

1 Parallel Hybrid Best-First Search

2 **Abdelkader Beldjilali** ✉

3 Université Fédérale de Toulouse, INRAE, UR 875, 31326 Toulouse, France

4 **Pierre Montalbano** ✉ 

5 Université Fédérale de Toulouse, ANITI, INRAE, UR 875, 31326 Toulouse, France

6 **David Allouche** ✉

7 Université Fédérale de Toulouse, ANITI, INRAE, UR 875, 31326 Toulouse, France

8 **George Katsirelos** ✉ 

9 Université Fédérale de Toulouse, ANITI, INRAE, MIA Paris, AgroParisTech, 75231 Paris, France

10 **Simon de Givry** ✉ 

11 Université Fédérale de Toulouse, ANITI, INRAE, UR 875, 31326 Toulouse, France

12 — Abstract —

13 While processor frequency has stagnated over the past two decades, the number of available cores in
14 servers or clusters is still growing, offering the opportunity for significant speed-up in combinatorial
15 optimization. Parallelization of exact methods remains a difficult challenge. We revisit the concept
16 of parallel Branch-and-Bound in the framework of Cost Function Networks. We show how to adapt
17 the anytime Hybrid Best-First Search algorithm in a Master-Worker protocol. The resulting parallel
18 algorithm achieves good load-balancing without introducing new parameters to be tuned as is the
19 case, for example, in Embarrassingly Parallel Search (EPS). It has also a small overhead due to its
20 light communication messages. We performed an experimental evaluation on several benchmarks,
21 comparing our parallel algorithm to its sequential version. We observed linear speed-up in some
22 cases. Our approach compared favourably to the EPS approach and also to a state-of-the-art parallel
23 exact integer programming solver.

24 **2012 ACM Subject Classification** Computing methodologies → Parallel algorithms

25 **Keywords and phrases** Combinatorial Optimization, Parallel Branch-and-Bound, CFN

26 **Digital Object Identifier** 10.4230/LIPIcs.CP.2022.36

27 **Supplementary Material** <https://miat.inrae.fr/degivry/Beldjilali22Supp.pdf>

28 **Funding** This work has been partially funded by the French "Agence Nationale de la Recherche",
29 through grant ANR-19-P3IA-0004. It was performed using HPC resources from CALMIP (Grant
30 2022-P21010).

31 **Carbon footprint** The experiments in this paper took approximately 17,000 hours and emitted
32 68kg of CO₂, with an estimate of 4g/h per core.

33 **1** Introduction

34 Cost Function Networks (CFNs), also known as Weighted Constraint Satisfaction Problems
35 (WCSPs) [17] is a mathematical framework which has been derived from Constraint Sat-
36 isfaction Problems by replacing constraints with cost functions. In a CFN, we are given
37 a set of variables with an associated finite domain and a set of local cost functions. Each
38 cost function involves some variables and associates a non-negative integer cost to each of
39 the possible combinations of values they may take. The usual WCSP problem considered
40 is to assign all variables in a way that minimizes the sum of all costs. This minimization
41 problem is NP-hard, and exact methods usually rely on Branch and Bound (B&B) algorithms
42 exploring a binary search tree with *soft local consistency* maintained at each node in order



© David Allouche, Abdelkader Beldjilali, Simon de Givry, George Katsirelos, and Pierre Montalbano;
licensed under Creative Commons License CC-BY 4.0

28th International Conference on Principles and Practice of Constraint Programming (CP 2022).

Editor: Christine Solnon; Article No. 36; pp. 36:1–36:10

Leibniz International Proceedings in Informatics



LIPIC Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

43 to improve the problem lower bound (represented by c_\emptyset) and prune domain values with a
 44 forbidden cost (represented by a maximum cost k) [5].

45 Constraint Programming (CP) exact approaches usually rely on Depth-First Search (DFS)
 46 methods while Integer Linear Programming (ILP) approaches explore the tree in a best-first
 47 manner by exploiting strong bounds. We are interested in hybrid methods combining depth-
 48 first and best-first with possibly weaker bounds but faster to compute. This is the case of the
 49 Hybrid Best-First Search (HBFS) method [2]. HBFS is a B&B algorithm for solving WCSPs.

