



8. Paradigme Master-Worker

Résumé

L'implémentation du mécanisme de coordination Master-Worker utilise les bibliothèques openMPI et Boost.MPI. L'algorithme est d'abord détaillé puis des expérimentations sont exposées. Enfin la comparaison avec d'autres approches similaires exhibe des résultats intéressants qui dépassent parfois les performances de solveurs perfectionnés tels que cplex ou daopt.

8.1 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'échelonnabilité (*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 6 : 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 MILP¹, en utilisant le solveur CPLEX², 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 [24].

¹On 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.

²CPLEX 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>

8.2 Description de l'algorithme

8.2.1 Vue globale

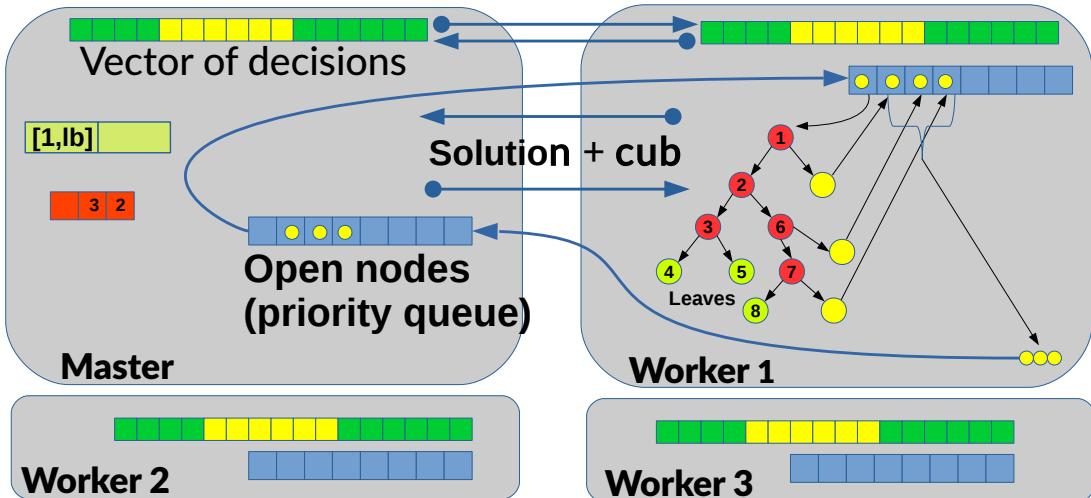


FIGURE 8.1 : HBFS Parallélisé : le master fournit le nœud 1 au worker qui lance un B&B dans le sous-arbre binaire de recherche associé à ce nœud. Les bornes inférieures lb sont propagées dans les nœuds enfants et servent à l'élagage de l'arbre. Le worker transmet au master les nœuds ouverts dès que le nombre maximum de backtracks, ici égal à 3, est atteint.

8.2.2 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 */  
    /* Le master attend le message d'un worker quelconque */  
11   recv(mpi :: any_source,tag0,work2); /* recv() bloquant */  
    /* A la réception d'une réponse, le master se remets au travail */  
12   cub := min(cub,cub(worker)); /* mets à jour le cub et la solution */  
13   cp := cpWorker; /* maj avec le vecteur de décisions du worker */  
14   open := openWorker; /* maj avec la file de priorité du worker */  
15   activeWork.erase(worker); /* efface la paire [worker,lb] */  
    /* calculate the min of the lb among the active workers */  
16   minLbWorkers = min(activeWork(worker));  
17   idleQ.push(worker); /* le worker devient inactif */  
18   clb := max(clb,min(lb(open),minLbWorkers)); /* calcule le clb global */  
19   showGap(clb,cub); /* affiche le gap d'optimalité */  
20   print(UB); /* affiche la valeur optimale si elle existe */  
    /* termine tous les processus workers */  
21 return (clb,cub); /* retourne le gap au hbfs parallel */
```

8.3 Implémentation de l'algorithme

8.3.1 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.

8.3.2 Utilisation du hbfs parallèle

Il suffit d'ajouter l'option -para à la ligne de commande de toulbar2 :

