INFORMEDQX: Informed Conflict Detection for Over-Constrained Problems

Viet-Man Le\textsuperscript{1,3}, Alexander Felfernig\textsuperscript{1}, Thi Ngoc Trang Tran\textsuperscript{1,4}, Mathias Uta\textsuperscript{2}

\textsuperscript{1} Graz University of Technology, Graz, Austria
\textsuperscript{2} Siemens Energy AG, Germany
\textsuperscript{3} University of Economics, Hue University, Hue, Vietnam
\textsuperscript{4} School of Hospitality and Tourism, Hue University, Hue, Vietnam

\{vietman.le,alexander.felfernig,ttrang\}@ist.tugraz.at, mathias.uta@siemens-energy.com

Abstract

Conflict detection is relevant in various application scenarios, ranging from interactive decision-making to the diagnosis of faulty knowledge bases. Conflicts can be regarded as sets of constraints that cause an inconsistency. In many scenarios (e.g., constraint-based configuration), conflicts are repeatedly determined for the same or similar sets of constraints. This misses out on the valuable opportunity for leveraging knowledge reuse and related potential performance improvements, which are extremely important, specifically interactive constraint-based applications. In this paper, we show how to integrate knowledge reuse concepts into non-instructive conflict detection. We introduce the INFORMEDQX algorithm, which is a reuse-aware variant of QUICKXPLAIN. The results of a related performance analysis with the Linux 2.6.3.33 configuration knowledge base show significant improvements in terms of runtime performance compared to QUICKXPLAIN.

Introduction

Conflict detection has many applications in constraint-based systems (and beyond) (Junker 2004; Rossi, van Beek, and Walsh 2006). Examples thereof are recommender systems (Felfernig and Burke 2008), knowledge-based configuration (Junker 2006), scheduling (Baptiste et al. 2006), and knowledge base testing and debugging (Felfernig et al. 2004). In such scenarios, the task of conflict detection is to identify minimal sets of constraints (so-called conflict sets or conflicts) that can be interpreted as an explanation for the given inconsistency. Often associated with conflict detection is conflict resolution (often denoted as diagnoses), which focuses on resolving all identified conflicts so that knowledge base consistency can be restored (Reiter 1987; de Kleer and Williams 1987).

Especially in interactive settings, there is often a need to identify preferred conflicts (Junker 2004; O’Sullivan et al. 2007; Walsh 2007; Rossi, Venable, and Walsh 2011), i.e., conflicts whose resolution could be regarded as acceptable for a user. For example, users of a car configurator with strong preferences regarding an upper price limit are more inclined (in the case that a configurator cannot find a solution) to accept some relaxations of technical car features before accepting to further extend the predefined price limit.

With an increasing size and complexity of the underlying knowledge bases, there is a need to further improve the performance of the reasoning engines as well as related algorithms for conflict detection and resolution (Jannach, Schmitz, and Schheekotykhin 2015). To tackle related scalability issues, different approaches have been developed in constraint solving (Papescu et al. 2018), algorithmic parallelization (Jannach, Schmitz, and Schheekotykhin 2015; Gent et al. 2018; Vidal et al. 2021; Le et al. 2023), and knowledge compression techniques (Cheng and Yap 2005).

On the basis of a simplified working example from the domain of smartwatch configuration, we introduce a hybrid approach to conflict detection. The related algorithm (INFORMEDQX) that helps to decide, based on the existing historic conflict data, if an activation of conflict detection is still needed or if a preferred conflict has already been determined in previous configuration sessions. In this context, we focus on scenarios where the complete set of conflicts cannot be determined ahead due to complexity reasons, i.e., we want to reuse pre-existing conflicts and to figure out on the algorithmic level when the activation of a conflict detection algorithm is still needed due to the fact that further (more preferred) conflicts exist in a given constraint set.

Conflict detection algorithms have been proposed in the context of different knowledge representations such as constraint solving (Junker 2004; Schheekotykhin, Jannach, and Schmitz 2015) and SAT solving (Lifiton and Sakallah 2008; Marques-Silva and Previti 2014a). In this context, the terms minimal unsatisfiable cores (MUC) or minimal unsatisfiable subsets (MUS) are often used as synonyms for minimal conflict sets. In this paper, we focus on complete and optimal conflict detection algorithms, i.e., algorithms that are able to find a minimal conflict if one exists and at the same time find the optimal solution with regard to a predefined optimality criteria.