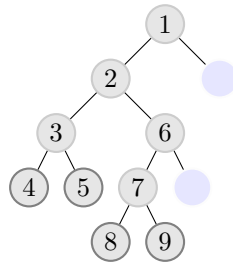
50 Dealing with parallel computers or grids to speed-up solving time of exact methods has
 51 been explored in many different ways. For grids, with slow network interconnection, MapRe-
 52 duce is a general approach exploiting problem decomposition into independent subproblems
 53 solved in parallel (*map* on the grid processors) and then sequentially *reduced* at the end
 54 of the resolution. In CP, this decomposition approach is called Embarrassingly Parallel
 55 Search (EPS) [15]. MapReduce has been applied also in the context of non serial dynamic
 56 programming in Graphical Models [19] and CFNs [3]. Message-passing approaches, on the
 57 other hand, take advantage of the low-latency communication of supercomputers, consisting
 58 of a large number of multiprocessor servers interconnected at high speed and low latency.
 59 This allows for finer granularity in B&B parallelization. According to a recent survey [9],
 60 parallelizing the search based on message-passing and parallel B&B in CP are difficult
 61 problems and still poorly explored. In CP, for example, COMET [18] uses *work-stealing*
 62 where workers which have run out of work take unexpanded nodes from other workers, leaving
 63 them less work to do and keeping all workers busy. In ILP, a recent review on parallel B&B
 64 was proposed in [21]. We selected the Master-Worker protocol as the basis for our approach.
 65 Other approaches rely on portfolios.

66 In this work we describe a parallel version of HBFS. We give an empirical evaluation
 67 on combinatorial optimization academic problems from Operations Research and real-life
 68 Graphical Model problems occurring in genetics and biology. Our experimental study analyses
 69 solving time and speed-ups of the parallel version compared to the original sequential HBFS.
 70 We also compare our approach with a parallel ILP solver (IBM Ilog `cplex`). Moreover, we
 71 performed experiments on a high-performance cluster to study the scalability of our algorithm
 72 and compare with EPS.

73 **2 Hybrid Best-First Search**

74 The sequential version of HBFS [2] is a B&B method for CFNs that combines Best-First
 75 Search (BFS) and Depth-First Search (DFS). Like BFS, HBFS provides an anytime global
 76 lower bound on the optimum, while also providing anytime upper bounds, like DFS. Hence, it
 77 provides feedback on the progress of search and solution quality in the form of an optimality
 78 gap. Besides, it exhibits highly dynamic behavior that allows it to perform on par with
 79 methods like Limited Discrepancy Search [11] and frequent restarting [10, 7] in terms of
 80 quickly finding good solutions. As in BFS, HBFS maintains a frontier of open search nodes. It
 81 expands each open node using DFS with a limit on its number of backtracks. Each bounded
 82 DFS returns a new list of open nodes to be inserted in the BFS frontier.

83 The pseudo-code of HBFS is given in Algorithm 1. The main procedure is in charge of the
 84 BFS frontier of open nodes. Here a node ν corresponds to a sequence of decisions $\nu.\delta$. The
 85 root node has an empty decision sequence (line 1). When a node is explored by DFS (line 5),
 86 an unassigned variable is chosen and a branching decision to either assign the variable to
 87 a chosen value (left branch, positive decision) or remove the value from the domain (right
 88 branch, negative decision) is taken. The number of decisions taken to reach a given node



■ **Figure 1** A tree that is partially explored by DFS with a backtrack limit $Z = 4$. Nodes with a bold border are leaves, nodes with no border are placed in the open list after the backtrack bound is exceeded. Nodes are numbered in the order they are visited.

89 ν is the depth of the node, $\nu.depth$. HBFS always chooses the next open node to explore
 90 with minimum lower bound $\nu.lb$ (best-first principle) and, in case of ties, maximum depth
 91 $\nu.depth$ (depth-first principle) in the frontier. The minimum of all open node lower bounds,
 92 denoted $lb(open)$, is a valid global lower bound (kept in clb at line 6) for the problem. HBFS
 93 also maintains the current upper bound (cub) as the cost of the best solution found so far by
 94 DFS (line 5). The search ends when the open list is empty or contains nodes with a lower
 95 bound greater than or equal to cub (line 2).

```

Function HBFS( $clb, cub$ ): integer ; /* Returns the optimum value */
1   $open := \{\nu(\delta = \emptyset, lb = clb)\}$  ; /* Initializes the open list with a root node */
2  while ( $open \neq \emptyset$  and  $clb < cub$ ) do
    $\nu := pop(open)$  ; /* Chooses a node with minimum lower bound and maximum depth */
3   Restores state  $\nu.\delta$ , leading to assignment  $A_\nu$ , maintaining soft local consistency ;
4    $NodesRecompute := NodesRecompute + \nu.depth$  ;
5    $cub := DFS(A_\nu, cub, Z)$  ; /* Increase Nodes and put all right open branches in open */
6    $clb := max(clb, lb(open))$  ;
   if ( $NodesRecompute > 0$ ) then
7     if ( $NodesRecompute/Nodes > \beta$  and  $Z \leq N$ ) then  $Z := 2 \times Z$  ;
8     else if ( $NodesRecompute/Nodes < \alpha$  and  $Z \geq 2$ ) then  $Z := Z/2$  ;
   return  $cub$  ;
```