```
./toulbar2 -para problem.wcsp
```

8.3.3 Description synoptique

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

- <https://github.com/toulbar2/toulbar2/blob/kad2/src/search/tb2solver.cpp>
- <https://github.com/toulbar2/toulbar2/blob/kad2/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()³ et les reçoivent en mode bloquant via recv().

Le master comme les workers possèdent leur propre file *open*. Le master distribue les nœuds aux workers et la solution courante, y compris la valeur courante *cub*, qui se chargent d'effectuer le B&B avec parcours DFS borné par le nombre de backtracks (DFBB). Les bornes inférieures locales *lb* sont calculées pour chaque nœud. Comme dans tout B&B, les *lb* servent à élaguer l'arbre de recherche. Les workers renvoient le travail non terminé au master sous la forme de nœuds ouverts.

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.

³isend() 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.

8.3.4 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 */  
/* A la réception d'un nœud, le worker se mets au travail */  
5   cub := min(cub,cub(master)) ; /* mets à jour le cub et la solution */  
6   cpWorker := cp[first,last] ; /* maj avec les décisions associées au nœud  
*/  
7   openWorker.push(node) ; /* maj sa file de priorité avec le nœud du  
master */  
/* le DFS B&B mets à jour openWorker avec les nœuds ouverts  
produits */  
8   cub :=DFS(Av,cub,Z);  
9   NodesRecompute := NodesRecompute + v.depth ;  
10  clb := max(clb,lb(open)) ; /* calcul du clb global courant */  
11  if (NodesRecompute > 0) then  
    /* heuristic Z adaptatif */  
    if (NodesRecompute/Nodes > β and Z ≤ N) then Z := 2 × Z;  
    else if (NodesRecompute/Nodes < α and Z ≥ 2) then Z := Z/2;  
    /* Le worker envoie les nœuds produits, son ub, la nouvelle  
    solution si elle est améliorante et son identifiant. isend() ->  
    envoi non bloquant */  
14  isend(cpWorker,openWorker,newWorkerUb,workerRank,workerSol)
```

8.4 Expérimentations

8.5 Comparaison entre EPS et Master-Worker

Le paradigme Master-Worker est meilleur dans tous les cas comme le montre la table 8.5.

Instance	<i>n</i>	<i>d</i>	S M-W	E M-W (%)	S EPS	E (%) EPS
graph11	340	44	2.22	9.24	0.25	1.05
capmp1	400	200	2.10	8.76	1.63	6.79
scen06	100	44	21.20	88.34	2.16	9.01
pedigree18	1184	5	11.18	51.03	1.32	5.50
nug12	12	12	20.14	83.92	5.63	23.46
404	100	4	16.25	67.70	5.09	21.21

TABLE 8.1 : Speed-up S et Efficacité E pour le Master-Worker(M-W) et l'*Embarrassingly Parallel Search* (ESP). Expérimentations faites sur serveur 24 cœurs. Le nombre de variables *n* et la taille maximale des domaines *d* donnent une idée de la complexité du problème.

Comparaison de toulbar2 avec HBFS séquentiel et HBFS parallèle

Les expérimentations ont été réalisées sur des serveurs 24 cœurs (Intel Xeon E5-2680/87 at 2.50/3GHz and 256 GB) ou sur la platefome GenoToul (64-core nodes of Intel Xeon E5-2683 at 2.10GHz).

Les premiers résultats présentés dans le tableau 8.2 montrent des performances très variables en fonction du problème mais globalement bonnes et toujours meilleures que celles de toulbar2 séquentiel.

Quand ce n'est pas le cas, on peut invoquer une fraction séquentielle du programme non négligeable comme pour le problème capmp1 et invoquer le speed-up théorique de Amdahl pour interpréter les résultats faibles dus à la durée de pré-traitement et de chargement du programme.

Cependant, un autre aspect semble influer sur les performances, celui des temps d'attente du master et des workers. On observe des performances plus faibles en terme de speed-up si les temps d'attente du master comme des workers sont élevés. Ainsi, pour scen06 le temps d'attente des workers est de l'ordre de 0.1s tandis que le master attend les réponses des workers durant 37 secondes. Pour graph11, les workers attendent entre 7 et 25s et le master 122s pour un temps d'exécution de 125s. Le master est donc bien disponible. Les temps de communications ou les temps d'accès et d'écriture dans la mémoire pourraient être invoqués pour expliquer des speed-ups faibles.

