





PROJET DE FIN D'ÉTUDES 2019 ECOLE NATIONALE DE L'AVIATION CIVILE, TOULOUSE,  
FRANCE. VERSION OF MONDAY 2<sup>ND</sup> SEPTEMBER, 2019 AT 01:59

VERSION OF MONDAY 2<sup>ND</sup> SEPTEMBER, 2019 AT 01:59

<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), 24 CHEMIN DE BORDE-ROUGE, CS 52627, 31326 AUZEVILLE-TOLOSANE CEDEX FRANCE.



## Preface

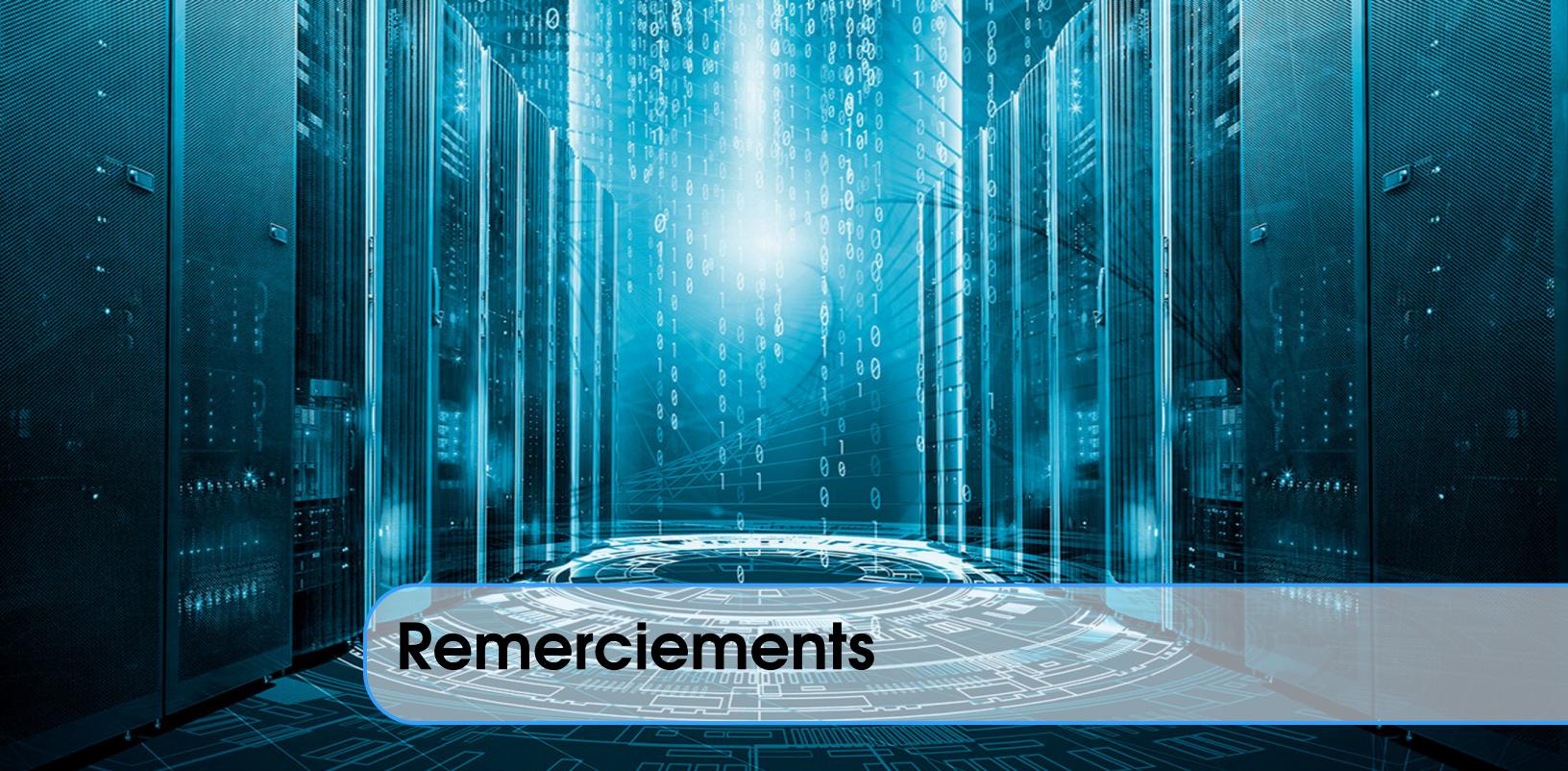
*"If you build it, they will come."*

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>Introduction</b>                               | <b>9</b>  |
| <b>1 Hybrid Best-First Search</b>                 | <b>11</b> |
| <b>1.1 Definitions</b>                            | <b>11</b> |
| 1.1.1 Depth-First Search .....                    | 11        |
| 1.1.2 Breadth-First Search .....                  | 11        |
| 1.1.3 Best-First Search .....                     | 12        |
| 1.1.4 Nœud ouvert .....                           | 12        |
| 1.1.5 Backtrack adaptatif .....                   | 12        |
| 1.1.6 Hybrid Best-First Search .....              | 13        |
| <b>2 Paradigme Master-Worker</b>                  | <b>17</b> |
| <b>2.1 Description du paradigme Master-Worker</b> | <b>17</b> |
| <b>2.2 Description de l'algorithme</b>            | <b>19</b> |
| 2.2.1 Algorithm Master .....                      | 19        |
| <b>2.3 Implémentation de l'algorithme</b>         | <b>19</b> |
| 2.3.1 Bibliothèques utilisées .....               | 19        |
| 2.3.2 Description synoptique .....                | 20        |
| 2.3.3 Algorithme Worker .....                     | 21        |
| <b>2.4 Premiers Résultats</b>                     | <b>21</b> |

|                     |    |
|---------------------|----|
| <b>Acronymes</b>    | 23 |
| <b>Glossaire</b>    | 25 |
| <b>Bibliography</b> | 29 |
| <b>Articles</b>     | 29 |
| <b>Books</b>        | 31 |
| <b>Index</b>        | 33 |



Bien que généralisées depuis les années 2000, les architectures parallèles présentent toujours un certain décalage avec les habitudes de programmation. Les algorithmes séquentiels ne bénéficient plus d'une amélioration automatique de leur performances avec l'augmentation de la fréquence des processeurs qui plafonne à quelques Giga Hertz (GHz). Le mur *thermique*, voire *quantique*, est passé par là. Le "*free lunch*" est terminé.