For demonstration purposes, we show how to exploit knowledge about existing conflicts in the context of the QUICKXPLAIN algorithm (Junker 2004) which is a divide-and-conquer based algorithm for the determination of preferred conflicts, i.e., conflicts that take into account pre-
defined preferences of users. QUICKXPLAIN is knowledge-representation agnostic, i.e., applicable to different knowledge representations such as constraint solving (Junker 2004), SAT solving (Marques-Silva and Previti 2014a), answer set programming (ASP) (Erdem, Gelfond, and Leone 2016), and description logics (DL) (McGuinness 2007), and does not exploit properties of a specific application domain.

Following a divide-and-conquer search regime, the underlying idea of QUICKXPLAIN is to divide the conflict solution space as efficiently as possible with the goal to identify subset-minimal conflict sets (one preferred minimal conflict at a time). QUICKXPLAIN has shown to work efficiently in many application scenarios (Rodler 2022), however, with an increasing size and complexity of the underlying knowledge bases, further mechanisms are needed to assure algorithm scalability (Vidal et al. 2021).

In this paper, we introduce a new algorithm (INFORMEDQX) that decides on a meta-level in which contexts there is a need to activate QUICKXPLAIN or whether a preferred conflict has already been determined in a previous configuration session. Interestingly, there exist scenarios in-between the two extreme cases that motivated the work presented in this paper. We introduce intelligent conflict reuse, which is specifically needed in scenarios without complete conflict knowledge, i.e., where not all conflicts of a knowledge base are known ahead – predetermining such conflicts in the general case is known to be a hard problem (Gregoire, Mazure, and Piette 2008). In such situations, we have to find a solution for exploiting incomplete conflict knowledge, which helps to improve the performance of conflict detection as a whole.

The major contributions of this paper are the following:

1. We introduce an algorithm (INFORMEDQX) and corresponding principles that help to make conflict search more efficient given a setting where some parts of the conflict space are already known.
2. On the basis of an evaluation with a real-world configuration knowledge base, we show a significantly improved conflict detection performance compared to the basic QUICKXPLAIN algorithm.
3. INFORMEDQX is not limited to the application in constraint solving scenarios but can as well be applied in the context of other knowledge representations such as ASP, SAT, and DL.

**Example Configuration Setting**

In the context of a simplified example from the domain of constraint-based smart watch configuration, we now introduce the definitions of a configuration task (Definition 1) and a corresponding configuration (Definition 2) (see also (Felfernig et al. 2014)). The following discussions are based on a constraint satisfaction problem (CSP) knowledge representation (Rossi, van Beek, and Walsh 2006).

**Definition 1 (Configuration task and Knowledge base).** A configuration task \( (V, C) \) can be defined as a CSP where
\[
V = \{v_1, v_2 \ldots v_n\}
\]
is a set of variables with finite domain definitions \( \text{dom}(v_i) \) indicating the domain of each variable \( v_i \in V \) and (2) \( C = C_{KB} \cup C_R \) is a set of constraints restricting possible solutions of a configuration task. In this context, \( C_{KB} \) represents a set of domain-specific constraints and \( C_R \) represents a set of user requirements. Finally, \( (V, C_{KB}) \) is denoted as configuration knowledge base.

Given such a definition of a configuration task, we now introduce the definition of a corresponding configuration.

**Definition 2 (Configuration).** A configuration (solution) \( S \) for a given configuration task \( (V, C) \) is an assignment \( A = \{a_1 = a_1 \ldots v_n = a_n\} \), \( a_i \in \text{dom}(v_i) \). \( S \) is valid if it is complete (each variable in \( V \) has a value) and consistent (\( S \) fulfills the constraints in \( C \)).