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

96 DFS increases a counter $Nodes$ at each branching decision. It can backtrack (taking right
 97 branches) up to a limit of Z backtracks. When this limit is reached, all the unexplored
 98 right branches are placed in $open$. HBFS controls the balance between best-first search
 99 (partially exploring more open nodes) and depth-first search (complete exploration from a
 100 given starting node). Best-first search requires recomputing the state $\nu.\delta$ of a node (line 3)
 101 which can be costly in practice. HBFS uses a simple rule to limit this recomputation effort
 102 (measured by $NodesRecompute$ at line 4). It tries to keep the ratio $\frac{NodesRecompute}{Nodes}$ in the
 103 interval $[\alpha, \beta]$ by increasing (by a power of two) the backtrack limit Z if the ratio value is
 104 above β or decreasing Z if it is below $alpha$ (lines 7–8). Initially, Z is set to 1. In order to
 105 avoid exponential DFS behavior, HBFS limits the maximum value taken by Z to N . We kept
 106 the same value $\alpha = 5\%$, $\beta = 10\%$, $N = 2^{14}$ in our experiments as in the original paper [2].

107 **3** Parallel HBFS

108 The parallel version of HBFS is based on the Master-Worker parallel paradigm [21] where
 109 the *Master* is in charge of the open node frontier and dispatches the current best (with
 110 minimum lower bound) open node plus the current best solution found so far to the next
 111 available *Worker*. The Worker performs a bounded DFS starting from the received node and
 112 returns to the Master the resulting list of open nodes (see Fig. 1, with a DFS limit here of 4
 113 backtracks). The Worker also returns the best solution found during its restricted search
 114 if any. Only the Master has a global view of the whole search and reports optimality gaps
 115 ($\frac{cub-clb}{cub}$) until the proof of optimality is reached: when the current best lower bound in the
 116 frontier of open nodes, including active worker starting nodes, is equal or greater than the
 117 cost of the best solution found so far or the frontier is empty and there are no active workers.
 118 When the problem is solved, the Master kills all the workers and returns the optimum value.

119 According to a round robin schema, the Master sends open nodes to every idle worker
 120 in a balanced way, ensuring a natural load balancing between the workers as soon as the
 121 number of open nodes in the frontier is larger than the number of workers. Moreover, an
 122 initial backtrack limit of $Z_i = 1$ associated to each Worker i favors the production of open
 123 nodes at the beginning of the search. Each Z_i is bounded by N as in sequential HBFS so
 124 that no worker takes too long.

125 The pseudo-code of the Master (resp. Worker) is given in Algorithm 2 (resp. Alg. 3).
 126 In the implementation, we avoid to send the same solution twice to a Worker. Moreover,
 127 workers send their solution only if it improves compared to the last solution sent by the
 128 Master. This strategy allows to shorten messages in the Master-Worker protocol.

```

Function HBFS-Master( $clb, cub, S$ ): integer ; /*  $S$  queue of workers, return the optimum */
   $open := \{\nu(\delta = \emptyset, lb = clb)\}$  ; /* Initializes the open list with a root node */
   $I := S$  ; /* Queue of idle workers */
   $A := \emptyset$  ; /* Maps active workers to open nodes currently being processed */
  while (( $open \neq \emptyset$  or  $A \neq \emptyset$ ) and  $clb < cub$ ) do
    while ( $open \neq \emptyset$  and  $I \neq \emptyset$ ) do
       $\nu := \text{pop}(open)$  ; /* Chooses a node with minimum lower bound and maximum depth */
       $i := \text{popFront}(I)$  ; /* Unqueue the first idle worker */
       $A := A \cup \{(i, \nu)\}$  ;
      Send  $\nu$  and best solution  $cub$  to Worker  $i$  ;
    9 Receive a list of open nodes  $\mathcal{V}$  and solution  $cub'$  by worker  $j$  ; /* Wait for message */
      push( $open, \mathcal{V}$ ) ; /* Adds worker open nodes to the Master open list */
       $cub := \min(cub, cub')$  ; /* Checks if a better solution as been found */
    10 pushBack( $I, j$ ) ; /* Pushes Worker  $j$  at the end of the idle worker queue  $I$  */
    11  $A := A \setminus \{(j, A[j])\}$  ; /* Removes Worker  $j$  from active workers */
       $clb := \max(clb, \min(lb(open), \min\{lb(\nu) \text{ for } (i, \nu) \in A\}))$  ; /* Global lower bound */
  return  $cub$ ;
```

■ **Algorithm 2** Parallel HBFS-Master. Initial call for p workers: HBFS-Master($c_0, k, (1, \dots, p)$).

