Exploiting Incomparability in Solution Dominance: Improving General Purpose Constraint-Based Mining

Gökberk Koçak[1] and **Özgür Akgün**[1] and **Tias Guns**[2] and **Ian Miguel**[1]

Abstract.

In data mining, finding interesting patterns is a challenging task. Constraint-based mining is a well-known approach to this, and one for which constraint programming has been shown to be a well-suited and generic framework. Constraint dominance programming (CDP) has been proposed as an extension that can capture an even wider class of constraint-based mining problems, by allowing us to compare relations between patterns. In this paper we improve CDP with the ability to specify an incomparability condition. This allows us to overcome two major shortcomings of CDP: finding dominated solutions that must then be filtered out after search, and unnecessarily adding dominance blocking constraints between incomparable solutions. We demonstrate the efficacy of our approach by extending the problem specification language ESSENCE and implementing it in a solver-independent manner on top of the constraint modelling tool CONJURE. Our experiments on pattern mining tasks with both a CP solver and a SAT solver show that using the incomparability condition during search significantly improves the efficiency of dominance programming and reduces (and often eliminates entirely) the need for post-processing to filter dominated solutions.

1 Introduction

Pattern Mining is the process of finding interesting patterns in large data sets. Common pattern mining tasks include problems like the well-known problem of frequent itemset mining (FIM) (sets of items that occur together frequently) from transactional databases. Standard pattern mining tasks that require enumerating all frequent itemsets are best performed using specialised tools and algorithms [1, 44, 22]. However, a complete enumeration of all frequent itemsets is rarely what a user needs, since the number of all frequent itemsets can be very large. The main goal of pattern mining is to find a smaller number of interesting patterns for further analysis. Domain-specific side constraints [9] which restrict the search with more limitations and methods for compactly representing the outcome of a particular pattern mining task [36, 40, 41] have been proposed to increase the utility of constraint-based pattern mining. While these methods allow us to focus on interesting patterns, and represent solution sets compactly, they also result in a significantly more computationally difficult data mining task.

Constraint Programming (CP) [38] is a general purpose method for specifying decision and optimisation problems in a declarative language and finding solutions to these problems using highly efficient black-box solvers. Recent work demonstrates the utility of CP for performing constraint-based data mining tasks [15, 20, 30, 39, 19]. The main advantage of these methods is their generic nature and hence flexibility: once a CP model is developed for a certain pattern mining task, additional side constraints can be introduced without difficulty. In contrast to specialised algorithms, where incorporating domain knowledge is often difficult, side constraints often improve the performance of a black-box constraint solver. Local constraints

```
language Essence 1.3
letting ITEM be domain int(...)
letting SUPPORT be domain int(...)
given db : mset of set of ITEM
given minSupport : int
find itemset: set of ITEM
find support: SUPPORT
such that
 support = sum entry in db .
             toInt(itemset subsetEq entry),
 support >= minSupport,
 SideConstraints
```

```
dominanceRelation
  (itemset subsetEq fromSolution(itemset))
  -> (support != fromSolution(support))
```

```
incomparabilityFunction
  descending |itemset|
```

Figure 1: Closed Frequent Itemset Mining in ESSENCE. The dominance relation defines the closedness property between the currently sought solution and the previous solutions via `fromSolution`. The incomparability function is defined on cardinality using a descending order, since closedness is defined by a superset relation.

(or side constraints on a single solution) can be expressed as standard constraints in a constraint-based mining system. A number of mining tasks require constraints that are not local. The most well-known example is closed frequent itemset mining (CFIM) with side constraints [9], which we will consider in detail in Section 6.1. Enforcing the general property of closedness (and maximality [27]) for frequent itemset mining requires adding *constraints among solutions*.

[1] School of Computer Science, University of St Andrews, UK. email: {gk34,ozgur.akgun,ijm}@st-andrews.ac.uk
[2] Vrije Universiteit Brussel, Belgium. email: Tias.Guns@vub.be

In a CFIM task, a frequent itemset is only a solution if its support is greater than all of its supersets. An itemset's support is the number of transaction that contain the itemset as a subset. Consider the following database of transactions for an illustration of the closedness property.

Example 1.

$$DB = \begin{cases} T_1 = \{Bacon, Lettuce, Tomato, Cheese\} \\ T_2 = \{Bacon, Lettuce, Tomato, Onion\} \\ T_3 = \{Lettuce, Tomato, Egg, Fish\} \end{cases}$$

In this example, the itemset $\{Bacon, Lettuce, Tomato\}$ with the support 2 is a closed frequent itemset and this means all of its subsets with an equal support value are not closed frequent itemsets: $\{Bacon, Lettuce\}$ and $\{Bacon, Tomato\}$. In contrast, $\{Lettuce, Tomato\}$ is a closed frequent itemset since it has support 3.

Constraint Dominance Programming (CDP) has been suggested as a way of formulating such properties in a general way [32, 21] such that they are compatible with other arbitrary constraints. In addition to the main model where the decision variables and constraints relating to a single solution are declared in the usual way, a CDP model specifies constraints among solutions using dominance blocking constraints. Every time a solution is found during search, a new blocking constraint is added. This way, potential solutions that are dominated by a previously found solution are blocked. Following this semantics, CDP always finds all non-dominated solutions. However, without a perfect search ordering the set of solutions can include dominated solutions. Dominated solutions can then be removed using a post-processing step [32].

Our approach starts from a specification of a problem class in the abstract constraint specification language ESSENCE [18], such as the Closed Frequent Itemset Mining example in Figure 1. An ESSENCE specification comprises: problem class parameters (`given`); combinatorial objects to be found (`find`); constraints the objects must satisfy (`such that`); identifiers declared (`letting`); and an optional objective function (`min/maximising`). The key feature of the language is support for abstract decision variables, such as multiset, relation and function, and *nested* types, such as the multiset of sets in Figure 1. This makes the language ideally suited to express data mining problems (Section 4).

