Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

Uncovering Hidden Structure through Parallel Problem Decomposition for the Set Basis Problem: Application to Materials Discovery

Yexiang Xue
Cornell University, USA
yexiang@cs.cornell.edu

Stefano Ermon
Stanford Univerity, USA
ermon@cs.stanford.edu

Carla P. Gomes
Cornell University, USA
gomes@cs.cornell.edu

Bart Selman
Cornell University, USA
selman@cs.cornell.edu

Abstract

Exploiting parallelism is a key strategy for speeding up computation. However, on hard combinatorial problems, such a strategy has been surprisingly challenging due to the intricate variable interactions. We introduce a novel way in which parallelism can be used to exploit hidden structure of hard combinatorial problems. Our approach complements divide-and-conquer and portfolio approaches. We evaluate our approach on the minimum set basis problem: a core combinatorial problem with a range of applications in optimization, machine learning, and system security. We also highlight a novel sustainability related application, concerning the discovery of new materials for renewable energy sources such as improved fuel cell catalysts. In our approach, a large number of smaller sub-problems are identified and solved concurrently. We then aggregate the information from those solutions, and use this information to initialize the search of a global, complete solver. We show that this strategy leads to a substantial speed-up over a sequential approach, since the aggregated sub-problem solution information often provides key structural insights to the complete solver. Our approach also greatly outperforms state-of-the-art incomplete solvers in terms of solution quality. Our work opens up a novel angle for using parallelism to solve hard combinatorial problems.

1 Introduction

Exploiting parallelism and multi-core architectures is a natural way to speed up computations in many domains. Recently, there has been great success in parallel computation in fields such as scientific computing and information retrieval [Dean and Ghemawat, 2008; Chu *et al.*, 2007].

Parallelism has also been taken into account as a promising way to solve hard combinatorial problems. However, it remains challenging to exploit parallelism to speed up combinatorial search because of the intricate non-local nature of the interactions between variables in hard problems [Hamadi and Wintersteiger, 2013]. One class of approaches in this domain is divide-and-conquer, which dynamically splits the search space into sub-spaces, and allocates each sub-space to a parallel node [Chrabakh and Wolski, 2003; Chu *et al.*, 2008; Rao and Kumar, 1993; Regin *et al.*, 2013; Moisan *et al.*, 2013; Fischetti *et al.*, 2014]. A key challenge in this approach is that the solution time for subproblems can vary by several orders of magnitude and is highly unpredictable. Frequent load re-balancing is required to keep all processors busy, but the load re-balancing process can result in a substantial overhead cost. Another class of approaches harnesses portfolio strategies, which runs a portfolio of solvers (of different type or with different randomization) in parallel, and terminates as soon as one of the algorithms completes. [Xu *et al.*, 2008; Leyton-Brown *et al.*, 2003; Malitsky *et al.*, 2011; Kadioglu *et al.*, 2011; Hamadi and Sais, 2009; Biere, 2010; Kottler and Kaufmann, 2011; Schubert *et al.*, 2010; O'Mahony *et al.*, 2008]. Parallel portfolio approaches can be highly effective. They do require however the use of a collection of effective solvers that each excel at different types of problem instances. In certain areas, such as SAT/SMT solving, we have such collections of solvers but for other combinatorial tasks, we do not have many different solvers available.

In this paper, we exploit parallelism to boost combinatorial search in a novel way. Our framework complements the two parallel approaches discussed before. In our approach parallelism is used as a preprossessing step to identify a promising portion of the search space to be explored by a complete sequential solver. In our scheme, a set of parallel processes are first deployed to solve a series of related subproblems. Next, the solutions to these subproblems are aggregated to obtain an initial guess for a candidate solution to the original problem. The aggregation is based on a key empirical observation that solutions to the subproblems, when properly aggregated, provide information about solutions for the original problem. Lastly, a global sequential solver searches for a solution in an iterative deepening manner, starting from the promising portion of the search space identified by the previous aggregation step. At a high level, the initial guess obtained by aggregating solutions to subproblems provides the so-called backdoor information to the sequential solver, by forcing it to start from the most promising portion of the search space. A backdoor set is a set of variables, such that once their values are set correctly, the remaining problem can be solved in polynomial time [Williams *et al.*, 2003; Dilkina *et al.*, 2009; Hamadi *et al.*, 2011].

We empirically show that a global solver, when initialized with proper information obtained by solving the subproblems, can solve a set of instances in seconds, while it takes for the same solver hours to days to find the solution without initialization. The strategy also outperforms state-of-the-art incomplete solvers in terms of solution quality.

We apply the parallel scheme to an NP-complete problem called the Set Basis Problem, in which we are given a collection of subsets $\mathcal{C}$ of a finite set U. The task is to find another, hopefully smaller, collection of subsets of U, called a "set basis", such that each subset in $\mathcal{C}$ can be represented exactly by a union of sets from the set basis. Intuitively, the set basis provides a compact representation of the original collection of sets. The set basis problem occurs in a range of applications, most prominently in machine learning, e.g., used as a special type of matrix factorization technique [Miettinen *et al.*, 2008]. It also has applications in system security and protection, where it is referred to as the role mining problem in access control [Vaidya *et al.*, 2007]. It also has applications in secure broadcasting [Shu *et al.*, 2006] and computational biology [Nau *et al.*, 1978].

While having many natural applications, our work is motivated by a novel application in the field of computational sustainability [Gomes, Winter 2009], concerning the discovery of new materials for renewable energy sources such as improved fuel cell catalysts [Le Bras *et al.*, 2011]. In this domain, the set basis problem is used to find a succinct explanation of a large set of measurements (X-ray diffraction patterns) that are represented in a discrete way as sets. Mathematically, this corresponds to a generalized version of the set basis problem with extra constraints. Our parallel solver can be applied to this generalized version as well, and we demonstrate significant speedups on a set of challenging benchmarks. Our work opens up a novel angle for using parallelism to solve a set of hard combinatorial problems.

2 Set Basis Problem

Throughout the paper, sets will be denoted by uppercase letters, while members of a set will be denoted by lowercase letters. A collection of sets will be denoted using calligraphic letters.

The classic Set Basis Problem is defined as follows:

- **Given:** a collection $\mathcal{C}$ of subsets of a finite universe U, $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ and a positive integer K;
- **Find:** a collection $\mathcal{B} = \{B_1, \ldots, B_K\}$ where each B_i is a subset of U, and for each $C_i \in \mathcal{C}$, there exists a subcollection $\mathcal{B}_i \subseteq \mathcal{B}$, such that $C_i = \cup_{B \in \mathcal{B}_i} B$. In this case, we say $\mathcal{B}_i$ *covers* C_i, and we say $\mathcal{C}$ is *collectively covered* by $\mathcal{B}$. Following common notations, $\mathcal{B}$ is referred to as a *basis*, and we call each $B_j \in \mathcal{B}$ a *basis set*. If $B_j \in \mathcal{B}_i$ and $\mathcal{B}_i$ covers C_i, we call B_j a *contributor* of C_i, and call C_i a *sponsor* of B_j. $C_1, C_2, \ldots, C_m$ are referred to as *original sets*.

