When the Decomposition Meets the Constraint Satisfaction Problem

YOUCEF DJENOURI[ID][1]**, DJAMEL DJENOURI**[2,3]**, ZINEB HABBAS**[4]**,**
JERRY CHUN-WEI LIN[ID][5]**, (Senior Member, IEEE),**
TOMASZ P. MICHALAK[6]**, AND ALBERTO CANO**[ID][7]**, (Senior Member, IEEE)**

[1]SINTEF Digital, 0314 Oslo, Norway
[2]Department of Computer Science and Creative Technologies, University of the West of England, Bristol BS16 1QY, U.K.
[3]CERIST Research Center, Algiers 16000, Algeria
[4]Informatique Department, Lorraine University, 54000 Metz, France
[5]Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, 5063 Bergen, Norway
[6]Institute of Informatics, Warsaw University, 00-927 Warsaw, Poland
[7]Department of Computer Science, Virginia Commonwealth University, Richmond, VA 23284-2512, USA

Corresponding author: Alberto Cano (acano@vcu.edu)

ABSTRACT This paper explores the joint use of decomposition methods and parallel computing for solving constraint satisfaction problems and introduces a framework called Parallel Decomposition for Constraint Satisfaction Problems (PD-CSP). The main idea is that the set of constraints are first clustered using a decomposition algorithm in which highly correlated constraints are grouped together. Next, parallel search of variables is performed on the produced clusters in a way that is friendly for parallel computing. In particular, for the first step, we propose the adaptation of two well-known clustering algorithms (k-means and DBSCAN). For the second step, we develop a GPU-based approach to efficiently explore the clusters. The results from the extensive experimental evaluation show that the PD-CSP provides competitive results in terms of accuracy and runtime.

INDEX TERMS CSP, decomposition, scalability, GPU.

I. INTRODUCTION

A Constraint Satisfaction Problem (CSP) is defined as a triple $P = <X, D, C>$ where $X = \{x_1, x_2, \ldots, x_n\}$ is a set of n variables, $D = \{d_1, d_2, \ldots, d_n\}$ is a set of finite domains of the variables, i.e., each variable x_i takes its value in a domain d_i, $C = \{c_1, c_2, \ldots, c_m\}$ is a set of m constraints. Each constraint, c_i, is a pair $(S(c_i), R(c_i))$ where $S(c_i) \subseteq X$ is a subset of variables called *the scope* of c_i and $R(c_i) \subseteq \prod_{x_k \in S(c_i)} d_k$ is *the constraint relation* that specifies the legal combinations of values. Many problems in artificial intelligence can be represented as CSPs. For example, scene labeling in computer vision involves testing a possible interpretation of objects against relation rules [1]. Other constraint satisfaction problems include theorem proving [2], scheduling [3], inference relations [4], and expert systems [5], [6]. Typically, as the CSP itself, these problems are NP-Complete and require extensive search over the solution space. The canonical exact method for solving CSPs is the backtrack search which performs a systematic exploration of a search tree until finding a total instantiation of values to variables that satisfies all the constraints [7]–[9]. However, since CSP is NP-Complete, it is extremely difficult to have an efficient general search algorithm. In fact, while generic solvers are sometimes surprisingly competitive, in many situations they are defeated because of some particularly difficult subsets of constraints. The attempts to address this drawback of enumerative search CSP algorithms can be divided into:

- Improvements based on **decomposition**, which is a data mining technique to divide hard problem instances into many independent but solvable subinstances. As a motivating example, Figure 1 shows that in some cases half of the of variables are shared by just a little bit more than 10% of constraints.[1] This suggests to treat these subgroup of constraints separately, solve this subgroup as a subproblem, and then use the obtained values of

The associate editor coordinating the review of this manuscript and approving it for publication was Zhaojie Ju [ID].

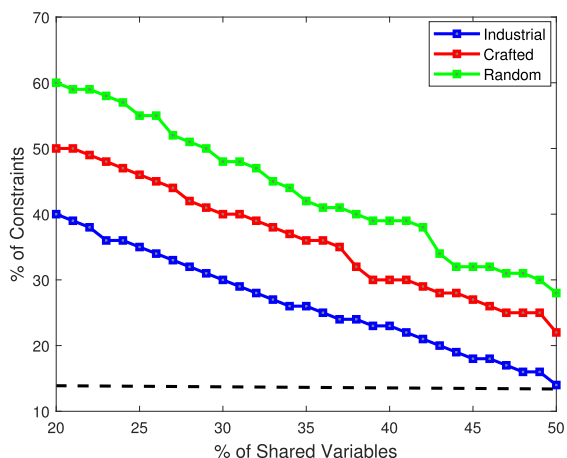[1]https://maxsat-evaluations.github.io/2018/benchmarks.html

FIGURE 1. The percentage of shared variables of CSP instances.

the shared variables when solving the remaining constraints. While the CSP algorithms improved in this spirit typically used graph-based decomposition techniques, this approach has been recently strengthened by more advanced data-mining techniques, and tree decomposition [10]–[12]. The main idea is to treat the clustering process as a preprocessing phase to extract as much relevant information as possible about the problem at hand and its resulting subproblems. This information is then used to enhance exploration of each subproblem. These data mining-based solutions are faster than the traditional approaches. For instance, [11] showed that their k-means-DPLL algorithm is 13 times faster than the classical DPLL algorithm for a random instance of 220 variables.

- Improvements based on **parallelization**. In order to fully utilize recent technical developments in hardware, the algorithms are adapted to exploit high-performance computing architectures, such as Graphics Processing Units (GPU). GPU is a piece of hardware consisting of multiple computational entities, or "cores", just like a standard CPU, designed mainly for rendering 3D computer graphics [13]. Several solutions based on GPU computing have been recently developed for solving CSP problems [14]–[16]. For instance, MaxSAT [14] is 56 times faster than the classical DPLL for solving the same instance of 220 variables mentioned above.

However, while both the above directions are promising, they have never been combined together to resolve CSP. In other words, to the best of our knowledge, no solution in the current literature is based on both data mining and high-performance computing techniques. The challenge is to develop an efficient combination of a clustering technique (that minimizes the number of shared variables between clusters) and a parallel search technique (that is capable of using the information obtained from clusters in a productive way).

To fill this gap in the literature, this paper proposes the first CSP approach that takes advantage of both data mining and high-performance computing techniques on the GPU. Our framework, called Parallel Decomposition for Constraint Satisfaction Problems (PD-CSP), includes decomposition methods to split a given CSP into a collection of smaller sub-problems related pair-wise by a set of shared variables and a dedicated approach for parallel search process. We adapt data mining clustering algorithms to decompose the set of constraints into highly correlated clusters and in the way that can be readily used in the parallel search process. In this work, we focus on two well-known algorithms: k-means [17] and DBSCAN [18]. We also propose a dedicated parallel search strategy that uses the clusters for instantiating the variables. This strategy takes into account the variables shared between clusters. While our framework is general, we present a use case on a GPU architecture. We show how to take advantage of the GPU massively threaded programming model by providing efficient mapping between the GPU blocks and the clusters of constraints. We evaluate our approach by studying its time complexity and comparing it with some existing CSP solvers applied to different CSP problems. Two case studies are presented applying PD-CSP to i) *MAXSAT*, and ii) to *Job Scheduling* problems. The results show that our approach advances the state-of-the-art in terms of both the runtime and effectiveness. For instance, PD-CSP is 105 times faster than DPLL when dealing the random instance of 220 variables. Moreover, the efficient GPU implementation achieves speedup factors, up to 350x for the *MAXSAT* problem and 480x for the *Job Scheduling* problem on a single GPU machine.