Based on these definitions, the definition of a simplified smart watch configuration task looks like as follows (see Example 1). A configurable Smartwatch must have at least one type of Connector and a Screen. The Connector can be of one or more of the types of GPS, Cellular, Wifi, and Bluetooth. The Screen can be either Analog, High Resolution, or E-ink. Besides, a Smartwatch may include a Camera, a Compass, and a Speaker. Furthermore, Compass requires a GPS, Camera requires a High Resolution, and Cellular and Analog exclude each other. A constraint-based representation of these restrictions can be found in Table 1.

**Example 1 (Smartwatch configuration task).** A CSP-based Smartwatch configuration task \( (V, C) \) is the following:

\[
\begin{align*}
V &= \{\text{Smartwatch}, \text{Connector}, \text{Screen}, \text{Camera}, \\
&\quad \text{Compass}, \text{Speaker}, \text{GPS}, \text{Cellular}, \text{Wifi}, \\
&\quad \text{Bluetooth}, \text{Analog}, \text{High Resolution}, \text{E-ink}\} \\
\text{dom(Smartwatch)} &= \{t\text{true}, f\text{alse}\} \ldots \text{dom(E-ink)} = \{t\text{true}, f\text{alse}\} \\
C_{KB} &= \{c_0 \ldots c_{10}\}, C_R = \{c_{11} \ldots c_{19}\}.
\end{align*}
\]

<table>
<thead>
<tr>
<th>CSP Representation</th>
<th>Domain-specific Constraints (( C_{KB} ))</th>
<th>User Requirements (( C_R ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_0 )</td>
<td>Smartwatch = t</td>
<td>( c_{11} )</td>
</tr>
<tr>
<td>( c_1 )</td>
<td>Smartwatch ( \leftrightarrow ) Connector</td>
<td>( c_{12} )</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>Smartwatch ( \leftrightarrow ) Screen</td>
<td>( c_{13} )</td>
</tr>
<tr>
<td>( c_3 )</td>
<td>Camera ( \rightarrow ) Smartwatch</td>
<td>( c_{14} )</td>
</tr>
<tr>
<td>( c_4 )</td>
<td>Compass ( \rightarrow ) Smartwatch</td>
<td>( c_{15} )</td>
</tr>
<tr>
<td>( c_5 )</td>
<td>Speaker ( \rightarrow ) Smartwatch</td>
<td></td>
</tr>
<tr>
<td>( c_6 )</td>
<td>Connector ( \rightarrow ) (GPS ( \lor ) Cellular ( \lor ) Wifi ( \lor ) Bluetooth)</td>
<td></td>
</tr>
<tr>
<td>( c_7 )</td>
<td>Screen ( \leftrightarrow ) xor(Analog, High Resolution, E-ink)</td>
<td>( c_{16} )</td>
</tr>
<tr>
<td>( c_8 )</td>
<td>Camera ( \rightarrow ) High Resolution</td>
<td>( c_{17} )</td>
</tr>
<tr>
<td>( c_9 )</td>
<td>Compass ( \rightarrow ) GPS</td>
<td>( c_{18} )</td>
</tr>
<tr>
<td>( c_{10} )</td>
<td>( \neg ) (Cellular ( \land ) Analog)</td>
<td>( c_{19} )</td>
</tr>
</tbody>
</table>

| Table 1: Configuration task constraints \( C_{KB} = \{c_0 \ldots c_{10}\} \) and \( C_R = \{c_{11} \ldots c_{19}\} \). \( c_0 : \text{Smartwatch} = t \) is a node constraint, avoiding the derivation of empty configurations. \( c_7 : \text{Screen} \leftrightarrow \) xor(Analog, High Resolution, E-ink) returns true, if exactly one out the three variables is true, otherwise it returns false. |
In our example, some of the user requirements $C_R$ (see Table 1) are inconsistent with the constraints in $C_{KB}$, i.e., the solver is not able to find a related configuration. For instance, $c_{11}$ and $c_{13}$ in $C_R$ are inconsistent with $c_{10}$ in $C_{KB}$. Therefore, no solution can be found for this configuration task. In such situations, we are interested in explanations as to why no solution could be identified. Minimal conflict sets, also denoted as minimal unsatisfiable subsets, are a means often used to explain such inconsistent situations. In the following, we formally introduce the notion of a minimal conflict set and also introduce related preference criteria, i.e., criteria regarding the degree of preferredness of individual conflict sets. In this context, we also discuss the major properties of QuickXplain (Junker 2004) which is used for demonstration purposes throughout this paper.