Intuitively, similar to the basis vectors in linear algebra, which provides a succinct representation of a linear space, a set basis with smallest cardinality K plays the role of a compact representation of a collection of sets. The Set Basis Problem is shown to be NP-hard in [Stockmeyer, 1975]. We use $\mathcal{I}(\mathcal{C})$ to denote an instance of the set basis problem which finds the basis for $\mathcal{C}$. A simple instance and its solution is reported in Table 1.

C_0	$\{x_0, x_1, x_2, x_3\}$	$B_0 \cup B_2$
C_1	$\{x_0, x_2, x_3\}$	B_0
C_2	$\{x_0, x_1, x_4\}$	$B_2 \cup B_4$
C_3	$\{x_2, x_3, x_4\}$	$B_1 \cup B_4$
C_4	$\{x_0, x_1, x_3\}$	$B_2 \cup B_3$
C_5	$\{x_3, x_4\}$	$B_3 \cup B_4$
C_6	$\{x_2, x_3\}$	B_1
B_0	$\{x_0, x_2, x_3\}$	$C_0 \cap C_1$
B_1	$\{x_2, x_3\}$	$C_3 \cap C_6$
B_2	$\{x_0, x_1\}$	$C_0 \cap C_2 \cap C_4$
B_3	$\{x_3\}$	$C_4 \cap C_5$
B_4	$\{x_4\}$	$C_2 \cap C_3 \cap C_5$

Table 1: (An example of set basis problem) $C_0, \ldots, C_6$ are the original sets. A basis of size 5 that cover these sets is given by $B_0, \ldots, B_4$. The rightmost column at the top shows how each original set can be obtained from the union of one or more basis sets. The given cover is minimum (i.e., containing a minimum number of basis sets). The rightmost column at the bottom shows the duality property: each basis set can be written as an intersection of several original sets.

Most algorithms used in solving set basis problems are incomplete algorithms. These algorithms are based on heuristics that work well in certain domains, but often fail at covering sets exactly. For a survey, see Molloy et al [Molloy *et al.*, 2009]. The authors of [Ene *et al.*, 2008] implement the only complete solver we are aware of. The idea is to translate the set basis problem as a graph coloring problem, and then use existing graph coloring solvers. They also develop a useful prepossessing technique, which can significantly reduce the problem complexity.

The Set Basis Problem has a useful dual property, which has been implicitly used by previous researchers [Vaidya *et al.*, 2006; Ene *et al.*, 2008]. We formalize the idea by introducing Theorem 2.1.

Definition (Closure) For a collection of sets $\mathcal{C}$, define the *closure* of $\mathcal{C}$, denoted as $\overline{\mathcal{C}}$, which includes the collection of all possible intersections of sets in $\mathcal{C}$:

- $\forall C_i \in \mathcal{C}, C_i \in \overline{\mathcal{C}}$.
- For $A \in \overline{\mathcal{C}}$ and $B \in \overline{\mathcal{C}}$, $A \cap B \in \overline{\mathcal{C}}$.

Theorem 2.1. *For original sets $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$, suppose $\{B_1, \ldots, B_K\}$ is a basis that collectively covers $\mathcal{C}$. Define $\mathcal{C}_i = \{C_j \in \mathcal{C} | B_i \subseteq C_j\}$. Then $B'_i = \cap_{C \in \mathcal{C}_i} C$ ($i = 1 \ldots K$) collectively covers $\mathcal{C}$ as well. Note for every B'_i ($i = 1 \ldots K$), $B'_i \in \overline{\mathcal{C}}$.*

One can check Theorem 2.1 by examining the example in Table 1. The full proof is available in the supplementary materials [Xue *et al.*, 2015]. From the theorem, any set basis problem has a solution of minimum cardinality, where each basis set is in $\overline{\mathcal{C}}$. Therefore, it is sufficient to only search for basis within the closure $\overline{\mathcal{C}}$. Hence throughout this paper, we assume all basis sets are within its closure for any solutions to set basis problems. Theorem 2.1 also implies that *each basis set $B_i \in \overline{C}$ is an intersection of all its sponsor sets.* One can observe this fact in Table 1. It motivates our dual approach to solve the set basis problem, in which we search for possible sponsors for each basis set.

3 Parallel Scheme

The main intuition of our parallel scheme comes from an empirical observation on the structure of the solutions of the benchmark problems we considered. For each benchmark, we solve a series of related simplified subproblems, where we restrict ourselves to finding basis for *a subset of the original collection of sets $\mathcal{C}$*. Interestingly, the solutions found by solving these subproblems are connected to the basis in the global solution. Although strictly speaking, the basis found for one sub-problem can only be expected to be a solution for that particular sub-problem, we observe empirically that *almost all basis sets from sub-problems are supersets of one or more basis sets for the original, global problem.* One intuitive explanation is as follows: Recall that from Theorem 2.1 each basis set can be obtained as the intersection of its sponsors. This fact applies both to the original global problem and its relaxed versions (subproblems). Since there are fewer sets to be covered in the subproblems, basis sets for the subproblems are likely to have fewer sponsors, compared to the ones for the global problem. When we take the intersection of fewer sets, we get a larger intersection. Hence we observe that it is often the case that a basis set for a subproblem is a superset of a basis set for the global problem.

Now suppose two subproblem basis sets A and B are both supersets of one basis set C in the global solution. If we intersect A with B, then the elements of C will remain in the intersection, but other elements from A or B will likely be removed. In practice, *we can often obtain a basis set in the global solution by intersecting only a few basis sets from the solutions to subproblems*.

Let us walk through the example in Table 1. First consider the subproblem consisting of the first 5 original sets, $C_0 ... C_4$. It can be shown that a minimum set basis is $B_{1,1} = \{x_0, x_1\}$, $B_{1,2} = \{x_0, x_3\}$, $B_{1,3} = \{x_2, x_3\}$, $B_{1,4} = \{x_4\}$. As another subproblem we consider the collection of all original sets except for C_0 and C_2. We obtain a minimum basis $B_{2,1} = \{x_0, x_3\}$, $B_{2,2} = \{x_2, x_3\}$, $B_{2,3} = \{x_3, x_4\}$, $B_{2,4} = \{x_0, x_1, x_3\}$. We see that each basis set of these two subproblems contains at least one of the basis sets of the original, full set basis problem. For example, $B_2 = \{x_0, x_1\} \subseteq B_{1,1}$ and $B_2 \subseteq B_{2,4}$. Moreover, one can obtain all basis sets except B_0 for the original problem by intersecting these basis sets. For example, $B_3 = \{x_3\} = B_{1,2} \cap B_{2,2}$.