L'objectif du stage est d'étudier la parallélisation d'une méthode de recherche arborescente hybride développée par l'équipe **Statistiques et Algorithmique pour la Biologie (SAB)** de l'INRA Toulouse et baptisé **Hybrid Best-First Search (HBFS)**; HBFS est un des algorithmes utilisé dans un solveur C++ de réseaux de contraintes *souples* performant : **toulbar2**. HBFS, contrairement aux algorithmes implémentant des méta-heuristiques : *tabu search*, recuit simulé, génétiques, à population, etc., est de fournir, quand elle existe, une solution exacte avec en outre sa preuve d'optimalité. Il permet de résoudre des problèmes de satisfaction de contraintes souples(WCSP) qui généralisent les CSP.

Un état de l'art sur les wcsp et le parallélisme est d'abord présenté<sup>6</sup>. Puis, plusieurs stratégies de parallélisation sont considérées qui ont donné lieu à deux choix de parallélisation et d'implémentation : l'**Embarrassingly Parallel Search (EPS)** et le paradigme **Master-Worker**.

---

<sup>6</sup>Un résumé est rédigé en tête de chaque chapitre pour faciliter la lecture. En effet, ce rapport a aussi été rédigé pour éventuellement servir d'outil de prise en main de toulbar2 et des formalismes associés.

Enfin, Les résultats des tests sont présentés qui vont confirmer ou non l'intérêt des pistes de parallélisations choisies ...



# 1. Hybrid Best-First Search

## Abstract

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<sup>1</sup> fixé dynamiquement pour essayer de trouver le meilleur compromis entre *propagation* et *diversification*.

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.

## 1.1 Definitions

### 1.1.1 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).

### 1.1.2 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

---

<sup>1</sup>Un backtrack désigne un retour arrière dans l'arbre de recherche.

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.

### 1.1.3 Best-First Search

Désigne une recherche dans un graphe où les nœuds les plus prometteurs sont explorés en premier. Une heuristique en 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).

