,
- void Solver : :restore(...) : Restaure un nœud en rejouant la séquence de décisions dans l'arbre de recherche lorsque ce nœud est prélevé de la file open.
- pair<Cost, Cost> Solver : :hybridSolveSeq(Cost clb, Cost cub) : version HBFS séquentielle expurgée du BTD-HBFS

#### 4.4.2 Fonction général de toulbar2

Un objet "solver" de wcsp est créé qui possède un état caractérisé par les valeurs de ses attributs.

Le solveur comprend également un certain nombre de méthodes qui modifient les attributs du solveurs. Des méthodes permettent de sauvegarder ou de restaurer l'état du solveur.

Les attributs importants de la recherche tels que UB se trouvent dans l'objet wcsp accédé par adresse et qui représente un problème wcsp.

Les méthodes modifient donc les variables *globalement*, la portée étant celle de la classe Solver.

D'autres variables sont utilisées qui, elles, sont globales au niveau du programme, notamment les variables préfixées par "Toulbar2 ::" qui contrôlent toulbar2 notamment via les options passées en arguments au programme et lues par SimpleOpt.

La conséquence est qu'il serait probablement difficile et inefficace d'utiliser une parallelisation en mémoire partagée due à la nécessité de se prémunir des problèmes liés aux accès concurrents. L'utilisation de Pthreads<sup>7</sup> et openMP<sup>8</sup> n'a pas été explorée.

Après cette aperçu du code et du fonctionnement de toulbar2, il est de s'intéresser à l'algorithme [Hybrid Best-First Search \(HBFS\)](#) lui-même.

---

<sup>7</sup>Pthread ou POSIX threads API de bas niveau pour la programmation multithreadée. Permet un contrôle précis du parallélisme : [https://en.wikipedia.org/wiki/POSIX\\_Threads](https://en.wikipedia.org/wiki/POSIX_Threads)

<sup>8</sup>OpenMP (Open Multi-Processing) API de plus haut niveau que pthread pour le calcul parallèle sur architecture à mémoire partagée. Plus portable et facile à utiliser que Pthreads : <https://fr.wikipedia.org/wiki/OpenMP>. A ne pas confondre avec openMPI, qui est une des implémentations, comme MPICH, d'un standard de parallélisation appelé, Message Passing Interface, destiné aux architectures à mémoire distribuées.



## 5. Hybrid Best-First Search

### Résumé

Ce chapitre présente en détail l'algorithme HBFS séquentiel. Il combine une file de priorité *open* qui contient les nœuds ouverts, c'est à dire les nœuds dont le sous arbre n'a pas encore été totalement développé, et un Branch and Bound avec parcours en profondeur d'abord, borné par un nombre de backtracks fixé dynamiquement pour améliorer le compromis entre

Dans le cadre de ce PFE, on considère que l'algorithme HBFS décrit ci-après prend en entrée un problème wcsp, au format \*.wcsp, qui représente une réseau de fonction de coûts, c'est à dire un hypergraphe, qu'il s'agit d'*analyser* à l'aide d'un arbre de décisions ou de recherche développé au fur et à mesure que des décisions sur les domaines des variables sont prises. On restera à un niveau abstrait sans décrire les problèmes pratiques qu'ils modélisent qui relèvent de domaines variés : biologie, etc.

### 5.1 Definitions

#### 5.1.1 Depth-First Search

Désigne une recherche dans un graphe où les nœuds sont explorés jusqu'à atteindre la profondeur maximale avant d'effectuer un retour arrière et de rechercher à nouveau la profondeur maximale. Cf. par exemple : [https://en.wikipedia.org/wiki/Depth-first\\_search](https://en.wikipedia.org/wiki/Depth-first_search).

#### 5.1.2 Breadth-First Search

Désigne une recherche dans un graphe où les nœuds sont explorés dans l'ordre déterminé par leur profondeur : tous les nœuds de profondeur 1 sont d'abord explorés, puis ceux de

profondeur 2, etc. Une implémentation non récursive d'une telle recherche utilise une pile. Cf. par exemple : [https://en.wikipedia.org/wiki/Breadth-first\\_search](https://en.wikipedia.org/wiki/Breadth-first_search). Cette pile contient l'ensemble des nœuds situés sur la *frontière* qui sépare les nœuds explorés de ceux qui ne le sont pas encore.

### 5.1.3 Best-First Search

Désigne une recherche dans un graphe où les nœuds les plus prometteurs sont explorés en premier. Une heuristique détermine le caractère prometteur. Une implémentation non récursive d'une telle recherche utilise une file de priorité. Cf. par exemple : [https://en.wikipedia.org/wiki/Best-first\\_search](https://en.wikipedia.org/wiki/Best-first_search).

### 5.1.4 nœud ouvert

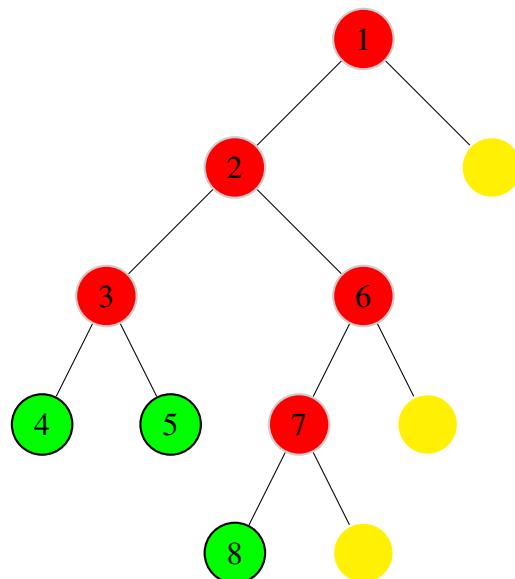


FIGURE 5.1 : Arbre binaire de décision partiellement exploré par un DFS avec une limite de backtrack,  $Z = 3$ . Les nœuds verts sont les feuilles de l'arbre donnant lieu à une affectation complète des variables. Les nœuds jaunes non cerclés sur les branches droites de l'arbre sont les nœuds ouverts i.e. le sous arbre associé n'a pas encore été exploré. Ces nœuds sont placés dans la file *open* par le DFS lorsqu'il atteint la limite  $Z = 3$ . En rouge, ce sont les nœuds fermés. Les nombres indiquent l'ordre de parcours DFS de l'arbre.

### 5.1.5 Backtrack adaptatif

Désigne une heuristique qui permet de trouver un compromis entre deux objectifs contradictoires, la diversification et les propagations répétées :

1. Diversification : le nombre de backtracks doit être suffisamment faible pour diversifier la recherche. En effet, l'arrêt du DFS donne la possibilité de choisir un nouveau nœud dans la file *open* plus prometteur situé à un endroit différent dans la frontière de

l'arbre de recherche. Cela permet donc de reconsidérer les choix faits précédemment et d'augmenter la probabilité de trouver une nouvelle meilleure solution.

2. Propagation : Le nombre de backtracks doit être suffisamment élevés car à chaque fois qu'un nœud est prélevé dans la file *open* deux opérations doivent être effectuées :
  - (a) Le rejet des  $v.\delta$  décisions qui ont conduit à ce dernier,
  - (b) Le re-calcul de  $w_\emptyset$  à partir de la racine (Soft Arc consistency re-enforcement)

### 5.1.6 Hybrid Best-First Search

La recherche hybride en meilleur nœud d'abord (HBFS) combine deux approches : une **Depth First Search with B&B (DFBB)** et une **Breadth First Search (BFS)**.

Les différentes étapes de l'algorithme de recherche dans l'arbre de décision associé au graphe du problème à résoudre sont développées ci-dessous. La figure 5.2 et l'algorithme 1 illustrent également son fonctionnement décrit dans l'article *Anytime Hybrid Best-First Search with Tree Decomposition for Weighted CSP* [2] dont on reprend les notations pour en faciliter une éventuelle lecture.

En particulier, un nœud  $v$  possède deux attributs qui caractérise son *état* :

1.  $v.\delta$  : la séquence de décisions, ou points de choix, qui ont conduit à ce nœud dans l'arbre de recherche ; un nœud est caractérisé par un chemin dans cet arbre. Il est associé à une affectation partielle des variables :  $A_v$ , voire à une affectation complète si le nœud est une feuille.
2.  $v.lb$  : la borne inférieure locale au nœud utilisée dans le HBFS pour élaguer l'arbre et pour calculer la borne inférieure globale  $LB = clb$ . Cette dernière étant égale au plus grand des  $v.lb$  des nœuds *frontière*. Son calcul est aisément puisque les nœuds sont classés par  $lb$  décroissant dans la file de priorité *open*.

Le gap d'optimalité est l'intervalle noté  $[clb, cub]$ .  $cub$  est la meilleure valeur de la fonction de coûts globales courante trouvée,  $clb$  est la borne inférieure globale décrite ci-dessus.

#### 1. Initialisation :

- Le nœud racine est initialisé avec  $v.\delta = \emptyset$  et  $v.lb = clb = w_\emptyset$  ; en effet, à ce stade, la borne inférieure locale au nœud  $lb$  est aussi la valeur globale. La meilleure valeur courante est initialisée par consistance d'arc locale. Dans HBFS, la méthode par défaut pour calculer cette borne inférieure  $w_\emptyset$  est l'**Existential Directional Arc Consistency (EDAC)**[8].
  - Le nœud racine est placé dans la file de priorité *open*.
  - La borne supérieure  $cub$ , qui est la meilleure valeur courante pour la fonction de coûts globale, est fixée à  $k^1$  ou à un coût très grand.
2. Le nœud est retiré de la file pour être traité par le DFBB, i.e. le Branch and Bound avec parcours DFS,
  3. Après un certains nombre de backtracks<sup>2</sup>, le DFBB est arrêté,

---

<sup>1</sup> $k$  est ce qu'on peut appeler l'élément absorbant dans la structure de valuation du problème wcsp. Avant la recherche, c'est la meilleure valeur que l'ont ait.

<sup>2</sup>le DFBB effectue une recherche bornée par le nombre de backtracks autorisés  $Z$ .  $Z$  n'est pas constant

4. L'algorithme DFBB place alors dans la file open les nœuds ouverts<sup>3</sup>, nœuds qu'il n'a pas eu le temps d'explorer,
5. La file de priorité se charge de classer les nœuds selon leur borne inférieure croissante ou selon leur profondeur décroissante en cas d'égalité. Le premier nœud de la file est censé être le plus prometteur. En tout cas, il permet de calculer efficacement la borne inférieur globale  $clb$ . Le second critère du tri lexicographique avec profondeur décroissante sélectionne le nœud le plus "profond" davantage susceptible d'aboutir à une affectation complète donc à un nouvel UB inférieur à  $clb$ , donc à des possibilités potentiellement accrues d'élagage. NB : il y a peut-être des améliorations possibles via la piste proposée sur <https://hal-enpc.archives-ouvertes.fr/hal-01230883>,
6. Si le DFBB parvient à une feuille de l'arbre de recherche<sup>4</sup>, une affectation complète des variables du problème est trouvée et permet de calculer une nouvelle valeur pour la fonction de coûts globale. Si cette valeur est meilleure i.e. inférieure à la valeur courante alors elle est mise à jour i.e. l'incumbent value is updated,
7. Le meilleur nœud est retiré de la file open,
8. l' état du nœud est re-calculé (restauration) ainsi que sa borne inférieure avant d'être traité par le DFBB qui ajoute à la file open des nœud ouverts s'il n'a pas eu le temps de traiter l'ensemble du sous-arbre associé.
9. L'opération se poursuit tant que la file open n'est pas vide,
10. Les nœuds tels que la borne inférieure locale  $clb \geq cub$  ne sont pas traités. On dit que l'arbre de recherche est élagué (pruning).

---

mais adaptatif. L'heuristique d'adaptation joue un rôle fondamental dans les performances de HBFS. En outre, en cas de manque de mémoire, hbfs passe en mode pure DFS jusqu'à ce que la pénurie cesse.

<sup>3</sup>Un nœud ouvert peut être vu comme un sous-problème ou un sous-arbre qu'il reste à explorer par opposition à un nœud fermé

<sup>4</sup>La séquence de décisions  $\delta$  est telle quelle détermine une affectation complète.

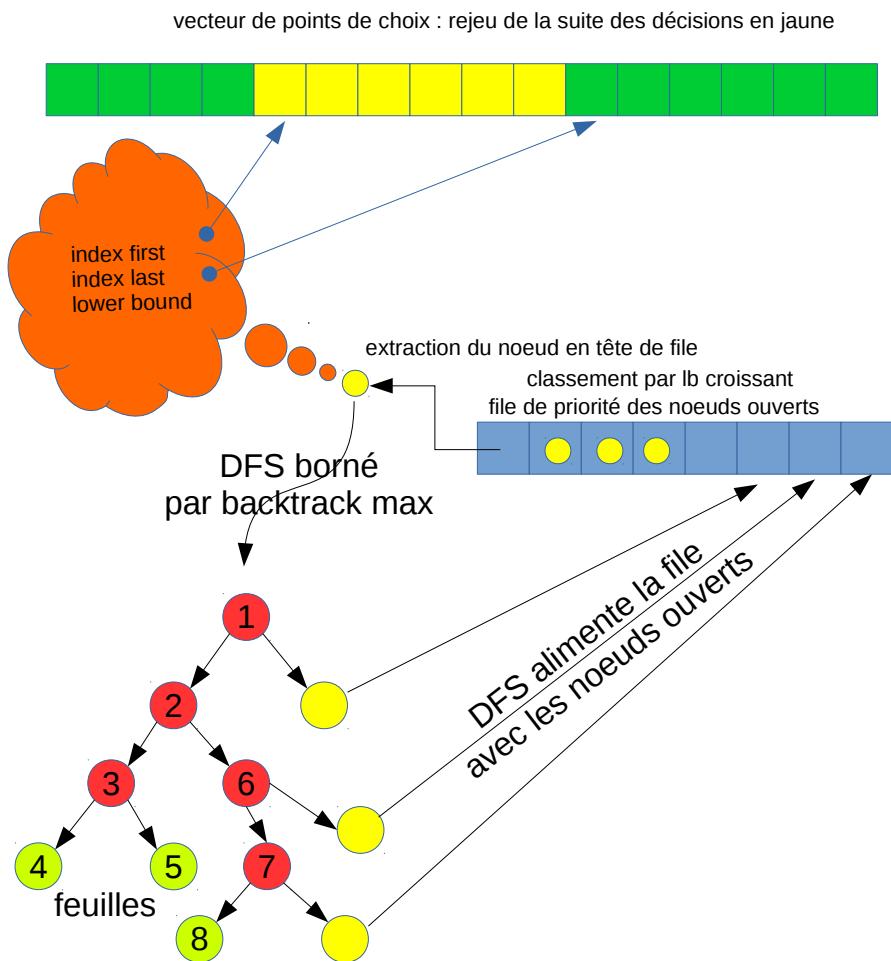


FIGURE 5.2 : Illustration de l'algorithme : un nœud est retiré de la file open, donné au DFS qui retourne des nœuds ouverts dans la file open, qui les classe par clb croissants. Un vecteur de décision, i.e. de points de choix, optimise l'espace mémoire en évitant les redondances. En effet, deux nœuds peuvent avoir en commun un même chemin dans l'arbre de décisions. Il est plus économique en mémoire d'utiliser deux index, first et last, qui pointent sur le chemin du nœud que de mémoriser dans chaque nœud le chemin complet.

---

**Algorithm 1:** Hybrid Best-First Search. Initial call : HBFS( $w_\emptyset, k$ ) with  $Z = 1$ .

---

```
/* clb : borne inf. globale courante, cub : borne sup. globale courante */  
/* [clb,cub] : gap d'optimalité */  
1 Function HBFS(clb, cub) : pair(integer,integer)  
| /* initialisation du nœud racine */  
2   open := v( $\delta = \emptyset, lb = clb$ ) ;  
| /* condition d'élagage  $clb \geq cub$  */  
3   while (open  $\neq \emptyset$  and clb  $< cub$ ) do  
4     v := pop(open) ;           /* on pop le nœud le plus "prometteur" */  
5     Restore state v. $\delta$ , leading to assignment  $A_v$ , maintaining local consistency ;  
6     NodesRecompute := NodesRecompute + v.depth ;  
7     cub := DFS( $A_v, cub, Z$ )/* puts all right open branches in open */ ;  
8     clb := max(clb, lb(open)) ;          /* calcul du clb global courant */  
|   /* heuristic Z adaptatif */  
9     if (NodesRecompute  $> 0$ ) then  
10      if (NodesRecompute/Nodes  $> \beta$  and  $Z \leq N$ ) then  $Z := 2 \times Z$ ;  
11      else if (NodesRecompute/Nodes  $< \alpha$  and  $Z \geq 2$ ) then  $Z := Z/2$ ;  
12    return (clb, cub);
```

---



## 6. Titre à préciser : Etat de l'art sur la po

### 6.1 Introduction

L'article "Parallel Solvers for Mixed Integer Linear Optimization" [19] présente une étude sur les solveurs parallèle en 6 points, les 3 principaux étant

1. Problèmes rencontrés lors de la parallélisation de ces algorithmes séquentiels
2. Revue des approches choisies par certains solveurs
3. Comment mesurer les performances d'un solveur

L'article traite de la résolution de problèmes d'optimisation mixte en nombres entiers sur les architectures parallèles. Les progrès importants effectués sur les algorithmes séquentiels ont ralenti la recherche sur les algorithmes parallèles. La parallélisation porte sur des versions simplifiées des algorithmes séquentiels car plus ce dernier est sophistiqué plus il est difficile à paralléliser. Ces derniers dépendent beaucoup de l'ordre de traitement des nœuds et la parallélisation a tendance casser cet ordre de traitement pourtant sélectionné pour son efficacité dans l'algo séquentiel.

les auteurs effectuent un état de l'art sur le MILP et rappellent la formulation du problème et l'utilisation du problème relaxé et des techniques de plan de coupe pour obtenir une meilleure borne inférieure dans le branch and bound.