129 **3.1 Improving the ramp-up phase**

130 We observed that at the beginning of the search the first active worker may take a long time
 131 to build its list of open nodes when it reaches the initial backtrack limit (equal to one). It
 132 can be explained by the fact that if it found a new solution then this improved upper bound
 133 will possibly imply more work in subsequent propagation made later when assessing the
 134 lower bound of each open node. This has the effect to slow-down the construction of the list

```

117 Procedure HBFS-Worker(cub,rank) ;                               /* rank: Worker ID */
118   while (true) do
119     openi := ∅ ;                                             /* local open list of Worker i */
120     Receive an open node  $\nu$  and solution cub' by Master ;    /* Wait for message */
121     cub := min(cub, cub') ;                                  /* Updates cub and best solution if any */
122     Restores state  $\nu, \delta$ , leading to assignment  $A_\nu$ , maintaining soft local consistency ;
123     NodesRecompute := NodesRecompute +  $\nu$ .depth ;
124     cub :=DFS( $A_\nu, cub, Z_i$ ) ; /* Increase Nodes ; put all right open branches in openi */
125     if (NodesRecompute > 0) then
126       if (NodesRecompute/Nodes >  $\beta$  and  $Z_i \leq N$ ) then  $Z_i := 2 \times Z_i$ ;
127       else if (NodesRecompute/Nodes <  $\alpha$  and  $Z_i \geq 2$ ) then  $Z_i := Z_i/2$ ;
128     Send openi and best solution cub to the Master ;      /* or closing-node mes. in burst
129     mode */

```

■ **Algorithm 3** Parallel HBFS-Worker. Initial call for Worker i : HBFS-Worker(k, i) with $Z_i = 1$.

of open nodes when HBFS stops backtracking. During this period, called the *ramp-up phase* (where some workers have not been assigned at least one task), no parallelism is exploited. We modified our communication protocol to send a message to the master as soon as an open-node has been collected or a new solution has been found by a worker inside its DFS subroutine (line 12). Such messages are received by the Master (line 9) which does not change the Worker state to idle (lines 10 and 11) until it receives a closing-node message by the Worker (sent at line 15). By doing so, it allows the Master to distribute open nodes to idle workers earlier before the first active worker has finished its initial DFS. We call this modified Master-Worker protocol the *burst mode*. However, the Worker can potentially send $O(nd)$ more messages and it disallows data compression of the open list messages.¹

4 Experimental Results

We implemented in C++ our parallel HBFS in the CFN solver *toulbar2*.² We used the boost MPI library for the Master-Worker communication protocol. We kept default parameters of *toulbar2* except no dichotomic branching in order to explore a binary search tree with DFS (option `-d:`). The variable ordering heuristic is *dom/wdeg* [4] combined with *last conflict* [14]. The value ordering heuristic exploits the last solution found if any [7] or else EDAC existential value [6]. EDAC is also used as soft local consistency during search. Instances were preprocessed by VAC [5] and the resulting CFNs saved to files before the experiments to reduce the setup sequential time of parallel HBFS. We compared both the sequential and parallel version of HBFS and also with the integer programming solver *cplex* (version 20.1 with non-premature stop parameters `EPAGAP=EPGAP=EPINT=0`). We set the number of threads used by *cplex* to the desired number of cores.

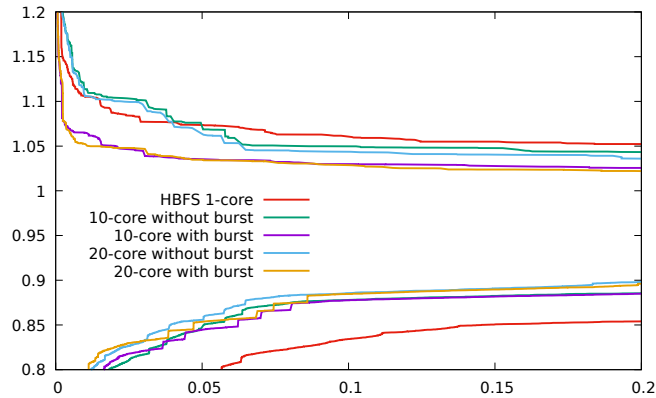
Experiments were performed either on medium-scale computers (24-core Intel Xeon E5-2687W v4 at 3 GHz and 256 GB) with 1-hour timeout or on a large-scale cluster with more than 10,000 cores (36-core per node of Intel Skylake 6140 at 2.3 GHz and 192 GB) with a longer 10-hour timeout for the sequential version only. Solving times are reported in seconds and correspond to CPU (resp. wall-clock) time for the sequential (resp. parallel) methods. No initial upper bounds were provided.