### 1.1.4 Nœud ouvert

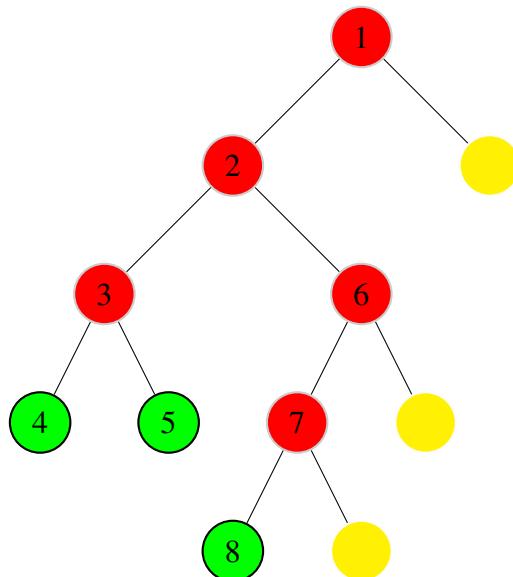


FIGURE 1.1 : Arbre binaire de décision partiellement exploré par un DFS avec une limite de backtrack,  $Z = 3$ . Les nœuds verts sont les feuilles de l'arbre donnant lieu à une affectation complète des variables. Les nœuds jaunes non cerclés sur les branches droites de l'arbre sont les nœuds ouverts 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.

### 1.1.5 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)

### 1.1.6 Hybrid Best-First Search

La recherche hybride en meilleur nœud d'abord (HBFS) combine deux approches : un **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 1.2 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  $clb$ . 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 globale 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  $k^2$  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 certain nombre de backtracks<sup>3</sup>, le DFBB est arrêté,

---

<sup>2</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>3</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>4</sup>. Ce sont les 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,
6. Si le DFBB parvient à une feuille de l'arbre de recherche<sup>5</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 nœuds 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  $lb \geq cub$  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>4</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>5</sup>La séquence de décisions  $\delta$  est telle quelle détermine une affectation complète.