In this work, we extend the necessary language and search components to ESSENCE in line with [21]. Figure 1 lists the problem specification for closed frequent itemset mining in ESSENCE (explained in more detail in Section 6.1). Our novel extension is to support two new kinds of statements `dominanceRelation` and `incomparabilityFunction`, and a new keyword `fromSolution` which is only defined when used as part of a dominance relation statement. On the solver side, CONJURE [4, 2] generates concrete models from ESSENCE problem specifications, then SAVILE ROW [34] is used to target a number of backend solvers. CONJURE and SAVILE ROW are both extended to support these new statements.

Hence we improve pure CDP [32] with an explicit incomparability condition between solutions: in addition to specifying the dominance relation between solutions, we also specify when many solutions are incomparable. Blocking constraints that are generated with standard-CDP are not useful when finding other incomparable solutions and there can be as many blocking constraints as the number of solutions. An explicit incomparability specification allows enumerating all mutually incomparable solutions without generating blocking constraints after each solution. The blocking constraints are collected and added at once after all incomparable solutions are enumerated at a given level.

Contributions. We extend ESSENCE to support CDP. We define and implement *incomparability* as an enhancement to standard-CDP and call it CDP+I. Our proposal overcomes the main bottleneck of dominance programming in a solver independent manner by generating many fewer blocking constraints, allowing a natural specification of the search order, and eliminating the post-processing step which is often required to filter dominated solutions.

2 Constraint Dominance Problems

A constraint satisfaction problem is a triple (V, D, C) of decision variables, domains and constraints. A constraint dominance problem extends a constraint satisfaction problem to a quadruple (V, D, C, R) by adding a dominance relation R [32]. Dominance blocking constraints are generated from an existing solution using a template provided by the modeller. They are used to prune all solutions dominated by the solution at hand.

When solving a constraint dominance problem, the goal is to enumerate all non-dominated solutions. Semantically, a solution is non-dominated if the dominance relation statement evaluates to true in comparison with every other solution. Operationally, this is achieved by iteratively finding a solution s, posting dominance blocking constraints to disallow solutions dominated by s, and using the modified model to find the next solution [21] (Algorithm 1). This procedure creates as many dominance blocking constraints as there are solutions and requires one solver call per solution. Moreover, without a perfect search order it may produce dominated solutions in addition to all the non-dominated solutions, which a post-processing step is required to remove.

Definition 1. *Solutions A and B are incomparable if neither A dominates B, nor B dominates A.*

A significant number of pairs of solutions A and B in the solution set of a CDP tend to be incomparable with each other. For any pair of solutions, the dominance blocking constraint generated from either solution is irrelevant when searching for the other solution. In the next section we present an explicit way of capturing such conditions declaratively in a new incomparability function statement.

Algorithm 1 CDP

```
1: (V, D, C, R) ← CDP
2: SAC(V, D, C) ▷ Singleton Arc Consistency preprocessing [16]
3: while True do
4:     CSP ← (V, D, C)
5:     S ← findSolution(CSP)
6:     if S = ∅ then
7:         break
8:     B ← generateDominance(R, S)
9:     C ← C ∪ B
```

3 Dominance and Incomparable solutions

We define a new type of statement to specify incomparable solutions explicitly. This statement is only valid in a CDP problem specification, i.e. one containing a dominance relation statement. The dominance relation statement defines the dominance relation itself, similar

to the dominance blocking constraints introduced in [21]. The incomparability function statement provides a function I mapping any solution to a single value that has an orderable ESSENCE type (typically an integer). Two solutions A and B such that $I(A) = I(B)$ are said to be incomparable.

Definition 2. *A CDP+I problem specification is a quintuple (V, D, C, R, I) where R is the dominance relation similar to that of CDP and I is the incomparability function.*

An example of dominance relation and our novel incomparability function for the closed frequent itemset mining problem can be seen in Figure 1.

The incomparability function partitions the search space with non-overlapping parts. Since we enumerate all solutions for each part of the partition, it necessarily checks the whole search space. However it is not guaranteed to eliminate non dominated solutions. We present CDP+I models for five problem classes in this paper, four of these models have complete incomparability functions whereas one (Relevant Subgroup Discovery) has an incomplete incomparability function. We discuss these problem classes in more detail in Section 6.

We make use of this explicit incomparability statement by enumerating all solutions that have an equal incomparability function value. This avoids the need to add any blocking constraints after each solution that has the same incomparability value. Then, all of the necessary blocking constraints are added at once before moving to the next incomparability level. This reduces the number of solver calls required, reduces the total number of dominance blocking constraints maintained, and allows the usage of efficient solution-enumeration solvers. In addition, thanks to the explicit search order specified in the incomparability function statement, we produce fewer (or in the best case no) dominated solutions. CDP enhanced with an explicit incomparability statement is implemented in CONJURE and SAVILE ROW.

Algorithm 2 presents the CDP+I algorithm we propose. In contrast to the pure CDP algorithm proposed in [21], which iterates over the solution set, our algorithm iterates over levels jointly defined. The levels correspond to the set of values that the incomparability function takes. All solutions at a particular level i are known to be incomparable to each other, and we exploit this by running the solver to enumerate all solutions at that level. We then generate one dominance blocking constraint per solution using the template provided by the modeller in the `dominanceRelation` statement of the model. Having access to a set of blocking constraints generated from the same template presents an opportunity that is unique to level-wise search. We optionally perform a model reformulation step provided by SAVILE ROW (line 8) to reduce the size and number of constraints and to achieve better propagation. In this work we use the partial evaluator and the common subexpression elimination methods (Section 5).