Instance	Serial Time (s)	Parallel time (s)	speedup	Efficacité (%)	pre time
graph11	323.12	125.55	2.57	10.72	1.90
capmp1	173.27	82.23	2.11	8.78	6.80
scen06	1026.48	39.22	26.18	109.07	0.16
pedigree18	6.76	0.57	11.86	49.42	0.10
nug12	201.40	8.75	23.02	95.90	0.004
404.wcsp	34.66	1.30	26.66	111.09	0.005

TABLE 8.2 : Exécutions sur serveur 24 cœurs. Le temps CPU est utilisé pour le calcul des speed-ups. La colonne Parallel time indique le temps CPU total. La colonne **pre time** indique la durée du préprocessing.

Eléments nécessaires à la reproduction des résultats

- Le problème graph11 est exécuté avec l'option -A pour utiliser la méthode d'arc cohérence : [Virtual Arc Consistency \(VAC\)](#),
- Options toulbar2 pour le problème pedigree18 : -A -O=-3 -p=-8,
- Exécution en parallèle : mpirun -np 24 ./toulbar2 -para problem.wcsp,
- Exécution séquentielle : ./toulbar2 problem.wcsp,
- Exécution faite sur serveur 24 cœurs de l'unité MIAT : sullo et enfer avec tous les processeurs libres,
- Efficacité = $100 * (\text{speedup}/24)\%$.

Dans les Tables 8.3 and 8.4, on trouvera les temps en secondes nécessaires à la résolution et à la preuve d'optimalité d'instances de type Warehouse et Conception de protéines par ordinateur (CPD). Les tests ont été automatisés sur les serveurs 24 cœurs. Les exécutions sont arrêtées au bout de 1 heure. On notera que la virgule est le séparateur des milliers et non décimal.

On constate dans la table 8.3 une amélioration des performances dans presque tous les cas de figure pour le HBFS parallèle (PHBFS). Les speed-ups restent cependant faibles.

Instance	<i>n</i>	<i>d</i>	Time (sec.)		Speed-up
			HBFS	HBFS-24	
capmo1	200	100	10.92	5.14	2.12
capmo2	200	100	1.80	2.04	0.88
capmo3	200	100	6.09	3.73	1.63
capmo4	200	100	4.36	3.21	1.36
capmo5	200	100	2.69	2.58	1.04
capmp1	400	200	172.57	80.64	2.14
capmp2	400	200	95.15	61.35	1.55
capmp3	400	200	75.04	56.25	1.33
capmp4	400	200	107.86	78.4	1.38
capmp5	400	200	81.15	52.9	1.53
capmq1	600	300	679.43	412.52	1.65
capmq2	600	300	841.80	503.64	1.67
capmq3	600	300	647.67	431.47	1.5
capmq4	600	300	1,093.55	514.42	2.13
capmq5	600	300	1,388.41	701.61	1.98

TABLE 8.3 : Benchmark warehouse [16] où *n* désigne le nombre de variables du problème et *d* la taille du plus grand domaine de ces variables.

Dans la table 8.4, on voit que PHBFS a résolu 3 instances non résolues par HBFS.

Instance	<i>n</i>	<i>d</i>	Time (sec.)		Speed-up
			HBFS	HBFS-24	
1xaw	107	412	721.43	568.50	1.27
3lf9	120	416	407.28	407.92	1
5dbl	130	384	122.84	171.82	0.71
5e10	133	400	147.73	198.23	0.75
5e0z	136	420	148.26	193.41	0.77
5eqz	138	434	3,366.11	1,049.11	3.21
1dvo	152	389	622.03	463.29	1.34
4bxp	170	439	327.46	395.16	0.83
1is1	185	431	-	2,545.82	-
2gee	188	397	797.22	863.64	0.92
5jdd	263	406	-	2,758.98	-
3r8q	271	418	1,605.30	1,294.97	1.24
1f00	282	430	-	2,140.40	-

TABLE 8.4 : Benchmarks *Computational Protein Design* (CPD) [22]. Les tirets indiquent que le problème n'a pu être résolu en moins d'une heure.