Given this observation, we design a parallel scheme that works in two phases – an exploration phase, followed by an aggregation phase. The whole process is shown in Figure 1, and the two phases are detailed in subsequent subsections.

Exploration Phase: we use a set of parallel processes. Each one solves a sub-problem obtained by restricting the global problem to finding the *minimum basis* for a subset of the original collection of sets $\mathcal{C}$.

Aggregation Phase: we first identify an initial solution candidate by looking at all the possible intersections among basis sets found by solving sub-problems in the exploration phase. We then use this candidate to initialize a complete solver, which expands the search in an iterative deepening way to achieve completeness, iteratively adding portions of the search space that are "close" to the initial candidate.

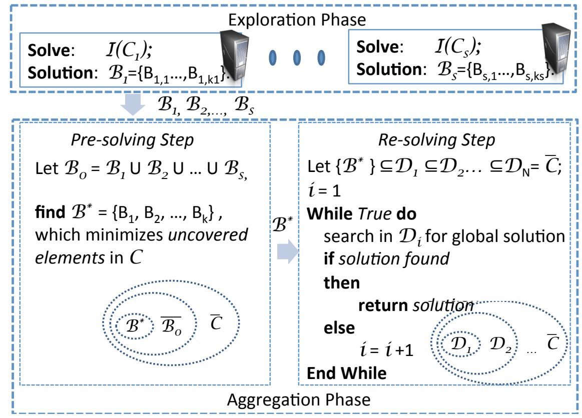

Figure 1: A diagram showing the parallel scheme.

3.1 Exploration Phase

The exploration phase utilizes parallel processes to solve a series of sub-problems. Recall the global problem is to find a basis of size K for the collection $\mathcal{C}$. Let $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_s\}$ be a decomposition of $\mathcal{C}$, which satisfies $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2 \cup \ldots \cup \mathcal{C}_s$. The sub-problem $\mathcal{I}(\mathcal{C}_i)$ restricted on $\mathcal{C}_i$ is defined as:

- **Given:** $\mathcal{C}_i \subseteq \mathcal{C}$;
- **Find:** a basis $\mathcal{B}_i = \{B_{i,1}, B_{i,2}, \ldots, B_{i,k_i}\}$ with smallest cardinality, such that every set $C' \in \mathcal{C}_i$ is covered by the union of a sub-collection of $\mathcal{B}_i$.

The sub-problem is similar to the global problem, however, with one key difference: we are solving an *optimization problem* where we look for a minimum basis, as opposed to the global problem, which is the decision version of the Set Basis Problem. In practice, the optimization is done by repeatedly solving the decision problem, with increasing values of K. We observe empirically that the optimization is crucial for us to get meaningful basis sets to be used in later aggregation steps. If we do not enforce the minimum cardinality constraint, the problem becomes under-constrained and there could be redundant basis sets found in this phase, which have no connections with the ones in the global solution.

Sets $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_s$ need not be mutually exclusive in the decomposition. We group similar sets into one subproblem in our algorithm, so the resulting subproblem will have a small basis. To obtain collection $\mathcal{C}_i$ for the i-th subproblem, we start from an initial collection of a singleton $\mathcal{C}_i = \{C_{i_1}\}$, where C_{i_1} is a randomly picked element from $\mathcal{C}$. We then add $T - 1$ sets most similar to C_{i_1}, using the Jaccard similarity coefficient $\frac{|A \cap B|}{|A \cup B|}$. This results in a collection of T sets which look similar. Notice that this is our method to find a collection of similar sets. We expect other approach can work equally well.

3.2 Aggregation Phase

In the aggregation phase, a centralized process searches for the exact solution, starting from basis sets that are "close" to a candidate solution selected from the closure of basis sets found by solving sub-problems, and then expands its search space iteratively to achieve completeness.

To obtain a good initial candidate solution, we begin with a pre-solving step, in which we restrict ourselves to find a good global solution only within the closure of basis sets found by solving sub-problems. This is of course an incomplete procedure, because the solution might lie outside the closure. However, due to the empirical connections between the basis sets found by parallel subproblem solving and the ones in the final solution, we often find the global solution at this step.

If we cannot find a solution in the pre-solving step, the algorithm continues with a re-solving step, in which an iterative deepening process is applied. It starts with the best K basis sets found in the pre-solving step[1], and iteratively expands the search space until it finds a global solution. The two steps are detailed as follows.

Pre-solving Step

Suppose $\mathcal{B}_i$ is the basis found for sub-problem $\mathcal{I}(\mathcal{C}_i)$, let $\mathcal{B}_0 = \cup_{i=1}^{s} \mathcal{B}_i$ and $\overline{\mathcal{B}_0}$ be the closure of $\mathcal{B}_0$. The algorithm solves the following problem in the pre-solving step:

- **Given:** $\mathcal{B}_0$;
- **Find:** Basis $\mathcal{B}^* = \{B_1^*, \ldots, B_K^*\}$ from $\overline{\mathcal{B}_0}$, such that $B_1^*, \ldots, B_K^*$ minimizes the total number of uncovered elements and falsely covered elements in $\mathcal{C}$[2].

In practice $\overline{\mathcal{B}_0}$ is still a huge space, so this optimization problem is hard to solve. We thus apply an incomplete algorithm, which only samples a small subset $\mathcal{U} \subseteq \overline{\mathcal{B}_0}$ and then select the best K basis sets from $\mathcal{U}$. It does not affect the later re-solving step, since it can start the iterative deepening process from any $\mathcal{B}^*$, whether optimal in $\overline{\mathcal{B}_0}$ or not.

The incomplete algorithm first forms $\mathcal{U}$ by conducting multiple random walks in the space of $\overline{\mathcal{B}_0}$. Each random walk starts with a random basis set $B \in \mathcal{B}_0$, and randomly intersects it with other basis sets in $\mathcal{B}_0$ to obtain a new member in $\overline{\mathcal{B}_0}$. All these sets are collected to form $\mathcal{U}$. With probability p, the algorithm chooses to intersect with the basis which maximizes the cardinality of the intersection. With probability $(1-p)$, the algorithm intersects with a random set. In our experiment, p is set to 0.95, and we repeat this random walk several times with different initial sets to make $\mathcal{U}$ large enough. Next the algorithm selects the optimal basis of size K from $\mathcal{U}$ which maximizes the coverage of the initial set collection, using a Mixed Integer Programming (MIP) formulation. The pseudocode of the incomplete algorithm and the MIP formulation are both in the supplementary materials [Xue *et al.*, 2015].