Algorithm 2 CDP+I

1: $(V, D, C, R, I) \leftarrow CDP{+}I$
2: $SAC(V, D, C)$
3: $levels \leftarrow getLevels(I)$
4: **for** $l \leftarrow levels$ **do**
5: $\quad CSP \leftarrow (V, D, C + levelRestriction(l))$
6: $\quad S \leftarrow findAllSolutions(CSP)$
7: $\quad B \leftarrow generateDominance(R, S)$
8: $\quad B \leftarrow reformulate(B)$ ▷ Optional
9: $\quad C \leftarrow C \cup B$

4 Extending the language: ESSENCE

ESSENCE is unique in the abstract modelling features it provides, such as complex domains (sets, multisets, sequences, etc) and quantification over decision variables [18]. A transactional dataset of items can be represented as a multiset of sets of integers in ESSENCE if the ordering of the items in a transaction does not matter (and a multiset of sequences if the ordering does matter). If any additional information is required, complex nested types of sets of records can be used to represent the data naturally. For example, in closed discriminative itemset mining (Section 6.4) and relevant subgroup discovery problems (Section 6.5), we use a multiset of records (containing a set of integers as the itemset and a Boolean as the class identifier) to represent the transactions.

In this work, we extend ESSENCE with a small number of necessary language features to allow the natural specification of CDP. We add a `dominanceRelation` statement which contains a template for the dominance blocking constraints to be posted with respect to other solutions of the problem instance. Inside the `dominanceRelation` statement, a newly introduced `fromSolution` keyword is used to refer to values coming from other solutions of the problem instance. Operationally in [21] a solution is tested only against solutions found up to that point in search. A solution is tested against other solutions found after it in a post-processing phase. We also extend ESSENCE to add an `incomparabilityFunction` statement, which allows the modeller to specify the incomparability condition.

5 Extending the compiler: CONJURE

Standard ESSENCE problem specifications are refined into solver-independent constraint models in ESSENCE PRIME by CONJURE [4]. CONJURE selects representations (in terms of Booleans, integers and arrays) for decision variables and problem parameters, and translates problem constraints to operate on the chosen representations. For sets, CONJURE has two main representation options. First is the Occurrence representation, which uses a Boolean array indexed by potential members of the set. In this representation a `true` entry represents set membership of the corresponding index. Second is the Explicit representation, which uses an array of values together with a marker variable to represent the set cardinality. Array entries up to the marker variable are considered to be members of the set. CONJURE automatically generates the necessary symmetry breaking constraints to ensure a 1-1 correspondence between the assignments to the original set and the chosen array representation [3]. In this work, we will not explore the effect of different set representations on performance, instead we rely on the *Compact* heuristic built into CONJURE [2].

The bodies of both the dominance relation and incomparability function statements are valid ESSENCE and therefore can be refined using the existing CONJURE infrastructure.

SAVILE ROW takes constraint models written in the ESSENCE PRIME language and translates them to a backend solver, MINION or SAT. It employs a number of reformulations to improve the model for the target solver. In this work we exploit two powerful reformulations performed by SAVILE ROW: common subexpression elimination (CSE) and domain filtering using singleton arc consistency (SAC) [33, 16]. CSE is useful particularly when applied to the dominance blocking constraints in CDP+I. SAC is useful in reducing the number of levels in CDP+I and hence reducing the number of solver calls.

When encoding to SAT, we use the standard encodings provided by SAVILE ROW: direct encoding and order encoding depending on the constraint expression [34]. We use the constraint programming solver MINION for domain filtering via SAC when targeting a SAT solver as well.

CDP Algorithm 1 is implemented in SAVILE ROW with minor modifications. In SAVILE ROW, the input CSP is represented using a model object. We define a new sub-model object for implementing CDP which is instantiated for each level with mutable reference to the main model's variables, domains and constants. The sub-model does not refer to the original problem constraints. After every solution a new sub-model is created with the corresponding dominance blocking constraints and then these are appended to the main model.

CDP+I Algorithm 2 is implemented in a similar fashion. Here, at each step we enumerate all solutions. The CP solver we use (MINION) supports finding a single solution as well as enumerating all solutions natively. For SAT, we use a fast SAT solver in the context of CDP (`glucose` [5]) and a non-blocking All-SAT solver (`nbc_minisat_all` [42]) in the context of CDP+I. `nbc_minisat_all` is much faster in enumerating solutions than repeated calls to a standard SAT solver since it is specifically crafted for this purpose using a non-blocking jumping mechanism.

SAVILE ROW is capable of translating solutions back from SAT/MINION automatically into ESSENCE PRIME. For each solution, dominance blocking constraints are generated. For CDP+I, we apply CSE on the set of dominance blocking constraints. Eliminating common sub-expressions has been shown to be a very effective model reformulation in previous work [35] and having a set of similar constraints presents a natural opportunity for them to arise. We demonstrate the effect of applying CSE on a part of an instance on the hepatitis dataset from our experiments.

Example 2.

$$\begin{gathered}((is_{29} \vee is_{37} \vee is_{55} \vee is_{56} \vee is_{58} \vee is_{60} \vee is_{62} \vee (76 < sup)) \\ \wedge \\ (is_{29} \vee is_{37} \vee is_{53} \vee is_{55} \vee is_{56} \vee is_{58} \vee is_{60} \vee (80 < sup))) \\ \rightsquigarrow \\ (aux = is_{37} \vee is_{55} \vee is_{56} \vee is_{58} \vee is_{60} \\ \wedge \\ (aux \vee is_{62} \vee (76 < sup)) \wedge (aux \vee is_{37} \vee (80 < sup)))\end{gathered}$$

In this example a disjunction with 5 literals is common to two dominance blocking constraints. It is extracted by the CSE algorithm by introducing an auxiliary Boolean decision variable. In larger examples, we typically identify many more common subexpressions.

6 Problem classes

All models operate on a transactional dataset in the form of a multi-set of set of items. The relevant subgroup discovery problem requires identifying which itemsets cover the pattern, and for this problem class we use a sequence instead of a multi-set as sequences allow positional indexing whereas multi-sets do not. The decision variables are also similar for each model; we always try to find a set of items to represent the pattern and its supports (using integers) or its cover (using a set of transaction ids).