8.5.1 Comparaison avec des travaux similaires

Des expérimentations sur d'autres instances difficiles de conception de protéines ont montré des speed-up intéressants.

Dans la table 8.5, on compare les solveurs cplex, daoopt et HBFS sur le benchmark *Linkage*. Les résultats du solveur daoopt sont obtenus sur un cluster de dual core 2.67 GHz Intel Xeon X5650 6-core CPUs et 24 GB de RAM. Ici, la meilleure scalabilité de PHBFS permet d'obtenir de meilleurs résultats sur cluster sur les instances pedigree31 et 44 mais PHBFS est dominé par cplex sur les instances pedigree19 et 51.

Le solveur daoopt est loin derrière mais avec des speed-ups et efficacités élevés.

	pedigree19	pedigree31	pedigree44	pedigree51
<i>n</i>	793	1,183	811	1,152
<i>d</i>	5	5	4	5
cplex	790	59.30	6.35	36.23
//10	191 (4.14)	9.00 (6.59)	2.48 (2.56)	9.43 (3.84)
//30	75 (10.53)	7.17 (8.27)	2.69 (2.36)	5.34 (6.78)
daoopt	375,110	16,238	95,830	101,788
//20	27,281 (13.75)	1,055 (15.39)	6,739 (14.22)	6,406 (15.89)
//100	7,492 (50.07)	201 (80.79)	1,799 (53.27)	1,578 (64.50)
HBFS	3,126	4.34	39.72	1,608
//10	434.27 (7.20)	1.51 (2.87)	6.08 (6.53)	179.22 (8.97)
//20	227.02 (13.77)	1.39 (3.12)	3.18 (12.49)	72.30 (22.24)
//100	119.43 (26.17)	0.97 (4.47)	1.64 (24.22)	31.40 (51.21)

TABLE 8.5 : Benchmark *linkage* [8] avec divers nombre de coeurs. Entre parenthèses, sont indiqués les speed-ups.

On voit dans la table 8.6 que HBFS parvient à résoudre toutes les instances contrairement à CPLEX et que PHBFS, utilisé avec 10 cœurs, améliore encore les résultats.

Instance	n	d	cplex	cplex-10	HBFS	HBFS-10	Speed-up
1UBI	13	198	-	-	1,023	214.02	4.78
2DHC	14	198	-	-	8.2	5.83	1.41
2DRI	37	186	-	-	135.5	30.00	4.52
1CDL	40	186	-	-	392.6	54.95	7.14
1CM1	42	186	-	6,177	6.6	6.11	1.08
1BRS	44	194	-	-	555.3	107.86	5.15
1GVP	52	182	-	-	596.1	185.75	3.21
1RIS	56	182	-	-	129.7	36.23	3.58
3CHY	74	66	-	5,259	88.7	20.71	4.28

TABLE 8.6 : Autre benchmark CPD [2]. Un tiret indique que la méthode n'est pas parvenue à prouver l'optimalité en moins de 9000s.

8.6 Bilan

La version Master-Worker de PHBFS produit des résultats intéressants qui améliorent la plupart du temps les performances du HBFS séquentiel mais surtout dépassent dans certains cas des solveurs perfectionnés tel que `cplex`⁴ ou `daoopt`⁵.

⁴Solveur activement développé par IBM : <https://www.ibm.com/analytics/cplex-optimizer>

⁵<https://github.com/lotten/daoopt>



9. Conclusion

L'objectif de ce stage, qui s'est déroulé du 18 mars au 15 septembre 2019 à l'INRA Toulouse, a consisté à paralléliser HBFS un algorithme séquentiel de recherche arborescente implémenté en C++ dans [toulbar2](#), un solveur de réseaux de fonctions de coûts représentables sous la forme d'hypergraphes.