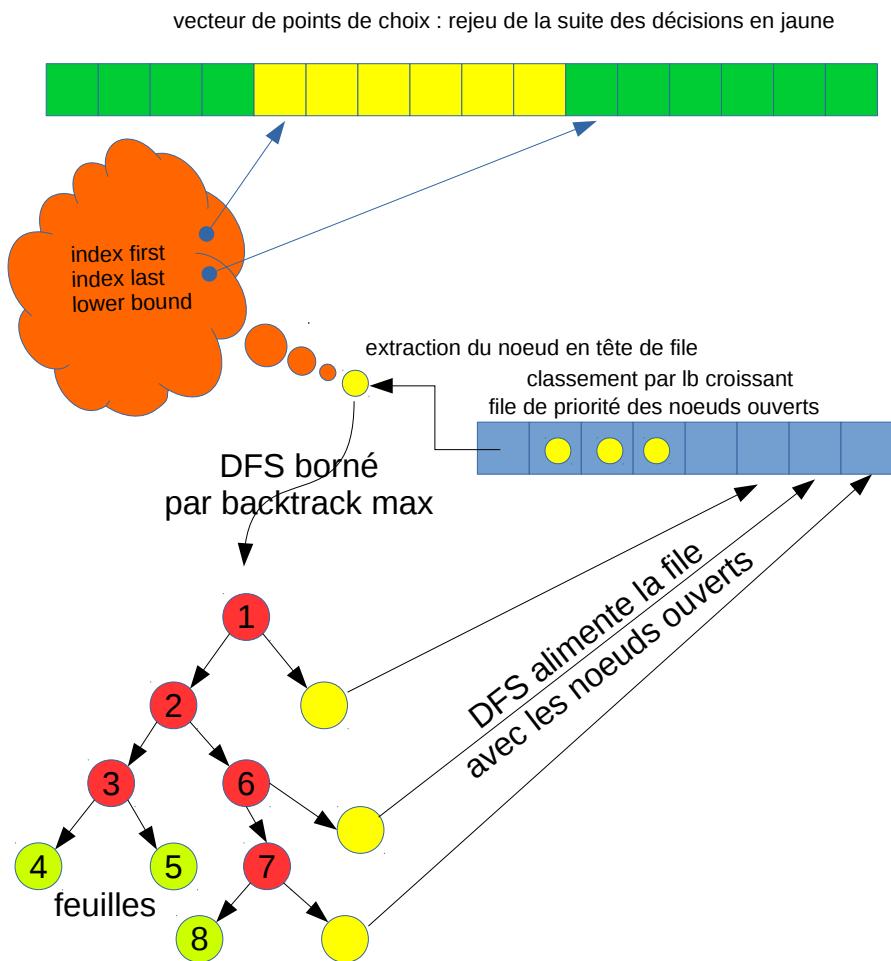


FIGURE 1.2 : Illustration de l'algorithme : un nœud est retiré de la file *open*, donné au DFS qui retourne des nœuds ouverts dans la file *open*, qui les classe par clb croissants. Un vecteur de décision, i.e. de points de choix, optimise l'espace mémoire en évitant les redondances. En effet, deux nœuds 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)  
2   /* initialisation du nœud racine */  
3   open := v( $\delta = \emptyset, lb = clb$ ) ;  
4   /* condition d'arrêt  $clb \geq cub$  ou open vide */  
5   while (open  $\neq \emptyset$  and clb  $< cub$ ) do  
6     v := pop(open) ;           /* on pop le nœud le plus "prometteur" */  
7     Restore state v. $\delta$ , leading to assignment  $A_v$ , maintaining local consistency ;  
8     NodesRecompute := NodesRecompute + v.depth ;  
9     cub := DFS( $A_v, cub, Z$ )/* puts all right open branches in open */ ;  
10    clb := max(clb, lb(open)) ;          /* calcul du clb global courant */  
11    /* heuristic Z adaptatif */  
12    if (NodesRecompute  $> 0$ ) then  
13      if (NodesRecompute/Nodes  $> \beta$  and  $Z \leq N$ ) then  $Z := 2 \times Z$ ;  
14      else if (NodesRecompute/Nodes  $< \alpha$  and  $Z \geq 2$ ) then  $Z := Z/2$ ;  
15    return (clb, cub);
```

---



## 2. Paradigme Master-Worker

### Abstract

#### 2.1 Description du paradigme Master-Worker

Le paradigme Master-Worker est très connu et largement utilisé dans la communauté du calcul haute performance (HPC). Dans son implémentation la plus simple, un seul processus master coordonne le travail d'un ensemble de processus workers. Les workers n'échangent qu'avec le master et les échanges avec le master se limitent à recevoir le travail et à restituer les résultats, rien entre les deux. Comme on l'a déjà vu, un worker ne *parle* pas en travaillant et le master ne le dérange pas lorsqu'il est occupé. Le rôle principal du master est d'assurer que les processeurs, utilisés par les workers, soient occupés la plus grande partie du temps. De part sa conception, le master assure donc le *load balancing*. Mais il peut être en charge d'autres opérations.

Cependant, comme toutes les communications passent par le master, on peut prévoir un phénomène de goulot d'étranglement du fait d'une augmentation linéaire des échanges avec le nombre de workers. Ceci a pour conséquence de limiter l'échelonnableté (*scalability*) du programme concrétisée par une *efficacité* décroissant avec le nombre de workers ; par construction le paradigme Master-Worker, implanté dans sa forme la plus simple, présentera un *speedup* par processeur décroissant. Cette limitation peut cependant être dépassée par exemple si le master demande aux workers d'envoyer directement le travail non terminé à ses *collègues*.

En outre, la granularité de niveau nœud du master-worker peut être modifiée dynamiquement pour passer à une granularité de niveau *sous-arbre* où le worker travaillera en toute indépendance tant que le master ne sera pas disponible. La centralisation de l'état de la

recherche permet aussi de conserver dans une certaine mesure l'ordre de la recherche du programme séquentiel. On retrouve la nécessité de trouver un compromis entre échanges d'informations qui vont permettre d'éviter aux workers de faire un travail qu'ils n'auraient pas fait en séquentiel et l'overhead de communication décrit au chapitre ?? : Stratégies de parallélisation.

Malgré ses défauts potentiels et la simplicité du son principe, cette approche a été utilisée avec succès pour résoudre des instances difficiles non encore résolues de problèmes MILP<sup>1</sup>, en utilisant le solveur CPLEX<sup>2</sup>, de la même manière que les sous-problèmes étaient générés par toulbar2 dans le chapitre précédent, pour générer les sous-problèmes qui sont ensuite traités par une grille de calcul [20].

---

<sup>1</sup>On appelle programme linéaire à variables mixtes, ou Mixed Integer Linear Programming, un programme linéaire contenant à la fois des variables continues donc dans  $\mathbb{R}$  et des variables discrètes donc dans  $\mathbb{Z}$ . Comme cette catégorie de problèmes généralise les programmes linéaires à variables entières, un MILP est un problème NP-difficile. Toutefois, dans de nombreux cas pratiques, ils sont faciles à résoudre à l'aide de solveurs entiers basés sur des B&B.

<sup>2</sup>CPLEX est un solveur commercialisé et maintenu par IBM. Son nom fait référence au langage C et à l'algorithme NP-hard du simplexe. Il est composé d'un exécutable et d'une bibliothèque de fonctions interfacées avec divers langages dont python, C, C++ et java : <https://www.ibm.com/analytics/cplex-optimizer>

## 2.2 Description de l'algorithme

### 2.2.1 Algorithm Master

**Algorithm 2:** Parallel Hybrid Best-First Search : Master