We use two side constraints on the pattern for each model; minimum value [20] and maximum cost [9, 10]. Moreover, the minimum value constraint is monotone and the maximum cost constraint is not monotone; the latter cannot be straightforwardly combined with the local formulation of the close itemset mining, as a semantically non-meaningful set of solution will be returned. Hence we demonstrate the correct handling of side constraints independent of whether they are monotone or not.

In the representation of the models, we use is to denote the decision variable itemset and $s()$ is used to access support. Additionally $prev()$ has been used to indicate previous solution's decision variables which is equivalent to `fromSolution` in ESSENCE syntax.

6.1 Closed Frequent Itemset Mining (CFIM)

Frequent itemset mining is a standard data mining problem where the task is enumerating itemsets whose *support* is above a given threshold value.

Closedness is a condition for the whole solution set of an frequent itemset mining task. Itemsets are called *closed* if and only if their support is greater than all of their supersets. CFIM also acts as a lossless way of compressing the solution set of FIM, since all frequent itemsets can easily be enumerated once the closed frequent itemsets are found [36].

Definition 3. *The dominance relation of closedness is:* $(is \subseteq prev(is)) \implies (s(is) \neq prev(s(is)))$.

Maximal frequent itemset mining is a slightly simplified version of this problem class which indicates no subset of any solution is allowed. It can be supported within the dominance framework with minimal changes, namely by removing the support condition in the dominance relation (i.e $\neg(is \subseteq prev(is))$).

We use the cardinality of the itemset is $|is|$ as the incomparability function since two itemsets with the same cardinality cannot dominate each other. This is because in the dominance relation, the left hand side of the implication is a subset operator. Itemsets with the same cardinality cannot be subsets of each other unless they are the same set (which is also not possible either since we enumerate each solution once). Descending order for subset operator is correct order to avoid dominated solutions.

6.2 Generator Itemset Mining

Generator itemsets (also called free itemsets or key itemsets [12, 11]) are a related compressed representation of all frequent itemsets. A generator itemset is a frequent itemset which does not have any frequent subsets with the same support.

Generator itemsets are useful as part of a larger association rule mining task, together with closed frequent itemsets to find minimal non-redundant association rules [29].

Definition 4. *The dominance relation* $(prev(is) \subseteq is) \implies (s(is) \neq prev(s(is)))$ *follows the definition very closely. A frequent itemset is a generator itemset if its support is not equal to the support of any of its subsets.*

The incomparability function for generator itemsets is almost the same as for closed itemsets: it uses the itemset cardinality. This condition is complete when paired with an ascending direction of search on the itemset cardinality. In contrast to CFIM, smaller itemsets dominate larger ones by definition since they do not have any frequent subsets. Then, dominance blocking constraints are added and we only find generator itemsets in ascending order.

6.3 Minimal Rare Itemset Mining

A minimal rare itemset is an infrequent itemset whose subsets are all frequent. They are closely related to maximal, closed and generator itemsets. Minimal rare itemset mining is useful for dense datasets where the number of frequent itemsets may be very large [41]. In this paper we constrain the support of an itemset to be *less* than a given frequency threshold, so a maximum frequency constraint, and greater than a given epsilon value.

Definition 5. *The dominance relation follows from the definition of minimal rare itemsets: an itemset is a solution if none of its subsets are solutions:* $\neg(prev(is) \subseteq is)$

The incomparability function uses set cardinality as well. It is complete when the search order is ascending on the itemset cardinality, since we first find small itemsets which cannot have any infrequent subsets. Then, dominance blocking constraints are added and we only find minimal rare itemsets in the successive levels.

6.4 Closed Discriminative Itemset Mining

Discriminative itemset mining operates on a slightly different dataset: in addition to transactions, each entry has an associated class label (positive/negative). We calculate two support values for a discriminative itemset, the support among positive labeled itemsets, and the support among the negatives. A discriminative itemset is one where the difference between the positive support and the negative support is greater than a frequency threshold [14]. In closed discriminative itemset mining we add an additional closedness condition.

In order to represent the dataset, we use a multi-set of records in ESSENCE. Each record contains the transaction and the class rather than only the transaction.

Definition 6. *After calculating the positive and negative support of the itemset with limiting constraints, we can apply the closure dominance on positive cover as follows:* $(is \subseteq prev(is)) \implies (s_+(is) > prev(s_+(is))$ *, where* $s_+()$ *is used to access the positive support.*

6.5 Relevant Subgroup Discovery

Relevant subgroup discovery is related to discriminative itemset mining. While discriminative itemset mining reasons on the support numbers of different classes of transactions, relevant subgroup discovery reasons using the actual sets of transactions that provide the support [31, 32]. A relevant subgroup X is an itemset such that at least one of following conditions hold; 1) For positive transactions, no other itemset covers a superset of the transactions covered by X, 2) For negative transactions, no other itemset covers a subset of the transactions covered by X or 3) For both kinds of transactions, no other itemset that has the same total cover is a superset of X.

We represent the dataset for this model, using a sequence of records instead of the multi-set that we had in discriminative itemset mining. This is to allow us to store references to transactions using their index in the sequence.

The decision variables are modified accordingly to encode the transaction identifiers instead of just the total number of transactions that provide the support. We post the following constraints to calculate the cover sets and enforce the minimum support condition on the positive transactions.

Definition 7. *The dominance relation for relevant subgroup discovery is more complex, but it follows the definition given above.*

$$\begin{aligned}\neg(c_+(is) \subseteq prev(c_+(is))) \vee \neg(prev(c_-(is)) \subseteq c_-(is)) \\ \vee \neg((c_+(is) \cup c_-(is) = prev(c_+(is)) \\ \cup prev(c_-(is))) \implies is \subseteq prev(is))\end{aligned}$$

Here, c_+ and c_- is used for positive and negative cover on transactions.

For the incomparability function, the descending order on the cardinality of the itemset can be applied. This incomparability function is not complete though: some dominated solutions may be found when using it. Even though it is not complete, it helps CDP+I in solution performance because it still eliminates a large number of dominated solutions.