II. RELATED WORK

Constraint Satisfaction Problems (CSP) have attracted interest in the research community for a long time, and many algorithms have been proposed to date. The first canonical CSP algorithm is Davis Putnam Logemann Loveland (DPLL) [19] which uses heuristic techniques to solve hard constraints including the SAT problem. It starts by designing the tree's solver and instantiating one variable at each level. The process is repeated until the possible values of all variables are explored or all constraints are satisfied. DPLL-based methods [7], [20]–[22] are enumerative and they are successive versions of backtracking strategy. As mentioned in the introduction, the attempts to address the drawback of enumerative search algorithms can be divided into the following categories:

A. DECOMPOSITION-BASED SOLUTIONS

Several decomposition methods of CSPs have been proposed to date, e.g., tree decomposition techniques [23]–[25] that have good theoretical properties but their practical efficiency has not been proven. Generally speaking, the decomposition methods for CSP can be naturally derived from graph partitioning techniques or graph clustering techniques. This is because the structure of a CSP can be captured by a constraint graph where the nodes represent the variables and the edges correspond to the constraints. The exact decomposition

methods such as the tree decomposition [25] and the hyper-tree decomposition [26] are not practical. On other hand, clustering techniques combined with heuristics associated with dedicated functions representing the quality of the decomposition were shown to be promising alternatives [11].

B. PARALLEL-BASED SOLUTIONS

Parallel approaches for solving CSP problems [27]–[32] commence with the partitioning of the search tree for parallel solving search, i.e the search tree is split into a collection of subtrees that will be distributed later among different processors. The crucial point leading to efficiency of parallel algorithms is the load balancing issue. The question here is to choose between static or dynamic load balancing. In the static case, an initial partitioning of the root nodes in the search tree is performed before the search is initiated. The problem is then divided into a number of subproblems pools; one pool for each processor. Each processor runs a sequential search algorithm on the pool assigned to it. In the dynamic case, the subproblems are dynamically distributed. The different dynamic load balancing strategies in the context of parallel Backtracking are studied in [33], [34]. These methods have been intensively studied from theoretical point of view. Unfortunately, these methods have not shown to be practical when facing real world problems. Parallel approaches can be grouped into two categories:

i) The first one commences with the partitioning of the search tree for parallel solving search [27], [28], i.e., the search tree is split into a collection of subtrees that will be distributed later among different processors. The crucial point leading to efficiency of parallel algorithms is the load balancing issue. The question here is to choose between static or dynamic load balancing. In the static case, an initial partitioning of the root nodes in the search tree is performed before the search is initiated. The problem is then divided into a number of subproblems pools; one pool for each processor. Each processor runs a sequential search algorithm on the pool assigned to it. In the dynamic case, the subproblems are dynamically distributed. Different dynamic load balancing strategies in the context of parallel Backtracking are studied in [33], [34].

ii) The second approach is parallel solving on shared men [29], [35], [36], which consists in splitting the problem to be solved into sub-problems and proposing algorithms to solve the sub-problems separately and then to combine the solutions of the sub-problems to find the global one. These methods have been intensively studied from theoretical point of view. Unfortunately, these methods have not shown to be practical when facing real world problems.

C. DISCUSSION

The existing CSP-based solutions only consider accurate decomposition methods such as the tree and hypertree decomposition mentioned above. These methods generate huge graphs with a large number of shared variables between different vertices. It is hard to make these methods run efficiently in a parallel way, and proceed independent jobs on the whole resulting graph. Moreover, the parallel CSP-based solutions consider a naive partitioning of the whole constraints, which degrades the overall performance of such methods in terms of load balancing and the communication between different workers. To address these challenging issues, we propose in the next section a new parallel approach that explore the different correlation provided by clustering algorithms for solving CSP problems.

III. THE GENERIC PD-CSP FRAMEWORK

In this section we first present the overview of the PD-CSP framework (See Fig. 2 for more details) and then we detail its components. Finally, we analyze its theoretical complexity.
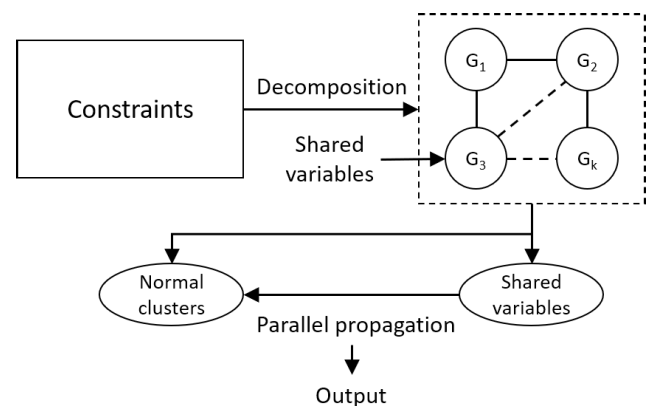

FIGURE 2. PD-CSP framework.

A. OVERVIEW

PD-CSP is composed of two main steps: *decomposition* and *parallel propagation*. In the first step, the constraints are divided into homogeneous groups using data mining decomposition techniques, where a group may be viewed as a subset of the whole set of constraints. An interesting decomposition approach is to minimize the number of the shared variables between clusters, while gathering in the same cluster constraints that are highly correlated, i.e., those that share the maximum number of variables. For each cluster, four types of information are stored: the constraints in the cluster, the variables shared among them, the variables shared with other clusters, and the conflicting constraints. These pieces of information are then used in the second step to establish parallel search when the best instantiation of variables is searched for.

B. STEP 1: DECOMPOSITION

A set of constraints C is decomposed into several groups, $G = \{G_i\}$, where each group, G_i, is a subset of constraints such that $G_i \cap G_j = \emptyset$ for any $i \neq j$. We denote the set of variables of group G_i by $\mathcal{X}(G_i)$ and define it as

$\mathcal{X}(G_i) = \{\bigcup \mathcal{X}(C_j) \setminus C_j \in G_i\}$. We define G as the set of groups of the constraints C. We define F as the set of conflict constraints, the constraints of different clusters sharing variables among them. Suppose that groups in G do not share any variables, which means $\forall (i, j) \in [1 \ldots k]^2 \; \mathcal{X}(G_i) \cap \mathcal{X}(G_j) = \emptyset$. This yields the following proposition:

Proposition 1: $L = \{\bigcup_{i=1}^{k} L_i\}$ *where* L_i *is the set of all instantiations of variables of the cluster* G_i.