¹ In non-burst mode, all right branches share a common prefix in their ν, δ and only the deepest δ information need to be sent to the Master.

² <https://toulbar2.github.io/toulbar2> version 1.2.

163 We tested the methods on four benchmarks selected from [12] with a total of 134 instances:
 164 two academic benchmarks taken in Operations Research, uncapacitated warehouse location
 165 problem (Warehouses) with 15 instances [13] and DIMACS maximum clique problem with
 166 62 instances (MaxClique)³ and two real-life Graphical Model benchmarks, linkage analysis
 167 problem occurring in genetics (Linkage) with 22 instances coming from UAI Evaluation 2008⁴
 168 and computational protein design problem in biology (CPD) with 35 instances [1]. We
 169 applied the *tuple encoding* to convert Linkage and CPD to integer linear programs [12]. For a
 170 comparison on MaxClique with another parallel branch and bound implementation, see [16].

171 4.1 Comparison of parallel HBFS with its sequential version



■ **Figure 2** Comparison on a medium-scale computer between sequential versus parallel HBFS with or without burst mode. The x-axis represents normalized time (with 0.2 corresponding to 720 seconds). The y-axis corresponds to normalized lower and upper bounds on 134 instances (with 1 corresponding to the optimum or best known cost, see the text description).

172 We compared the anytime behavior of sequential (HBFS-1) and parallel HBFS (with 10
 173 or 20 cores) with or without burst mode (see Sec. 3.1) on a medium-scale computer. We
 174 summarize the evolution of lower (*clb* in Alg. 1 and 2) and upper bounds (*cub*) for each
 175 method over all instances in Fig. 2. Specifically, for each instance we normalize all costs
 176 as follows: the initial lower bound c_0 produced by EDAC is 0; the best but potentially
 177 suboptimal solution found by any method is 1; the worst solution is 2. This normalization
 178 is invariant to translation and scaling. Additionally, we simply normalize time from 0 to
 179 1, corresponding to 1 hour. A point x, y on the lower bound line for method M in Fig. 2
 180 means that after normalized runtime x , method M has proved on average over all instances
 181 a normalized lower bound of y and similarly for the upper bound.

182 First, we observed that all parallel versions significantly outperformed the sequential
 183 HBFS lower bound curve. Concerning upper bound curves, the burst mode gave a clear
 184 advantage to parallel HBFS especially at the beginning of the search. In the sequel of the
 185 paper, we always report results of parallel HBFS with burst mode. As shown in the figure,
 186 increasing the number of cores from 10 to 20 slightly improved the bounds.

187 In Table 1 we report the number of instances solved by sequential and parallel HBFS
 188 for each benchmark. Parallel HBFS solved 1 more instance than the single core version in

³ We removed the largest instances keller6 and p_hat1500-1,2,3 from the original 66 DIMACS instances.

⁴ Linkage instances were further preprocessed by variable elimination limited to at most 8 neighbors [8].

189 Linkage and 1 (resp. 3) in MaxClique using 10 (resp. 20) cores. We made local comparisons
 190 of solving times (shown in parentheses) by averaging on the subset of instances solved by the
 191 three methods (HBFS-1, HBFS-10, HBFS-20). It allows us to display overall speed-up of
 192 parallel approaches by giving the ratio of total sequential over parallel time. Parallel HBFS
 193 obtained near linear speed-up on MaxClique. Recall that 1 core is used by the master and the
 194 rest by the workers in the Master-Worker approach preventing us from full linear speed-up.
 195 On CPD and Linkage the speed-up was halved. For Warehouses, only 50% of reduction in
 196 overall time was observed. This can be explained partly by the limited number of search
 197 nodes (Table 4 in Supplementary Material). We also observed that the evaluation of right
 198 branches made by the first active worker starting from the root node took most of the time.
 199 This is due to the fact that a first solution has been found by the worker resulting in more
 200 propagation on the right branches especially near the root. This pathological phenomenon
 201 did not appear on the other benchmarks.