[1] Best in terms of coverage of the initial set collection.

[2] An *uncovered* element of set C_j is one element contained in C_j, but is not covered by any basis set that are contributors to C_j. A *falsely covered* element of set C_j is one element that is in one basis set that is a contributor to set C_j, but is not contained in C_j.

Re-solving Step

The final step is the re-solving step. It takes as input the basis $\mathcal{B}^* = \{B_1^*, B_2^*, \ldots, B_K^*\}$ from the pre-solving step, and searches for a complete solution to $\mathcal{I}(C)$ in an iterative deepening manner. The algorithm starts from a highly restricted space $\mathcal{D}_1$, which is a small space close to $\mathcal{B}^*$. If the algorithm can find a global solution in $\mathcal{D}_1$, then it terminates and returns the solution. Otherwise, it expands its search space to $\mathcal{D}_2$, and searches again in this expanded space, and so on. At the last step, searching in $\mathcal{D}_n$ is equivalent to searching in the original unconstrained space $\overline{\mathcal{C}}$, which is equivalent to solving the global set-basis problem without initialization at all. However, this situation is rarely seen in our experiments.

In practice, $\mathcal{D}_1, \ldots, \mathcal{D}_n$ are specified by adding extra constraints to the original MIP formulation for the global problem, then iteratively removing them. $\mathcal{D}_n$ corresponds to the case where all extra constraints are removed.

The actual design of $\mathcal{D}_1, \ldots, \mathcal{D}_n$ relies on the MIP formulation. In our MIP formulation (which is detailed in the supplementary materials [Xue *et al.*, 2015]), there are indicator variables $y_{i,k}$ ($1 \le i \le n$ and $1 \le k \le K$), where $y_{i,k} = 1$ if and only if the i-th element is contained in the k-th basis set B_k. We also have indicator variables $z_{k,j}$, where $z_{k,j}$ is one if and only if the basis set B_k is a contributor of the original set C_j (or equivalently, C_j is a sponsor set for B_k).

Because we empirically observe that $B_1^*, B_2^*, \ldots, B_K^*$ are often super-sets of the basis sets in the exact solution, we construct the constrained space $\mathcal{D}_1, \ldots, \mathcal{D}_n$ by enforcing the sponsor sets of certain basis sets. Notice that this is a straightforward step in the MIP formulation, since we only need to fix the corresponding indicator variables $z_{k,j}$ to 1 to enforce C_j as a sponsor set for B_k. The hope is that these clamped variables will include a subset of backdoor variables for the original search problem [Williams *et al.*, 2003; Dilkina *et al.*, 2009; Hamadi *et al.*, 2011]. The runtime of the sequential solver is dramatically reduced when the aggregation phase is successful in identifying a promising portion of the search space.

As pointed out by Theorem 2.1, we can represent $B_1^*, B_2^*, \ldots, B_K^*$ in terms of their sponsors:

$$\begin{aligned} B_1^* &= C_{11} \cap C_{12} \cap \ldots \cap C_{1s_1} \\ B_2^* &= C_{21} \cap C_{22} \cap \ldots \cap C_{2s_2} \\ &\ldots \\ B_K^* &= C_{K1} \cap C_{K2} \cap \ldots \cap C_{Ks_K} \end{aligned}$$

in which $C_{11}, C_{12}, \ldots, C_{21}, \ldots, C_{Ks_K}$ are all original sets in collection $\mathcal{C}$. For the first restricted search space $\mathcal{D}_1$, we enforce the constraint that the sponsors for the i-th basis set B_i must contain all the sponsors of B_i^* for all $i \in \{1, \ldots, K\}$. Notice this implies $B_i \subseteq B_i^*$.

In later steps, we gradually relax these extra constraints, by freeing some of the indicator variables $z_{k,j}$'s which were clamped to 1 in previous steps. $\mathcal{D}_n$ denotes the search space when all these constraints are removed, which is equivalent to searching the entire space. The last thing is to decide the order used to remove these sponsor constraints. Intuitively, if one particular set is discovered many times as a sponsor set in the solutions to subproblems, then it should have a high chance to be the sponsor set in the global solution, because it fits in the solutions to many subproblems. Given this intuition, we associate each sponsor set with a *confidence score*, and define n thresholds: $0 = p_1 < \ldots < p_n = +\infty$. In the k-th round (search in $\mathcal{D}_k$), we remove all the sponsor sets whose confidence score is lower than p_k. We define the confidence score of a particular set as the number of times it appears as a sponsor of a basis set in subproblem solutions, which can be aggregated from the solutions to subproblems.

4 Experiments

We test the performance of our parallel scheme on both the classic set basis problem, and on a novel application in materials science.

4.1 Classic Set Basis Problem

Setup We test the parallel scheme on synthetic instances. We use a random ensemble similar to Molloy et al [Molloy *et al.*, 2009], where every synthetic instance is characterized by n, m, k, e, p. To generate one synthetic instance, we first generate k basis sets. Every set contains $[n\frac{p}{100}]$ objects, uniformly sampled from a finite universe of n elements. We then generate m sets. Each set is a union of e randomly chosen basis sets from those initially generated.

We develop a Mixed Integer Programming (MIP) model to solve the set basis problem (detailed in the supplementary materials [Xue *et al.*, 2015]). The MIP model takes the original sets $\mathcal{C}$ and an integer K, and either returns a basis of size K that covers $\mathcal{C}$ exactly, or reports failure. We compare the performance of the MIP formulation with and without the initialization obtained using the parallel scheme described in the previous section.

We empirically observe high variability in the running times of the sub-problems solved in the exploration phase, as commonly observed for NP-hard problems. Luckily, our parallel scheme can still be used without waiting for every sub-problem to complete. Specifically, we set up a cut-off threshold of 90%, such that the central process waits until 90% of sub-problems are solved before carrying out the aggregation phase. We also run 5 instances of the aggregation phase in parallel, each with a different random initialization, and terminate as soon as the fastest one finds a solution. In our experiment, $n = m = 100$. All MIPs are solved using IBM CPLEX 12.6, on a cluster of Intel x5690 3.46GHz core processors with 4 gigabytes of memory. We let each subproblem contain $T = 15$ sets for all instances.