Example 3.

$$DB = \begin{cases} T_1 = \{1,2,3,4\}, Class_1 = 1 \\ T_2 = \{1,2,3,4\}, Class_2 = 1 \\ T_3 = \{1,2,3,5\}, Class_3 = 1 \end{cases}$$

An example follows to demonstrate that the cardinality is not complete for incomparability. The itemset $\{1,2\}$ dominates $\{3,4\}$ because the positive cover of the former includes all transactions whereas the positive cover of the latter only includes the first two. This violates the first component of the dominance relation given above.

7 Empirical analysis

In our experiments, we use 12 transactional datasets from the CP4IM website[3]. They are derived from UCI datasets, meant to be used for constraint-based itemset mining.

We included a minimum value and a maximum cost side constraints in all of our models to demonstrate their ability to handle arbitrary side constraints. Due to the inclusion of side constraints, specialised data mining algorithms are applicable on these tasks. We generate values between 0 and 5 for item values and costs using uniform randomness. In addition, we generate a threshold for the minimum value and the maximum cost constraints as well. We systematically generate several candidate instances and choose instances which have a reasonable number of solutions (in the order 10,000 at most for all problem classes except minimal rare itemset mining and in the order of 100,000 for minimal rare) and those that can be solved within our time limit of 6 hours[4]. We generate instances at 5 frequency levels: 10%, 20%, 30%, 40%, 50%. For CFIM we use the instances published in the appendix of [26].

For each problem class, we solve each instance using 4 pairs of solver configurations. The first two configurations are for standard CDP with the default search order provided by the solver, one with a CP solver (MINION) and one with a SAT solver (`glucose`). The second two configurations are for CDP with the search order specified in the incomparability function statement. The third two configurations are for CDP+I CP/SAT and the last two are for CDP+I with reformulation enabled. We use a CP solver (MINION) and an AllSAT solver (`nbc_minisat_all`) for CDP+I configurations.

[3] https://dtai.cs.kuleuven.be/CP4IM/datasets/

[4] Our Github repository of the data and results is available at https://github.com/stacs-cp/ECAI2020-CDP [28]

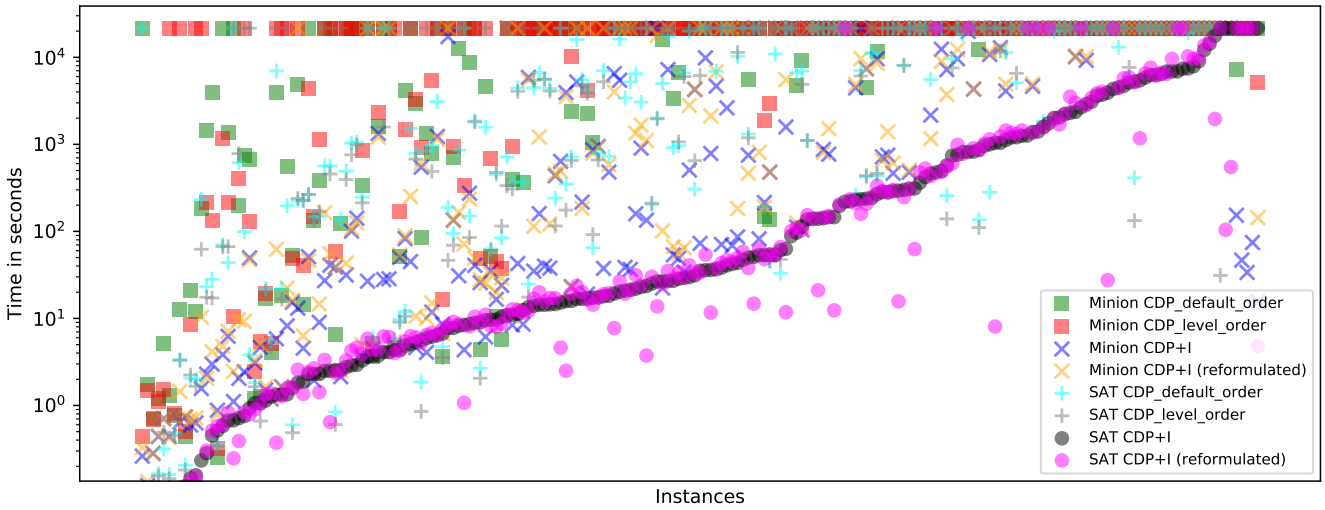


Figure 2: Solver time for all instances, sorted by SAT CDP+I. Timeouts are also shown on the top of the plot.

We run every SAT instance 3 times with different seeds and present averages. We run MINION instances only once with a deterministic static search ordering.

We run our experiments on two identical 32 core AMD Opteron 6272 machines, at 2.1 GHz and with 256GB memory. We run 31 cores in parallel on each machine and left one core idle to account for system processes. Each separate experiment was given a single CPU core, 8GB of memory and a 6-hours time limit using cgroups. Total time to run all the experiments was roughly 3 CPU years.

7.1 Results

The experiments are conducted in a way to examine and explain the following:

- Impact of adding a level-wise search order to CDP (CDP-default-order vs CDP-level-order, see Figure 3a)
- Impact of CDP+I in general with respect to standard CDP (CDP-level-order vs CDP+I, see Figure 3b)
- Impact of model reformulations in CDP+I (CDP+I vs CDP+I with model reformulation, see Figure 3c)
- Comparison of the solving methodology on CDP-level-order and CDP+I (MINION vs SAT, see Figure 3d)

For a general picture, Figure 2 shows the time spent solving all instances of the five problem classes using the 8 configurations. The results are sorted by time spent by SAT CDP+I without reformulation. The timed out instances are also included at the 6-hour mark near the top of the plot. The results show that CDP+I configurations are significantly better than CDP configurations on most instances. While SAT CDP+I configurations require the least time in the majority of the cases, for a small number of instances the SAT CDP configuration is the fastest; these instances have a very small number of levels of solutions. For a small number of instances Minion CDP+I is the best configuration; these instances contain large amounts of data and the size of the SAT encoding gets prohibitively large.