```
/* INITIALISATIONS */  
1 cp := Ø ; /* cp = δ vecteur de choice points. */  
2 open := v(cp,lb = clb) ; /* initialisation du nœud racine */  
3 idleQ := {1,2,...} ; /* Au début, tous les workers sont inactifs */  
4 activeWork := Ø ; /* map avec les rangs des workers actifs et le lb min */  
/* Tant que clb < cub et qu'il reste du travail à distribuer ou en cours */  
/* TRAITEMENTS */  
5 while (clb < cub and (open ≠ Ø or activeWork ≠ Ø)) do /* Tant qu'il reste du travail et des workers pour le faire */  
6   while (open ≠ Ø and idleQ ≠ Ø) do /* le master envoie un nœud, cub et la solution au worker */  
7     isend(node,ub,masterSol) ; /* isend() non bloquant */  
8     pop(idleQ) ; /* le worker devient actif */  
9     activeWork[worker] := lb ; /* mémorise le lb du worker */  
10    v := pop(open) ; /* on pop le nœud avec lb minimum */  
11    /* Le master attend le message d'un worker quelconque */  
12    recv(MPI_ANY_SOURCE,MPI_ANY_TAG,work2) ; /* recv() bloquant */  
13    /* A la réception d'une réponse, le master se remets au travail */  
14    cub := min(cub,cub(worker)) ; /* mets à jour le cub et la solution */  
15    cp := cpWorker ; /* maj avec le vecteur de décisions du worker */  
16    open := openWorker ; /* maj avec la file de priorité du worker */  
17    activeWork.erase(worker) ; /* efface la paire [worker,lb] */  
18    /* calculate the min of the lb among the active workers */  
19    minLbWorkers = min(activeWork(worker));  
20    idleQ.push(worker) ; /* le worker devient inactif */  
21    clb := max(clb,min(lb(open),minLbWorkers)) ; /* calcule le clb global */  
22    showGap(clb,cub) ; /* affiche le gap d'optimalité */  
23    print(UB) ; /* affiche la valeur optimale si elle existe */  
24    /* termine tous les processus workers */  
25  return (clb,cub) ; /* retourne le gap au hbfs parallel */
```

## 2.3 Implémentation de l'algorithme

### 2.3.1 Bibliothèques utilisées

L'implémentation du Master-Worker a été effectuée en utilisant le standard MPI via son implémentation openMPI et l'interface C++ Boost associée : [https://www.boost.org/doc/libs/1\\_71\\_0/doc/html/mpi.html](https://www.boost.org/doc/libs/1_71_0/doc/html/mpi.html).

### 2.3.2 Description synoptique

Les deux fichiers principaux ci-dessous sont accessibles sur GitHub :

- <https://github.com/toulbar2/toulbar2/blob/kad/src/search/tb2solver.cpp>
- <https://github.com/toulbar2/toulbar2/blob/kad/src/search/tb2solver.hpp>

Dans le fichier hpp, la class Work définit les messages échangés entre master et workers. Un constructeur permet de formater les messages envoyés par le master aux workers, un autre se charge des communications dans l'autre sens.

Dans le fichier cpp, la méthode pair<Cost, Cost> Solver : :hybridSolvePara(Cost clb, Cost cub) contient l'implémentation parallèle avec Boost.MPI de l'algorithme HBFS obtenu à partir de l'algorithme séquentiel de la méthode pair<Cost, Cost> Solver : :hybridSolve(Cluster \*cluster, Cost clb, Cost cub).

Le master comme les workers envoient les messages en mode non-bloquant via la fonction isend()<sup>3</sup> et les reçoivent en mode bloquant via recv().

Le master comme les workers possèdent leur propre file *open*. Le master distribue les nœuds et la solution courante, y compris la valeur courante *cub*, aux workers qui se chargent d'effectuer le B&B avec parcours DFS borné par le nombre de backtracks (DFBB). Les workers renvoient les nœuds de leur propre file *open* au master qui les rangent dans la sienne.

Le master comprend deux boucles while imbriquées. La boucle externe assure la terminaison du programme ce qui n'est pas trivial dans le cas de la programmation parallèle. La boucle interne permet au master de distribuer le travail aux workers.

Pour des raisons d'optimisation spatiale un vecteur de *choice points*, ou vecteur de décisions, noté *cp*, est utilisé. Les nœuds en tant qu'objets possèdent comme attributs, outre le coût local *lb*, deux index qui pointent sur une partie du vecteur *cp*. La séquence de décisions associée au nœud n'est donc pas directement mémorisée dans le nœud, ce qui conduirait à des duplications de données. En effet, dans l'arbre de recherche, certains nœuds sont appelés à partager une partie de leur chemin.

---

<sup>3</sup>isend() correspond à un *immediate send* donc non bloquant. La fonction recv(), non préfixée par la lettre i, est bloquante ; le processus attend de recevoir un message pour poursuivre l'exécution du code.

### 2.3.3 Algorithme Worker

Algorithm 3: Parallel Hybrid Best-First Search : Worker