| Method | CPD (35) | | Warehouses (15) | | Linkage (22) | | MaxClique (62) | |
|--------------------|------------------|-----------------|--------------------|-----------------|-------------------|-----------------|----------------------|-----------------|
| | | <i>Speed-up</i> | | <i>Speed-up</i> | | <i>Speed-up</i> | | <i>Speed-up</i> |
| HBFS-1 | 30 (43.44s) | | 15 (128.96s) | | 20 (23.24s) | | 37 (364.25s) | |
| HBFS-10 | 30 (8s) | <i>5.43</i> | 15 (80.174s) | <i>1.61</i> | 21 (3.5s) | <i>6.64</i> | 38 (40.24s) | <i>9.05</i> |
| HBFS-20 | 30 (4.43s) | <i>9.81</i> | 15 (85.39s) | <i>1.51</i> | 21 (2s) | <i>11.62</i> | 40 (19.9s) | <i>18.3</i> |
| cplex-1 | 24 (331.2s) | | 15 (123.83s) | | 22 (8.04s) | | 42 (282.16s) | |
| cplex-10 | 24 (226.51s) | <i>1.46</i> | 15 (68.82s) | <i>1.8</i> | 22 (2.56s) | <i>3.14</i> | 45 (55.48s) | <i>5.08</i> |
| cplex-20 | 24 (198.49s) | <i>1.67</i> | 15 (72.06s) | <i>1.72</i> | 22 (2.29s) | <i>3.51</i> | 46 (71.47s) | <i>3.95</i> |
| HBFS-1 (cluster) | 30 (66.46s) | | 15 (392.30s) | | 21 (427.21s) | | 37 (504s) | |
| HBFS-180 (cluster) | 30 (3.7s) | 17.96 | 15 (126s) | 3.11 | 22 (4.15s) | 102.94 | 45 (6.44s) | 78.26 |

■ **Table 1** Number of solved instances within 1 hour (except for sequential HBFS-1 run on the cluster with a larger timeout of 10 hours) and average time in seconds in parentheses. To compute the mean we only consider for a given method (toulbar2 HBFS or cplex) the instances solved with any number of cores on the same computer (server with 3 GHz cores or cluster with 2.3 GHz cores).

202 4.2 Comparison of parallel HBFS with integer programming

203 In Table 1 we also report the number of instances solved and their average solving time
 204 (as explained above) by cplex using multithreading. It clearly dominates HBFS on Linkage
 205 (Supp. Fig. 5). For Warehouses, the differences are less important still in favor of cplex.
 206 For MaxClique, although the global picture shows that it solved six more instances than
 207 HBFS with 20 cores, both methods performed well on different subsets of instances (e.g.,
 208 HBFS-20 solved two instances – brock400_4 and sanr400_0.7 – unsolved by cplex-20 whereas
 209 cplex-20 solved eight instances unsolved by HBFS-20). For CPD, the CFN approach largely
 210 dominates the integer programming approach for all the instances. Concerning anytime
 211 curves shown in Fig. 3 (see also Supp. Fig. 4 and 5), the CFN approach is also significantly
 212 superior to cplex on average in producing good upper bounds faster, HBFS-20 being the best
 213 method. Concerning overall speed-up, cplex had difficulties to benefit from parallelism on
 214 CPD, Linkage, and Warehouses where it usually develops a small amount of search nodes
 215 (less than 7,059 nodes except on Linkage/pedigree19 and pedigree40), resulting in poor
 216 speed-up except in a few cases. The speed-up is better on MaxClique but seems to stagnate
 217 when going from 10 to 20 cores (it was even slower on four instances).

218 **4.3 Comparison of parallel HBFS with EPS on a cluster**

219 The EPS approach is a two-phase procedure. First, the problem to be solved is decomposed
 220 into a list of l independent subproblems. Next, all the subproblems are solved in parallel
 221 (with at most p workers running at the same time) based on a particular scheduling strategy
 222 with no communication between the workers. For optimization problems, we need to provide
 223 a good initial upper bound. Otherwise the search tree can be much larger than needed. In
 224 the first phase, we used the original HBFS method to collect l subproblems. As soon as
 225 HBFS has more than l open nodes in its frontier it stops and returns the current upper
 226 bound (cub) and the list of open nodes (without those having a lower bound $lb(\nu) \geq cub$).
 227 Each open node ν defines an independent subproblem with partial assignment $\nu.\delta$. In order
 228 to collect open nodes more rapidly we fix the (maximum) backtrack limit $Z = N = 1$. Ideally
 229 l should be $30 \times p$ with p the number of available cores [15]. In the second phase, we schedule
 230 on the cluster the subproblems that are solved by the original HBFS method using a simple
 231 scheduling heuristic based on increasing $|\nu.\delta|$.