The optional model reformulation brings a negligible overhead on most instances and on some instances it helps significantly.

For a small number of instances the default search order performs better for SAT; these instances have a very small number of solutions all of which are non-dominated. In these cases the small overhead of applying a specific search order does not pay off.

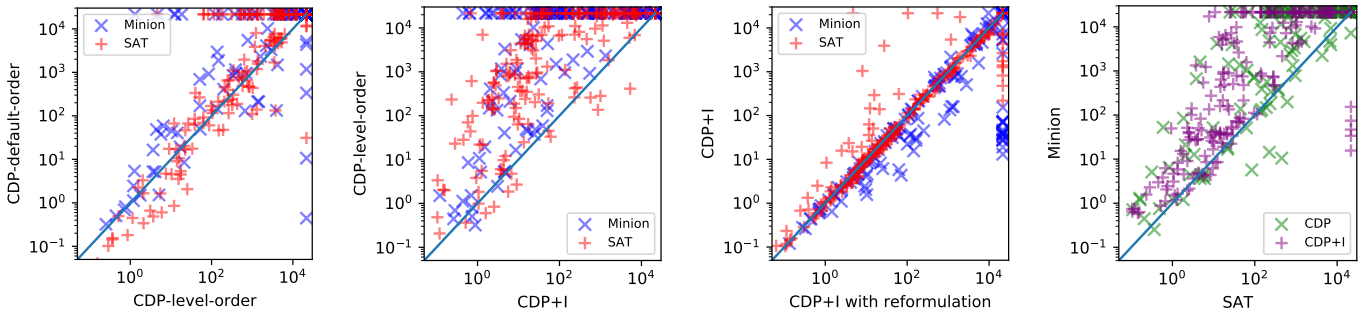
Effect of CDP order Figure 3a presents a comparison plot between two different CDP configurations with MINION and SAT. On easy instances, level ordering creates an overhead. However, for difficult instances using the same search order defined in the incomparability function helps. CDP-level-order solves many more instances than CDP-default order where a large number of instances time out.

CDP vs CDP+I Figure 3b presents a comparison plot between CDP-level-order and CDP+I on the same instances, using both MINION and SAT. We only compare against CDP-level-order since it performs better than CDP-default order in general. This plot shows the direct effect of using a level-wise search and adding the dominance blocking constraints in batches. A point above the diagonal line means CDP+I performs better, which is a significant majority of the instances. Both solving methodologies clearly benefit from CDP+I thanks to typically fewer solver calls (once per level as opposed to once per solution) and retaining learned clauses for SAT.

Effect of reformulation on CDP+I Figure 3c presents a comparison plot between two different CDP+I configurations for MINION and SAT: with and without the optional model reformulation. The results show that the reformulation does not consistently help solving time. The reformulation hurts the performance of the CP solver MINION more. However, the performance of the SAT solver `nbc_minisat_all` up to median difficulty is improved. For the most difficult instances, the reformulation has mixed effects. The auxiliary variables added by CSE create a connection between the dominance blocking constraints and this can help propagation only for a subset of the instances. When the help is not significant enough, they create an unnecesary overhead.

MINION vs SAT Figure 3d presents a comparison plot between MINION and SAT for CDP and CDP+I. We use CDP-level-order and CDP+I without reformulations in this plot. SAT performs better for most instances and it benefits from using incomparability more. This is likely to be due to the learning employed by SAT solvers. In contrast, MINION is not a learning solver.

In Table 1 we focus on the five problem classes separately to see the effect of the complexity of the dominance relation. Results show that SAT is always significantly better for all problem classes except the Relevant Subgroup Discovery problem where the complexity of



(a) CDP default vs level-order (b) CDP-level-order vs CDP+I (c) Reformulation in CDP+I (d) MINION vs SAT

Figure 3: Comparing the eight configurations. All plots present solver times (in seconds) with a 6-hour timeout. Timed out instances are near the top and the right borders.

dominance relation is the highest.

Problem	CP wins	SAT wins	Similar time
Closed	0	40	18
Generator	0	36	17
Minimal	0	17	13
Discriminating	0	18	14
Relevant	5	19	19

Table 1: Solver time comparison on CP vs SAT on CDP+I. Substantial differences ($> 50s$) are reported as wins. Similar time indicates the CP and the SAT solver reached the solutions approximately around the same time (± 50 seconds).

Number of solver calls CDP variants perform better than CDP+I equivalents for 25 instances. All of these instances have a shared characteristic: they have a small number of solutions (less than 10, which is smaller than the number of levels). In the extreme case of instances with significantly fewer solutions than levels using CDP+I has a large overhead. In these cases the cost of making one solver call per solution becomes negligible.

8 Related work

Our novel system is inspired by level-wise search [13, 8], the dominant form of search in early pattern mining work while being inside a general CP-modelling environment in a purely declarative fashion.

There is significant related work on specialised SAT encodings for itemset mining problems [24, 25], maximal itemset mining with side constraints [23] and work on global constraints for itemset mining problems [30, 39, 6]. These can be fruitfully integrated into our framework to improve the performance of CDP and CDP+I even further. Using the standard constraint formulations and encodings produced by CONJURE and SAVILE ROW was fair in our work to evaluate the effect of CDP+I, since we use the same encodings for both CDP and CDP+I. They are both likely to benefit from improved encodings in similar ways.

Model reformulations in constraint programming and SAT have been shown to be effective ways of increasing solution performance [34]. We use the CSE algorithm found in SAVILE ROW in this work. For general CNF formulas, finding an equivalent CNF formula that is minimal in the number of literals is NP-hard [43]. There are cheaper alternative methods of reducing the size of CNF formulas [17]. Exploring other ways of reformulation and compressing CP models and SAT encodings is possible thanks to the batch addition of the dominance blocking constraints.