```
/* boucle infinie */  
1 while (1) do  
2   cpWorker := Ø ;           /* vecteur de choice points ou décisions du worker. */  
3   openWorker := Ø ;          /* initialisation de la PQ du master */  
4   recv(master,tag0,work);    /* Le worker attend le travail du master */  
5   /* A la réception d'un nœud, le worker se mets au travail */  
6   cub := min(cub,cub(master)) ; /* mets à jour le cub et la solution */  
7   cpWorker := cp[first,last] ; /* maj avec les décisions associées au nœud */  
8   openWorker.push(node) ; /* maj sa file de priorité avec le nœud du master */  
9   /* le DFS B&B mets à jour openWorker avec les nœuds ouverts produits */  
10  cub :=DFS(Av,cub,Z);  
11  NodesRecompute := NodesRecompute + v.depth ;  
12  clb := max(clb,lb(open)) ; /* calcul du clb global courant */  
13  if (NodesRecompute > 0) then  
14    /* heuristic Z adaptatif */  
15    if (NodesRecompute/Nodes > β and Z ≤ N) then Z := 2 × Z;  
16    else if (NodesRecompute/Nodes < α and Z ≥ 2) then Z := Z/2;  
17    /* Le worker envoie les nœuds produits, son ub, la nouvelle solution si elle  
18       est améliorante et son identifiant. isend() -> envoi non bloquant */  
19    isend(cpWorker,openWorker,newWorkerUb,workerRank,workerSol)
```

## 2.4 Premiers Résultats

Les premiers résultats présentés dans le tableau 2.1 montrent des performances très variables en fonction du problème mais globalement satisfaisantes. Les problèmes tels que capmp1 qui nécessitent un pré-traitement et/ou un temps de chargement long, augmentent la fraction séquentielle du programme. Ainsi, si le prétraitement du problème capmp1 est de 60s, la fraction séquentielle est proche de  $r = 171/60$  et la loi de Amdahl indique que le speedup est majoré par  $1/r = 2.85$ . On mesure un speedup de 2.10.

| wcsp file  | Serial Time (s) | Parallel time (s) | speedup | efficacité | optimal ub |
|------------|-----------------|-------------------|---------|------------|------------|
| graph11    | 318,3           | 143,5             | 2,22    | 9          | 3080       |
| capmp1     | 171,8           | 81,7              | 2,10    | 9          | 2460099    |
| scen06     | 1024,9          | 48,3              | 21,20   | 88         | 3389       |
| pedigree18 | 311,9           | 25,5              | 12,25   | 51         | 620119799  |
| nug12      | 197,0           | 9,8               | 20,14   | 84         | 578        |
| 404.wcsp   | 33,3            | 2,1               | 16,2    | 68         | 114        |

option -A pour graph11