Après une première étape bibliographique (1.5 mois), de prise en main des outils et de [toulbar2](#) (0.5 mois), la piste EPS a été explorée durant 1 mois comprenant l'analyse du code, sa modification pour produire les sous-problèmes et les expérimentations et tests associés. La piste Master-Worker a pris environ 2 mois comprenant l'analyse du code, le choix de solutions techniques, les tests, les expérimentations et comparaisons avec d'autres approches utilisées dans des solveurs tels que [cplex](#). La rédaction de ce rapport et d'autres documents, notamment pour le séminaires des stagiaires à l'INRA, a nécessité 1 mois.

La démarche utilisée a consisté à explorer des pistes de parallélisation, à en choisir deux : la piste EPS et la piste Master-Worker et à les explorer c'est à dire à réaliser les développements, les tests, les expérimentations, la comparaison et l'analyse des résultats.

HBFS a finalement été parallélisé avec succès en donnant des résultats probants qui feront l'objet d'une présentation à la 25ème conférence CP 2019¹ sur la programmation par contraintes.

Des améliorations pourraient cependant être apportées :

- En terme d'échelonnableté avec l'aide d'autres paradigmes déjà cités dans ce rapport² : *Superviseur-Worker, Multiple-Masters-Worker, Master-Hub-Worker*
- En terme de speedup, en parallélisant les pré-traitements si toutefois c'est possible,
- En parallélisant l'algorithme : Backtracking Tree Decomposition HBFS (BTD-HBFS).

¹<https://cp2019.a4cp.org/>

²Le présent rapport est publié sur <https://github.com/kad15/SandBoxToulbar2/blob/master/kad/rapport.pdf>



10. Annexes

10.1 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
                ={{},{}}, {{}} ...
            }
        }
        vv = tmp;
    }
}

```

```

        }
    }

    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;
}
```

10.2 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
4 PREREQUISITES:
5 sudo apt-get update
6
7 - Connecting to git without login :
8 create an ssh key without pass phrase.
9 cd ~/.ssh && ssh-keygen -t ed25519 -C "toto@free.fr"
10 cat id_ed25519.pub, copy the output, use it to create a new ssh key on github. see
    Settings of your account.
11 git 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 ccmake .. 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 : ccmake .. means execute ccmake (ccmake 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 wil 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
```

10.3 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 [10.2](#).

```
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 coredumpsize 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 pour 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 -DILOGCPLEX -DIL_STD -
DILOSTRICTPOD -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_regexp "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-regexp "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 COMPIRATION 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(Toulebar_NAME "toulbar2") # default project name

IF(ALL_APP)
  MESSAGE(STATUS "#####")
  MESSAGE(STATUS " TOULBAR2 and MENDELSOFT COMPIRATION 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 COMPILEMENT 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 "ILOGUE COMPILATION" 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/mpi/gcc/openmpi-1.10.3rc4/bin/mpicxx)
#  SET(CMAKE_SYSTEM_PREFIX_PATH /usr/mpi/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++)
  # 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(MENDELSOFT)
    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(MENDELSOFT)

# 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)

#####
# Ilog Solver 6.0
#####
include(${My_cmake_script}/ilogue.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 ${doc_destination}/${Toolbar_NAME} PATTERN
".svn" EXCLUDE)
install(FILES ${CMAKE_CURRENT_BINARY_DIR}/toulbar2.1 DESTINATION share/man/man1)

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

10.4 Annexe D : Utilisation de toulbar2 avec l'option eps

Génération des sous-problèmes avec HBFS

L'option `-eps` a été ajoutée à la longue liste d'options de toulbar2. On trouvera ci-dessous un exemple d'utilisation avec mesure du *wall time* en secondes.

```
/usr/bin/time -f "Generation Time: %es" ./toulbar2 404.wcsp -eps
```

Cette commande produit la sortie ci-dessous et 3 fichiers :

1. `eps.sh` : script d'exécution de toulbar2 en parallel à adapter si nécessaire,
2. `nbProcess.txt` : contient le nombre de sous-problèmes(SP) à générer. Ce n'est qu'une valeur cible approximative. Le nombre effectif de sous-problèmes générés sera en général différent. Si le fichier existe, le nombre-cible de SP sera lu dans ce fichier, sinon il est créé avec un nombre = $30k$ avec k nombre de processus. L'idée derrière ce fichier était de mettre en place un apprentissage pour déterminer automatiquement, en fonction de la plateforme, le nombre-cible optimal de SP à générer.
3. `subProblems.txt` : chaque ligne contient un sous-problème de la forme ci-dessous qui décrit une séquence de décisions $v.\delta$. Elle correspond à un nœud dans l'arbre de recherche.