Results Results obtained with and without initialization from parallel sub-problem solving are reported in Table 2. First, we see it takes much less wall-clock time (typically, by several orders of magnitude) for the complete solver to find the exact solution if it is initialized with the information collected from the sub-problems. The improvements are significant even when taking into account the time required for solving sub-problems in the exploration phase. In this case, we obtain several orders of magnitude saving in terms of solving time. For example, it takes about 50 seconds (wall-clock time) to solve A6, but about 5 hours without parallel initialization. Because we run $s = 100$ sub-problems in the exploration phase, another comparison would be based on CPU time, which is given by $(100 \cdot Exploration + 5 \cdot Aggregation)$. Under this measurement, our parallel scheme still outperforms the sequential approach on problem instances A2, A3, A5, A6, A8. Even though our CPU time is longer for some instances, our parallel scheme can be easily applied to thousands of cores. As parallel resources are becoming more and more accessible, it is obvious to see the benefit of this scheme. Note that we can also exploit *at the same time* the built-in parallelism of CPLEX to solve these instances. However, because CPLEX cannot explore the problem structure explicitly, it cannot achieve significant speed-ups on many instances. For example, it takes 12813.15, 259100.25 and 113475.12 seconds to solve the largest A6, A7 and A8 instances using CPLEX on 12-cores.

Although our focus is on improving the run-time of exact solvers, Table 2 also shows the performance of several state-of-the-art incomplete solvers on these synthetic instances. We implemented *FastMiner* from [Vaidya *et al.*, 2006], *ASSO* from [Miettinen *et al.*, 2008], and *HPe* from [Ene *et al.*, 2008], which are among the most widely used incomplete algorithms. *FastMiner* and *ASSO* take the size of the basis K as input, and output K basis sets. They are incomplete in the sense that their solution may contain false positives and false negatives, which are defined as follows. c is a false positive element if $c \notin C_i$, but c is in one basis set that is a contributor to C_i. c is a false negative element, if $c \in C_i$, but c is not covered by any basis sets contributing for C_i. *FastMiner* does not provide the information about which basis set contributes to an original set. We therefore give the most conservative assignment: B is a contributor to C_i if and only if $B \subseteq C_i$. This assignment introduces no false positives. Both *FastMiner* and *ASSO* have parameters to tune. Our report are based on the best parameters we found. We report the maximum error rate in Table 2, which is defined as $\max_{C_i \in \mathcal{C}} \{(ft_i + ff_i)/|C_i|\}$, where ft_i and ff_i are the number of false positive and false negative elements at C_i, respectively. As seen from the table, neither of these two algorithms can recover the exact solution. *ASSO* performs better, but it still has 51.06% error rate on the hardest benchmark. We think the reason why *FastMiner* performs poorly is because it is usually used in situations where certain number of false positives can be tolerated. *HPe* is a graph based incomplete algorithm. It is guaranteed to find a complete cover, however it might require a number of basis sets K' larger than the optimal number K. We implemented both the greedy algorithm and the lattice-based post-improvement for *HPe*, and we used the best heuristic reported by the authors. As we can see from Table 2, *HPe* often needs five times more basis sets to cover the entire collection.

The authors in [Ene *et al.*, 2008] implemented the only complete solver we are aware of. Unfortunately, we can not obtain their code, so a direct comparison is not possible. However, the parallel scheme we developed does not make any assumption on the specific complete solver used. We expect other complete solvers (in addition to the MIP one we experi-

[3]For this instance, 73 out of 100 subproblem instances complete within 2 hours. Thus the aggregation phase is conducted based on these instances. This exploration time here is calculated based on the slowest of the 73 instances.

Instance		Solution Quality								Run-time for Complete Method (seconds)			
		HPe		*Fast-Miner*		*ASSO*		*Complete*				*Total Time*	*Total Time*
No.	K	K'	$E\%$	K'	$E\%$	K'	$E\%$	K'	$E\%$	*Exploration*	*Aggregation*	*Parallel*	*Sequential*
A1	8	65	0	8	100	8	26.56	8	0	30.61	7.21	**37.82**	2199.07
A2	8	86	0	8	100	8	18.18	8	0	42.32	149.18	**191.5**	11374.34
A3	10	41	0	10	96.67	10	25	10	0	12.09	0.42	**12.51**	1561.46
A4	10	47	0	10	100	10	18.92	10	0	136.82	10.94	**147.76**	332.93
A5	10	58	0	10	100	10	52	10	0	55.52	2.60	**58.12**	57004.13
A6	12	51	0	12	96.3	12	25	12	0	46.85	0.75	**47.6**	17774.04
A7	12	63	0	12	100	12	38.46	12	0	6963.42[3]	13.47	**6967.89**	$>$ 72 hours
A8	12	93	0	12	100	12	51.06	12	0	176.4	5.77	**182.17**	$>$ 72 hours

Table 2: Comparison of different methods on classic set basis problems. K is the number basis sets used by the synthetic generator. In the solution quality block, we show the basis size K' and the error rate $E\%$ for incomplete method *HPe*, *FastMiner* and *ASSO* and the complete method. $K' > K$ means more basis sets are used than optimal. $E\% > 0$ means the coverage is not perfect. The running time for incomplete solvers are little, so they are not listed. In the run-time block, *Exploration*, *Aggregation*, *Total Time Parallel* and *Sequential* show the wall times of the corresponding phases in the parallel scheme and the time to solve the instance sequentially (*Total Time Parallel* = *Exploration* + *Aggregation*).

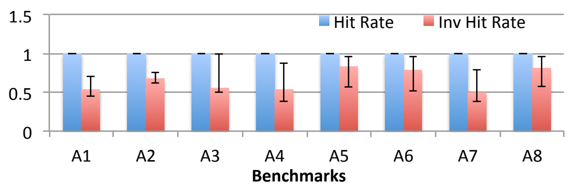

Figure 2: The overlap between the s-basis sets and the g-basis sets in each benchmark. The bars show the median value for (inverse) hitting rate, and error bars show the 10-th and 90-th percentile.

mented with) will improve from the initialization information provided by solving subproblems.

Discussion We now provide empirical evidence that justifies and explains the surprising empirical effectiveness of our method. For clarity, we call a basis set found by solving subproblems an s-basis set, and a basis set in the global solution a g-basis set. For any set S, we define the *hitting rate* as: $p(S) = \max_{B \in \mathcal{B}} |S \cap B|/|B|$, and the *inverse hitting rate* as: $ip(S) = \max_{B \in \mathcal{B}} |S \cap B|/|S|$, where B is chosen among all g-basis set. Intuitively, $p(S)$ and $ip(S)$ measure the distance between S and the closest g-basis set. Note that $p(S) = 1$ (respectively $ip(S) = 1$) implies the basis S is the superset (respectively subset) of at least one g-basis in the global solution. If $p(S)$ and $ip(S)$ are both 1, then S matches exactly to one g-basis set[4].

First, we study the overlap between a single s-basis set and all g-basis sets. As shown in Figure 2, across the benchmarks we considered the hitting rate is almost always one (with the lowest mean is for A2, which is 0.9983). This means that *the s-basis sets are almost always supersets of at least one g-basis set in the global solution*.