Proof: Consider $\forall (i, j) \in [1 \ldots k]^2 \; \mathcal{X}(G_i) \cap \mathcal{X}(G_j) = \emptyset$. We have $\forall i \in [1 \ldots k]: L_i = \{v \in D \mid Consistent(C, X, v)\}$. Note that *Consistent(C, X, v)* is valid if we instantiate the variable X by a value v, the set of constraints became satisfied or at least remain consistent. The consistency of the instantiation of v is based on its consistency in the whole set of constraints C. The value of v have to be checked in all constraints in C, and constraints satisfying v should be returned for further processing. Now, consider a variable x, the value v exists in $\mathcal{X}(G_i)$, $\Rightarrow x \notin \mathcal{X}(G_j)$, $(\forall j \in [1 \ldots k], \vee j \neq i)$ $\Rightarrow x \nsubseteq \mathcal{X}(G_j) \Rightarrow L_i = \{v \in D \mid Consistent(C, X, v)\} \Rightarrow L = \{\bigcup_{i=1}^{k} L_i\}$. □

If the whole set of constraints is decomposed as per Proposition 1, we will end up with independent clusters, i.e., no cluster of constraints share variables with any other cluster. Then, the clusters could be solved separately. Unfortunately, this is rather unlikely in the general case.

Given the above, our aim is to develop a decomposition technique that does not necessarily look for a decomposition with completely independent clusters but rather a decomposition where the number of shared variables, and the conflict constraints between the clusters are minimized. To this end, we propose a multi-objective k-means algorithm to efficiently decompose the constraints, and minimize both the shared variables and the conflict constraints.

Definition 1 (Constraints Similarity): We define constraints similarity by adopting the Jaccard similarity as

$$\mathcal{J}_C(C_i, C_j) = \frac{\sum_{x \in (C_i \cap C_j)} Sim(C_i, C_j, x)}{|C_i| + |C_j| + \sum_{x \in (C_i \cap C_j)} Sim(C_i, C_j, x)} \quad (1)$$

Note that $Sim(C_i, C_j, x) = 1$, *if* $x^i = x^j$, 0, *otherwise.* x^i *is the value of the variable* x *in the constraint* C_i.

Definition 2 (Centroids): We define the centroids of the group of constraints G_i, denoted $\overline{G_i}$, by $\overline{G_i} = \{\bigcup \max_{x_l}(x_l^i) \mid x_l \in \mathcal{X}(G_i)\}$, where $\max_{x_l}(x_l^i)$ is the highest value of the variable x_l in group G_i.

The multi-objective k-means for constraints decomposition aims to optimize the function $J = \sum_{j=1}^{k} \sum_{t \in G_j} |t - \mu_j|^2$ where μ_j is the centroid of constraints in G_j. First, the constraints are assigned randomly to k clusters. Second, a centroid is computed for each cluster. Third, every constraint is assigned to a cluster with the closest centroid. The latter two steps are repeated until there is no further assignment of the constraints to the clusters. The output of our decomposition step is a triplet, $< G, S, F >$, where G is the set of clusters of constraints, S is the set shared variables—the element $s^{ij} \in S$, denoted s_l^{ij}, will be used to represent the set

of shared variables between the adjacent clusters (G_i and G_j). F represents the set of the conflict constraints of all clusters in G.

C. STEP 2: PARALLEL PROPAGATION

The clusters derived in the decomposition step are explored in parallel. Indeed, the search process is applied on each cluster independently. To use the output of the decomposition step, i.e., network O, in *any* parallel architecture, we propose the following steps:

Step 2.1: Instantiation of the variables shared between clusters: We begin with instantiating all the variables shared between clusters because they are a potential source of conflict when solving each cluster in parallel. We choose to give the priority of the instantiation to the variables with high degree of correlation which we define as $|\{G' \mid \exists C_i \in G' \vee x' \in C_i\}|$ for any variable x'. In other words, the degree of correlation of x' is the number of clusters in which at least one constraint contains x'. By doing so, we first instantiate variables which potentially may cause the most conflict. This step is performed on the CPU, where a conventional generate-and-test strategy is used to find the best instantiation. All the constraints satisfied the instatiated variables are removed. Now, we are ready to move on to the parallel computation.

Step 2.2: Computing and storing local results: Each parallel node is assigned a unique cluster and it applies a generate-and-test strategy therein, i.e., it generates all the instantiation of variables from that cluster and stores them in the set of all the instantiations of all the variables. This latter set is built following the logic known from the serial implementations [22]. Once the local instantiation of variables are calculated, they are sent to the CPU for further processing.

Step 2.3: Merging The local results: The CPU merges the local instantiation of variables from each cluster into the global solution. A concatenation of all the local instantiations and the best instantiation of the shared variables are used.

D. COMPLEXITY

The time complexity of the PD-CSP framework depends on the clustering algorithm and the CSP solver used. In our case, the complexity of the k-means algorithm [17] or the complexity of DBSCAN algorithm [18] is $O(m \times n)$. As for the mining process, let us denote the complexity of a constraint CSP solver, A, by $O(Cost(A, n, m))$. Note that m and n are the number of constraints and the number of variables, respectively. For the parallel propagation, the instantiation of the variables shared between clusters (Step 2.1) requires $O(2^{|S|})$ in the worst case. The cost for parallel instantiation of the clusters' variables (Step 2.2) is $Cost(A, |G^\star|, |\mathcal{X}(G^\star)|)$, where $G^\star$ is the cluster in G having the maximum number of variables. Thus, the complexity of the PD-CSP is $O((n \times m) + 2^{|S|} + Cost(A, |G^\star|, |\mathcal{X}(G^\star)|))$.

Table 1 compares the complexity of some existing CSP solvers using the PD-CSP framework by varying the function $Cost(A, m, n)$. Note that, the worst complexity is computed by considering all exploration of shared variables and all

TABLE 1. Complexity of the existing constraint satisfaction solvers using the PD-CSP framework.

Solver	Cost(A, m, n)	With PD-CSP
DPLL	$m \times 2^n$	$n \times m + 2^{|S|} + |G^\star| \times 2^{|\mathcal{X}(G^\star)|}$
BSAT	$m \times 1.6181^n$	$n \times m + 2^{|S|} + |G^\star| \times 1.6181^{|\mathcal{X}(G^\star)|}$
AC	$m \times n^3$	$n \times m + 2^{|S|} + |G^\star| \times |\mathcal{X}(G^\star)|^3$

constraints in the designed clusters. From this table, we may conclude that by using the PD-CSP framework, the complexity of all solvers is reduced by considering the cluster noted $G^\star$ having maximum number of variables. In the worst case, we have $|\mathcal{X}(G^\star)| = n$. Tis happens when the number of cluster is set to 1. Then, the shared variable is an empty set. In an average case, $|\mathcal{X}(G^\star)| \ll n$ and the existing CSP solvers perform much better when applied in the PD-CSP framework.