Le branch and bound utilise une borne inférieure lb et une borne supérieure globale UB qui permettent l'élagage de l'arbre de recherche lorsque lb, la borne inf du sous-problème, est supérieure ou égale à UB. Un nœud de l'arbre de recherche correspondant à un SP, ce nœud est alors supprimé.

La notion de sous problème (SP) dans l'article se réfère au fait que le SP porte sur une espace réalisable plus petit que celui du problème d'origine. La nature intrinsèque du problème n'est pas modifiée permettant un traitement récursif. Les espaces réalisables de l'ensemble des SP d'un problème P forment un recouvrement voire une partition de l'espace réalisable de P.

Les solveurs commerciaux Xpress, Gurobi, Cplex sont parallélisés en mémoire partagée. Les solveurs libres CBC, Glpk, Lpsolve, SYMPHONY, DIP et SCIP possèdent pour certains des algorithmes parallèles.

## 6.2 Algorithmes parallèles

Cette partie expose certains problèmes rencontrés lors de la conception, de l'implémentation et de la mesure des performances des algorithmes de résolution de MILP.

### 6.2.1 Définitions

:

#### **Algorithme parallèle :**

Généralement, désigne la partie de l'algorithme qui contrôle le parallélisme : affectation de tâches aux processeurs, déplacement des données entre processeurs et non le programme parallélisé dans sa globalité.

#### **Algorithme séquentiel sous-jacent :**

Ensemble de sous tâches associées à l'algorithme purement séquentiel telles que les opérations de branching et du calcul des bornes dans le B&B. Ces éléments constituent souvent des boîtes noires pour l'algorithme parallèle.

#### **Wallclock time**

: temps global d'exécution d'un algorithme par opposition à un temps qui mesurerait le temps passé à effectuer des tâches de communications.

#### **Speed-up S :**

Souvent définie comme le rapport du temps de l'algorithme parallélisé sur le temps nécessaire à l'algorithme séquentiel pour résoudre le même problème. Un algorithme peut être plus performant mais moins scalable qu'un autre.

#### **Efficacité parallèle E :**

speed-up par nombre de coeurs. S et E caractérisent la performance globale du programme. D'autres métriques ou critères peuvent être utilisées e.g. la capacité d'un programme à résoudre des instances difficiles.

### **Scalabilité ou échelonnabilité :**

Aptitude du programme parallélisé à tirer partie des ressources à disposition en particulier on cherchera à obtenir des performances croissant linéairement avec le nombre de processeurs.

L'existence d'un compromis, ou trade-off, à trouver entre scalabilité et performances globales est une des difficultés fondamentale dans l'évaluation des performances.

### **Phases :**

Un algorithme parallélisé se déroule en 3 phases : ramp-up ou montée en puissance, primary ou principale et ramp-down ou terminaison. La phase ramp-up débute avec le programme et se termine lorsque tous les processeurs/workers ont reçu pour la première fois du travail à effectuer. La phase de ramp-down débute lorsqu'au moins un worker devient inoccupé de manière permanente et se termine lorsque la solution globale est affichée.

### **Overhead ou surcoût :**

quantité de travail nécessaire à la parallélisation qui ne serait pas effectuée dans l'algorithme séquentiel. Les auteurs de l'article distinguent 4 types d'overhead :

1. communication : temps passé à déplacer les données entre processeurs,
2. inactivité due aux phases de ramp-up et ramp-down e.g. où tous les processeurs ne sont pas encore alimentés en travail,
3. latence associée à l'attente de données ou au temps d'allocation de mémoire,
4. travail redondant : temps perdu à répéter les mêmes tâches alors que cette redondance n'existerait pas dans le programme séquentiel.

Une parallélisation efficace doit contrôler l'overhead mais il n'existe pas de solution standard car forte dépendance aux propriétés de l'algorithme et à l'architecture matérielle. En particulier l'overhead de communication dépendra du type réseau entre circuit intégrés (ces derniers sont nommés sockets sous linux cf. sortie de la commande lscpu), du nombre de coeurs par socket, du nombre de thread par cœur. Une machine à 8 processeurs peut comporter 1 socket, 4 coeurs, 2 threads par coeur (hyperthreading : un thread par processeur peut s'exécuter sur l'ALU alors que le second utilise l'unité de gestion mémoire. Le système d'exploitation voit deux coeurs "logiques" au lieu d'un seul cœur physique. Il peut affecter deux threads ou deux processus à ces deux coeurs logiques pour exploiter la structure superscalaire (parallelisme au niveau instruction). Cette dernière permet l'exécution de plusieurs instructions par cycle d'horloge. L'hyperthreading permet d'augmenter le nombre d'instructions indépendantes dans le pipeline.

### **Granularité :**

4 principaux types de granularités, classée ici de la plus granuleuse à la plus fine, peuvent caractériser les algorithmes parallèles. La granularité définit la quantité de travail atomique de l'algorithme.

1. Tree parallelism : Le parallélisme porte sur la recherche en parallèle dans plusieurs arbres avec des stratégies différentes ; le même espace d'état est parcouru au même moment. C'est l'approche portfolio qui vise à déterminer l'algorithme le plus efficace pour une instance donnée ; une course est lancée entre les solutions ou les stratégies.

2. Subtree parallelism : plusieurs sous-arbres sont explorés simultanément mais indépendamment sans échange d'informations. Une possibilité serait de passer un sous-arbre à un algorithme séquentiel et d'exécuter l'algorithme séquentiel durant un temps déterminé. Cette granularité est utilisée sous une forme ou une autre par les algorithmes les plus avancés pour limiter l'overhead.
3. Node parallelism : Plusieurs nœuds sont traités en parallèle avec un contrôle centralisé et des échanges d'informations. Le schéma de coordination Master-Worker constitue une implémentation immédiate. Le master distribue les nœuds à traiter présents dans une liste Q et collecte les résultats et les nœuds, dits ouverts ou actifs, à traiter produits par le worker. C'est le mode de parallélisation le plus courant. ndlr : par rapport à la granularité subtree, on constate qu'un sous arbre est aussi un nœud, puisqu'un nœud représente un sous arbre en voie de développement en sous arbre.
4. Subnode parallelism : parallélisation des traitements effectués au sein de chaque nœud. Ces traitements varient en fonction des problèmes et des solutions utilisées.

#### **Adaptivity :**

Propriété d'un algorithme à modifier son comportement par exemple à effectuer un traitement différent, e.g. changer la granularité, en fonction des 3 phases : montée en charge ou ramp-up, principale et ramp-down.

#### **worker :**

Désigne une entité qui effectue un traitement sur demande. ndlr : Généralement, un worker désignera un processus en mémoire distribuée ou un thread en mémoire partagé. Ce qui correspondra à un cœur physique contenu dans une puce i.e. socket voire virtuel dans le cas d'une architecture hyperthreadée notamment des processeurs Intel.

#### **Load balancing (équilibrage de charge) :**

Désigne la distribution du travail utile de manière à minimiser les temps d'inactivité des workers.

#### **Warm start, Sur Load balancing, sur-équilibrage :**

Les calculs effectués par un nœud parent, s'ils sont utilisés par le noeud fils constituent le warm start ou démarrage à chaud car lui évite des traitements. Parfois l'équilibrage à tout prix est contre productif car aboutit à casser ce lien avec nécessité pour le fils affecté un worker d'effectuer à nouveau ces calculs.

#### **Partage de connaissances :**

Données générées durant la recherche qui sont utiles aux autres workers : bornes, solutions réalisables produites par un worker utiles à d'autres

#### **Détermiminisme :**

capacité d'un algorithme à produire la même solution quel que soit le nombre d'exécution. Le Déterminisme "fort" impose un ordre des opérations toujours identique mais ceci est difficile à garantir notamment dans le cas de l'hyperthreading.

### **Synchronisation :**

Méthode qui permet aux workers de travailler sur un état commun pour assurer la cohérence et/ou l'efficacité du programme. Implémenté par exemple sous forme de barrière ou les processus s'attendent en un point donné. nécessaire pour agréger des résultats intermédiaires ou finaux. Permet de renforcer le déterminisme. Inconvénient : les workers les plus rapides doivent attendre les plus lents donc plus il y a de workers plus il y a de temps et la scalabilité est alors mauvais. **La synchronisation est donc à éviter dans la mesure du possible.** Utilisation la plus évidente : obtenir des bornes globales les plus à jour possible. Heureusement, dans les B&B il n'est pas strictement nécessaire de maintenir à jour des bornes inférieures et supérieures globales. Le pruning nécessite uniquement la borne inférieure locale au nœud. Quant à la borne supérieure, une borne valide est suffisante pour le pruning même s'il serait souhaitable d'avoir une borne à jour et connue de tous les workers.

### **Coordination :**

La coordination des actions effectuée par un algorithme introduit un compromis à trouver entre l'augmentation de l'overhead et l'amélioration des performances de l'algorithme séquentiel sous-jacent. La coordination ne nécessite pas forcément une stricte synchronisation. exemple de schéma de coordination : load balancing, ...

### **Latence :**

Temps de transmission entre deux nœuds.

### **Topologie :**

Agencement global du réseau. La latence dépend de la topologie.

### **Framework :**

Lorsque le programme parallèle qui gère le parallélisme et le programme séquentiel sous-jacent sont suffisamment découpé il est possible d'utiliser un framework dont le rôle sera de gérer le parallélisme d'un algorithme séquentiel. Par exemple, la fonction de branching du branch and bound pourrait se faire partie de l'interface du framework sous forme de signature, charge à l'utilisateur de ce framework d'implémenter le code séquentiel correspondant dans cette fonction. Un framework peut se présenter sous forme d'un ensemble de classe de base qu'il s'agit de dériver et compléter les méthodes. L'avantage est d'éviter d'avoir à implémenter soi-même des stratégies de load balancing et autres coordinations nécessaires aux parallélismes.

## **6.2.2 Compromis**

Compromis partage de données pour augmenter l'efficacité de l'algorithme vs limiter les communications et donc l'overhead.

Compromis entre le temps passé dans les couches superficielles de l'arbre i.e. dans le traitement de la racine et des quelques couches suivantes vs temps nécessaire à l'algorithme

pour exploiter complètement tous les coeurs disponibles. Or cette phase de ramp-up peut être très chronophage par rapport à la recherche dans les couches profondes de l'arbre et varie selon les problèmes.

#### **Parallélisation naïve :**

Dans ce type de parallélisation, une tâche est constituée par la résolution d'un sous-problème dans son ensemble. Or, selon les auteurs, les ressources sont dans ce cas sous-exploitées lors des phases de ramp-up et ramp-down.

### **6.2.3 Load balancing**

L'équilibrage de charge défini précédemment est caractérisé par :

1. la fréquence à laquelle il est effectué,
2. par les informations qui doivent être déplacées.
3. par sa nature statique, load balancing(LB) effectué une seule fois au début ou dynamic, LB effectué durant tout le programme.

#### **Static load balancing**

Quatre catégories de méthodes d'initialisations ou LB statique

1. Root initialization : La plus utilisée pour sa facilité d'implémentation. Une processus traite le nœud racine et crée les nœuds enfants. Ces enfants sont distribués à tous les workers et cette technique est répétée intégralement jusqu'à ce que tous les workers soient occupés. Méthode plus efficace si Le temps de calcul d'un nœud est faible et le branchement est élevé de manière à mettre au travail les workers le plus rapidement possible. Méthode moins efficace si le nombre worker est important.
2. Enumerative initialization : le nœud root est transmis à tous les workers qui développent chacun le même arbre (il faut donc que le programme soit déterministe). lorsque le nombre de nœuds enfant est égal au nombre de workers, le worker i sélectionne le nœud i et efface le reste.
3. Selective initialization : La racine est broadcastée à chaque worker qui génère un seul chemin. Les chemins de chaque worker doit être distincts d'où la nécessité d'un schéma sophistiqué pour assurer que tous les workers travaillent sur des chemins différents.
4. Direct initialization : pas de création explicite de l'arbre. A la place, ce sont les workers qui créent directement un nœud à partir d'une certaine profondeur.
5. autres méthodes implémentées dans des framework : racing ramp-up dans UG, spiral et root initialization à deux niveaux dans CHiPPS.

#### **Dynamic load balancing : DLB**

Les schémas de load balancing sont fortement dépendants du type de l'algorithme. Caractérisés par le degré de centralisation et par l'initialisateur de l'équilibrage : processus dédiés (work-sharing), fournisseur ou receveur de travail (work-stealing).

quelques DLB :

1. Asynchronous round robin : exemple avec 8 workers de 0 à 7. Le worker 0 maintient une

variable target qui pointe sur le worker auquel demander du travail en cas d'innoccupation de ce worker 0. Initialement target =  $(i+1) \bmod 8 = (0+1) \bmod 8 = 1$ . Si le worker 0 passe en mode idle ; il envoie une requête au worker 1 et incrémente modulo 8 sa target. target := target++ mod 8 = 2, etc.

2. nearest neighbor : A chaque worker est associé un ensemble de voisins. Si un worker manque de travail, il envoie une requête à ses voisins définis précédemment. Avantage : communications localisée. Inconvénient : possible temps long pour atteindre le load balancing global.
3. Random polling scheme : Un worker envoie une requête à un worker choisi au hasard. Équiprobabilité de choix d'un worker. très simple et assez efficace dans certaines applications.

Le travail transmis entre workers fait partie des "connaissances" échangées ; il existe donc de nombreux compromis à trouver. Un LB agressif réduit le temps d'innovation des workers et le travail redondant mais augmente l'overhead. Les solveurs modernes emploient des stratégies plus sophistiquées et adaptatives.

### **6.3 Revue des approches choisies par certains solveurs**

### **6.4 Comment mesurer les performances d'un solveur**

### **6.5 conclusion**





## 7. Embarrassingly Parallel Search

### 7.1 Définition

- parallélisation naturelle
- division en sous problèmes indépendants
- env. 30 sous problèmes (SP) par cœur sur architecture 4/8
- load balancing assuré par sous problèmes suffisamment "petits"
- Utilisation de la file de priorité des noeuds ouverts de HBFS pour produire ces SP.

### 7.2 Benchmarks

### 7.3 Généralités sur la Embarrassingly Parallel Search (EPS)

Le terme *pittoresque* d'Embarrassingly Parallel Computation est dû à Geoffrey Fox et date de 1995. Ce paradigme désigne une parallélisation naturelle qui consiste à découper

wcsp file	Time (s) of subproblem (sp) generation	Serial Time (s)	Parallel time (s)	speed up	% CPU	obs
1PGB.11p.19aa.using	2,5	4,3	6,2	0,69	1422	75 sp
graph11	1,1	550,1 trop long	<1			464 sp*
capmp1	68,3	266,0	96,6	2,76	2226	172 sp – speed up total : 1,61
scen06	0,6	945,0	437,0	2,16		175 sp – speed up total : 1,82
capmo1	4,6	14,2	3,2	4,46	276	85 sp
pedigree18	0,3	340,1 trop long	<1			126 sp
pedigree7	0,9	2,4	4,5	0,54	393	82 sp
nug12	1,0	207,3	49,5	4,19	385	
nug12	0,9	207,0	35,9	5,77	510	153 sp
404.wcsp	0,4	36,7	20,9	1,8	203	731 sp -j60
404,wcsp	0,3	37,1	4,8	7,73	541	141 sp -j+0

un programme, si ce dernier le permet, en parties indépendantes et à les assigner à des "workers" c'est à dire aux coeurs ou processeurs disponibles. Les workers résolvent les sous-problèmes indépendamment et retourne leur résultat au master qui calcule la solution globale optimale.