```
-x=", 44#0, 46=1, 60=1, 45=1, 50=1, 71=0, 58=1, 88=1, 61=1, 89=1, 62=1, 11=1  
, 63=1, 86#1"
```

L'option `-x` de toulbar2 permet de résoudre ce SP spécifique. `44#0` signifie par exemple que la décision sur la variable de rang 44 est de retirer la valeur 0 de son domaine, etc.

Dans cet exemple, 120 sous-problèmes ont été demandés mais 141 sont produits par toulbar2 pour obtenir une *partition* du problème global ; si ce n'est pas une *partition*, l'espace d'états ne sera pas exploré exhaustivement et donnera probablement une solution sous-optimale.

```
c ./toulbar2 version : 1.0.1-449-gb040efb-kad (1567241515),  
copyright (c) 2006-2019, toulbar2 team  
HBFS Embarrassingly Parallel Search activated.  
loading wcsp file: ./404.wcsp  
Read 100 variables, with 4 values at most, and 710 cost functions,  
with maximum arity 3.  
Cost function decomposition time : 4e-05 seconds.  
Reverse DAC dual bound: 65 (+13.846%)  
Reverse DAC dual bound: 66 (+1.515%)  
Preprocessing time: 0.004 seconds.  
88 unassigned variables, 226 values in all current domains  
(med. size:2, max size:4) and 594 non-unary cost functions (med. degree:1)  
Initial lower and upper bounds: [66, 158] 58.228%
```

```

SEQUENTIAL HBFS MODE!!! ADD -para OPTION FOR PARALLEL MODE
New solution: 122 (0 backtracks, 17 nodes, depth 18)
Optimality gap: [67, 122] 45.082 % (17 backtracks, 34 nodes)
New solution: 120 (17 backtracks, 77 nodes, depth 43)
Optimality gap: [73, 120] 39.167 % (59 backtracks, 119 nodes)
Optimality gap: [83, 120] 30.833 % (94 backtracks, 195 nodes)
Optimality gap: [84, 120] 30.000 % (123 backtracks, 261 nodes)
TOTAL NUMBER OF NODES ADDED IN OPEN LISTE :146
NUMBER OF NODES IN OPEN LISTE WHEN openNodeLimit is reached : 141
BEST CURRENT SOLUTION FOUND AT DUMP TIME : UB = 120
NUMBER OF SUBPROBLEMS REALLY GENERATED : 141
Text has been written in file subProblems.txt
Text has been written in file eps.sh
Temps d'execution : 0.01s

```

Remarque 1 : Outre la partition en sous-problèmes, hbfs fournit la meilleure solution au moment de la production des SP, c'est à dire au moment du *dump*. Cette dernière peut être utilisée par toulbar2 dans la phase parallèle.

Remarque 2 : Une instruction tel que "if(node.getCost() < wcsp->getUb())" élimine les nœuds qui ne peuvent de donner de solution. C'est la condition de non-élagage.

Remarque 3 : La production des sous-problèmes par toulbar2 bénéficie du pré-processing. Les 3 remarques précédentes montrent que les SP générés seront de bonne qualité.

Phase parallel

La commande ci-dessous donne un exemple d'utilisation de GNU parallel sur une machine à 4 cœurs. On utilise la solution $UB = 120$ trouvée au moment du dump. Chaque sous-problème du fichier *subProblems.txt* est transmis à GNU Parallel qui assure l'assignation des processus à 4 processeurs. Ainsi, dans cet exemple, on a demandé $30k = 30 * 4 = 120$ sous-problèmes, 146 nœuds sont passés par la file open, 141 ont effectivement été produits. La génération a pris 0.1s. Ce temps est négligeable mais ce n'est pas le cas pour d'autres problèmes.