In the context of itemset mining, finding undominated solutions under constraints is proven to be coNP-Hard [7] which eliminates one-shot CSP approaches and justifies the usage of CDP.

Skypatterns is another problem class that can be specified using dominance relations [37]. This problem class can be straightforwardly modelled in ESSENCE using our CDP extensions.

MiningZinc [20, 19] is a declarative framework for specifying data mining problems on top of the solver independent CP modelling language MiniZinc. Dominance programming features (including incomparability) can be implemented on top of MiningZinc as well to benefit from similar performance gains. However, the representations of the nested sets, multisets and sequences available natively in ESSENCE are not available in MiningZinc and would have to be implemented.

9 Conclusion

In this paper we extended the high-level problem specification language ESSENCE to support dominance programming features. In addition, we introduced a novel incompatibility function statement, which allows the modeller to specify a condition that can be used to identify incomparable solutions. We demonstrated significant performance gains thanks to the combined CDP+I framework. CDP+I produces a drastically reduced number of dominated solutions and requires fewer solver calls for difficult problems. Equipped with CDP+I capabilities, ESSENCE becomes a particularly suitable language for specifying and solving constraint-based itemset mining problems that both contain problem specific side constraints and constraints among solutions (such as closedness). The implementation is solver independent thanks to CONJURE, SAVILE ROW and our additional effort. Furthermore, we have shown that we can apply CDP+I to a wider range of mining tasks beyond itemsets, such as relevant subgroup discovery.

In the CDP+I algorithm, a set of dominance blocking constraints are collected within an incomparability level and are added as a single batch. This allows us to evaluate the effect of model reformulation on the dominance blocking constraints before they are added to the model, which is uniquely possible thanks to the level-wise search.

Future work includes the application of CDP+I to a wider range of problem classes, both in data mining and beyond. We believe multi-objective optimisation problems will be a natural next application area for CDP+I. Other possible model reformulation mechanisms can be examined. Specifically for the SAT backend, techniques to find shorter and/or more efficient CNF formulas can be applied to improve efficiency of the CDP+I framework even further.

REFERENCES

[1] Rakesh Agrawal, Ramakrishnan Srikant, et al., 'Fast algorithms for mining association rules', in *20th int. conf. very large data bases, VLDB*, volume 1215, pp. 487–499, (1994).

[2] Özgür Akgün, Alan M Frisch, Ian P Gent, Bilal Syed Hussain, Christopher Jefferson, Lars Kotthoff, Ian Miguel, and Peter Nightingale, 'Automated symmetry breaking and model selection in conjure', in *International Conference on Principles and Practice of Constraint Programming*, pp. 107–116. Springer, (2013).

[3] Özgür Akgün, Ian P Gent, Christopher Jefferson, Ian Miguel, and Peter Nightingale, 'Breaking conditional symmetry in automated constraint modelling with conjure.', in *ECAI*, pp. 3–8, (2014).

[4] Özgür Akgün, Ian Miguel, Chris Jefferson, Alan M Frisch, and Brahim Hnich, 'Extensible automated constraint modelling', in *Twenty-Fifth AAAI Conference on Artificial Intelligence*, (2011).

[5] Gilles Audemard and Laurent Simon, 'Predicting learnt clauses quality in modern sat solvers', in *Twenty-first International Joint Conference on Artificial Intelligence*, (2009).

[6] Mohamed-Bachir Belaid, Christian Bessiere, and Nadjib Lazaar, 'Constraint programming for association rules', in *Proc. of the 2019 SIAM International Conference on Data Mining*, pp. 127–135. SIAM, (2019).

[7] Mohamed-Bachir Belaid, Christian Bessiere, and Nadjib Lazaar, 'Constraint programming for mining borders of frequent itemsets', (2019).

[8] Francesco Bonchi, Fosca Giannotti, Alessio Mazzanti, and Dino Pedreschi, 'Examiner: Optimized level-wise frequent pattern mining with monotone constraints', in *International Conference on Data Mining*, pp. 11–18. IEEE, (2003).

[9] Francesco Bonchi and Claudio Lucchese, 'On closed constrained frequent pattern mining', in *Fourth IEEE International Conference on Data Mining (ICDM'04)*, pp. 35–42. IEEE, (2004).

[10] Francesco Bonchi and Claudio Lucchese, 'Extending the state-of-the-art of constraint-based pattern discovery', *Data & Knowledge Engineering*, **60**(2), 377–399, (2007).

[11] J-F Boulicaut and Baptiste Jeudy, 'Mining free itemsets under constraints', in *International Database Engineering and Applications Symposium*, pp. 322–329. IEEE, (2001).

[12] Jean-Francois Boulicaut, Artur Bykowski, and Christophe Rigotti, 'Approximation of frequency queries by means of free-sets', in *European Conference on Principles of Data Mining and Knowledge Discovery*, pp. 75–85. Springer, (2000).

[13] Raymond Chan, Qiang Yang, and Yi-Dong Shen, 'Mining high utility itemsets', in *Third IEEE international conference on data mining*, pp. 19–26. IEEE, (2003).

[14] Hong Cheng, Xifeng Yan, Jiawei Han, and Chih-Wei Hsu, 'Discriminative frequent pattern analysis for effective classification', in *2007 IEEE 23rd International Conference on Data Engineering*, pp. 716–725. IEEE, (2007).

[15] Luc De Raedt, Tias Guns, and Siegfried Nijssen, 'Constraint programming for itemset mining', in *SIGKDD international conference on Knowledge discovery and data mining*, pp. 204–212. ACM, (2008).

[16] Romuald Debruyne and Christian Bessiere, 'Some practicable filtering techniques for the constraint satisfaction problem', in *In Proceedings of IJCAI'97*. Citeseer, (1997).

[17] Thomas Eiter, Kazuhisa Makino, and Georg Gottlob, 'Computational aspects of monotone dualization: A brief survey', *Discrete Applied Mathematics*, **156**(11), 2035–2049, (2008).