IV. PD-CSP ON THE GPU

This section presents the use of GPU for boosting the PD-CSP performance. The GPU programming model involves two components: the CPU (host) and the GPU (device). The former corresponds to the system's processor and main memory. The latter comprises a massively multi-threading system that consists of multiple computing *cores*, where each core executes a block of threads. Threads of a block in the same core communicate with one another using a shared memory, whereas the communication between blocks relies on a global memory. The CPU/GPU communication is made possible by system buses. The application of the three steps defined above must be carefully designed to fit the hardware at hand. In this section, we case study of our generic approach on the GPU. The degree of correlation of all shared variables, and the best instantiation of these variables are determined on the CPU. The set of designed clusters except the satisfied constraints are then sent to the GPU. Each block of threads is mapped onto one cluster, where the conventional generate-and-test strategy process is applied on each block. If we consider the size of the shared memory of each block is r, the first r non-satisfied constraints of the cluster G_i are allocated to the shared memory of the block, and the remaining non-satisfied constraints of the cluster G_i is allocated to the global memory of the GPU host. We define a *local table*, $table_i$, for storing the instantiation of variables of the cluster G_i. The local table of each cluster is sent to the CPU for further processing. In this context, the CPU host performs the merging step to find the global instantiation of variables by concatenating all local instantiations and the best instantiation of the shared variables. Algorithm 1 presents the pseudo-code of the parallel search using standard GPU-CUDA operations.

From a theoretical standpoint, a GPU implementation improves the serial implementation by exploiting the massively threaded computing of GPUs while instantiating the variables of the non-satisfied constraints in all clusters. It also minimizes the CPU/GPU communication, by defining only two points of CPU/GPU communication. The first one takes place when the non-satisfied constraints is loaded into the

Algorithm 1 Parallel Search on GPU

1: /*********************CPU
    Host************************/
2: **Input:**
    $C = \{C_1, C_2 \ldots, C_m\}$: The set of m constraints
    $X = \{x_1, x_2 \ldots, x_n\}$: The set of n variables
    $G = \{G_1, G_2 \ldots, G_k\}$: The set of k clusters
    S: The set of shared variables
    Algo: The CSP algorithm.
3: **Output:**
    L: The set of all instantiations
4: $L \leftarrow \emptyset$
5: Solving(L, S, G, Algo)
6: RemoveSatisfiedConstraint(G, L)
7: *cudaMemcpy(G', G, $n \times m$, cudaMemcpyHostToDevice)*
8: ParallelSearch$\ll$<k, 1024$\gg$>(L, G', Algo)
9: **return** L
10: /*********************GPU
    Device************************/
11: **Kernel ParallelSearch**(L, G', Algo)
12: **input**
    Shared NC []: Array of non-satisfied constraints in shared memory
13: **Output:**
    L': The set of the local instantiations of all blocks
14: idx $\leftarrow$ blockIdx.x $\times$ blockDim.x + threadIdx.x
15: NC[idx] $\leftarrow G'_{blockIdx.x}$[idx]
16: Solving(L'[blockIdx.x], NC, G', Algo)
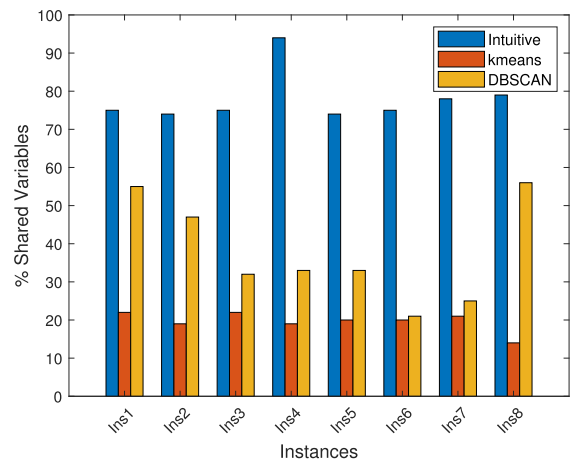17: *cudaMemcpy(L', L, $|L'|$, cudaMemcpyDeviceToHost)*

FIGURE 3. The quality of decomposition determined by different clustering algorithms (Intuitive, *k*-means, and DBSCAN).

GPU device, and the second one when the local tables are returned to the CPU. It also provides an efficient memory management by using different levels of memories including global and shared memories. However, it may suffer from synchronization problems between the GPU blocks. This takes place when the GPU blocks process clusters with different number of non-satisfied constraints. This issue degrades
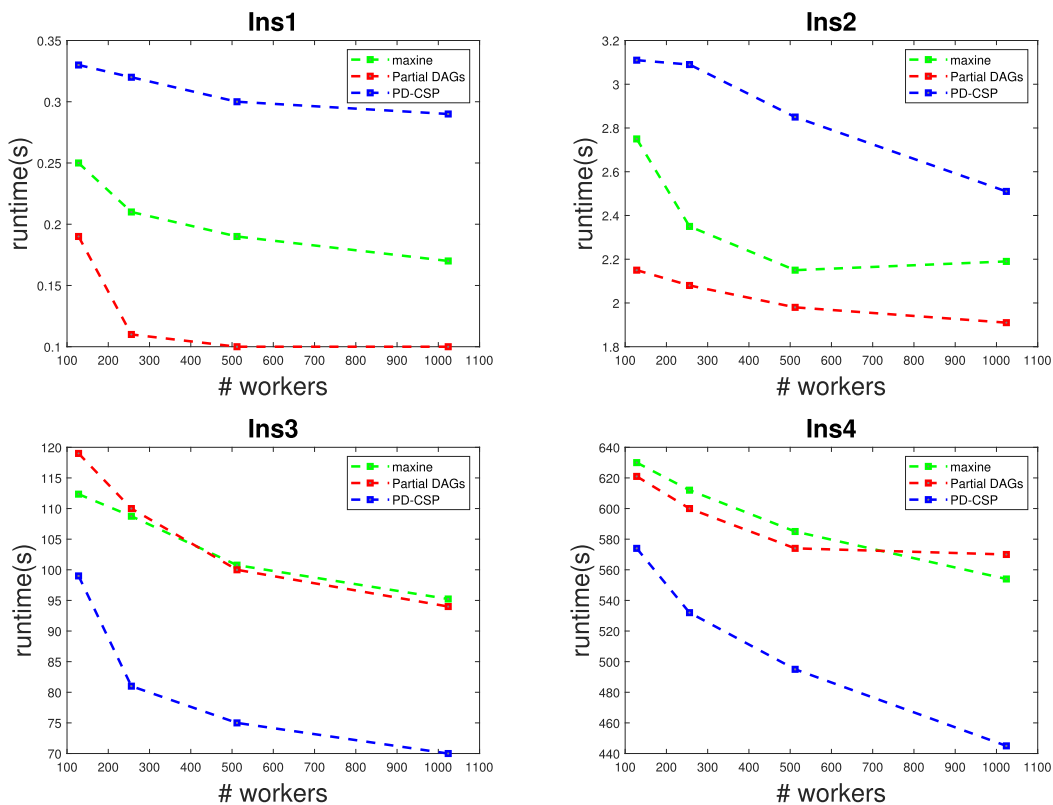
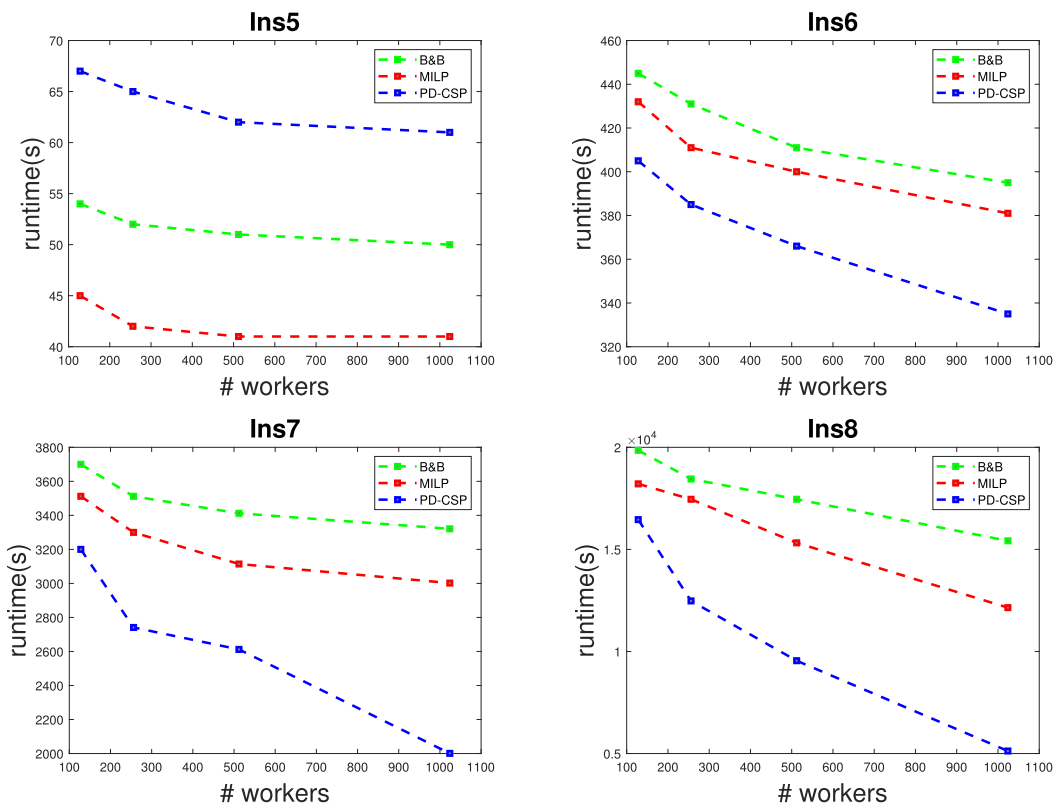**FIGURE 4.** Runtime of PD-CSP vs. the state-of-the-art *MAXSAT* algorithms.