## 7.4 Génération a priori des sous problèmes

### Génération par produit cartésien

Un programme C++ a été créé qui produit les assignations partielles donnant ainsi un nombre de sous problèmes tel que  $|A_{k-1}| < q \leq |A_k|$  où q est la valeur cible du nombre de sous problèmes désirés  $|A_k|$  correspond au cardinal du produit cartésien des domaines de k variables. cf [20].

```
./generate name of input wcsp file, space,
name of output subproblems file,space,
approximate number of sub problem
Example: ./generate 404.wcsp job_file 250
Parallel command : cat job_file | parallel -j+0 --eta -k ./toulbar2 404.wcsp {}
```

Le programme, dont on trouvera le listing en annexe ??, génère des affectations/décisions de type égalité comme ci-dessous :

```
256 sous problèmes correspondant aux 4 premières variables
du problème 404 de domaines {0,1,2,3}
```

```
-x=",0=0,1=0,2=0,3=0"
-x=",0=0,1=0,2=0,3=1"
-x=",0=0,1=0,2=0,3=2"
-x=",0=0,1=0,2=0,3=3"
-x=",0=0,1=0,2=1,3=0"
-x=",0=0,1=0,2=1,3=1"
-x=",0=0,1=0,2=1,3=2"
...

```

qui correspondent au produit cartésien calculé via la fonction:

```
/** \brief compute the cartesian product of a vector vectors
domains : set of domains dm represented by c++ vectors
*/
vector<vector<long>> cartProd (const vector<vector<long>>& domains)
{
    vector<vector<long>> vv = {{}}, tmp; // init vector of vectors
    for (const vector<long> & dm : domains) { // for each domain dm in the set of domains
        tmp= {};
        ...
        for (const long & d : dm) {
            for (const long & v : vv) {
                v.push_back(d);
            }
        }
        vv = tmp;
    }
    return vv;
}
```

```

for (const vector<long> & vv : vvv) { // for each sub vector v in vvv (v is {} at first)
for (const long value : dm) { // for each value in domain dm

    tmp.push_back(v); // add domain v in tmp here v ={} at first
    // cout << "x ="<<x << endl;
    tmp.back().push_back(value); // add value to the last domain in tmp, here v={} tmp ={ }

}
}

vvv.swap(tmp); // vvv become tmp and tmp become vvv
}
//cout<< tmp << endl;
return vvv;
}

```

## Résultats

Les résultats obtenus sur les problèmes au format wcsp : 404, scen06 sont décevants. Le temps d'exécution est largement majoré par rapport au programme séquentiel.

## 7.5 Génération des sous problèmes via hbfs

L'algorithme HBFS de Toulbar2 utilise une file de priorité, nommée open, pour stocker les nœuds ouverts qui constituent les sous problèmes qui restent à résoudre. Donc naturellement, en fixant une taille maximale pour open qui lorsqu'elle est atteinte ou dépassée aboutit à un dump des sous-problèmes et à la terminaison du programme, on obtient une série de sous problèmes de *bonne qualité* au sens où ils ont pu bénéficier des optimisations du programme séquentiel HBFS.

En outre de manière à obtenir un load balancing correct les sous problèmes doivent être de taille suffisamment petits. Pour ce faire, on se fixe une valeur N, qui donne de bons résultats, égale à environ 30 fois le nombre de cœurs[20]. Lorsque la taille de la file open dépasse cette valeur, on extrait les nœuds et on reconstitue l'affectation partielle associée. On utilise ensuite cette liste d'affectations partielles en tant que paramètre dans toulbar2 pour produire un ensemble de processus qu'il s'agit d'exécuter en parallèle par exemple en utilisant GNU parallel.

Pour le problème 404.wcsp, les courbes ci-dessous décrivent les résultats obtenus qui exhibent un speedup maximum de 4.9 sur une machine 4/8, 4 cœurs réels, 8 avec l'hyper-

threading avec un nombre de sous problèmes par cœur entre 28 et 31. On retrouve le résultat de Regin et al. mais ce dernier est probablement architecture et problème dépendant. Dans le cas présent, le calculateur n'est composé que d'un seul processeur physique (i.e. socket unique).

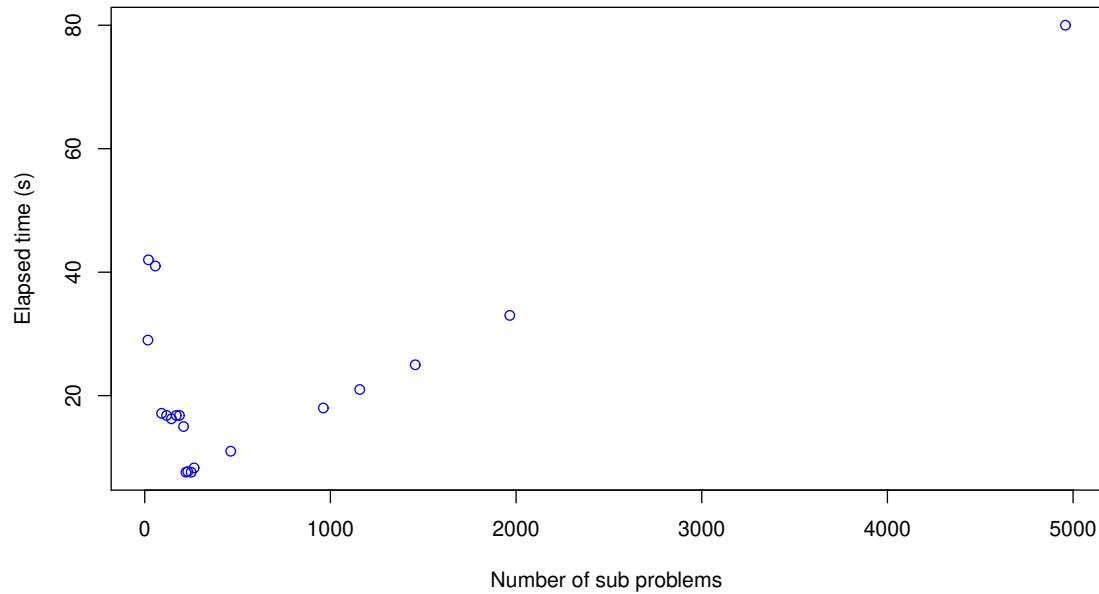


FIGURE 7.1 : a minimum for the elapse time is observed near 200 sub problems

Un zoom de la figure 7.10 donne une valeur plus précise de l'optimum.

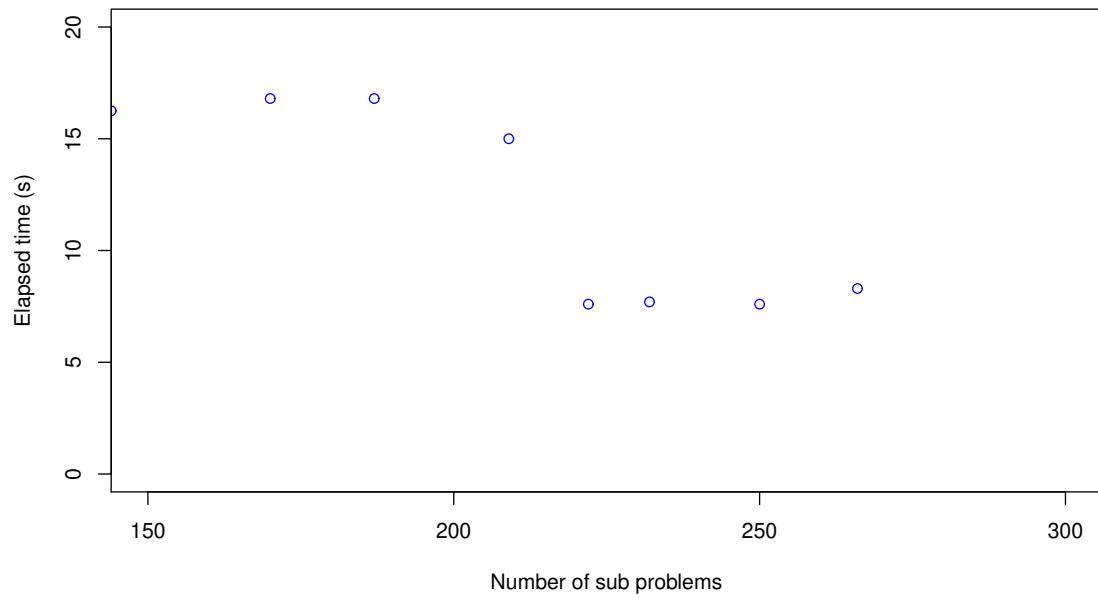


FIGURE 7.2 : Minimum clockwall time is reached for 220 to 250 subproblems

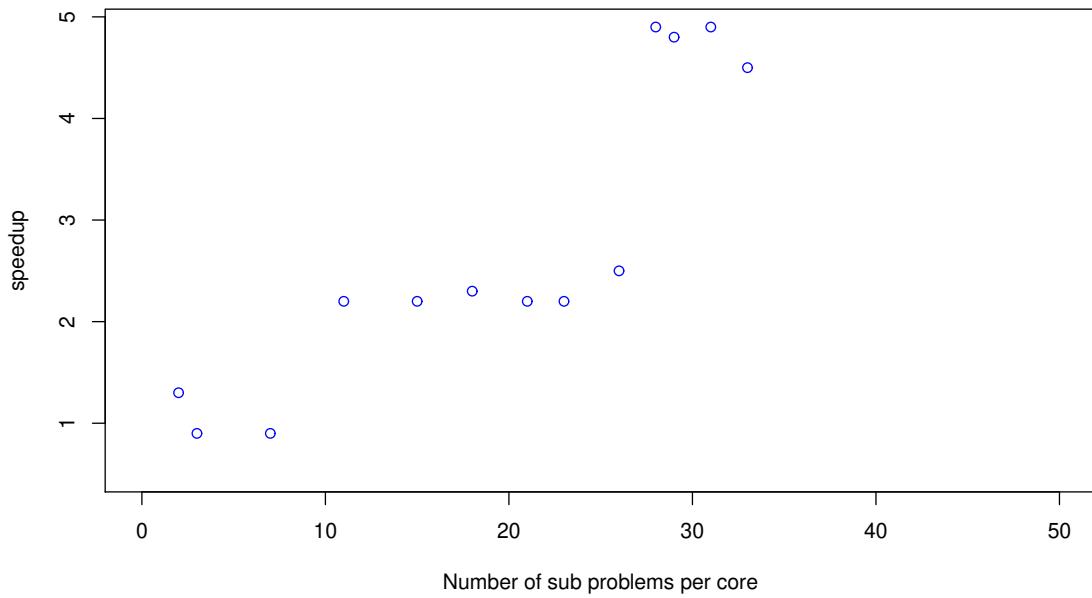


FIGURE 7.3 : Maximum speedup observé pour des valeurs de 28 à 31 voire 33 sous problème par cœur

## 7.6 Eléments descriptifs des résultats obtenus

### 7.6.1 Performances

#### Méthode de mesure

1. Toulbar2 est exécuté en séquentiel dans le dossier bin/Linux avec la commande `./toulbar2 pb.wcsp` pour obtenir le temps réel qui serait chronométré du lancement à la terminaison du programme.
2. On lance Toulbar2 -eps pb.wcsp pour produire le script `eps.sh` et les sous problèmes dans `subProblem.txt`. Les sous problèmes sont constitués d'une suite de décisions associées au branchement dans l'arbre de recherche qu'on peut considérer comme des affectations partielles. A titre d'exemple, l'option à donner à toulbar2 peut être de la forme :

```
-x="" ,44#0,46=1,60=1,45=1,50=1,71=0,58=1,88=1,61#1,59=3,49#1,9#3"
```

3. On utilise ensuite gnu parallel en lui passant en arguments les processus à exécuter en parallèle. L'option `-j` indique le nombre maximum de processus à exécuter en parallèle à un instant donné sur les  $N = 240$ . Sur une machine à 8 coeurs, on pourrait penser que 8 processus maximum donneraient un résultat optimum mais il a été observé sur une machine donnée et le problème `404.wcsp` que le temps d'exécution est optimal avec  $20 = 2.5 * 8$  processus en parallèle. Ce facteur de 2.5 optimal dans

le cas précédent est malheureusement architecture et problème dépendant d'après quelques expérimentations. Ce comportement est probablement dû à la manière dont gnu parallel est programmé, au scheduler linux et à l'architecture.

```
cat subProblems.txt | time parallel -j20 -eta ./toulbar2 404.wcsp -ub=114
```

### 7.6.2 Nombre de nœuds totals traités vs nombre de nœuds ouverts

La file open varie en taille au cours de l'exécution du programme. Des nœuds sont retirés de la file pour alimenter l'algorithme de recherche en profondeur (DFS) et ce dernier retourne des nœuds ouverts qui alimentent la liste open. Les expériences faites sur différents problèmes au format wcsp que lors de la génération des sous problèmes seules une minorité de nœuds ont été traités en totalités : nœuds fermés. Le problème scen06.wcsp par exemple, avec un nombre de problèmes cibles fixé à 240, fournit 257 nœuds ouverts pour 260 nœuds explorés au total. rapport 257/260. 404.wcsp -> 251/264, nug12.wcsp -> 243/286.

Le temps de génération des sous problèmes ne dépasse pas la demi-seconde sur une processeur 8 cœurs hyperthreadé : 4 cœurs réels, 2 threads par cœur. Le système d'exploitation voit 8 cœurs donc la valeur qui déclenchera l'extraction des sous problèmes est  $N = 30 * 8 = 240$ . Ce temps est négligeable devant la phase parallèle.

### 7.6.3 Optimisation de la génération

Seuls les sous problèmes pouvant donner lieu à solution sont générés : condition d'élagage borne inf lb  $\geq$  borne sup ub.

En outre, la meilleure solution ub trouvée au moment de la génération des sous problèmes (SP), ou incumbent solution, est utilisée dans les sous problèmes à exécuter en parallèle. Pour le problème 404.wcsp, la résolution sans utiliser l'option -ub=114 de toulbar2 pour tenir compte de la meilleure solution trouvée lors de la génération des SP est de 17s contre 7.5 secondes avec ub. Cette amélioration est bien sûre elle aussi problème dépendante.

L'intérêt d'utiliser le ub obtenu à la génération des SP est illustré sur le problème 404 par la figure 7.4 ci-dessous. Seule une minorité de sous problème trouve la valeur optimale globale, les autres n'aboutissent qu'à une solution sub optimale imposée par leur assignation partielle qui ne permet pas d'atteindre l'optimum global. D'où, d'une part d'utiliser toute information préalable sur le ub, et d'autre part, d'échanger dynamiquement les ub trouvés entre les processus. Ce qui peut se faire soit par passage de messages entre processus ou par mémoire partagée avec mutuelle exclusion. Cependant le principe de l'EPS ne prévoit pas d'échanges entre processus durant le traitement.

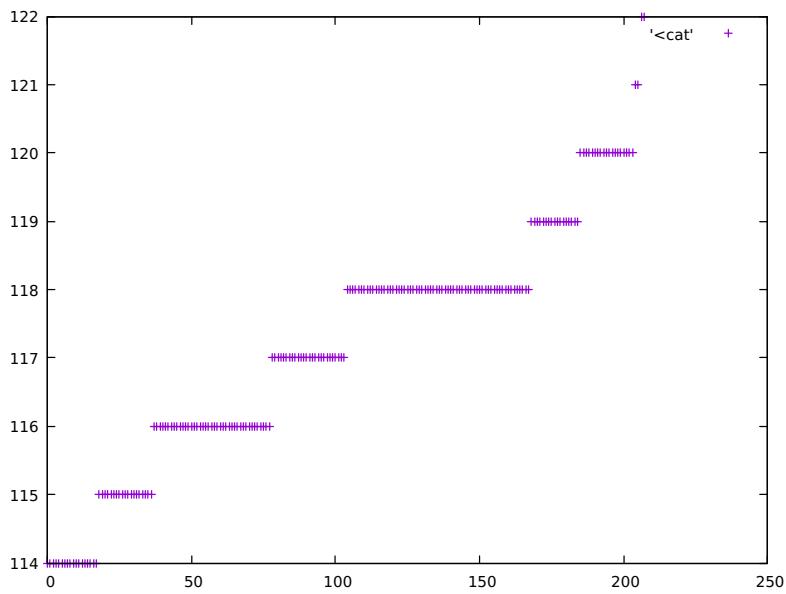


FIGURE 7.4 : Valeur de l’optimum trouvé par les 208 SP. Seule une minorité trouve l’optimum 114.

#### 7.6.4 Distribution en nombre de variables non assignées

Le % de nombre de variables non assignées pour les problèmes 404.wcsp et scen06.wcsp fournissent une indication sur la taille du problème si les domaines des variables sont de taille comparable.

exemple : pour le problème 404 qui compte nbVar=100 variables , on calcule le % comme suit :  $((\text{nbVar} - (\text{nb.last}-\text{nb.first}))/\text{nbVar}) * 100$ . Pour scen06, on trouve certains % négatifs !