232 In Table 2 we report for nine difficult instances their optimum value, the upper bound
 233 found at the end of EPS Phase-1, the actual number of generated subproblems, the average
 234 solving time of all subproblems, the maximum solving time, the number of failed subproblems
 235 (timeout of 1 hour) and the overall solving time of EPS Phase-2 using 180 cores on the cluster.
 236 We compare with HBFS using the same number of cores. Our EPS strategy failed on 4/9
 237 instances. In parentheses, we indicate the maximum depth $\nu.depth$ of failed subproblems.
 238 Clearly, finding the right number l of not-too-difficult subproblems corresponding to partial
 239 assignments greater than a given depth is a challenging task. In our experiments, we tried
 240 with different values for $l \in [50, 6000]$, selecting the largest threshold value with a Phase-1
 241 duration being less than 1 second for Linkage ($l = 6000$), 6 seconds for MaxClique ($l = 6000$)
 242 and 44 seconds for CPD ($l = 1000$). On the opposite, we did not tune any specific parameter
 243 for our parallel HBFS method.

244 In Table 1 we also report the overall speed-up of HBFS-180 compared to HBFS-1 on the
 245 cluster. HBFS-180 got a two-order-of-magnitude speed-up on Linkage.

246 **5 Conclusion**

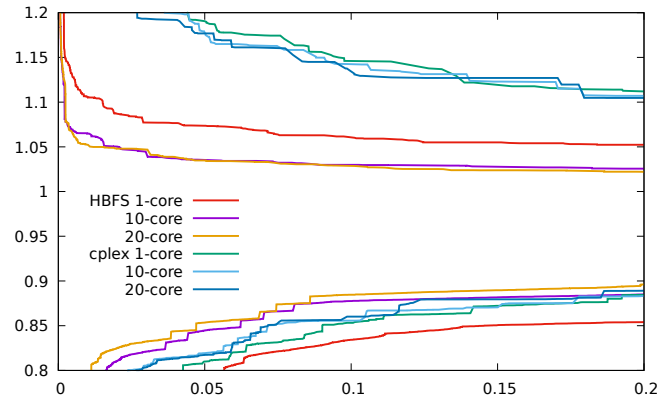
247 Although the speed-up offered by the parallel version of HBFS was very instance dependent,
 248 we observed significant gain on several instances, outperforming in some cases state-of-the-
 249 art solvers like `cplex`. Even if the scalability of our approach must be subject of deeper
 250 investigation, due to the minimal size of the information shared between the Master and the
 251 Workers, our approach is very likely compliant with a larger number of cores.

252 A more challenging task which remains as future work is to exploit the structure of
 253 CFNs by parallelizing Backtrack with Tree Decomposition (BTD-HBFS) [2]. Shared memory
 254 protocols may be more suitable for this task to make learnt nogoods available to all Workers.

255 On the practical side, our parallel HBFS could ran in conjunction with a parallel large
 256 neighborhood search strategy [20] offering even better anytime lower and upper bounds.

257 **References**

- 258 1 D Allouche, J Davies, S de Givry, G Katsirelos, T Schiex, S Traoré, I André, S Barbe,
 259 S Prestwich, and B O’Sullivan. Computational protein design as an optimization problem.
 260 *Artificial Intelligence*, 212:59–79, 2014.



■ **Figure 3** Comparison on a medium-scale computer between toulbar2 using parallel HBFS (with burst mode) and cplex using multiple threads. The x-axis represents normalized time (with 0.2 corresponding to 720 seconds). The y-axis corresponds to normalized lower and upper bounds on 134 instances (with 1 corresponding to the optimum or best known cost, see the text description).

| instance | n | d | opt. | cub | l | av. time | max. t. | #fail(depth) | EPS-180 | HBFS-180 |
|---------------------|-----|-----|---------|---------|------|----------|---------|--------------|---------|-------------|
| linkage/pedigree19 | 259 | 5 | 4625 | 5684 | 5114 | 20.57 | - | 1 (4) | - | 69.1 |
| linkage/pedigree40 | 274 | 6 | 7300 | 8838 | 5641 | 101.99 | - | 49 (21) | - | 1680 |
| linkage/pedigree51 | 295 | 5 | 6406 | 6802 | 5798 | 0.61 | 497.38 | 0 | 499 | 5.7 |
| cpd/1BRS | 38 | 178 | 4007610 | 4007679 | 956 | 2.94 | 38.90 | 0 | 44 | 37.5 |
| cpd/1CDL | 38 | 170 | 3590514 | 3590825 | 1001 | 6.66 | 79.04 | 0 | 79 | 18.3 |
| cpd/1GVP | 52 | 170 | 5196719 | 5196841 | 979 | 14.59 | 170.66 | 0 | 171 | 17.0 |
| maxcl./brock400_1 | 400 | 2 | 373 | 379 | 6010 | 63.95 | - | 12 (149) | - | 1812 |
| maxcl./brock400_2 | 400 | 2 | 371 | 379 | 5975 | 65.27 | - | 18 (149) | - | 880 |
| maxcl./san400_0.5_1 | 400 | 2 | 387 | 392 | 6073 | 5.07 | 414.96 | 0 | 3652 | 1220 |

■ **Table 2** EPS and HBFS-180 results on hard instances (with n variables and maximum domain size d). A '-' indicates that some (see #failed) subproblems could not be solved in less than 3,600sec.