FIGURE 5. Runtime of PD-CSP vs. the state-of-the-art *JOB Scheduling* algorithms.
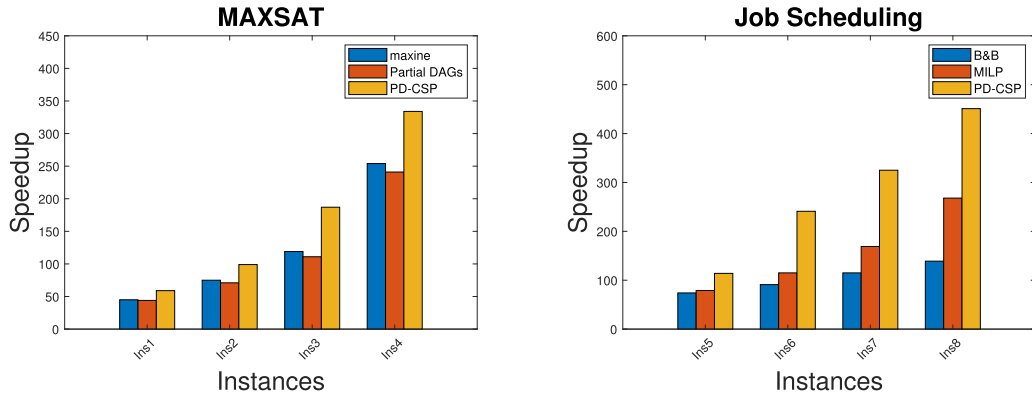
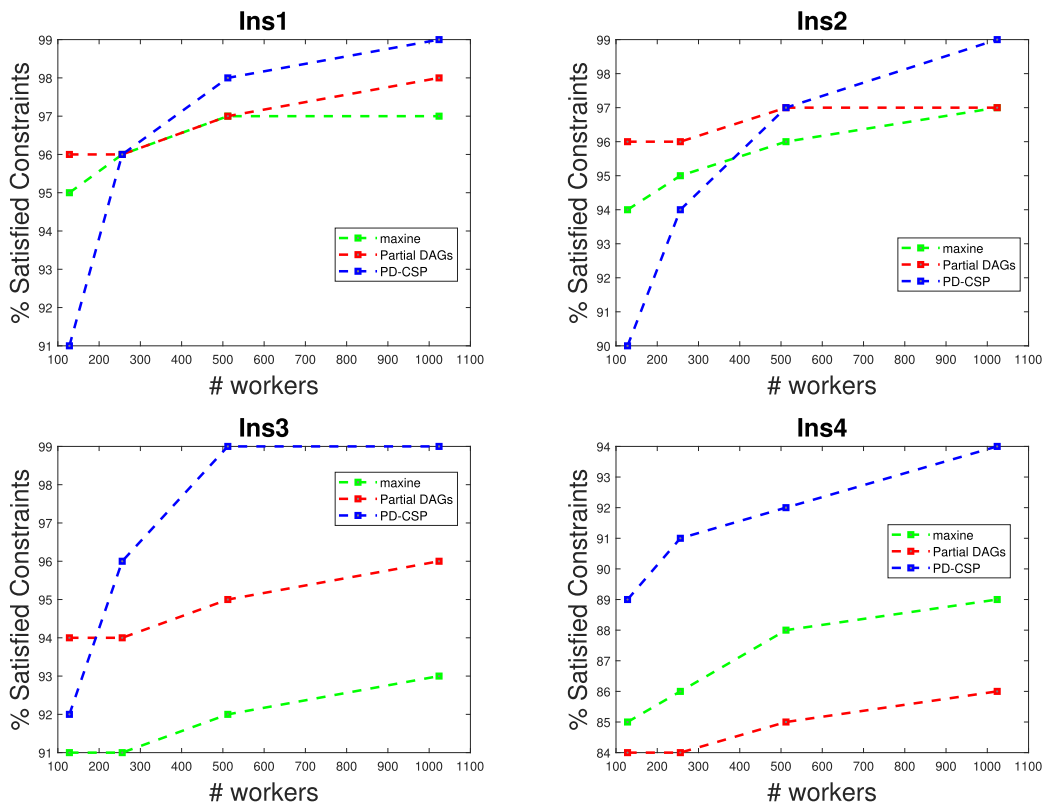**FIGURE 6.** The speedup of PD-CSP vs. the state-of-the-art algorithms.