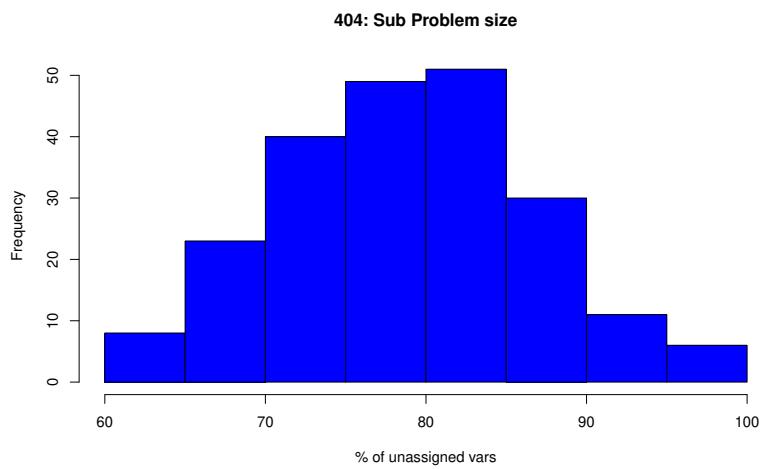


FIGURE 7.5 : Taille des sous problèmes en nombre de variables

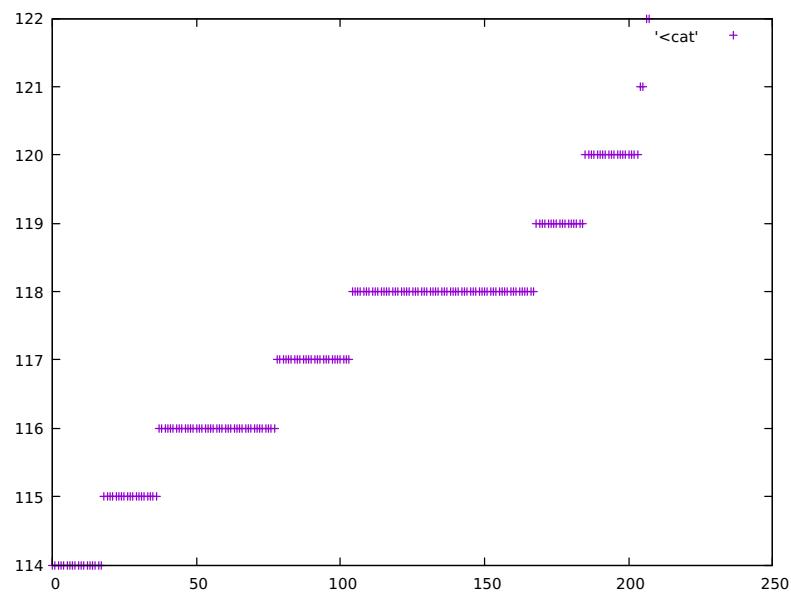


FIGURE 7.6 : Taille des sous problèmes en nombre de variables

```
ff

SubProblem      size...
Min.    :61.00  Mode:logical
1st Qu.:74.00  NA's:218
Median  :79.00
Mean    :79.36
3rd Qu.:85.00
Max.    :98.00
```

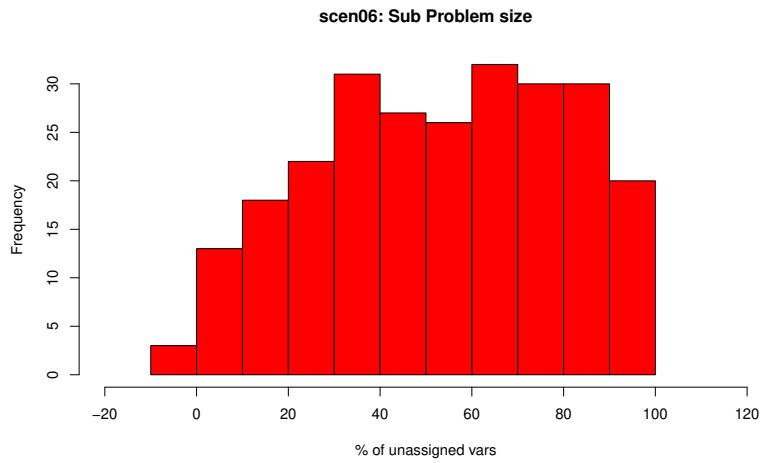


FIGURE 7.7 : Taille des sous problèmes en nombre de variables

```
> summary(size_scen06)
SubProblem      size...
Min.    :-4.00   Mode:logical
1st Qu.:32.00   NA's:252
Median  :55.00
Mean    :53.37
3rd Qu.:76.00
Max.    :98.00
```

### 7.6.5 Distribution en complexité des sous problèmes

Les complexités sont évaluées sous la forme du cardinal des produits cartésiens des domaines des variables non assignées, c'est à dire des variables non concernées par des décisions de type `:=`, `#`, `<`, `>`. Si les domaines sont réduits par propagation de contrainte au moment de l'extraction des sous problèmes alors ces complexités sont sur évaluées. Les résultats ci-dessous montre une grande disparité des complexités théoriques avec une majorité de problèmes complexes de l'ordre de  $10^{37}$  à  $10^{38}$ .

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
7.556e+22	5.071e+30	6.490e+32	6.535e+36	4.154e+34	3.403e+38

#### 404: Sub Problem Complexity

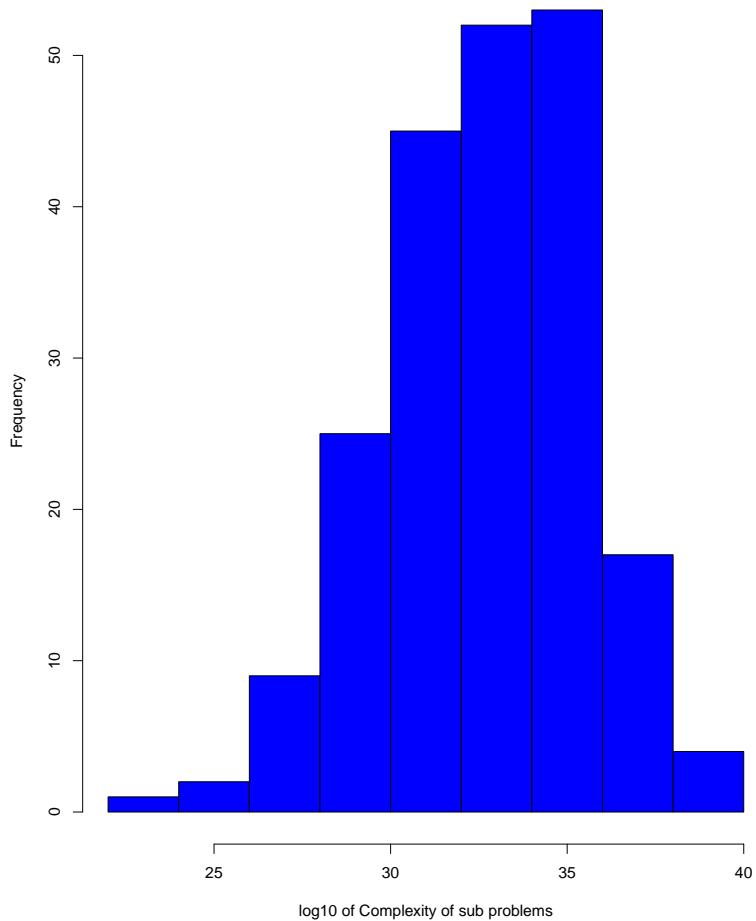


FIGURE 7.8 : Distribution du log10 des complexités

La complexité théorique est elle liée au temps de résolution du sous problème ? Ce qui permettrait de prévoir le temps de résolution. Le schéma ci-dessous indique que pour les complexités faibles, inférieures à 33 en échelle logarithmique le temps de résolution est faible. Par contre, au delà, les temps d'exécution des sous problèmes s'allongent et la prévision des temps associés est hasardeuse. On peut éventuellement inférer une application pratique pour la génération des sous problèmes à savoir calculer la complexité théorique des sous problèmes et remettre dans la file open ceux dont le CT dépassent un certain seuil. Encore faut-il pouvoir déterminer ce seuil et trouver le bon compromis entre la phase séquentielle de génération de SP et la phase parallèle.

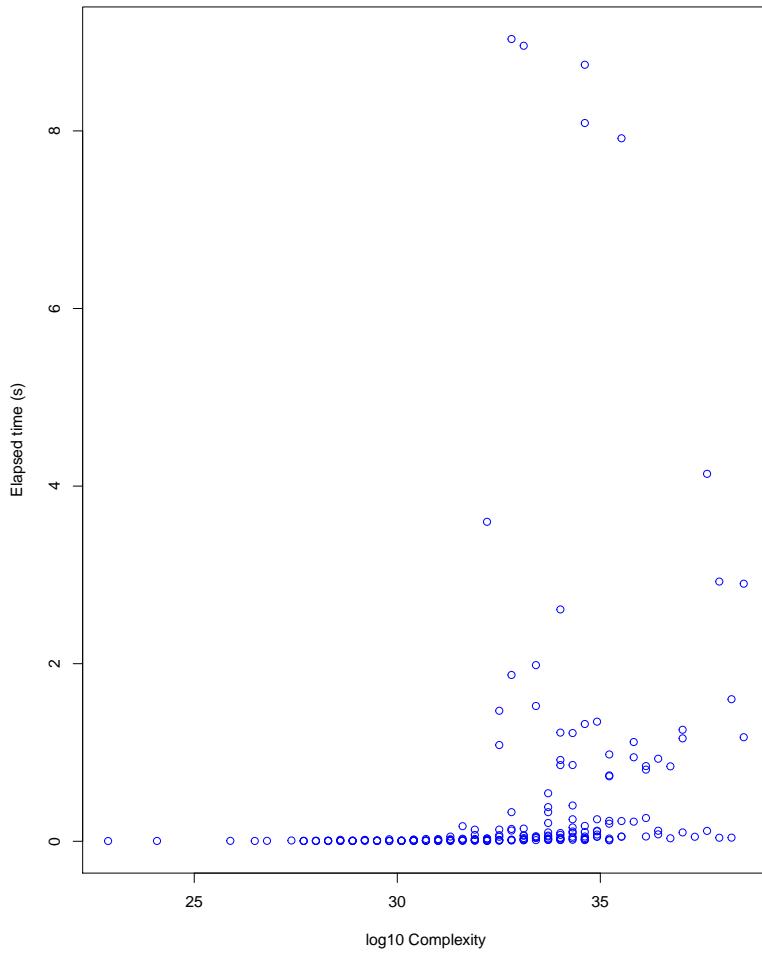


FIGURE 7.9 : time versus complexity of sub problems

#### 7.6.6 Temps individuel de résolution des sous problèmes

Les mesures sont effectuées sur le problème 404. Le temps total d'exécution de toulbar2 est de l'ordre de 17s sans utiliser le ub produit par lors de la production des sous problèmes. La majorité des sous problèmes prennent une fraction de seconde. Seuls quelques sous problèmes prennent plus de 7s. Une génération de sous problèmes de taille plus homogène permettrait sans doute d'améliorer le speedup en évitant que des coeurs ne soient inoccupés si le nombre de "petits" problèmes ne sont pas suffisamment nombreux pour compenser le temps nécessaire aux "gros" problèmes. Dans le problème 404, la visualisation de la charge des coeurs montre que ces derniers restent occupés jusqu'en fin de programme sans doute du fait d'une distribution de "taille" des sous problèmes qui permet de tenir occupé chaque cœur. (nb : similitude avec le problème du bin packing à ceci près qu'on ne cherche pas à minimiser le nombre de bin/coeurs utilisés mais à minimiser les différences de "remplissage" ; la dimension spatiale du bin packing étant remplacée par la dimension

temporelle : métriques analogues).

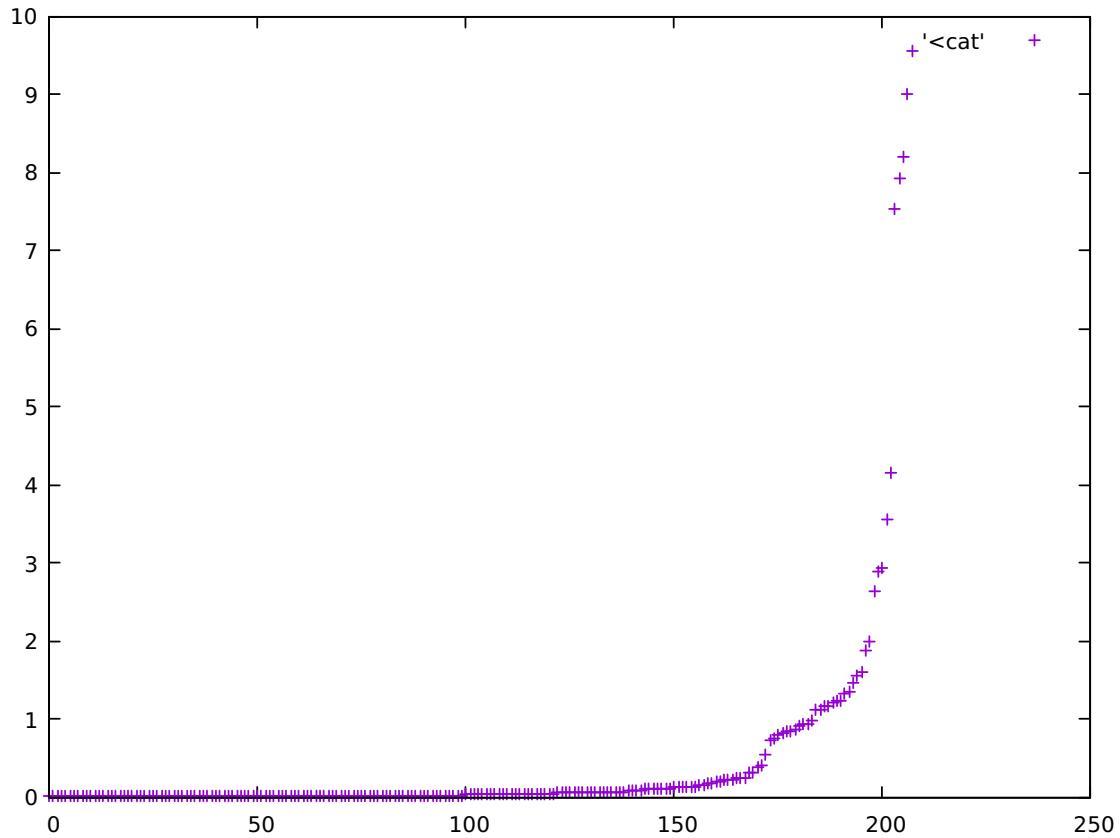


FIGURE 7.10 : courbe des temps de résolution en fonction des numéros des 208 SP du problème 404 sans utilisation de la borne ub calculée lors de la génération des SP

En récupérant et en utilisant l’incumbent solution ub obtenue au moment du dump, les performances sur le problème 404 sont fortement améliorées : gain de temps sur les problèmes les plus gros e.g. le 5ème sous problème passe de 9s à 4s. Ce qui au passage montre l’importance de pouvoir partager dynamiquement et facilement le ub entre les processus.

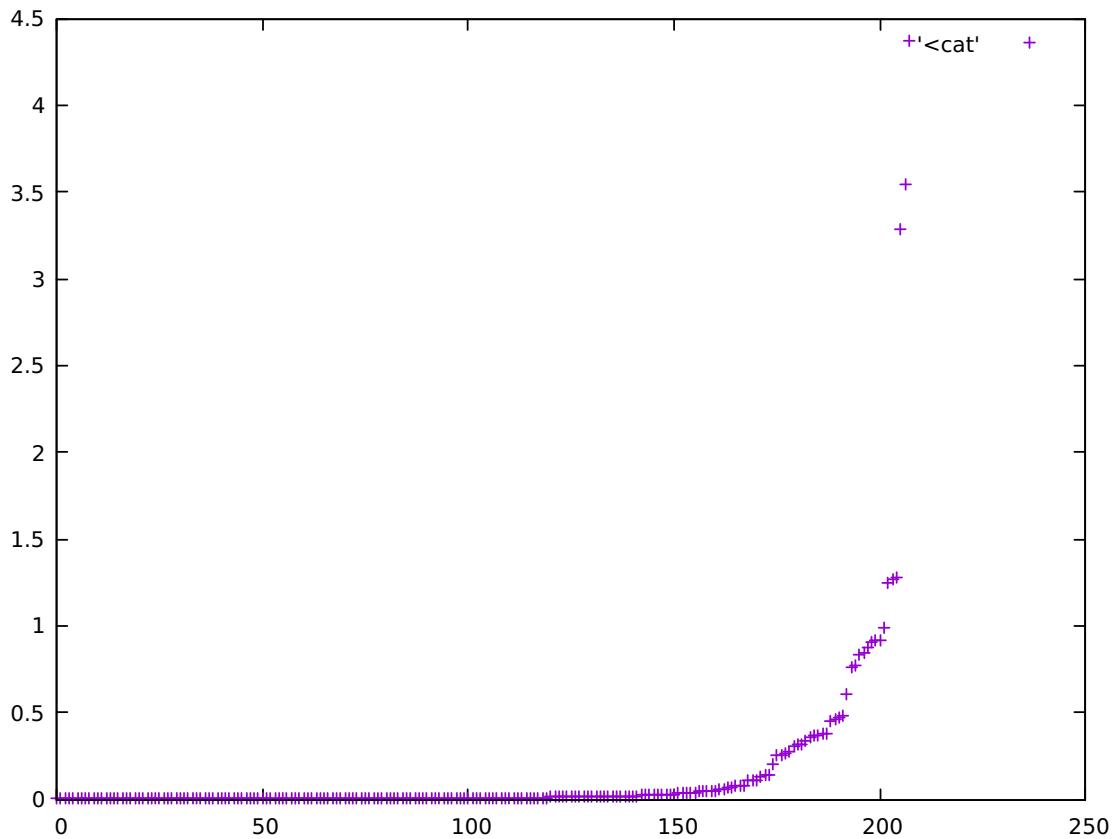


FIGURE 7.11 : courbe des temps de résolution en fonction des numéros des 208 SP du problème 404 avec -ub=114

### 7.6.7 Performances globales de la parallélisation : speed-up

Si des speed-up de 4.9 sur une machine 4/8 ont pu être atteint sur le pb 404.wcsp, les résultats sont décevants pour scen06.wcsp (speed-up = 1.2 à 2). Par ailleurs, sur un serveur à 2 processeurs avec 12 cœurs réels par processeur physique, les résultats ne sont pas améliorés en terme de speed-up, voire d'échelonnabilité (scalability) : perf moins bonnes des cœurs individuels, com. plus lentes entre les deux processeurs physiques (sockets), caches et processeurs moins bien exploités pour une cible de 30 SP/c.