```
cat subProblems.txt | time parallel -j4 --eta -k ./toulbar2 404.wcsp
-ub=120 {}
```




Acronymes

ABDFBB Adaptative 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=E7hJXsywOdA>.

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.

VAC Virtual Arc Consistency.

VCSP Valued Constraint Satisfaction Problem.

WCSP Weighted Constraint Satisfaction Problems.



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. 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"^[4] 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. 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 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, phy-

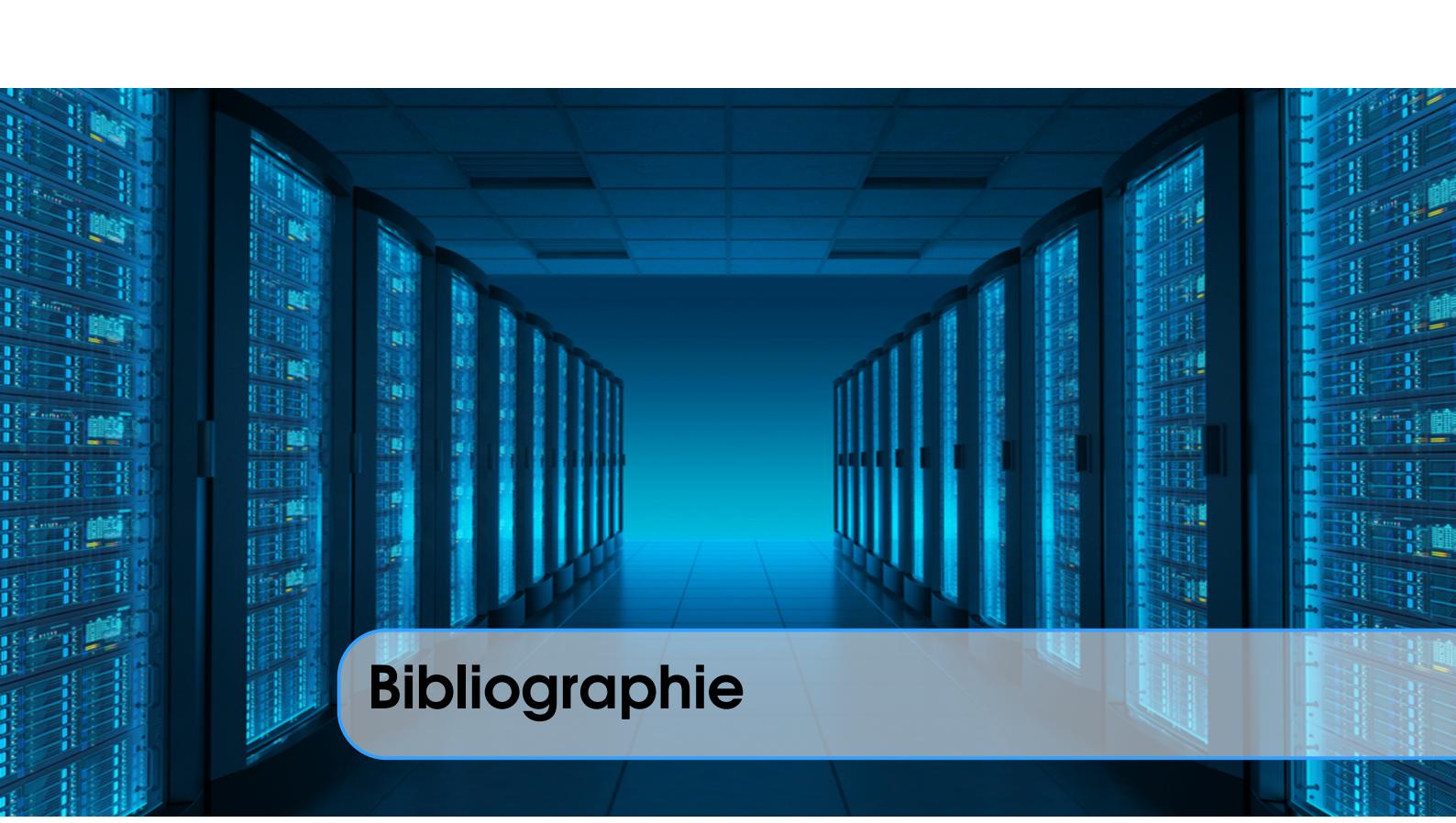
sique des particules et les super ordinateurs, aux traitements de tâches en parallèles dépendantes les unes des autres : mécanique des fluides, prévisions météorologiques. Les performances de HBFS seraient ainsi supérieures sur un super ordinateur. Exemple de supercomputer : IBM Blue Gene/Q baptisé MIRA 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)).

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 coûts : <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 coeurs. Nécessite d'éviter les goulets 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 25, 39).
- [2] D ALLOUCHE, J DAVIES, S de GIVRY, G KATSIRELOS, T SCHIEX, S TRAORÉ, I ANDRÉ, S BARBE, S PRESTWICH et B O’SULLIVAN. “Computational Protein Design as an Optimization Problem”. In : AI 212 (2014), pages 59-79 (cf. page 102).
- [3] 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 25, 48, 63).
- [4] 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 131).
- [6] 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 (cf. page 56).
- [7] 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).

- [8] A FAVIER, S de GIVRY, A LEGARRA et T SCHIEX. “Pairwise decomposition for combinatorial optim. in graphical models”. In : (2011), pages 2126-2132 (cf. page 101).
- [9] 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 : ().
- [10] 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 47, 56, 63).
- [11] Simon de GIVRY, Thomas SCHIEX et Gérard VERFAILLIE. “Exploiting Tree Decomposition and Soft Local Consistency In Weighted CSP”. In : (2006).