FIGURE 7. The percentage of the satisfied constraints by PD-CSP vs. the state-of-the-art *MAXSAT* algorithms.

the performance of the GPU-based implementation of the PD-CSP framework. In real scenarios, different number of non-satisfied constraints per cluster may be obtained. This depends to the way of the clustering used in the decomposition step, i.e., the more the sizes of the clusters are different, the higher the GPU synchronization cost.

V. PERFORMANCE EVALUATION

We carried out extensive experiments to evaluate the PD-CSP framework on two well-known CSP problems (*MAXSAT* and *Job Scheduling*). The two problems differ from the input values. The input of MAXSAT is the set of constraints, each

of which is a conjunction of Boolean variables, however the input of CSP is the set of constraints, each of which is a relation of continues variables. Therefore, different data instances are used for both problems. For *MAXSAT*, we used one *Random* instance "Ins1" containing 110 variables and 50 clauses, one *Crafted* instance "Ins2" containing 220 variables and 45 clauses, and two *Industrial* instances "Ins3" containing 360 variables and 685 clauses, and "Ins4" containing 675 variables and 1286 clauses. For *Job Scheduling* we used the large size benchmarks proposed by Taillard [37]. The different instances are denoted by the number of jobs and the number of machines. We used four instances:
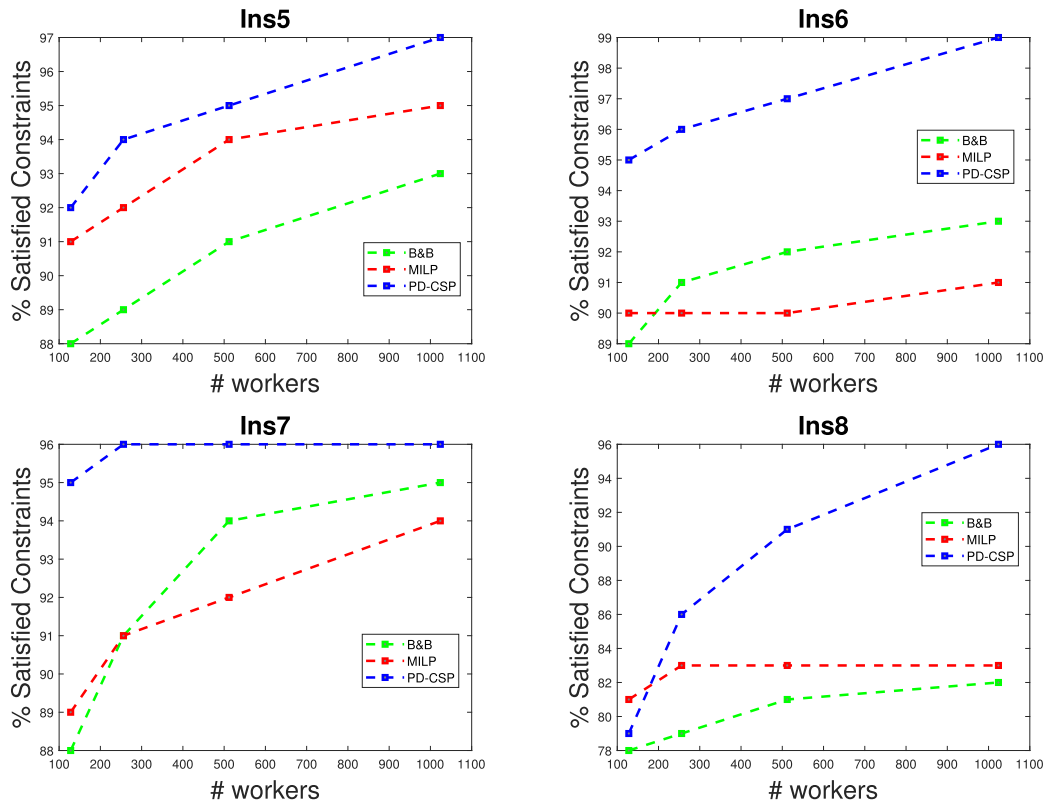
FIGURE 8. The percentage of the satisfied constraints of PD-CSP vs. the state-of-the-art *JOB Scheduling* algorithms.

"Ins5" containing 15 jobs and 15 machines, "Ins6" containing 30 jobs and 20 machines, "Ins7" containing 50 jobs and 20 machines and "Ins8" containing 100 jobs and 20 machines.

A. CLUSTERING QUALITY

Figure 3 presents the quality of decomposition of three different clustering algorithms: intuitive, multi-objective k-means, and DBSCAN, on different dataset instances (*MAXSAT* and *Job Scheduling*). The intuitive clustering is a naive decomposition that partitions constraints directly to the clusters, i.e., the first block is assigned to the first cluster, the second block is assigned to the second cluster, and so on, until all the blocks are assigned.

Many metrics have been developed to measure the quality of the decomposition. In our research work, we are interested to minimize the shared variables among the clusters of constraints. Therefore, the quality of decomposition is determined by the percentage of the shared variables between the clusters. The quality decreases as this percentage goes up. We also varied the number of clusters from 1 to 50 for each data instance, and the best value is kept for the experiments. The best values of the number of clusters are summarized in Table 2. Figure 3 reveals that k-means gives better decomposition comparing to two other algorithms for all the data instance. These results are explained by the fact that k-means is a pure partitioning algorithm inspired by the

TABLE 2. Parameter Setting of multi-objective kmeans.

Data	k
inst1	5
inst2	16
inst3	22
inst4	29
inst5	37
inst6	23
inst7	27
inst8	19

centroids representing the constraints of the same cluster. However, DBSCAN is inspired by neighborhood computation representing the dense regions. It is therefore possible to have two similar constraints belonging to two closer clusters. In the remaining of the experiments, we used k-means as the decomposition algorithm in the PD-CSP framework.

B. RUNTIME

Figures 4 and 5 present the runtime of the PD-CSP and the state-of-the art algorithms for both *MAXSAT* and *Job Scheduling*. For the *MAXSAT*, the baseline algorithms are maxine [14] and Partial DAGs [15]. For the *Job Scheduling*, the baseline algorithms are B&B [38] and MILP [39]. For various data instance sizes, the results reveal that for small number of variables, the baseline algorithms outperform the PD-CSP algorithm. This is explained by the fact that

PD-CSP performs further preprocessing step represented by the decomposition step. However, for larger number of variables and with using higher number of workers (GPU-blocks in our context), PD-CSP outperforms the baseline algorithms. This is due to the intelligent parallel strategy employed in the search step. Figure 6 shows the speedup of PD-CSP and the four baseline algorithms by considering 1024 workers (GPU-blocks). Note that the speedup is determined by the ratio of runtime of the parallel approach over the runtime of the sequential approach. This figure reveals the superiority of PD-CSP compared to the baseline algorithms. The speedup of PD-CSP exceeds 450 whereas the speedup of the baseline methods does not reach 300. These results confirm again that PD-CSP benefits from the intelligent strategy used in parallel search.