```
/usr/bin/time -f "Temps = %es" mpirun -np 24 ./toulbar2 -para problem.wcsp
```

```
/usr/bin/time -f "Temps = %es" ./toulbar2 problem.wcsp
```

Machine 2 \* 12 cores : sullo et enfer

Efficacité = 100\*(speedup/24)

FIGURE 2.1 : Test effectués sur les serveurs 24 coeurs *sullo* et *infer* de l’unité MIAT sur une sélection des problèmes.

## 2.5 Bilan

D’autres tests doivent être entrepris d’ici la fin du stage qui n’est pas encore terminé à la date de restitution du présent rapport



# 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.

**MILP** Mixed-Integer Linear Programming.

**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.

**SPMD** Single Program Multiple Data.

**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.

**cpu** Central Processing Unit : terme qu'on utilisera comme synonyme de cœurs ou de processeur qui désignent le processeur physique qui exécute un processus.

**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 compétences 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 coeurs 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.

**Latence** Dans un contexte MPI, la latence représente le temps nécessaire pour qu'un émetteur et un récepteur s'échangent un message vide.

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

**nœud** Elément constitutif d'un système comprenant au moins un processeur muni de sa propre mémoire.

**processus** programme en cours d'exécution auquel le système d'exploitation octroie des ressources, en particulier un espace mémoire privé. Il possède un état.

**Speedup** Rapport entre le temps d'exécution d'un programme séquentiel et le temps du d'exécution du même programme parallélisé.

**SPMD** Single Program, Multiple Data : modèle d'exécution où chaque cpu exécute le même programme indépendamment sauf en ce qui concerne les échanges de messages comme avec le standard MPI. Les données manipulées ne sont pas forcément identiques.

**supercomputer** Bien qu'un cluster puisse être considéré comme un super ordinateur, la définition retenue 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 paral-

lèles 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)).

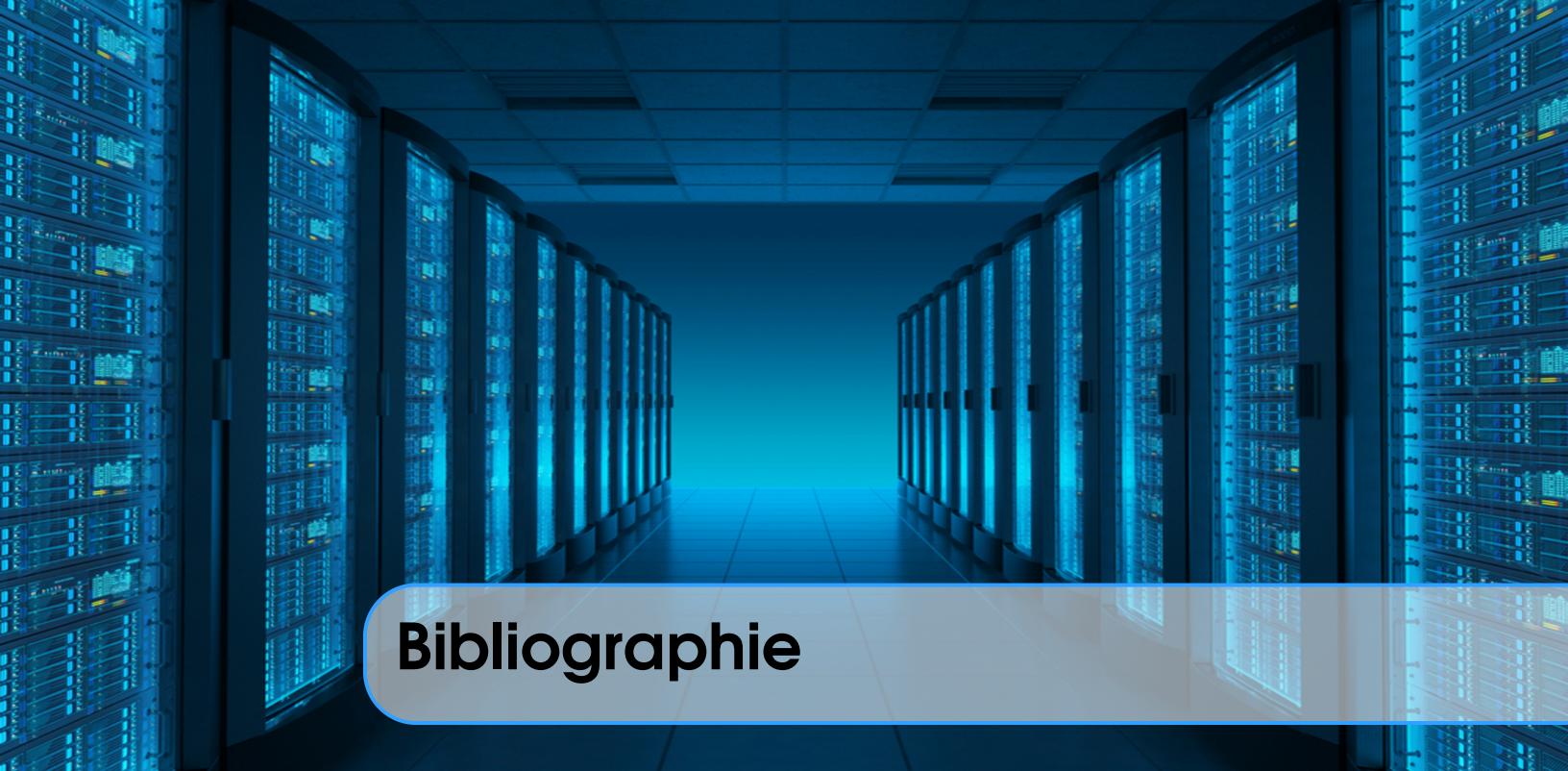