- 261 2 D Allouche, S de Givry, G Katsirelos, T Schiex, and M Zytnicki. Anytime Hybrid Best-First
262 Search with Tree Decomposition for Weighted CSP. In *Proc. of CP-15*, pages 12–28, Cork,
263 Ireland, 2015.
- 264 3 D. Allouche, S. de Givry, and T. Schiex. Towards parallel non serial dynamic programming
265 for solving hard weighted csp. In *Proc. of CP-10*, St Andrews, Scotland, 2010.
- 266 4 F Boussemart, F Hemery, C Lecoutre, and L Sais. Boosting systematic search by weighting
267 constraints. In *ECAI*, volume 16, page 146, 2004.
- 268 5 M. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki, and T. Werner. Soft arc
269 consistency revisited. *Artificial Intelligence*, 174(7–8):449–478, 2010.
- 270 6 S. de Givry, M. Zytnicki, F. Heras, and J. Larrosa. Existential arc consistency: Getting
271 closer to full arc consistency in weighted csps. In *Proc. of IJCAI-05*, pages 84–89, Edinburgh,
272 Scotland, 2005.
- 273 7 E Demirovic, G Chu, and P J. Stuckey. Solution-based phase saving for CP: A value-selection
274 heuristic to simulate local search behavior in complete solvers. In *Proc. of CP-18*, pages
275 99–108, Lille, France, 2018.
- 276 8 A Favier, S de Givry, A Legarra, and T Schiex. Pairwise decomposition for combinatorial
277 optimization in graphical models. In *Proc. of IJCAI-11*, Barcelona, Spain, 2011.
- 278 9 I Gent, I Miguel, P Nightingale, C McCreesh, P Prosser, N Moore, and C Unsworth. A
279 review of literature on parallel constraint solving. *Theory and Practice of Logic Programming*,
280 18(5-6):725–758, 2018.
- 281 10 C. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization.
282 In *Proc. of AAAI’98*, Madison, WI, 1998.
- 283 11 W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proc. of IJCAI’95*, Montréal,
284 Canada, 1995.
- 285 12 B Hurley, B O’Sullivan, D Allouche, G Katsirelos, T Schiex, M Zytnicki, and S de Givry. Multi-
286 Language Evaluation of Exact Solvers in Graphical Model Discrete Optimization. *Constraints*,
287 21(3):413–434, 2016.
- 288 13 J Kratica, D Tošić, V Filipović, and I Ljubić. Solving the simple plant location problem by
289 genetic alg. *RAIRO*, 35(1):127–142, 2001.
- 290 14 C. Lecoutre, L Saïs, S. Tabary, and V. Vidal. Reasoning from last conflict(s) in constraint
291 programming. *Artificial Intelligence*, 173:1592,1614, 2009.
- 292 15 A Malapert, J-C Régim, and M Rezgui. Embarrassingly parallel search in constraint program-
293 ming. *Journal of Artificial Intelligence Research*, 57:421–464, 2016.
- 294 16 C McCreesh and P Prosser. The shape of the search tree for the maximum clique problem and
295 the implications for parallel branch and bound. *ACM Trans. Parallel Comput.*, 2(1), 2015.
- 296 17 P. Meseguer, F. Rossi, and T. Schiex. Soft constraints processing. In F. Rossi, P. van Beek,
297 and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 9. Elsevier, 2006.
- 298 18 L Michel, A See, and P Van Hentenryck. Parallelizing constraint programs transparently.
299 In C Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, pages
300 514–528, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- 301 19 L Otten and R Dechter. And/or branch-and-bound on a computational grid. *JAIR*, 59:351–435,
302 2017.
- 303 20 Abdelkader Ouali, David Allouche, Simon de Givry, Samir Loudni, Yahia Lebbah, Francisco
304 Eckhardt, and Lakhdar Loukil. Iterative Decomposition Guided Variable Neighborhood Search
305 for Graphical Model Energy Minimization. In *Proc. of UAI-17*, pages 550–559, Sydney,
306 Australia, 2017.
- 307 21 T Ralphs, Y Shinano, T Berthold, and T Koch. Parallel solvers for mixed integer linear
308 optimization. In *Handbook of parallel constraint reasoning*, pages 283–336. Springer, 2018.