Next, we study the relationship between the intersection of multiple s-basis sets and g-basis sets. Figure 3 shows the median hitting rate and inverse hitting rate with respect to different number of s-basis sets involved in one intersection. The error bars show the 10-th and 90-th percentile. The result is

[4]Assuming no g-basis set is the subset of another g-basis set, which is the case in instances A1 to A8.

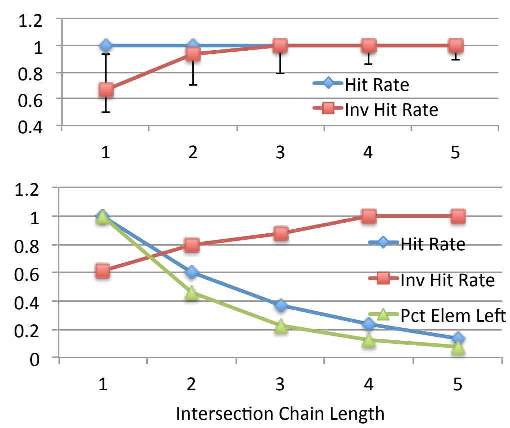

Figure 3: The median (inverse) hitting rate of a random intersection of multiple s-basis sets. X-axis shows the number of s-basis sets involved in the intersection. (Top) The basis sets in one intersection are supersets of one g-basis set. (Bottom) The basis sets in one intersection are randomly selected.

averaged over all instances A1 through A8, with equal number of samples obtained from each instance. In the top chart, the s-basis sets involved in one intersection are supersets of one common g-basis set. In this case, the hitting rate is always 1. However, by intersecting only a few (2 or 3) s-basis sets, the inverse hitting rate becomes close to 1 as well, which implies the intersection becomes very close to an *exact* match of one g-basis set. This is in contrast with the result in the bottom chart, where the intersection is among randomly selected s-basis sets. In this case, when we increase the size of the intersection, fewer and fewer elements remain in the intersection. The bottom chart of Figure 3 shows the percentage of elements left, defined as $|\cap_{i=1}^{k} A_i| / \max_{i=1}^{k} |A_i|$. When intersecting 5 basis sets, in median case less than 10% elements still remain in the intersection.

The top and bottom charts of Figure 3 provide an empirical explanation for the success of our scheme: as we randomly intersect basis sets from the solutions to the subproblems, some intersections become close to the empty set (as in the bottom

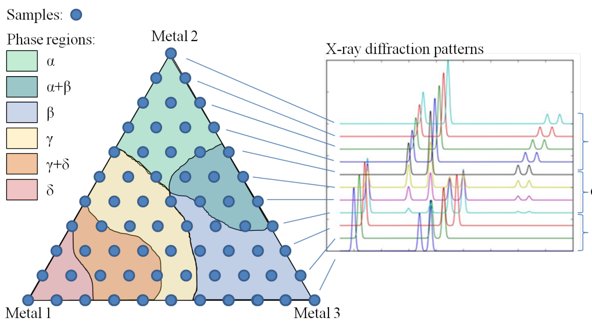

Figure 4: Demonstration of a phase identification problem. (Left) A set of of sample points (blue circles) on a silicon wafer (triangle). Colored areas show the regions where phase (basis pattern) α, β, γ, δ exist. (Right) the X-ray diffraction pattern (XRD) for sample points on the right edge of the triangle. The XRD patterns transform from single phase region α to composite phase region $\alpha + \beta$ to single phase region β, with small shiftings along neighboring sample locations.

System	# Points	Parallel (secs)	Sequential (secs)
A1	45	**119.22**	902.99
A2	45	**156.24**	588.85
A3	45	**74.37**	537.55
B1	60	**118.97**	972.8
B2	60	**177.89**	591.66
B3	60	**122.4**	1060.79
B4	60	**133.25**	633.52
C1	45	**3292.44**	17441.39
C2	45	**1186.70**	3948.41
D1	28	**207.92**	622.16
D2	28	**281.4**	2182.23
D3	28	**903.41**	2357.87

Table 3: The time for solving phase identification problems. # Points is the number of sample points in the system. Parallel and Sequential show the time to solve the problem with and without parallel initialization, respectively.

chart case), but others converge to one of the g-basis sets in the global solution (as in the upper chart case). In the second case, we obtain good solution candidates for the global problem by intersecting solutions to subproblems.

4.2 Phase Identification Problem

We also apply our parallel scheme to speed up solvers for a variant of the set basis problem with extra constraints. We show how this more general formulation can be applied to the so-called *phase identification problem* in combinatorial materials discovery [Le Bras *et al.*, 2011].

In combinatorial materials discovery, a *thin film* is obtained by depositing three metals onto a silicon wafer using guns pointing at three locations. As metals are sputtered on the silicon wafer, different locations have different combinations of the metals, due to their distances from the gun points. As a result, various crystal structures are formed across locations. Researchers then analyze the X-ray diffraction patterns (XRD) at a selected set of sample points. The XRD pattern at one sample point reflects the crystal structure of the underlying material, and is a mixture of one or more basis patterns, each of which characterizes one crystal structure. The overall goal of the phase identification problem is to explain all the XRD patterns using a small number of basis patterns.

The phase identification problem can be formulated as an extended version of the set basis problem. We begin by introducing some terminologies. Similar to [Ermon *et al.*, 2012], we use discrete representations of the XRD signals, where we characterize each XRD pattern with the locations of its *peaks*. In this model, we define a *peak* q as a set of (sample point, location) pairs: $q = \{(s_i, l_i) | i = 1, \ldots, n_q\}$, where $\{s_i | i = 1, \ldots, n_q\}$ is a set of sample points where peak q is present, and l_i is the location of peak q at sample point s_i, respectively. We use the term *phase* to refer to a basis XRD pattern. Precisely, a *phase* comprises set of peaks that occur in the same set of sample points. We use the term *partial phase* to refer to a subset of the peaks and/or a subset of the sample points of a phase. We use lower-case letters p, q, r to represent peaks, and use upper-case letters P, Q, R to represent phases. Given these definitions, the *Phase Identification Problem* is:

Given A set of X-ray diffraction patterns representing different material compositions and a set of detected peaks for each pattern; and K, the expected number of phases.

Find A set of K phases, characterized as a set of peaks and the sample points in which they are involved.

Subject to Physical constraints that govern the underlying crystallographic process. We use all the constraints in [Ermon *et al.*, 2012]. For example, one physical constraint is that a phase must span a continuous region in the silicon wafer.

Figure 4 shows an illustrative example. In this example, there are 4 peaks for phase α, and 3 peaks for phase β. Peaks in phase α exist in all sample points in the green region, and peaks in phase β exist in purple region. They co-exist in several sample points in the mid-right region of the triangle.