Cependant, des résultats plus intéressants sont obtenus en fixant un seuil cible de 150 pour le nombre de nœuds dans la file open au moment du dump. Ce qui correspond plus ou moins à 6 SP/c.

Les problèmes Graph11, capmp1 et pedigree18 sont les plus récalcitrants avec des speedup inférieurs à 1 et des temps de résolution en mode parallèle importants.

## Benchmarks sur serveur 4/8 : 4 coeurs réels, 8 avec multithreading

Benchmark sur architecture 4-8 coeurs

wcsp file	Time (s) of subproblem (sp) generation	Serial Time (s)	parallel time (s)	speedup	% CPU	obs
1PGB.11p.19aa.usingEref_self_digit2	1,6	4,1	10,4	0,4	740	53 sub problems
graph11	1,0	574,0	trop long*	< 1		464 sub problems
capmp1	152,3	446,6	397,4	1,1	787	172 sp – speed up total : 0,81
scen06	0,6	944,0	792,0	1,2	610	257 sp
capmo1	5,3	14,1	18,9	0,7	744	85 sub problem
pedigree18	0,3	348,0	<1			
pedigree7	0,3	2,7	13,7	0,2	583	
nug12	0,3	202,0	52,8	3,8	467	
404	0,3	35,1	7,7	4,6	530	

\*pas de résultats après + de 12 heures

FIGURE 7.12 : Benchmarks sur architecture 4-8 coeurs

## Benchmarks sur serveur 2\*12 cœurs réels

Benchmark sur architecture 24 coeurs

wcsp file	Time (s) of subproblem (sp) generation	Serial Time (s)	Parallel time (s)	speedup	% CPU	obs	Gub : ub at generation time	optimal ub	gap	relative gap (%)
1PGB.11p.19	2,1	4,3	5,1	0,83	1388	52 sp	286561	286135	426	0,1
graph11	1,7	325,0	1294,1	0,25	138	488 sp* -A	30703	3080	27623	896,9
capmp1	61,2	266,0	101,9	2,61	2149	172 sp – spe	2491566	2460099	31467	1,3
scen06	0,6	945,0	437,0	2,16	619	175 sp	8433	3389	5044	148,8
capmo1	4,6	14,2	3,2	4,46	276	85 sp	1163083	1156908	6175	0,5
pedigree18	0,3	340,1 trop long	<1		126	sp	768569542	620 119 799	148 449 743	23,9
pedigree7	0,9	2,4	4,5	0,54	393	82 sp	539491253	295341819	244149434	82,7
pedigree7	0,9	2,4	6,7	0,36	397	137 sp	517834901	295341819	222493082	75,3
nug12	1,0	207,3	49,5	4,19	385	716 sp	608,0	578,0	30,0	5,2
nug12	0,9	207,0	35,9	5,77	510	150 sp	620	578	42	7,3
404.wcsp	0,4	36,7	20,9	1,8	203699	sp -j+0	114	114	0	0
404.wcsp	0,3	37,1	4,8	7,73	541	141 sp -j+0	120	114	6	5,3

```
cat subProblems.txt | time parallel -j24 --eta -k ./toulbar2 404.wcsp -ub=114 {}
cat subProblems.txt | time parallel -j24 --eta -k ./toulbar2 nug12.wcsp -ub=608 {}
cat subProblems.txt | time parallel -j24 --eta -k ./toulbar2 graph11.wcsp.xz -A -ub=30703 {}

Machine 2 * 12 cores : sullo avec cores partiellement occupés par d'autres process
```

```
graph11 remplit très rapidement la file open
temps bcp trop long : programme interrompu
Le speed varie beaucoup avec le nombre de sous problèmes. Cf mesures multiples notamment 404.wcsp
%CPU = (temps cumulé des processus + temps system i.e. process en mode noyau)/ elapse time –
donne une information sur l'utilisation des ressources CPU. Ici Majoré par 2400 %
pour les problèmes cap, le temps de preprocessing est très lent d'où l'absence de gain.
pour pedigree7, le problème vient peut être d'un mauvais majorant ?
```

FIGURE 7.13 : Benchmarks sur architecture 24 cœurs

## 7.7 Conclusion

Globalement, l’itération 1 (EPS) ne donne pas de résultats satisfaisants en pratique malgré des speedups intéressants dans certains cas. En effet, il apparaît que le nombre de sous problèmes générés impacte de façon importante sur le speedup en fonction de l’architecture : alors que les meilleures performances sur une machine 4/8 sont obtenues en générant environ 30 SP/cœur (nbProcess.txt = 240) avec option -j20, sur un serveur 24 cœurs, les expériences effectuées montrent de meilleurs speedup vers 6 SP/cœur (nbProcess.txt = 240) avec option -j24. Les processeurs sont aussi mieux exploités. En effet, les %CPU sont plus faibles avec 30 sp/c, avec 6 sp/c ils sont plus élevés ( 500% pour le 404). On reste loin des 2400 % théoriques sur une machine 24 cœurs mais à sa décharge un process matlab multithreadé était en cours d’exécution. La mesure reste cependant pertinente car le temps séquentiel est mesuré dans les mêmes conditions. Sur ce serveur, l’augmentation du nombre

max de jobs à lancer en parallèle fixés par l'option "-j " vs le nombre de cœurs limite les performances.

L'itération 2 MPI/Master-Worker pourrait s'avérer plus prometteuse en évitant les paramétrages de création de SP et surtout d'échanger l'incumbent solution dynamiquement.





## 8. Paradigme Master-Worker

### 8.0.1 Master-workers

- un processus maître distribue les sous problèmes aux "workers" i.e. aux cœurs.
- pas d'échanges entre eux suite à l'envoi des SP
- si un worker trouve une solution, il la retourne au master





## 9. Conclusion





# 10. Perspectives

## 10.0.1 Paradigme Supervisor-workers

- Le superviseur envoie le problème root au worker 1
- Le superviseur reçoit les solutions trouvées par les workers et les sous problèmes produits par ces derniers et son rangés dans une file.
- Les workers communiquent périodiquement avec le superviseur.
- le superviseur contrôle la taille de la file via un message indiquant aux workers qu'il est ou pas en mode collecte de sous problèmes.
- algorithme avec davantage de communications entre worker et superviseur
- Problème de goulot d'étranglement au niveau superviseur





## 11. Annexes

### 11.1 Annexe A : Génération a priori des sous problèmes

Le code ci-après constitue une implémentation en C++ qui, notamment, calcule les ensembles issus de produits cartésiens d'un ensemble de variables  $x_i$  de domaine  $D_i$ , utilisée dans la production de sous problèmes a priori.

```
/** /brief generate subproblems such that Ak-1 < q <= Ak
*/
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
#include <vector>
#include <cassert>
#include <algorithm>
using namespace std;

/** \brief function that read a wcsp file and return
a vector of long containing the domain size of variables

*/
vector<long> readDomains(const string & fic)
{
    ifstream file(fic,ios::in);
    string line;
    if(file)
    {
        getline(file, line);
        getline(file, line); //get second line with domains of vars
    }
    else
    {
        cout << "ERREUR: no file" << endl;
    }

    // Vector of string to save tokens
    vector <long> domain;

    // stringstream class check1
    stringstream check1(line);
    string token;

    // Tokenizing w.r.t. space ' '
    while(getline(check1, token, ' '))
        domain.push_back(stoi(token));

    return domain;
}

/** \brief function that compute the cardinal of cartesian product of the domains
between two indexes min and max, max excluded
*/
long cardinalProd(const vector<long> & dom, const long min, const long max)
{
    assert(min <= max);
    assert(max < (long) dom.size());
    long prod = 1;
    for(long i = min; i<max; i++)
        prod *= dom[i];

    return prod;
}

/** \brief compute the complexity of a sub problem(sp)
which is the cartesian product of all the variables
divided by the cardinal of cartesian prod of the assigned vars of the sp
*/
long subPbComplexity(const vector<long> & subDomain, const long globaleCpx )
{
    return globaleCpx / cardinalProd(subDomain, 0, (long) subDomain.size());
}
```

```

}

/** \brief Compute the number of vars k in assignment Ak such that
the provide number q is in (card(Ak-1), card(Ak)] */

long sizeOfAk(const vector<long> & dom, const int q)
{
    long k = 0;
    long AkMinusOne = dom[0];
    assert(q>=dom[0]);
    if(q==dom[0]) return 1;
    long Ak = AkMinusOne * dom[1];
    if(q>AkMinusOne && q<= Ak) return 2;
    for(long i=2 ; i< (long) dom.size(); i++)
    {
        AkMinusOne = Ak;
        Ak *= dom[i];
        // cout << "AkMinusOne = "<< AkMinusOne<<endl;
        // cout << "Ak = "<< Ak<<endl;

        if(q==AkMinusOne)
        {
            k=i;
            break;
        }
        if(q>AkMinusOne && q<= Ak)
        {
            k=i+1;
            break;
        }
    }
    //cout << "nb of subproblems to produce = " << cartesianProd(dom,0,k)<<endl;
    cout << "nb of subproblems generated = " << Ak<<endl;
    return k;
}

ostream& operator<<( ostream &flux, const vector<vector<long>> &vv)
{
    for(size_t i = 0; i < vv.size(); i++) {
        for (size_t j = 0; j < vv[i].size(); j++) {
            cout << vv[i][j] << " ";
        }
        cout << std::endl;
    }
    return flux;
}

ostream& operator<<( ostream &flux, const vector<long> &v)
{
    for(size_t i = 0; i < v.size(); i++)
        cout << v[i] << " ";
    cout << std::endl;
    return flux;
}

/** \brief compute the cartesian product of a vector of vectors
domains : set of domains dm represented by c++ vectors
*/
vector<vector<long>> cartProd (const vector<vector<long>>& domains)
{
    vector<vector<long>> vv = {{}}, tmp; // init vector of vectors
    for (const vector<long> & dm : domains) { // for each domain dm in the set of domains dm
        tmp= {};
        for (const vector<long> & v : vv) { // for each sub vector v in vv (v is {} at first)
            for (const long value : dm) { // for each value in domain dm

                tmp.push_back(v); // add domain v in tmp here v ={} at first
                // cout << "x ="<<x << endl;
                tmp.back().push_back(value); // add value to the last domain in tmp, here v={} tmp
                ={{0},{1}} ...
            }
        }
    }
}

```

```

        }
    }

    vv.swap(tmp); // vv become tmp and tmp become vv
}
//cout<< tmp << endl;
return vv;
}

vector<vector<long>> convertVV(vector<long> dom, int k)
{
    // convert vector dom in a vector of vectors
    vector<vector<long>> vv;
    vector<long> v;
    for (long i =0; i<k; i++ )
    {
        for ( long j =0; j<dom[i]; j++)
        {
            v.push_back(j);
        }
        vv.push_back(v);
        v.clear();
    }
    // cout<< vv << endl;
    return vv;
}

void writeJobs(vector<vector<long>> cp, const string & fic)
{
    ofstream file(fic);
    if(file) // if ok
    {
        int k;
        for(size_t i = 0; i < cp.size(); i++)
        {
            k=0;
            file << "-x=\\"";
            for (size_t j = 0; j < cp[i].size(); j++)
            {
                file << ","<< k<<"="<< cp[i][j] ;
                k++;
            }
            file << "\\"<<endl;
        }
    }
    else
    {
        cout << "File error "<< fic << endl;
    }
    file.close();
}

int main(int argc, char *argv[])
{
    if(argc !=4)
    {
        cout << "usage: "<< argv[0] << " " << "name of input wcsp file, space, name of output
subproblems file,space, approximate number of sub problem"<< endl;
        cout << "Example: "<< argv[0]<< " 404.wcsp job_file 250"<< endl;
        cout << "Parallel command : cat job_file | parallel -j+0 --eta -k ./toulbar2 404.wcsp
    }
}
```

```
{}"<<endl;
}

vector<long> dom = readDomains(argv[1]); // domain of vars

long k = sizeOfAk(dom, atoi(argv[3])); // nb of vars to try to match q variables
cout << "nb of corresponding pre-assigned variables = "<< k << endl;
vector<vector<long>> vv = convertVV(dom, k);

vector<vector<long>> cp = cartProd(vv);
writeJobs(cp, argv[2]);

return 0;
}
```

## **11.2 Annexe B : Utilisation de toulbar2 avec Eclipse**

On trouvera ci-dessous une proposition de mode opératoire dont l'objectif est d'obtenir rapidement un environnement de développement efficace.

## HowTo.use.toulbar2\_with\_eclipse.cpp

```
1# TUTORIAL VERSION 3 : juillet 2019
2# AUTHOR : KAD
3
4PREREQUISITES:
5sudo apt-get update
6
7 - Connecting to git without login :
8create an ssh key without pass phrase.
9cd ~/.ssh && ssh-keygen -t ed25519 -C "toto@free.fr"
10cat id_ed25519.pub, copy the output, use it to create a new ssh key on github. see
    Settings of your account.
11git clone git@github.com:toulbar2/toulbar2.git
12
13 - sudo apt install cmake cmake-data cmake-curses-gui libboost-all-dev libboost-graph-
    dev zlib1g-dev liblzma-dev libjemalloc-dev libgmp3-dev htop sudo libomp-dev openmpi*
14
15 - sudo apt install texstudio doxygen graphviz texlive-latex-recommended texlive-
    fonts-recommended
16
17 - Install last version Eclipse IDE for Scientific Computing which support mpi,
    openmpi; Parallel Tools Platform (PTP)
18 for instance via https://www.eclipse.org/downloads/packages/
19
20 1 - SHELL PHASE
21
22 In toulbar2 folder create a build folder.
23 cd build
24 Do cmake .. to configure the build ( c to configure, g to generate, q to quit)
25 nb : do sudo apt-get install libboost-all-dev to install boost_mpi
    boost_serialize ...
26
27 Do cmake .. to create the Makefile ( cmake .. has to be done each time new files are
    added to the project)
28 Do make -j8 where 8 is the number of cores on your computer
29
30 nb : re-execute cmake .. in build directory to update the Makefile
31 if you add files in the code of toulbar2
32
33 nb : cmake .. means execute cmake (cmake configure cmake tool which in turn
    produce a makefile)
34 with the CMakeLists.txt file in the parent directory.
35
36 nb : build folder can be named as you wish : for instance you can
37 create 2 folders : one folder named "debug" for the debug version,
38 the other named "release" for the release version.
39
40 IMPORTANT IF YOU CAN UPDATE GIT :
41 put the following lines and others in .gitignore to avoid polluting the repo with
    build folder and eclipse project files.
42 release/
43 debug/
44 *.cproject
45 *.project
46
47 prerequisites : two following "parallel" folders :
48 1 - toulbar2/src
49 2 - toulbar2/release build folder has to be created if it does not exist
50
51
```