**thread** Appelé aussi processus léger. S'exécute dans l'espace mémoire d'un processus "lourd". Il possède sa propre pile mais partage l'espace mémoire avec d'autres threads.

**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>.

**Tâche** Programme constitué d'une séquences d'instructions qui décrivent les actions nécessaires à la réalisation de cette tâche. Une tâche décrit ce qu'il faut faire. Une tâche est effectuée par un processus qui utilise un processeur.

**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).
- [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. page 13).
- [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 25).
- [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).
- [6] A DJERRAH, Bertrand LECUN, Van-Dat CUNG et Catherine ROUCAIROL. “Bob++: Framework for Solving Optimization Problems with Branch-and-Bound methods”. In : (juin 2006), pages 369-370. DOI : [10.1109/HPDC.2006.1652188](https://doi.org/10.1109/HPDC.2006.1652188).
- [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. page 13).
- [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).
- [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>.
- [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>.
- [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>.
- [16] Tarek MENOUER. “Solving combinatorial problems using a parallel framework”. In : J. Parallel Distrib. Comput. 112 (2018), pages 140-153. DOI : [10.1016/j.jpdc.2017.05.019](https://doi.org/10.1016/j.jpdc.2017.05.019). URL : <https://doi.org/10.1016/j.jpdc.2017.05.019>.
- [17] Tarek MENOUER, Bertrand LECUN et P VANDER-SWALMEN. “Parallélisation d’un solveur de contraintes avec le framework parallèle BOBPP”. In : (jan. 2013).
- [18] 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>.
- [20] Ted RALPHS, Yuji SHINANO, Timo BERTHOLD et Thorsten KOCH. “Parallel Solvers for Mixed Integer Linear Optimization”. In : (2018), pages 283-336 (cf. page 18).
- [22] Jean-Charles RÉGIN, Mohamed REZGUI et Arnaud MALAPERT. “Embarrassingly Parallel Search”. In : (2013). Sous la direction de Christian SCHULTE, pages 596-610.
- [23] 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>.
- [24] “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).

## Books

- [4] Martin C. COOPER, Simon de GIVRY et Thomas SCHIEX. *Valued Constraint Satisfaction Problems*. Sous la direction de SPRINGER. Tome 2. Chapitre 7.
- [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.
- [19] Peter PACHECO. *An Introduction to Parallel Programming*. Elsevier Science Technology, 2011.
- [21] Jean-Charles RÉGIN et Arnaud MALAPERT. *Parallel Constraint Programming*. Springer, 2018. Chapitre 9, pages 337-379.





## C

cluster ..... 5

## G

GenoToul ..... 5