There is an analogy between the Phase Identification Problem and the classical Set Basis Problem. In the Set Basis Problem, each original set is the union of some basis sets. In the Phase Identification Problem, the XRD pattern at a given sample point is a mixture of several phases. Here, the phase is analogous to the basis set, and the XRD pattern at a given sample point is analogous to the original set. Because of this relationship, we employ a similar parallel scheme to solve the Phase Identification Problem, which also includes an exploration phase followed by an aggregation phase.

Exploration Phase

In the Exploration Phase, a set of subproblems are solved in parallel. For the Phase Identification Problem, a subproblem is defined as finding the minimal number of phases to explain a *contiguous* region of sample points on the silicon wafer.

This is analogous to the exploration phase defined for set basis problem – finding basis for a subset of sets. The reason why we emphasize a contiguous region is because of the underlying physical constraint: the phase found must span a contiguous region in the silicon wafer. Figure 5 shows a sample decomposition into subproblems. Here each colored small region represents a subproblem.

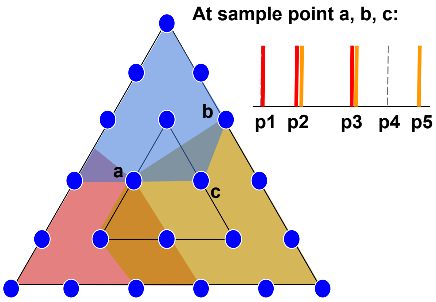

Figure 5: An example showing subproblem decomposition and merging of partial phases. (Subproblem decomposition) Each of the red, yellow and blue areas represents a subproblem, which is to find the minimal number of (partial) phases to explain all sample points in a colored area. (Merging of partial phases) Suppose partial phase A and B are discovered by solving the subproblem in the blue and the yellow region, respectively. A has peaks p_1, p_2, p_3 and all these peaks span the entire blue region, while B has peaks p_2, p_3, p_5 and all these peaks span the entire yellow region. Notice peaks p_2, p_3 match on sample points a, b and c, which are all the sample points in the intersection of the blue and yellow regions. Hence, the partial phases A and B can be merged into a larger phase C, which has peaks p_2 and p_3, but span all sample points in both the blue and yellow regions.

Aggregation Phase

The exploration phase produces a set of partial phases from solving subproblems. We call them partial because each of them describes only a subset of sample points.

As in the Set Basis Problem, we find partial phases can be merged together into larger phases. Figure 4 shows an illustrative example. Formally, two phases A and B may be *merged* into a new phase C, denoted as $C = A \circ B$, which contains all the peaks from A and B whose locations match across all the sample points they both present. The peaks in C then span the union of sample points of A and B. The merge operator $\circ$ plays the same role as the intersection operator of the Set Basis Problem. Similarly, we define $\overline{S}$ as the closure of (partial) phases S with respect to the merge operator $\circ$, which generates all possible merging of the phases in S.

Suppose $\mathcal{B}_0 = \cup_{i=1}^{s} \mathcal{B}_i$ is the set of all (partial) phases identified by solving subproblems, where $\mathcal{B}_i$ is the set of (partial) phases identified when solving subproblem i. As with the Set Basis Problem, the aggregation phase also has a pre-solving step, and a re-solving step. The pre-solving step takes as input the responses $\mathcal{B}_0$ from all subproblems, and extracts a subset of K partial phases from the closure $\overline{\mathcal{B}_0}$ as the candidate solution, which explains as many peaks on the silicon wafer as possible. The re-solving step searches in an iterative-deepening way for an exact solution, starting from the phases close to the candidate solution from the pre-solving step.

As in the pre-solving step of the Set Basis Problem, $\overline{\mathcal{B}_0}$ could be a large space and we are unable to enumerate all items in $\overline{\mathcal{B}_0}$ to find an exact solution. Instead, we take an approximate approach which first expands $\mathcal{B}_0$ to a larger set $\mathcal{B}' \subseteq \overline{\mathcal{B}_0}$ using a greedy approach. Then we employ a Mixed-Integer Program (MIP) formulation that selects the best K phases from $\mathcal{B}'$ which covers the largest number of peaks. The greedy algorithm and the MIP encoding are similar in concept to the ones used in solving the Set Basis Problem, but take into account extra physical constraints.

The Re-solving step expands the search from the pre-solving step in an iteratively deepening way to achieve completeness. Suppose the pre-solving step produces K phases $P_1^*, P_2^*, \ldots, P_K^*$. In the first round of the re-solving step, the complete solver is initialized such that the first phase must contain all the peaks of P_1^*, the second phase must contain all the peaks of P_2^*, etc. If the solver can find a solution with this initialization, then the solver terminates and returns the results. Otherwise, it usually detects a contradiction very quickly. In this case, we remove some peaks from $P_1^*, \ldots, P_K^*$ and re-solve the problem. We continue this re-solving process, until all the peaks from the Pre-solving step are removed, in which case the solver is free to explore the entire space without any restrictions. Again, this is highly unlikely in practice. In most cases, the solver is able to find solutions in the first one or two iterations.

We augmented the Satisfiability Modulo Theory formulation as described in [Ermon *et al.*, 2012] with our parallel scheme and use the Z3 solver [De Moura and Bjørner, 2008] in the experiments. We use Z3 directly in the exploration phase, and then use it as a component of an iterative deepening search scheme in the aggregation phase. Due to a rather more imbalanced distribution of the running times across different sub-problems, we only wait for 50% of sub-problem solvers to complete before conducting the aggregation phase.

Table 3 displays the experimental results for the phase identification problem. We run on the same benchmark instances used in the work of Ermon et al [Ermon *et al.*, 2012]. We can see from Table 3 that in all cases the solver completes much faster when initialized with information obtained by parallel subproblem solving. This improvement in the runtime allows us to analyze much bigger problems than previously possible in combinatorial materials discovery.

5 Conclusion

We introduced a novel angle for using parallelism to exploit hidden structure of hard combinatorial problems. We demonstrated empirical success in solving the Set Basis Problem, obtaining over an order of magnitude speedups on certain problem instances. We also identified a novel application area of the Set Basis Problem, concerning the discovery of new materials for renewable energy sources. Future directions include applying this approach to other NP-complete problems, and exploring its theoretical foundations.

Acknowledgments

We are thankful to the anonymous reviewers, Richard Beinstein and Ronan Lebras for their constructive feedback. This work was supported by NSF Expeditions grant 0832782, NSF Infrastructure grant 1059284, NSF Eager grant 1258330, NSF Inspire grant 1344201, and ARO grant W911-NF-14-1-0498.

References

[Biere, 2010] Armin Biere. Lingeling, plingeling, picosat and precosat at sat race 2010. Technical report, SAT race, 2010.