## HowTo.use.toulbar2\_with\_eclipse.cpp

```
52 - cd build (where build designate debug or release folder or whatever)
53
54 - Configure cmake with ccmake tool: ccmake .. (where the 2 point designate the parent
      directory namely toulbar2)
55 select the option you need e.g. replace Release by Debug, etc ...
56 type c to configure cmake then g to generate conf files.
57
58 - Run cmake .. (cmake with only one c this time).
59 this will generate make file and other stuff.
60
61 - make -j8 to compile on an 8 cores machine which produce the exe file
62 toulbar2/debug/bin/Linux/toulbar2
63
64
65
66 2 - ECLIPSE PHASE
67
68 Eclipse does not seem to deal properly with existing cmake project: plugin are more
      or less obsolete, ...
69 so we import a makefile project to use make -j8 through eclipse.
70 Open eclipse and create a workspace named e.g. toulbar2_eclipse_workspace for
      instance in your home directory.
71 note that toulbar2_eclipse_workspace and toulbar2 are distinct folders. Eclipse uses
      the former one to save its metadata,
72 the latter is used by eclipse through .project and .cproject files to write the
      workbench parameters, etc
73 the workbench designate the actual "workspace" where you write your code.
74
75 - Create the toulbar2 eclipse project :
76 File -> Import -> C/C++ -> Existing code as Makefile project
77 nb : the version used here is eclipse scientific computing 2019-06 (eclipse cdt with
      parallel tools)
78
79 - enter a name for the project : toulbar2_debug for instance
80 - enter the path to toulbar2 folder
81 - choose linux GCC as a toolchain
82 - validate with finish button
83
84=> the project is created in eclipse
85
86 - select project folder on the upper left and go to menu Project -> properties -> C/
      C++ build
87 uncheck use default build command and add type make -j8 ( j = jobs : where 8 is
      your number of processors, more than 8 is also possible with only 8 cores)
88 this will trigger a faster parallel compilation. (alternatively, go in Behavior tab
      and choose enable parallel build)
89 add release to build directory i.e. ${workspace_loc:/toulbar2_release}/release
90 to tell eclipse where to build toulbar2.
91 clic Manage configurations > rename Default with release then create
92 a debug configuration if you have compiled and created a debug folder too.
93 then in eclipse Projet menu you will be able to set the default configuration.
94
95 toulbar2 executable will be created in release/bin/Linux directory.
96
97 To use the toulbar2 you just build, and not a possibly other version installed in
      your system,
98 go to release/bin/Linux directory and use ./toulbar2 file.wcsp put your wcsp files
      in this current folder.
99
```

## HowTo.use.toulbar2\_with\_eclipse.cpp

```
100
101- in C/C++ general -> code analysis check use project settings and uncheck all to
      avoid false warning and errors
102
103- in formatter choose a format for your code that you like. Normally toulbar2 uses
      K&R format style
104
105- to have the code formatted when saving the file : windows -> Preferences -> C/C++ -
      > editor -> Saves Actions -> select Format source Code -> format for all lines
106- mannually format the code with ctrl+shift+F
107
108- To change font in editor : Windows -> Preferences -> General -> Appearance ->
      Color and font -> C/C++ -> C/C++ editor text font -> button Edit
109
110
111- To execute toulbar2 from eclipse with arguments : Menu Run -> Run configuration ->
      C/C++ Application -> toulbar2 release default
112 add toulbar2/build/bin/Linux/toulbar2 the path to the program toulbar2
113 then in tab Argument add the argument to use with toulbar2 e.g. a wcsp file name and
      other toulbar2 options.
114 Choose your working folder e.g. ${workspace_loc:toulbar2}/release/bin/Linux so that
      to avoid to place wcsp files in toulbar2 folder.
115 You can also run a parallel run configuration.
116
117
118
119
120
121
```

### **11.3 Annexe C : Compilation sur le cluster genotoul**

On trouvera ci-dessous un fichier tcl (Tool Command Language) utilisé par l'utilitaire "module" et un fichier cmake modifié pour compiler toulbar2 sur le cluster genotoul. Pour compiler toulbar2, on peut utiliser le fichier \*.tcl pour configurer l'environnement puis suivre la "SHELL PHASE" du mode opératoire en annexe [11.2](#).

```
module load -f fichier.tcl
```

Le fichier cmake doit être renommé en CMakeList.txt et placé dans le dossier toulbar2. Il faut alors créer un répertoire de build : release dans le dossier toulbar2, aller dans release et lancer le configurateur cmake via la commande : ccmake .. ce qui permet à ccmake d'accéder au fichier CMakeList.txt situé dans le répertoire parent toulbar2.

```
#%Module1.0#####
# description : module to load environment on SLURM cluster
# kad version 1.0
# usage: module load -f path_to/my_own_module
# To unload one module: module unload bioinfo/bowtie2-2.2.9
# To unload all module and specific variable: module purge

#limit coredumpsizer 0
#module purge
module load compiler/gcc-7.2.0
module load mpi/openmpi-2.1.2
module load compiler/cmake-3.12.3
#module load compiler/intel-2018.0.128

setenv BOOST_ROOT /tools/libraries/Boost/boost_1_70_0_openmpi-2.1.2
setenv BOOST_INCLUDEDIR /tools/libraries/Boost/boost_1_70_0_openmpi-2.1.2/include
setenv BOOST_LIBRARYDIR /tools/libraries/Boost/boost_1_70_0_openmpi-2.1.2/lib

setenv MPI_INCLUDE_PATH /tools/cluster/mpi/openmpi/2.1.2/gcc-4.5.8/include
setenv JEMALLOC_ROOT /home/allouche/work/kad/jemalloc

prepend-path PATH /tools/cluster/mpi/openmpi/2.1.2/gcc-4.5.8/bin
prepend-path LD_LIBRARY_PATH /tools/libraries/Boost/boost_1_70_0_openmpi-2.1.2/lib
prepend-path LD_LIBRARY_PATH /home/allouche/work/kad/jemalloc/lib
```

```
# modif kad aout 2019 : pour compilation sous cluster
# avec boost mpi 1.70 et jemalloc recompilée en local
# jemalloc installée dans /home/allouche/work/kad/jemalloc
# pour compiler toulbar2 sur genologin faire :
# cd ....../toulbar2
# module purge
# module load -f ./toulbar2_module.tcl
# cd vers le répertoire de build désiré
# ccmake .. vérifier les options touche t pour avoir tout
# c pou config puis g pour generate
# cmake ..
# make -j8
# COPIER UN PB DANS BIN LINUX OU SE TROUVE le binaire toulbar2 compilé
# cd bin/Linux pour tester srun ./toulbar2 404.wcsp
# srun -N 3 --ntasks-per-node=32 -n 96 --time=00:04 --exclusive=user ./toulbar2
404.wcsp -para

#####
# cmake file for toulbar2 framework building , test and packaging
# version 0.9
# David allouche 17/10/10
#####

cmake_minimum_required(VERSION 2.6)
set (CMAKE_EXPORT_COMPILE_COMMANDS ON)
set (My_cmake_script "${CMAKE_CURRENT_SOURCE_DIR}/cmake-script") # location of
cmake script needed
set (My_Source src)      # source location
set (doc_destination "share/doc")           #path of doc installation
set (My_misc_source misc/src/)
list(APPEND CMAKE_MODULE_PATH "${My_cmake_script}/Modules")

set (MAINTAINER "David Allouche <david.allouche@inra.fr>") #used in packaging
set (CONTACT "https://github.com/toulbar2/toulbar2") #used in packaging

#####
# PROJECT NAME
#####

project("toulbar2")

MESSAGE(STATUS "#####")
MESSAGE(STATUS "project :${PROJECT_NAME} toolkit compilation cmake file version
0.99")
MESSAGE(STATUS "source:${CMAKE_CURRENT_SOURCE_DIR} ")
MESSAGE(STATUS "MAKE_BUILD_TYPE : ${CMAKE_BUILD_TYPE}")
MESSAGE(STATUS "#####")

include(CMakeDependentOption)

#####
# cmake option definition
#####

#      OPTION(TOULBAR2 "toulbar2 solver compilation [default: on]" ON)
#      OPTION(MENDELSOFT "mendelsoft compilation and packaging option
[default:off]" OFF)
OPTION(WIN32 "toulbar2 cross compilation flag [default: off]" OFF)
OPTION(ALL_APP "dedicated applications and toulbar2 solver compilation [default:
off]" OFF)
OPTION(MENDELSOFT_ONLY "mendelsoft compilation and packaging ONLY ==> remove other
exe and lib compilation [default:off]" OFF)
OPTION(TOULBAR2_ONLY "toulbar2 solver compilation ONLY [default: on]" ON)
OPTION(LIBTB2 "lib toulbar2 compilation [default: off]" OFF)
OPTION(ILOG "ilog solver binding [default: off]" OFF)
#      OPTION(LIBTB2INT " lib toulbar2 compilation INT mode required with ilog
```

```

and windows [default: off]" OFF)
OPTION(XML "add a reader for the (W)CSP xml input format [default: off]" OFF)
##      OPTION(CPLEX "encode global cost functions into a linear program solved by
CPLEX [default: off]" OFF)
OPTION(Boost "boost graph binding [default: on]" ON)
OPTION(MPI "MPI [default: off]" OFF)
OPTION(HBFS_MPI "HBFS parallelization using MPI [default: on]" ON)
OPTION(WIDE_STRING "use wide string to encode long domains in n-ary cost
functions/separators [default:on]" ON)
OPTION(LONG_COSTS "use long long to encode costs [default:on]" ON)
OPTION(LONG_PROBABILITY "use long double to encode probabilities [default:on]" ON)
OPTION(BUILD_API_DOC "build and install HTML documentation with doxygen
[default:off]" OFF)
OPTION(BUILD_API_DOC_LATEX "build and install LaTeX PDF documentation with doxygen
[default:off]" OFF)
OPTION(STATIC "static compilation flag [default: off]" OFF)
OPTION(verbose "verbose mode [default:on]" ON)
OPTION(COVER_TEST "cover test [default:on]" ON)
OPTION(BENCH "benchmarking [default:off]" OFF)
OPTION(WITH_MEM_JEMALLOC "Enable malloc replacement (http://www.canonware.com/jemalloc) [default on]" ON)
mark_as_advanced(WITH_MEM_JEMALLOC BUILD_API_DOC_LATEX)

#####
#Default profile for compilation
#####
IF(NOT CMAKE_BUILD_TYPE)
    SET(CMAKE_BUILD_TYPE Release CACHE STRING
        "Choose the build type, options are: None Debug Release RelWithDebInfo
MinSizeRel."
        FORCE)
ENDIF(NOT CMAKE_BUILD_TYPE)

SET(CMAKE_CXX_FLAGS "-Wall -std=c++17" )

IF(CPLEX)
    SET(CPLEX_LOCATION "/opt/ibm/ILOG/CPLEX_Studio126")
    SET(CPLEXFLAGS "-fPIC -fexceptions -fno-strict-aliasing -DIL0GCPLEX -DIL_STD -
DIL0STRICTPOD -pthread -I${CPLEX_LOCATION}/cplex/include -I${CPLEX_LOCATION}/
concert/include")
    SET(CPLEXLIB "-L${CPLEX_LOCATION}/cplex/lib/x86-64_linux/static_pic -L${
CPLEX_LOCATION}/concert/lib/x86-64_linux/static_pic")
    SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${CPLEXFLAGS}")
    SET(CMAKE_EXE_LINKER_FLAGS " ${CPLEXLIB}")
    SET(all_depends ${all_depends} "ilocplex")
    SET(all_depends ${all_depends} "cplex")
    SET(all_depends ${all_depends} "concert")
    SET(all_depends ${all_depends} "m")
    SET(all_depends ${all_depends} "pthread")
    SET(STATIC off)
ENDIF(CPLEX)

#####
# Compile Options on recent 64 bytes Macs.
# thanks to alex rudnick
#####
IF(APPLE)
    SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -m64" )
    SET(CMAKE_SHARED_LINKER_FLAGS "-Wall -m64" )
    SET(STATIC off)
ENDIF(APPLE)

#####
# default OPTION for toolbar2 test phase
#####
SET(FOPT "test-opt.cmake" CACHE string "filename containing local options used for