C. SOLUTION QUALITY

Figures 7 and 8 present the percentage of the satisfied constraints. The runtime of the running algorithms are set as follows, 0.5 sec for "Ins1", 2 sec for "Ins2", 400 sec for "Ins4", 40 sec for "Ins5", 350 sec for "Ins6", 2, 500 sec for "Ins7", and 10, 000 sec for "Ins8". By varying the number of workers (GPU-blocks) from 128 to 1, 024, our solution outperforms the baseline algorithms in all cases. This is explained by the fact that the proposed algorithms explore large search space compared to the other baseline algorithms, due to the decomposition and intelligent parallel search steps.

VI. DISCUSSION AND CONCLUSION

We introduced a framework, PD-CSP, that benefits from the decomposition by analyzing the correlations among the constraints. The set of constraints are first partitioned using the decomposition algorithm, where the highly correlated constraints are grouped together. As an instatiation of our general approach, we adapted two clustering algorithms: k-means and DBSCAN. Next, we developed a dedicated GPU-based parallel search to find the best assignment of variables by efficiently exploring the clusters of constraints. The experimental results on *MAXSAT* and *Job Scheduling* problems show the superiority of PD-CSP against the state-of-the-art algorithms in terms of reduced computational time and high accuracy represented by the number of satisfied constraints. Note that the number of satisfied constraints is obtained by enumerating the satisfied constraints for the final instantiation derived during the search process. The first key finding of this study is that the proposed framework can deal with big constraint data instances. This is different from previous constraint satisfaction solvers, which have long execution times as the whole set of constraints is considered in the solving step. The proposed framework is able not only to make instantiation of variables but also to analyze correlations and similarities between the constraints and to find out disjoint groups among them. We argue that, in the context of CSP, considering the decomposition techniques in the preprocessing step allows to quickly derive the best assignment of variables. Our framework is generic and can be applied to any CSP, contrary to the other algorithms that can only deal with a particular type of CSP. Motivated by our promising results, we suggest the following directions for further research. we plan to develop new strategies for dealing with load balancing. One way to address this issue is to develop decomposition strategies allowing to find out equitable clusters in terms of number of constraints per cluster. Another way is to develop new strategies for repairing clusters and finding clusters with similar number of constraints.

REFERENCES

[1] A. A. Amini, T. E. Weymouth, and R. C. Jain, "Using dynamic programming for solving variational problems in vision," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 9, pp. 855–867, Sep. 1990.

[2] F. E. Curtis, A. Wächter, and V. M. Zavala, "A sequential algorithm for solving nonlinear optimization problems with chance constraints," *SIAM J. Optim.*, vol. 28, no. 1, pp. 930–958, Jan. 2018.

[3] S. M. Pour, J. H. Drake, L. S. Ejlertsen, K. M. Rasmussen, and E. K. Burke, "A hybrid constraint programming/mixed integer programming framework for the preventive signaling maintenance crew scheduling problem," *Eur. J. Oper. Res.*, vol. 269, no. 1, pp. 341–352, Aug. 2018.

[4] C. Beierle and S. Kutsch, "Computation and comparison of nonmonotonic skeptical inference relations induced by sets of ranking models for the realization of intelligent agents," *Appl. Intell.*, vol. 49, no. 1, pp. 28–43, 2019.

[5] X.-L. Zheng and L. Wang, "A two-stage adaptive fruit fly optimization algorithm for unrelated parallel machine scheduling problem with additional resource constraints," *Expert Syst. Appl.*, vol. 65, pp. 28–39, Dec. 2016.

[6] A. A. Qamhan and I. M. Alharkan, "Note on 'a two-stage adaptive fruit fly optimization algorithm for unrelated parallel machine scheduling problem with additional resource constraints'," *Expert Syst. Appl.*, vol. 128, pp. 81–83, Aug. 2019.

[7] V. Kumar, "Algorithms for constraint-satisfaction problems: A survey," *AI Mag.*, vol. 13, no. 1, p. 32, Mar. 1992.

[8] M. Mouhoub and A. Sukpan, "Managing dynamic CSPs with preferences," *Int. J. Speech Technol.*, vol. 37, no. 3, pp. 446–462, Oct. 2012.

[9] K. W. Yong and M. Mouhoub, "Using conflict and support counts for variable and value ordering in CSPs," *Appl. Intell.*, vol. 48, no. 8, pp. 2487–2500, 2018.

[10] S. De Givry, T. Schiex, and G. Verfaillie, "Exploiting tree decomposition and soft local consistency in weighted CSP," in *Proc. AAAI*, vol. 6, 2006, pp. 1–6.

[11] Y. Djenouri, Z. Habbas, and D. Djenouri, "Data mining-based decomposition for solving the MAXSAT problem: Toward a new approach," *IEEE Intell. Syst.*, vol. 32, no. 4, pp. 48–58, 2017.

[12] Y. Djenouri, Z. Habbas, D. Djenouri, and P. Fournier-Viger, "Bee swarm optimization for solving the MAXSAT problem using prior knowledge," *Soft Comput.*, vol. 23, no. 9, pp. 3095–3112, May 2019.

[13] K. Pawłowski, K. Kurach, K. Svensson, S. Ramchurn, T. P. Michalak, and T. Rahwan, "Coalition structure generation with the graphics processing unit," in *Proc. AAMAS*, 2014, pp. 293–300.

[14] M. Alviano, C. Dodaro, and F. Ricca, "A MaxSAT algorithm using cardinality constraints of bounded size," in *Proc. IJCAI*, 2015, pp. 2677–2683.

[15] W. Haaswijk, M. Soeken, A. Mishchenko, and G. De Micheli, "SAT-based exact synthesis: Encodings, topology families, and parallelism," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 4, pp. 871–884, Apr. 2020.

[16] A. Belhadi, Y. Djenouri, J. Lin, and A. Cano, "A general-purpose distributed pattern mining system," *Appl. Intell.*, vol. 50, pp. 2647–2662, 2020.

[17] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Math. Statist. Probab.*, 1967, vol. 1, no, 14, pp. 281–297.

[18] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. KDD*, 1996, vol. 96, no. 34, pp. 226–231.

[19] M. Davis and H. Putnam, "A computing procedure for quantification theory," *J. ACM*, vol. 7, no. 3, pp. 201–215, Jul. 1960.

[20] S. O. Chan, J. R. Lee, P. Raghavendra, and D. Steurer, "Approximate constraint satisfaction requires large LP relaxations," *J. ACM*, vol. 63, no. 4, pp. 1–22, Nov. 2016.

[21] C. Carbonnel and M. C. Cooper, "Tractability in constraint satisfaction problems: A survey," *Constraints*, vol. 21, no. 2, pp. 115–144, Apr. 2016.

[22] D. A. Cohen, M. C. Cooper, P. G. Jeavons, and S. Živný, "Binary constraint satisfaction problems defined by excluded topological minors," *Inf. Comput.*, vol. 264, pp. 12–31, Feb. 2019.

[23] R. Dechter and J. Pearl, "Tree clustering for constraint networks," *Artif. Intell.*, vol. 38, no. 3, pp. 353–366, Apr. 1989.