[Chrabakh and Wolski, 2003] Wahid Chrabakh and Rich Wolski. Gradsat: A parallel sat solver for the grid. Technical Report 2003-05, UCSB Computer Science, 2003.

[Chu *et al.*, 2007] Cheng Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. *NIPS*, 2007.

[Chu *et al.*, 2008] Geoffrey Chu, Peter J. Stuckey, and Aaron Harwood. Pminisat: A parallelization of minisat 2.0. Technical report, SAT race, 2008.

[De Moura and Bjørner, 2008] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[Dean and Ghemawat, 2008] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[Dilkina *et al.*, 2009] B. Dilkina, C. Gomes, Y. Malitsky, A. Sabharwal, and M. Sellmann. Backdoors to combinatorial optimization: Feasibility and optimality. *CPAIOR*, pages 56–70, 2009.

[Ene *et al.*, 2008] Alina Ene, William G. Horne, Nikola Milosavljevic, Prasad Rao, Robert Schreiber, and Robert Endre Tarjan. Fast exact and heuristic methods for role minimization problems. In Indrakshi Ray and Ninghui Li, editors, *SACMAT*, pages 1–10. ACM, 2008.

[Ermon *et al.*, 2012] S. Ermon, R. Le Bras, C. P. Gomes, B. Selman, and R. B. van Dover. Smt-aided combinatorial materials discovery. In *SAT'12*, SAT'12, 2012.

[Fischetti *et al.*, 2014] Matteo Fischetti, Michele Monaci, and Domenico Salvagnin. Self-splitting of workload in parallel computation. In *CPAIOR*, pages 394–404, 2014.

[Gomes, Winter 2009] Carla P. Gomes. Computational Sustainability: Computational methods for a sustainable environment, economy, and society. *The Bridge, National Academy of Engineering*, 39(4), Winter 2009.

[Hamadi and Sais, 2009] Youssef Hamadi and Lakhdar Sais. Manysat: a parallel sat solver. *Journal On Satisfiability, Boolean Modeling and Computation (JSAT)*, 6, 2009.

[Hamadi and Wintersteiger, 2013] Youssef Hamadi and Christoph M. Wintersteiger. Seven challenges in parallel SAT solving. *AI Magazine*, 34(2):99–106, 2013.

[Hamadi *et al.*, 2011] Youssef Hamadi, João Marques-Silva, and Christoph M. Wintersteiger. Lazy decomposition for distributed decision procedures. In *PDMC*, pages 43–54, 2011.

[Kadioglu *et al.*, 2011] Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm selection and scheduling. In *CP'11*, pages 454–469, 2011.

[Kottler and Kaufmann, 2011] Stephan Kottler and Michael Kaufmann. SArTagnan - A parallel portfolio SAT solver with lockless physical clause sharing. In *Pragmatics of SAT*, 2011.

[Le Bras *et al.*, 2011] R. Le Bras, T. Damoulas, J. M. Gregoire, A. Sabharwal, C. P. Gomes, and R. B. van Dover. Constraint reasoning and kernel clustering for pattern decomposition with scaling. In *CP'11*, pages 508–522, 2011.

[Leyton-Brown *et al.*, 2003] Kevin Leyton-Brown, Eugene Nudelman, Galen Andrew, Jim Mcfadden, and Yoav Shoham. A portfolio approach to algorithm selection. In *IJCAI'03*, pages 1542–1543, 2003.

[Malitsky *et al.*, 2011] Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Non-model-based algorithm portfolios for sat. In *SAT'11*, pages 369–370, Berlin, Heidelberg, 2011. Springer-Verlag.

[Miettinen *et al.*, 2008] Pauli Miettinen, Taneli Mielikainen, Aristides Gionis, Gautam Das, and Heikki Mannila. The discrete basis problem. *IEEE Transactions on Knowledge and Data Engineering*, 20(10):1348–1362, 2008.

[Moisan *et al.*, 2013] Thierry Moisan, Jonathan Gaudreault, and Claude-Guy Quimper. Parallel discrepancy-based search. In Schulte [2013], pages 30–46.

[Molloy *et al.*, 2009] Ian Molloy, Ninghui Li, Tiancheng Li, Ziqing Mao, Qihua Wang, and Jorge Lobo. Evaluating role mining algorithms. In Barbara Carminati and James Joshi, editors, *SACMAT*, pages 95–104. ACM, 2009.

[Nau *et al.*, 1978] Dana S. Nau, George Markowsky, Max A. Woodbury, and D. Bernard Amos. A mathematical analysis of human leukocyte antigen serology. *Mathematical Biosciences*, 40(34):243 – 270, 1978.

[O'Mahony *et al.*, 2008] Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, and Conor Nugent. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Irish Conference On Artificial Intelligence And Cognitive Science*, 2008.

[Rao and Kumar, 1993] V.N. Rao and V. Kumar. On the efficiency of parallel backtracking. *Parallel and Distributed Systems, IEEE Transactions on*, 4(4):427–437, Apr 1993.

[Regin *et al.*, 2013] Jean-Charles Regin, Mohamed Rezgui, and Arnaud Malapert. Embarrassingly parallel search. In Schulte [2013], pages 596–610.

[Schubert *et al.*, 2010] Tobias Schubert, Matthew Lewis, and Bernd Becker. Antom solver description. Technical report, SAT race, 2010.

[Schulte, 2013] Christian Schulte, editor. *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013*, 2013.

[Shu *et al.*, 2006] Guoqiang Shu, D. Lee, and Mihalis Yannakakis. A note on broadcast encryption key management with applications to large scale emergency alert systems. In *IPDPS 2006*, pages 8 pp.–, 2006.

[Stockmeyer, 1975] Larry J. Stockmeyer. The set basis problem is np-complete. Technical Report RC-5431, IBM, 1975.

[Vaidya *et al.*, 2006] Jaideep Vaidya, Vijayalakshmi Atluri, and Janice Warner. Roleminer: Mining roles using subset enumeration. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, 2006.

[Vaidya *et al.*, 2007] Jaideep Vaidya, Vijayalakshmi Atluri, and Qi Guo. The role mining problem: Finding a minimal descriptive set of roles. In *In Symposium on Access Control Models and Technologies (SACMAT)*, pages 175–184, 2007.

[Williams *et al.*, 2003] R. Williams, C.P. Gomes, and B. Selman. Backdoors to typical case complexity. In *IJCAI'03*, volume 18, pages 1173–1178, 2003.

[Xu *et al.*, 2008] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for sat. *J. Artif. Intell. Res. (JAIR)*, 32:565–606, 2008.

[Xue *et al.*, 2015] Yexiang Xue, Stefano Ermon, Carla P. Gomes, and Bart Selman. Uncovering hidden structure through parallel problem decomposition for the set basis problem: Application to materials discovery, supplementary materials, 2015.