- [13] 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 25).
- [14] 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 25, 39, 40).
- [15] 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 40).
- [16] Javier LARROSA, Simon S DE GIVRY, F HERAS et M ZYTNICKI. “Existential arc consistency: getting closer to full arc consistency in weighted CSPs”. In : (2005), pages 84-89 (cf. page 100).
- [17] 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 25, 39).
- [19] 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> (cf. page 79).
- [20] Tarek MENOUER, Bertrand LECUN et P VANDER-SWALMEN. “Parallélisation d’un solveur de contraintes avec le framework parallèle BOBPP”. In : (jan. 2013) (cf. page 79).
- [21] L OTTEN et R DECHTER. “AND/OR Branch-and-Bound on a Computational Grid”. In : *JAIR* 59 (2017), pages 351-435.

- [22] 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 25, 39, 40, 101).
- [24] Ted RALPHS, Yuji SHINANO, Timo BERTHOLD et Thorsten KOCH. “Parallel Solvers for Mixed Integer Linear Optimization”. In : (2018), pages 283-336 (cf. pages 40, 67, 75, 76, 78-81, 94).
- [26] Jean-Charles RÉGIN, Mohamed REZGUI et Arnaud MALAPERT. “Embarrassingly Parallel Search”. In : (2013). Sous la direction de Christian SCHULTE, pages 596-610 (cf. pages 25, 39, 40, 77, 81-83).
- [27] 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 47).
- [28] “Weighted constraint satisfaction problem”. In : (2016). URL : https://en.wikipedia.org/wiki/Weighted_constraint_satisfaction_problem (cf. pages 47, 48).

Books

- [5] Martin C. COOPER, Simon GIVRY et Thomas SCHIEX. *Valued Constraint Satisfaction Problems*. Sous la direction de SPRINGER. Tome 2. Chapitre 7 (cf. pages 47, 49).
- [12] Ananth GRAMA, Anshul GUPTA, George KARYPIS et Vipin KUMAR. *Introduction to Parallel Computing*. Addison Wesley, 2003.
- [18] Timothy MATTSON, Beverly SANDERS et Berna MASSINGILL. *Patterns For Parallel Programming*. Addison-Wesley, 2004.
- [23] Peter PACHECO. *An Introduction to Parallel Programming*. Elsevier Science Technology, 2011 (cf. page 72).
- [25] Jean-Charles RÉGIN et Arnaud MALAPERT. *Parallel Constraint Programming*. Springer, 2018. Chapitre 9, pages 337-379 (cf. pages 25, 39, 77, 78, 84).