[24] G. Gottlob, N. Leone, and F. Scarcello, "A comparison of structural CSP decomposition methods," *Artif. Intell.*, vol. 124, no. 2, pp. 243–282, Dec. 2000.

[25] Z. Miklós, "Understanding tractable decompositions for constraint satisfaction," Ph.D. dissertation, Univ. Oxford, Oxford, U.K., 2008.

[26] G. Gottlob, G. Greco, N. Leone, and F. Scarcello, "Hypertree decompositions: Questions and answers," in *Proc. 35th ACM SIGMOD-SIGACT-SIGAI Symp. Princ. Database Syst. (PODS)*, 2016, pp. 57–74.

[27] C. McCreesh and P. Prosser, "The shape of the search tree for the maximum clique problem and the implications for parallel branch and bound," *ACM Trans. Parallel Comput.*, vol. 2, no. 1, pp. 1–27, May 2015.

[28] S. Smirnov and V. Voloshinov, "On domain decomposition strategies to parallelize branch-and-bound method for global optimization in everest distributed environment," *Procedia Comput. Sci.*, vol. 136, pp. 128–135, 2018.

[29] A. Goldman, J. Lepping, Y. Ngoko, and D. Trystram, "Combining parallel algorithms solving the same application: What is the best approach?" in *Proc. IEEE Int. Symp. Parallel Distrib. Process., Workshops Phd Forum*, May 2013, pp. 1859–1868.

[30] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara, "The distributed constraint satisfaction problem: Formalization and algorithms," *IEEE Trans. Knowl. Data Eng.*, vol. 10, no. 5, pp. 673–685, Sep./Oct. 1998.

[31] R. Marino, G. Parisi, and F. Ricci-Tersenghi, "The backtracking survey propagation algorithm for solving random K-SAT problems," *Nature Commun.*, vol. 7, no. 1, pp. 1–8, Dec. 2016.

[32] N. A. El-Taweel and H. E. Z. Farag, "Voltage regulation in islanded microgrids using distributed constraint satisfaction," *IEEE Trans. Smart Grid*, vol. 9, no. 3, pp. 1613–1625, May 2018.

[33] Y. Hamadi, S. Jabbour, and L. Sais, "ManySAT: A parallel SAT solver," *J. Satisfiability, Boolean Model. Comput.*, vol. 6, no. 4, pp. 245–262, Jun. 2009.

[34] C. Truchet, A. Arbelaez, F. Richoux, and P. Codognet, "Estimating parallel runtimes for randomized algorithms in constraint solving," *J. Heuristics*, vol. 22, no. 4, pp. 613–648, Aug. 2016.

[35] Z. Habbas, M. Krajecki, and D. Singer, "Decomposition techniques for parallel resolution of constraint satisfaction problems in shared memory: A comparative study," *IJCSE*, vol. 1, nos. 2–4, pp. 192–206, 2005.

[36] R. Martins, V. Manquinho, and I. Lynce, "An overview of parallel SAT solving," *Constraints*, vol. 17, no. 3, pp. 304–347, Jul. 2012.

[37] E. Taillard, "Benchmarks for basic scheduling problems," *Eur. J. Oper. Res.*, vol. 64, no. 2, pp. 278–285, Jan. 1993.

[38] A. Dabah, A. Bendjoudi, A. AitZai, D. El-Baz, and N. N. Taboudjemat, "Hybrid multi-core CPU and GPU-based B&B approaches for the blocking job shop scheduling problem," *J. Parallel Distrib. Comput.*, vol. 117, pp. 73–86, Jul. 2018.

[39] K. Fleszar and K. S. Hindi, "Algorithms for the unrelated parallel machine scheduling problem with a resource constraint," *Eur. J. Oper. Res.*, vol. 271, no. 3, pp. 839–848, Dec. 2018.

YOUCEF DJENOURI received the Ph.D. degree in computer engineering from the University of Science and Technology Houari Boumediene (USTHB), Algiers, Algeria, in 2014. He was a Postdoctoral Research with Southern Denmark University, Norwegian University of Science and Technology, in 2017, where he has working on urban traffic data analysis. He is currently a Research Scientist with SINTEF, Oslo, Norway. He has published more than 60 research papers in peer-reviewed international conferences and journals, which have received more than 900 citations.

DJAMEL DJENOURI received the Ph.D. degree in computer science from the University of Science and Technology (USTHB), Algiers, Algeria, in 2007. He is currently an Associate Professor with the University of the West of England, Bristol, U.K. He published more than 100 papers in international peer-reviewed journals and conference proceedings and two books. He is holding two national patents. His research interests include the Internet of Things, wireless and mobile networks, machine learning, and application for smart cities and green applications.

ZINEB HABBAS received the Ph.D. degree in computer science from the Institute National Polytechnique of Grenoble. She is currently a Professor with Lorraine University. Her research interests include solving optimization problems, constraint satisfaction problems, synergy between data mining techniques and optimization, exact and heuristic methods, and parallel algorithms.

JERRY CHUN-WEI LIN (Senior Member, IEEE) received the Ph.D. degree in computer science and information engineering from National Cheng Kung University, Tainan, Taiwan, in 2010. He is currently working as an Associate Professor with the Department of Computing, Mathematics, and Physics, Western Norway University of Applied Sciences (HVL), Bergen, Norway. He is the co-leader of the popular SPMF open-source data-mining library. He has published more than 300 research papers in peer-reviewed international conferences and journals, which have received more than 4000 citations. His research interests include data mining, privacy-preserving and security, big data analytics, and social networks. He is a member of the Editorial Board of *Intelligent Data Analysis*. He is also the Editor-in-Chief of *Data Mining and Pattern Recognition* (DSPR) and an Associate Editor of the *Journal of Internet Technology* and IEEE Access.

TOMASZ P. MICHALAK received the Ph.D. degree in applied economics from the University of Antwerp, Belgium, in 2009. He was a Postdoctoral Researcher with the University of Liverpool, Southampton, and Oxford. He is currently an Assistant Professor with the Institute of Informatics, Faculty of Mathematics, Informatics, and Mechanics, University of Warsaw. His research interests include artificial intelligence, machine learning applications, social networks, and game theory.

ALBERTO CANO (Senior Member, IEEE) received the B.Sc. degree in computer engineering and the B.Sc. degree in computer science from the University of Cordoba, Spain, in 2008 and 2010, respectively, and the M.Sc. and Ph.D. degrees in intelligent systems and computer science from the University of Granada, Spain, in 2011 and 2014, respectively. He is currently an Assistant Professor with the Department of Computer Science, Virginia Commonwealth University, USA, where he heads the High-Performance Data Mining Laboratory. He has published over 50 articles in high-impact factor journals, 50 contributions to international conferences, two book chapters, and one book in the areas of machine learning, data mining, and parallel, distributed, and GPU computing. His research interests include machine learning, data mining, general-purpose computing on graphics processing units, Apache Spark, and evolutionary computation. His research is supported by an Amazon AWS Machine Learning Award, in 2018, and the VCU Presidential Research Quest Fund, in 2018. He is an Associate Editor of IEEE Access, *Applied Intelligence*, and *PeerJ Computer Science*.

● ● ●