```

```
validation")
SET(Default_test_option CACHE STRING "Define toulbar2 option used in command line
for testing: cf toulbar2 command line")
SET(Default_test_timeout 300 CACHE INTEGER "default test timeout")
SET(Default_Nb_cpu 4 CACHE INTEGER "default ncpus number for mpirun ")
SET(Default_validation_dir "validation" CACHE string "default location of
validation files")
SET(Default_cover_dir "cover" CACHE string "default location of cover test files")
SET(Default_regex "end." CACHE string "default regex searched in test output")

SET(Default_BenchDir "benchmarks" CACHE string "default location of benchmark
files")
SET(Default_BenchFormat "wcsp" CACHE string "default format extension for
benchmark files search")
SET(Default_bench_timeout 30 CACHE INTEGER "default timeout used for benchmarking
")
SET(Default_bench_option "TOULBAR2_OPTION" CACHE STRING " option used in command
line for benchmarking.")
SET(Default_bench_regex "test ok" CACHE string "default regex searched in bench
output")

MARK_AS_ADVANCED(FORCE LIBTB2 verbose Boost FOPT CMAKE_INSTALL_PREFIX
Default_test_timeout WIDE_STRING LONG_PROBABILITY LONG_COSTS)
MARK_AS_ADVANCED(FORCE BUILD_SHARED_LIBS GMP_LIBRARY)

IF (MPI)
    SET(Boost ON)
    SET(STATIC OFF)
ENDIF(MPI)

#####
# static building
#####

IF(STATIC)
    MESSAGE(STATUS "STATIC COMPILATION ON (warning: jemalloc, xml, and MPI options
not compliant with static link)")
    SET(WITH_MEM_JEMALLOC OFF)
    SET(XML OFF)
    SET(MPI OFF)
    SET(BUILD_SHARED_LIBS OFF)
    set(CMAKE_FIND_LIBRARY_SUFFIXES ".a")
    set(CMAKE_EXE_LINKER_FLAGS "-static -static-libgcc -static-libstdc++")
ELSE()
    SET(BUILD_SHARED_LIBS ON) #shared library building
ENDIF(STATIC)

if(verbose)
    set (CMAKE_VERBOSE_MAKEFILE ON)
endif(verbose)

#####
# OPTION DEPENDANCES
#####
SET(Toulbar_NAME "toulbar2") # default project name

IF(ALL_APP)
    MESSAGE(STATUS "#####")
    MESSAGE(STATUS " TOULBAR2 and MENDELSOFT COMPILEMENT AND PACKAGING ")
    MESSAGE(STATUS "#####")
    SET(TOULBAR2_ONLY OFF)
    SET(MENDELSOFT_ONLY OFF)
    SET(MENDELSOFT ON)
    SET(TOULBAR2 ON)
    SET(ILOG OFF)
```

```
SET(LIBTB2 ON)
SET(XML ON)
SET(CPLEX OFF)
SET(boost ON)
set (Toulbar_NAME "toulbar2-all")
project(${Toulbar_NAME})
#description used in the package building
    SET(Toulbar_PACKAGE_DESCRIPTION "${Toulbar_NAME} is an open source C++
exact solver and library for graphical model optimization. It can solve MAP/Markov
Random Fields or Cost Function Networks/Weighted CSP.")
        SET(Toulbar_PACKAGE_SUMMARY "${Toulbar_NAME} exact solver for graphical
models.")

ENDIF(ALL_APP)

IF(MENDELSPORT_ONLY)
MESSAGE(STATUS "#####
MESSAGE(STATUS " MENDELSPORT COMPILATION AND PACKAGING ONLY")
MESSAGE(STATUS "#####

SET(MENDELSPORT ON)
SET(TOULBAR2 OFF)
SET(TOULBAR2_ONLY OFF)
SET(ILOG OFF)
SET(LIBTB2 OFF)
SET(XML OFF)
SET(CPLEX OFF)
SET(Boost OFF)
SET(MPI OFF)

#basename for packaging and versioning
set (Toulbar_NAME "mendelsoft")
#description used in the package building
SET (Toulbar_PACKAGE_DESCRIPTION
    "MendelSoft is an open source software which detects marker genotyping
incompatibilities (Mendelian errors only) in complex pedigrees using weighted
constraint satisfaction techniques. The input of the software is a pedigree data
with genotyping data at a single locus. The output of the software is a list of
individuals for which the removal of their genotyping data restores consistency.
This list is of minimum size when the program ends.")
    SET(Toulbar_PACKAGE_SUMMARY "${Toulbar_NAME} is an open source software which
detect Mendelian errors in complex pedigrees using weighted constraint
satisfaction techniques")

ENDIF(MENDELSPORT_ONLY)

IF(TOULBAR2_ONLY)
MESSAGE(STATUS "#####
MESSAGE(STATUS " TOULBAR2 Solver Compilation and Packaging Only")
MESSAGE(STATUS "#####

SET(MENDELSPORT OFF)
SET(TOULBAR2 ON)
SET(ILOG OFF)
SET(LIBTB2 OFF)
#basename for packagin and versionning
SET(Toulbar_NAME "toulbar2")
#description used in the package building
    SET(Toulbar_PACKAGE_DESCRIPTION "${Toulbar_NAME} is an open source C++
exact solver and library for graphical model optimization. It can solve MAP/Markov
Random Fields or Cost Function Networks/Weighted CSP.")
        SET(Toulbar_PACKAGE_SUMMARY "${Toulbar_NAME} exact solver for graphical
models.")
ENDIF(TOULBAR2_ONLY)
```

```
#####
IF(ILOG)
  set(LIBTB2INT ON)
ENDIF(ILOG)

SET(EXECUTABLE_OUTPUT_PATH bin/${CMAKE_SYSTEM_NAME})
SET(LIBRARY_OUTPUT_PATH lib/${CMAKE_SYSTEM_NAME})

#####
IF(WIN32)
  # option used for cross compilation
  set( EXE ".exe")
  set(COST INT_COST)
  set(LIBTB2INT OFF)
  set(XML OFF)
  set(CPLEX OFF)
  set(Boost OFF)
  set(MPI OFF)
  set(ILOG OFF)
  set(LIBTB2 OFF)

  MESSAGE STATUS "WIN32 on ."
  MESSAGE STATUS "COST ==> int."
  
# mingW32 env setup
include(${My_cmake_script}/mingw32-config.cmake)

ELSE(WIN32)

  SET(COST LONGLONG_COST)
  MESSAGE STATUS "COST ==> long long"

ENDIF(WIN32)

#####
# find opt libs
#####
#if(WITH_MEM_JEMALLOC)
#  find_package(Jemalloc)
#  if(JEMALLOC_FOUND)
#    link_directories(${JEMALLOC_LIBPATH})
#    SET(all_depends ${all_depends} "jemalloc")
#  endif()
#endif()

if(WITH_MEM_JEMALLOC)
  set( JEMALLOC_FOUND 0 )

if ( UNIX )
  FIND_PATH( JEMALLOC_INCLUDE_DIR
    NAMES
      jemalloc/jemalloc.h
    PATHS
      /home/allouche/work/kad/jemalloc/include
      $ENV{JEMALLOC_ROOT}
      $ENV{JEMALLOC_ROOT}/include
      ${CMAKE_SOURCE_DIR}/externals/jemalloc
  DOC
    "Specify include-directories that might contain jemalloc.h here."
  )
  FIND_LIBRARY( JEMALLOC_LIBRARY
    NAMES
      jemalloc libjemalloc JEMALLOC
    PATHS
      /home/allouche/work/kad/jemalloc/lib
```

```
$ENV{JEMALLOC_ROOT}/lib
$ENV{JEMALLOC_ROOT}
DOC "Specify library-locations that might contain the jemalloc library here."
)

if ( JEMALLOC_LIBRARY )
  if ( JEMALLOC_INCLUDE_DIR )
    set( JEMALLOC_FOUND 1 )
    message( STATUS "Found JEMALLOC library: ${JEMALLOC_LIBRARY}")
    message( STATUS "Found JEMALLOC headers: ${JEMALLOC_INCLUDE_DIR}")
  else ( JEMALLOC_INCLUDE_DIR )
    message(FATAL_ERROR "Could not find jemalloc headers! Please install
jemalloc libraries and headers")
  endif ( JEMALLOC_INCLUDE_DIR )
endif ( JEMALLOC_LIBRARY )

mark_as_advanced( JEMALLOC_FOUND JEMALLOC_LIBRARY JEMALLOC_EXTRA_LIBRARIES
JEMALLOC_INCLUDE_DIR )
endif (UNIX)

endif()

MESSAGE(STATUS "search for GMP library")
INCLUDE(FindPkgConfig)
include(${My_cmake_script}/FindGmp.cmake)
SET (all_depends ${all_depends} "gmp")
INCLUDE_DIRECTORIES(${GMP_INCLUDE_DIR})

#CMAKE_DEPENDENT_OPTION(ILOG "ILOGLUE COMPILED" OFF "LIBTB2INT" OFF)
#####
INCLUDE(FindPkgConfig)
INCLUDE(FindGit)

# list of files used for compilation are included in source_file.cmake
# new file need to be added to this list
# you can also define your own list and add it to the wall list
# for example: SET (source_files ${source_files} ${my_file_2add})

include(${My_cmake_script}/source_files.cmake)

#####
IF(Boost)
  include(${My_cmake_script}/Boost_option.cmake)
  link_directories(${Boost_LIBRARY_DIR})
  INCLUDE_DIRECTORIES(${Boost_INCLUDE_DIR})
ENDIF(Boost)

#####
IF(MPI)
#  SET(CMAKE_C_COMPILER mpicc)
#  SET(CMAKE_CXX_COMPILER mpicxx)
#  SET(CMAKE_CXX_COMPILER /usr/mpc/gcc/openmpi-1.10.3rc4/bin/mpicxx)
#  SET(CMAKE_SYSTEM_PREFIX_PATH /usr/mpc/gcc/openmpi-1.10.3rc4)
  find_package(MPI REQUIRED)
  include_directories(${MPI_INCLUDE_PATH})
  SET (mpiflag OPENMPI)
ENDIF(MPI)

#####
# FLAG XML ON ==> xmlcps supported
#####
include(${My_cmake_script}/xmlcsp.cmake)

#####
# build executable
#####
```

```
INCLUDE_DIRECTORIES ( ${CMAKE_CURRENT_SOURCE_DIR}/${My_Source} )

IF(TOULBAR2)
    add_executable(toulbar2 ${EXE} ${source_files})
    ADD_CUSTOM_TARGET(gen_version ALL /bin/sh ${CMAKE_CURRENT_SOURCE_DIR}/cmake-script/genVersionFile.sh)
    SET_SOURCE_FILES_PROPERTIES(ToulbarVersion.hpp PROPERTIES GENERATED 1)
    ADD_DEPENDENCIES(toulbar2 ${EXE} gen_version)
    IF(MPI)
        TARGET_LINK_LIBRARIES(toulbar2 ${EXE} ${all_depends} ${MPI_LIBRARIES})
        IF(MPI_COMPILE_FLAGS)
            set_target_properties(toulbar2 ${EXE} PROPERTIES COMPILE_FLAGS "${MPI_COMPILE_FLAGS}")
        ENDIF(MPI_COMPILE_FLAGS)
        IF(MPI_LINK_FLAGS)
            set_target_properties(toulbar2 ${EXE} PROPERTIES LINK_FLAGS "${MPI_LINK_FLAGS}")
        ENDIF(MPI_LINK_FLAGS)
    ELSE(MPI)
        TARGET_LINK_LIBRARIES(toulbar2 ${EXE} ${all_depends})
    ENDIF(MPI)

    IF(HBFS_MPI)
        SET(CMAKE_CXX_COMPILER mpic++)
    ENDIF(HBFS_MPI)

# BOOST
#   set(BOOST_LIBRARYDIR /usr/lib64/mpich/lib) # to test boost mpich on fedora
#   set(BOOST_LIBRARYDIR /usr/lib/x86_64-linux-gnu) # boost openmpi on ubuntu
#   set(BOOST_LIBRARYDIR /tools/libraries/Boost/boost_1_70_0_openmpi-2.1.2/lib)
#   include_directories(/tools/libraries/Boost/boost_1_70_0_openmpi-2.1.2/include)
# -I/tools/libraries/Boost/boost_1_70_0_openmpi-2.1.2/include
# find_package(Boost 1.70.0 REQUIRED) # Fail with error if Boost is not found
# find_package(Boost 1.70.0 REQUIRED mpi serialization)
# include_directories(${Boost_INCLUDE_DIR})
# set(Boost_NO_SYSTEM_PATHS ON) #Set to ON to disable searching in locations not specified by these hint variables. Default is OFF.

# MPI
find_package(MPI 2.1.2 REQUIRED)
include_directories(${MPI_INCLUDE_PATH})
set(MPI_CXX_LIBRARIES /tools/cluster/mpi/openmpi/2.1.2/gcc-4.8.5/lib)
set(MPI_CXX_INCLUDE_PATH /tools/cluster/mpi/openmpi/2.1.2/gcc-4.8.5/include)
include_directories(${MPI_INCLUDE_PATH})

set(MPI_CXX_HEADER_DIR /tools/cluster/mpi/openmpi/2.1.2/gcc-4.8.5/include)

    TARGET_LINK_LIBRARIES(toulbar2 ${EXE} ${all_depends} ${MPI_LIBRARIES} ${Boost_LIBRARIES}
"/tools/libraries/Boost/boost_1_70_0_openmpi-2.1.2/lib/libboost_serialization.so"
"/tools/libraries/Boost/boost_1_70_0_openmpi-2.1.2/lib/libboost_mpi.so" "/tools/libraries/Boost/boost_1_70_0_openmpi-2.1.2/lib/libboost_iostreams.so" "/home/allouche/work/kad/jemalloc/lib/libjemalloc.so")
set(MPI_COMPILE_FLAGS "-I/tools/libraries/Boost/boost_1_70_0_openmpi-2.1.2/include")
    IF(MPI_COMPILE_FLAGS)
        set_target_properties(toulbar2 ${EXE} PROPERTIES COMPILE_FLAGS "${MPI_COMPILE_FLAGS}")
    ENDIF(MPI_COMPILE_FLAGS)
    IF(MPI_LINK_FLAGS)
        set_target_properties(toulbar2 ${EXE} PROPERTIES LINK_FLAGS "${MPI_LINK_FLAGS}")
    ENDIF(MPI_LINK_FLAGS)
    ELSE(HBFS_MPI)
        TARGET_LINK_LIBRARIES(toulbar2 ${EXE} ${all_depends})
    ENDIF(HBFS_MPI)
```

```
    INSTALL( TARGETS toulbar2${EXE} DESTINATION bin)
ENDIF(TOULBAR2)

#####
# mendelsoft compilation
#####
IF(MENDELSPORT)
    add_executable(mendelsoft${EXE} ${source_files})
    ADD_CUSTOM_TARGET(gen_version ALL /bin/sh ${CMAKE_CURRENT_SOURCE_DIR}/cmake-
script/genVersionFile.sh)
    SET_SOURCE_FILES_PROPERTIES(ToulbarVersion.hpp PROPERTIES GENERATED 1)
    TARGET_LINK_LIBRARIES(mendelsoft${EXE} ${all_depends})
    install( TARGETS mendelsoft${EXE} DESTINATION bin)
ENDIF(MENDELSPORT)

# we must delete INT_COST
#####
# LIBTB2INTCOST GENERATION
#####
IF(LIBTB2)
    IF(WIN32)
        MESSAGE(STATUS "LIBTB2 not compliant with win32 in cost=LONG / cost type is
now int ....")
    ENDIF(WIN32)
    # INCLUDE_DIRECTORIES ( ${CMAKE_CURRENT_SOURCE_DIR}/${My_Source} )
    ADD_CUSTOM_TARGET(gen_version ALL /bin/sh ${CMAKE_CURRENT_SOURCE_DIR}/cmake-
script/genVersionFile.sh)
    SET_SOURCE_FILES_PROPERTIES(ToulbarVersion.hpp PROPERTIES GENERATED 1)

    LINK_DIRECTORIES(${CMAKE_CURRENT_SOURCE_DIR})

    add_library(
        tb2
        ${LIBTB2FILE}
    )
    target_link_libraries(tb2 gmp)

    INSTALL(TARGETS tb2
        RUNTIME DESTINATION bin
        LIBRARY DESTINATION lib
        ARCHIVE DESTINATION lib
    )
ENDIF(LIBTB2)

#####
# LIBTB2INTCOST GENERATION
#####
IF(LIBTB2INT)
    MESSAGE(STATUS "COMPILING LIBTB2 INT ....")
    # INCLUDE_DIRECTORIES ( ${CMAKE_CURRENT_SOURCE_DIR}/${My_Source} )
    LINK_DIRECTORIES(${CMAKE_CURRENT_SOURCE_DIR})
    ADD_CUSTOM_TARGET(gen_version ALL /bin/sh ${CMAKE_CURRENT_SOURCE_DIR}/cmake-
script/genVersionFile.sh)
    SET_SOURCE_FILES_PROPERTIES(ToulbarVersion.hpp PROPERTIES GENERATED 1)
    add_library(
        tb2int
        ${LIBTB2FILE}
    )

    INSTALL(TARGETS tb2int
        RUNTIME DESTINATION bin
        LIBRARY DESTINATION lib
        ARCHIVE DESTINATION lib
    )
ENDIF(LIBTB2INT)
```

```
ENDIF(LIBTB2INT)

#####
# Ilog Solver 6.0
#####
include(${My_cmake_script}/iloglue.cmake)

#####
# define option
#####
include(${My_cmake_script}/define_option.cmake)
MESSAGE(STATUS "##### define_option passed#####")

#####
# toulbar2test
#####
include(${My_cmake_script}/toulbar2test.cmake)

#####
# unit tests
#####
IF(COVER_TEST)
include(${My_cmake_script}/test.cmake)

MESSAGE(STATUS "##### test.cmake ==> tests script generated")
MESSAGE(STATUS "##### CTEST : toulbar2 default option = ${TOPT} (-DTOPT= ...to
change it) #####")

#####
# Cover tests
#####
include(${My_cmake_script}/cover-test.cmake)
ENDIF(COVER_TEST)

IF(BENCH)
include(${My_cmake_script}/test_bench.cmake)
include(${My_cmake_script}/add_make_command.cmake)

configure_file(${CMAKE_CURRENT_SOURCE_DIR}/misc/script/MatchRegexp.txt
${CMAKE_CURRENT_BINARY_DIR}/MatchRegexp.txt COPYONLY)

configure_file(${CMAKE_CURRENT_SOURCE_DIR}/misc/script/run_test.pl
${EXECUTABLE_OUTPUT_PATH}/run_test.pl COPYONLY)

configure_file(${CMAKE_CURRENT_SOURCE_DIR}/misc/script/make_report.pl
${CMAKE_CURRENT_BINARY_DIR}/make_report.pl COPYONLY)

configure_file(${CMAKE_CURRENT_SOURCE_DIR}/misc/script/exp_opt.pl
${CMAKE_CURRENT_BINARY_DIR}/exp_opt.pl COPYONLY)

ENDIF (BENCH)

#####
# Doc generation
#####
IF (BUILD_API_DOC)
include(${My_cmake_script}/UseDoxygen.cmake)
ENDIF (BUILD_API_DOC)

ADD_CUSTOM_TARGET(man ALL)

ADD_CUSTOM_COMMAND(
TARGET man
DEPENDS ${CMAKE_CURRENT_SOURCE_DIR}/man/toulbar2.1
COMMAND cp ${CMAKE_CURRENT_SOURCE_DIR}/man/toulbar2.1 $
```

```
{CMAKE_CURRENT_BINARY_DIR}/toulbar2.1
    OUTPUTS ${CMAKE_CURRENT_BINARY_DIR}/toulbar2.1
)

#####
# doc install
#####
install(DIRECTORY ${My_doc} DESTINATION ${doc_destination}/${Toulbar_NAME} PATTERN
".svn" EXCLUDE)
install(FILES ${CMAKE_CURRENT_BINARY_DIR}/toulbar2.1 DESTINATION share/man/man1)

#####
# examples installation
#####
install(DIRECTORY ${Default_validation_dir} DESTINATION ${doc_destination}/${
{Toulbar_NAME}/examples/ PATTERN ".svn" EXCLUDE)
install(DIRECTORY ${Default_cover_dir} DESTINATION ${doc_destination}/${
{Toulbar_NAME}/examples/ PATTERN ".svn" EXCLUDE)

#####
# PACKAGING
#####
set(CPACK_PROJECT_CONFIG_FILE ${CMAKE_CURRENT_SOURCE_DIR}/src/MyCPackConf.cmake)
INCLUDE(InstallRequiredSystemLibraries)
include(${My_cmake_script}/package.cmake)

##### END
```





# Acronymes

**ABDFBB** Adaptive Bounded Depth First Search B&B : Le B&B avec parcours DFS est borné par le nombre Z de retours arrière ou backtracks autorisés ; une fois atteinte cette limite Z, le DFS s'interrompt . Il est adaptatif car cette borne Z varie selon une certaine heuristique.

**B&B** Branch and Bound ou Evaluation-Séparation : méthode générique très utilisée en recherche opérationnelle. Cf. par exemple <https://www.techno-science.net/definition/6348.html> ou <https://www.youtube.com/watch?v=E7hJXsyw0dA>.

**BFS** Breadth First Search.

**BNFS** Best Node First Search : L'acronyme BNFS est utilisé ici pour éviter l'ambiguïté avec BFS : <https://www.techiedelight.com>. *The defining characteristic of BNFS is that, unlike DFS or BFS (which blindly examines/expands a cell without knowing anything about it or its properties), best-first search uses an evaluation function (sometimes called a "heuristic") to determine which object is the most promising, and then examines this object. This "best first" behaviour is implemented with a Priority Queue.* Cf. <https://courses.cs.washington.edu/courses/cse326/03su/homework/hw3/bestfirstsearch.html>.

**BTD-HBFS** Backtracking with Tree Decomposition Hybrid Best-node First Search.

**CFN** Cost Function Network : Le problème est un hyper-graphe dont les nœuds sont les variables et les arêtes les contraintes.