[18] Alan M Frisch, Warwick Harvey, Chris Jefferson, Bernadette Martínez-Hernández, and Ian Miguel, 'Essence: A constraint language for specifying combinatorial problems', *Constraints*, **13**(3), 268–306, (2008).

[19] Tias Guns, Anton Dries, Siegfried Nijssen, Guido Tack, and Luc De Raedt, 'Miningzinc: A declarative framework for constraint-based mining', *Artificial Intelligence*, **244**, 6–29, (2017).

[20] Tias Guns, Anton Dries, Guido Tack, Siegfried Nijssen, and Luc De Raedt, 'Miningzinc: A modeling language for constraint-based mining', in *Twenty-Third International Joint Conference on Artificial Intelligence*, (2013).

[21] Tias Guns, Peter J Stuckey, and Guido Tack, 'Solution dominance over constraint satisfaction problems', *arXiv:1812.09207*, (2018).

[22] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao, 'Mining frequent patterns without candidate generation: A frequent-pattern tree approach', *Data mining and knowledge discovery*, **8**(1), 53–87, (2004).

[23] Said Jabbour, Fatima Ezzahra Mana, Imen Ouled Dlala, Badran Raddaoui, and Lakhdar Sais, 'On maximal frequent itemsets mining with constraints', in *International Conference on Principles and Practice of Constraint Programming*, pp. 554–569. Springer, (2018).

[24] Said Jabbour, Lakhdar Sais, and Yakoub Salhi, 'Decomposition based sat encodings for itemset mining problems', in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 662–674. Springer, (2015).

[25] Said Jabbour, Lakhdar Sais, and Yakoub Salhi, 'Mining top-k motifs with a sat-based framework', *Artificial Intelligence*, **244**, 30–47, (2017).

[26] Gökberk Koçak, Özgür Akgün, Ian Miguel, and Peter Nightingale, 'Closed frequent itemset mining with arbitrary side constraints', in *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 1224–1232. IEEE, (2018).

[27] Gökberk Koçak, Özgür Akgün, Ian Miguel, and Peter Nightingale, 'Maximal frequent itemset mining with non-monotonic side constraints', *24th International Conference on Principles and Practice of Constraint Programming - Doctoral Program*, 1, (2018).

[28] Gökberk Koçak, Özgür Akgün, Tias Guns, and Ian Miguel. Experiments for ECAI2020 CDP paper. DOI: 10.5281/zenodo.3675340, February 2020.

[29] Marzena Kryszkiewicz, 'Representative association rules and minimum condition maximum consequence association rules', in *European Symposium on Principles of DM and KD*, pp. 361–369. Springer, (1998).

[30] Nadjib Lazaar, Yahia Lebbah, Samir Loudni, Mehdi Maamar, Valentin Lemière, Christian Bessiere, and Patrice Boizumault, 'A global constraint for closed frequent pattern mining', in *International Conference on Principles and Practice of Constraint Programming*, pp. 333–349. Springer, (2016).

[31] Florian Lemmerich, Mathias Rohlfs, and Martin Atzmueller, 'Fast discovery of relevant subgroup patterns', in *Twenty-Third International FLAIRS Conference*, (2010).

[32] Benjamin Negrevergne, Anton Dries, Tias Guns, and Siegfried Nijssen, 'Dominance programming for itemset mining', in *International Conference on Data Mining*, pp. 557–566. IEEE, (2013).

[33] Peter Nightingale, Özgür Akgün, Ian P Gent, Christopher Jefferson, and Ian Miguel, 'Automatically improving constraint models in savile row through associative-commutative common subexpression elimination', in *Principles and Practice of Constraint Programming*, pp. 590–605. Springer, (2014).

[34] Peter Nightingale, Özgür Akgün, Ian P Gent, Christopher Jefferson, Ian Miguel, and Patrick Spracklen, 'Automatically improving constraint models in savile row', *Artificial Intelligence*, **251**, 35–61, (2017).

[35] Peter Nightingale, Patrick Spracklen, and Ian Miguel, 'Automatically improving sat encoding of constraint problems through common subexpression elimination in savile row', in *Principles and Practice of Constraint Programming*, pp. 330–340. Springer, (2015).

[36] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal, 'Discovering frequent closed itemsets for association rules', in *International Conference on Database Theory*, pp. 398–416. Springer, (1999).

[37] Willy Ugarte Rojas, Patrice Boizumault, Samir Loudni, Bruno Crémilleux, and Alban Lepailleur, 'Mining (soft-) skypatterns using dynamic csp', in *International Conference on AI and OR Techniques in CP for Combinatorial Optimization Problems*, pp. 71–87, (2014).

[38] Francesca Rossi, Peter Van Beek, and Toby Walsh, *Handbook of constraint programming*, Elsevier, 2006.

[39] Pierre Schaus, John OR Aoga, and Tias Guns, 'Coversize: a global constraint for frequency-based itemset mining', in *International Conference on Principles and Practice of Constraint Programming*, pp. 529–546. Springer, (2017).

[40] Arnaud Soulet and François Rioult, 'Efficiently depth-first minimal pattern mining', in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 28–39. Springer, (2014).

[41] Laszlo Szathmary, Amedeo Napoli, and Petko Valtchev, 'Towards rare itemset mining', in *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*, volume 1, pp. 305–312. IEEE, (2007).

[42] Takahisa Toda and Takehide Soh, 'Implementing efficient all solutions sat solvers', *Journal of Experimental Algorithmics (JEA)*, **21**, 1–12, (2016).

[43] Christopher Umans, 'The minimum equivalent dnf problem and shortest implicants', *Journal of Computer and System Sciences*, **63**(4), 597–611, (2001).

[44] Mohammed Javeed Zaki, 'Scalable algorithms for association mining', *IEEE transactions on knowledge and data engineering*, **12**(3), 372–390, (2000).