**DFBB** Depth First Search with B&B.

**DFS** Depth First Search.

**EDAC** Existential Directional Arc Consistency.

**EPA** Etablissement Public à Caractère Administratif.

**EPS** Embarrassingly Parallel Search.

**EPST** Etablissement Public à Caractère Scientifique et Technologique.

**EPT** Equivalent Preserving Transformations.

**FOSS** Free Open Source Software.

**HBFS** Hybrid Best-First Search.

**HPC** High Performance Computing.

**IDE** Integrated Development Environment.

**ILP** Integer Linear Programming.

**INRA** Institut National de Recherche Agronomique.

**INRAE** Institut National de Recherche pour l’Agriculture, l’Alimentation et l’Environnement.

**IRSTEA** Institut National de Recherche en Sciences et Technologies pour l’Environnement et l’Agriculture.

**json** JavaScript Object Notation.

**MIA** département de Mathématiques et Informatique.

**MIAT** unité de Mathématiques et Informatique de Toulouse.

**MPI** Message Passing Interface.

**PFE** Projet de Fin d’Etudes.

**S&E** Séparation et Evaluation.

**SAB** équipe Statistiques et Algorithmique pour la Biologie.

**SLURM** Simple Linux Utility for Resource Management.

**VCSP** Valued Constraint Satisfaction Problem.

**WCSP** Weighted Constraint Satisfaction Problems.



## Glossaire

**Arité** Nombre de variables impliquées dans une fonction de coûts. L'ensemble de ces variables est appelé Support. L'arité est donc le cardinal du support de la contrainte.

**Cluster** Un cluster est une grappe d'ordinateurs, appelés nœuds en référence à la théorie des graphes. Relativement homogènes, ces derniers sont connectés par un réseau local, propriété d'une entité déterminée et unique qui en assure la gestion. Les clusters fournissent la partie calculatoire des services fournis par les data centers. [https://fr.wikipedia.org/wiki/Grappe\\_de\\_serveurs](https://fr.wikipedia.org/wiki/Grappe_de_serveurs). Chaque nœud se présente souvent sous la forme d'une carte mère montée dans des racks, eux-mêmes montés dans des baies, elles-mêmes installées dans un local climatisé et sécurisé. Mais on peut monter son propre cluster avec un ensemble d'ordinateurs, de consoles de jeux, etc. Il est aussi possible d'acheter des services de calcul dans les data centers des "clouds"<sup>[3]</sup> qui désignent une infrastructure également centralisée qui peut faire usage de clusters, de superordinateur, pour fournir divers niveaux de prestations.

**contrainte souple** synonyme de contrainte valuée et de fonction de coûts.

**contrainte souple d'arité nulle** notée  $w_\emptyset$ , elle correspond à une fonction de coûts constante, de support vide donc indépendante de toute variable du problème.

**Gap d'optimalité** Intervalle qui encadre la valeur optimale, noté dans ce rapport  $[clb, cub]$  ou  $[LB, UB]$ .

**genologin** Désigne la nouvelle tranche du cluster installée en 2017. C'est aussi le nom des frontaux genologin1 et genologin2. Il comprend 48 nœuds, du node101 au node148, 32 cœurs et 256 GB de RAM par nœud, interconnexion par InfiniBand 56GB/s. Workload manager SGE remplacé par SLURM. Total : 1584 cœur / 3168 threads / 51 TFlops.

**GenoToul** Plateforme bioinformatique faisant partie du réseau GenoToul qui met à disposition des ressources (calcul, stockage, banques de données, logiciels) et des compé-

tences pour accompagner les programmes de biologie et de bioinformatique aux plans régional, national et international. La capacité de calcul, en août 2019, est assurée par un cluster comprenant environ 3000 cœurs réels installé dans le data center, ou arche de données, du centre de recherche de l'INRA Toulouse. <http://bioinfo.genotoul.fr/>.

**grid computing** Les grilles de calcul, à la différence des clusters, constituent un assemblage d'ordinateurs hétérogènes connectés par un réseau parfois de portée mondiale, e.g. l'Internet. Les propriétaires/administrateurs sont multiples, non connus en général, le matériel et les logiciels sont hétérogènes.[https://fr.wikipedia.org/wiki/Grid\\_computing](https://fr.wikipedia.org/wiki/Grid_computing). Exemples de projets de calcul distribué par grille : détection d'intelligence extra-terrestre : <https://fr.wikipedia.org/wiki/SETI@home>.

**hypergraphe** graphe dont les nœuds peuvent être reliés par plus de 1 arrête, ou plus de 2 arcs s'il est orienté.

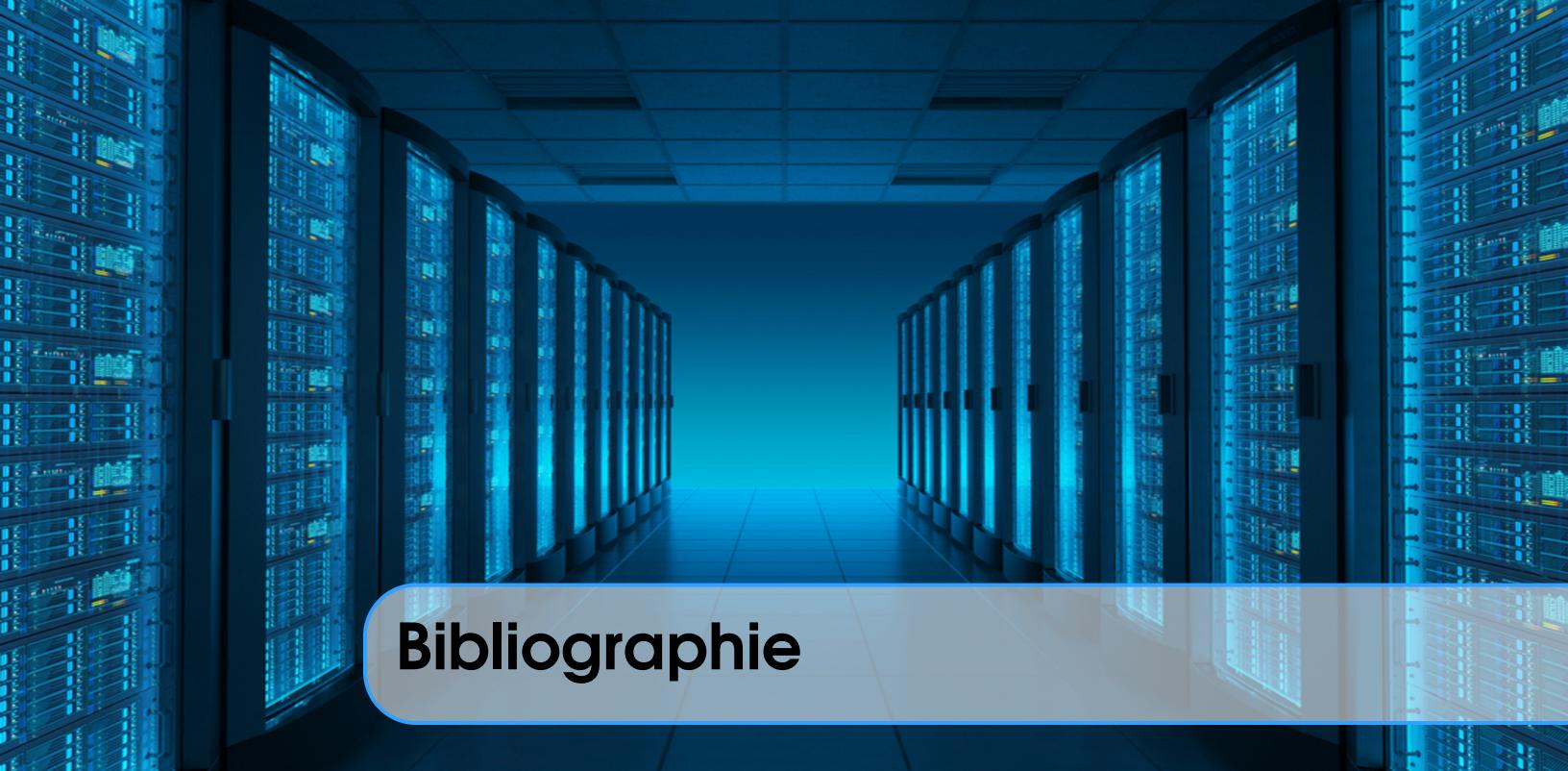
**InfiniBand** Standard de communication à haut débit de l'ordre d'une dizaine ou centaine de Gbit/s et de latence inférieure à la microseconde.

**Master-Worker** Paradigme de parallélisation qui utilise un processus master qui distribue le travail au workers et centralise les résultats obtenus par ces derniers..

**supercomputer** Bien qu'un cluster puisse être considéré comme un super ordinateur, la définition retenues ici concerne des systèmes qui sont davantage intégrés et non simplement constitués d'un ensemble de nœuds indépendants. Dans un super ordinateur, vu comme une unité d'exécution unique, l'intégration des processeurs permet des communications plus rapides entre processeurs et entre mémoire et processeurs. Probablement que les clusters sont plus adaptés à l'exécution de programmes se prêtant au paradigme *Embarrassingly Parallel* : bio-informatique, physique des particules et les super ordinateurs, aux traitements de tâches en parallèles dépendantes les unes des autres : mécanique des fluides, prévisions météorologiques. Les performances de HBFS seraient ainsi supérieures sur un super ordinateur. Exemple de supercomputer : IBM Blue Gene/Q baptisé MIRA dont une photo illustre l'en-tête de ce glossaire. By Courtesy Argonne National Laboratory, CC BY 2.0, [https://en.wikipedia.org/wiki/Mira\\_\(supercomputer\)](https://en.wikipedia.org/wiki/Mira_(supercomputer)).

**toulbar2** Contraction de Toulouse et Barcelone. Le 2 indique la version C++ de toulbar programmé en C. Solveur exact de divers problèmes d'optimisation combinatoire décrits par un réseau de fonctions de coût écrit en C++ : <http://www7.inra.fr/mia/T/toulbar2/>, <https://github.com/toulbar2/toulbar2>.

**échelonnabilité** Scalabilité ou Scalability : Capacité d'un algorithme à tirer partie d'une augmentation des ressources, principalement du nombre de cœurs. Nécessite d'éviter les goulots d'étranglement dans les communications.



# Bibliographie

## Articles

- [1] D. ALLOUCHE, S. De GIVRY et T. SCHIEX. “Towards Parallel Non Serial Dynamic Programming for Solving Hard Weighted CSP”. In : (2010) (cf. pages 19, 33).
- [2] David ALLOUCHE, Simon de GIVRY, George KATSIRELOS, Thomas SCHIEX et Matthias ZYTNICKI. “Anytime Hybrid Best-First Search with Tree Decomposition for Weighted CSP”. In : (2015). Sous la direction de Gilles PESANT, pages 12-29 (cf. pages 19, 42, 57).
- [3] Michael ARMBRUST, Armando FOX, Rean GRIFFITH, Anthony D. JOSEPH, Randy H. KATZ, Andrew KONWINSKI, Gunho LEE, David A. PATTERSON, Ariel RABKIN, Ion STOICA et Matei ZAHARIA. “Above the Clouds: A Berkeley View of Cloud Computing”. In : UCB/EECS-2009-28 (2009). URL : <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html> (cf. page 117).
- [5] Martin COOPER, Simon DE GIVRY, Marti SANCHEZ, Thomas SCHIEX, Matthias ZYTNICKI et Tomas WERNER. “Soft arc consistency revisited”. anglais. In : *Artificial Intelligence* 174 (2010), pages 449-478. URL : [http://www.irit.fr/publis/ADRIA/VAC\\_OSAC\\_final.pdf](http://www.irit.fr/publis/ADRIA/VAC_OSAC_final.pdf) (cf. page 50).
- [7] Ian P. GENT, Chris JEFFERSON, Ian MIGUEL, Neil C. A. MOORE, Peter NIGHTINGALE, Patrick PROSSER et Chris UNSWORTH. “A Preliminary Review of Literature on Parallel Constraint Solving”. In : () .
- [8] Simon de GIVRY, Federico HERAS, Matthias ZYTNICKI et Javier LARROSA. “Existential arc consistency: Getting closer to full arc consistency in weighted CSPs”. In : (2005) (cf. pages 42, 50, 57).
- [9] Simon de GIVRY, Thomas SCHIEX et Gérard VERFAILLIE. “Exploiting Tree Decomposition and Soft Local Consistency In Weighted CSP”. In : (2006).

- [11] Philippe JÉGOU, Hélène KANSO et Cyril TERRIOUX. “Adaptive and Opportunistic Exploitation of Tree-Decompositions for Weighted CSPs”. In : (nov. 2017). DOI : [10.1109/ICTAI.2017.00064](https://doi.org/10.1109/ICTAI.2017.00064) (cf. page 19).
- [12] Akihiro KISHIMOTO, Radu MARINESCU et Adi BOTEA. “Parallel Recursive Best-first AND/OR Search for Exact MAP Inference in Graphical Models”. In : NIPS’15 (2015), pages 928-936. URL : <http://dl.acm.org/citation.cfm?id=2969239.2969343> (cf. pages 19, 33, 34).
- [13] Javier LARROSA, Emma ROLLON et Rina DECHTER. “Limited Discrepancy AND/OR Search and Its Application to Optimization Tasks in Graphical Models”. In : IJCAI’16 (2016), pages 617-623. URL : <http://dl.acm.org/citation.cfm?id=3060621.3060708> (cf. page 34).
- [14] Bernard MANS, Thierry MAUTOR et Catherine ROUCAIROL. “A parallel depth first search branch and bound algorithm for the quadratic assignment problem”. In : *European Journal of Operational Research* 81.3 (mar. 1995), pages 617-628. URL : <https://ideas.repec.org/a/eee/ejores/v81y1995i3p617-628.html> (cf. pages 19, 33).
- [16] Tarek MENOUER, Bertrand LECUN et P VANDER-SWALMEN. “Parallélisation d’un solveur de contraintes avec le framework parallèle BOBPP”. In : (jan. 2013).
- [17] Abdelkader OUALI, David ALLOUCHE, Simon DE GIVRY, Samir LOUDNI, Yahia LEBBAH, Francisco ECKHARDT et Lakhdar LOUKIL. “Iterative Decomposition Guided Variable Neighborhood Search for Graphical Model Energy Minimization”. In : (août 2017). URL : <https://hal.archives-ouvertes.fr/hal-01628162> (cf. pages 19, 33, 34).
- [20] Jean-Charles RÉGIN, Mohamed REZGUI et Arnaud MALAPERT. “Embarrassingly Parallel Search”. In : (2013). Sous la direction de Christian SCHULTE, pages 596-610 (cf. pages 19, 33, 34, 70, 71).
- [21] Thomas SCHIEX, Helene FARGIER et Gerard VERFAILLIE. “Valued Constraint Satisfaction Problems: Hard and Easy Problems”. In : IJCAI’95 (1995), pages 631-637. URL : <http://dl.acm.org/citation.cfm?id=1625855.1625938> (cf. page 41).
- [22] “Weighted constraint satisfaction problem”. In : (2016). URL : [https://en.wikipedia.org/wiki/Weighted\\_constraint\\_satisfaction\\_problem](https://en.wikipedia.org/wiki/Weighted_constraint_satisfaction_problem) (cf. pages 41, 42).

## Books

- [4] Martin C. COOPER, Simon de GIVRY et Thomas SCHIEX. *Valued Constraint Satisfaction Problems*. Sous la direction de SPRINGER. Tome 2. Chapitre 7 (cf. pages 41, 44).

- [6] Youssef EDITORS: HAMADI et Lakhdar (Eds.) SAIS. *Handbook of Parallel Constraint Reasoning*. Springer, 2018 (cf. pages [19](#), [33](#)).
- [10] Ananth GRAMA, Anshul GUPTA, George KARYPIS et Vipin KUMAR. *Introduction to Parallel Computing*. Addison Wesley, 2003.
- [15] Timothy MATTSON, Beverly SANDERS et Berna MASSINGILL. *Patterns For Parallel Programming*. Addison-Wesley, 2004.
- [18] Peter PACHECO. *An Introduction to Parallel Programming*. Elsevier Science Technology, 2011.
- [19] Ted RALPHS, Yuji SHINANO, Timo BERTHOLD et Thorsten KOCH. *Parallel Solvers for Mixed Integer Linear Optimization*. Sous la direction d'Youssef HAMADI et Lakhdar SAIS. Springer International Publishing, 2018. Chapitre Parallel Solvers for Mixed Integer Linear Optimization, pages 283-336 (cf. pages [34](#), [61](#)).





## A

Arité d'une contrainte souple ..... 37

## C

cluster ..... 5

contrainte souple = fonction de coûts = contrainte  
valuée ..... 37

## G

GenoToul ..... 5

## H

hypergraphe ..... 38

## I

InfiniBand ..... 21

## S

SLURM ..... 21