**Determining Preferred Minimal Conflicts**

Conflict sets are constraint sets that are responsible for an inconsistency, i.e., a situation in which no solution can be found. With consistent($C$), we express that a constraint set $C$ is consistent, and inconsistent($C$) reflects situations where no solution can be found for $C$. Definition 3 introduces the concept of conflict set minimality on the basis of subset minimality (i.e., not minimal cardinality): if $CS$ is a minimal conflict, no proper subset of $CS$ can be a minimal conflict.

**Definition 3 (Conflict set).** A conflict set is a set $CS \subseteq C_R :$ inconsistent($CS \cup C_{KB}$). $CS$ is minimal iff $\not\exists CS' : CS' \subset CS$.

An example of minimal conflict sets in the context of our example configuration task is the following (see Example 2).

**Example 2 (Minimal conflict sets).** Given the configuration task $(V, C = C_{KB} \cup C_R)$ presented in Example 1, we are able to identify the following minimal conflict sets: $CS_1 = \{c_{11}, c_{13}\}, CS_2 = \{c_{13}, c_{18}\},$ and $CS_3 = \{c_{14}, c_{16}\}$. The minimality property is fulfilled since $\not\exists CS_4 : CS_1 \subset CS_4 \subset \not\exists CS_5 : CS_5 \subset CS_2,$ and $\not\exists CS_6 : CS_6 \subset CS_3.$

**Preferred Minimal Conflict.** To resolve inconsistencies in interactive settings such as configuration (O’Sullivan et al. 2007; Felfernig et al. 2014), a user has to resolve conflicts consisting of constraints that represent user requirements ($c_i \in C_R$). In this context, a constraint that is of low importance for the user is a preferred candidate for being part of a conflict set.

To make the preference degree of a conflict set more transparent, we introduce the following definitions of a strict total order (Definition 4) and a corresponding preferred conflict (Definition 5). These definitions provide a way to clearly identify a preferred conflict set among a set of candidates (see also (Junker 2004; Marques-Silva and Previti 2014b)).

**Definition 4 (Strict total order).** A strict total order $< \in C_R = \{c_1 \ldots c_m\}$ is represented as $c_1 < c_2 \ldots < c_m$ where $\forall (c_i, c_{i+1}); c_i$ is preferred over $c_{i+1}$.

On the basis of such an ordering of individual constraints part of a conflict set, we are able to characterize the preference degree of a conflict set on the basis of a pairwise comparison (see (Junker 2004)). Given a strict total order $< \in$ of a set of constraints, there exists a unique preferred conflict set (Junker 2004).

**Definition 5 (Preferred conflict).** Given a strict total order $< \in C_R$, a set $X \subseteq C_R$ is preferred over another set $Y \subseteq C_R$ (denoted $X >_{\text{lex}} Y$) iff $\exists k \leq m : c_k \in Y \setminus X$ and $X \cap \{c_1 \ldots c_{k-1}\} = Y \cap \{c_1 \ldots c_{k-1}\}$. A minimal conflict set $CS$ is a (lexicographically) preferred conflict iff $\forall CS' \not\subset CS : CS >_{\text{lex}} CS'$.

Following Definition 5, a preferred conflict in our example configuration task is the following (see Example 3).

**Example 3 (Preferred conflict).** Given the minimal conflict sets $CS_1 = \{c_{11}, c_{13}\}, CS_2 = \{c_{13}, c_{18}\}, CS_3 = \{c_{14}, c_{16}\},$ and a corresponding total ordering of $\{c_{11} < c_{13} < c_{12} < \ldots < c_{18}\},$ the preferred conflict (conflict set) can be determined as follows:

- $CS_3$ is preferred over $CS_1$ (denoted as $CS_3 >_{\text{lex}} CS_1$) since $\exists c_{11} \in CS_1 \setminus CS_3$ with $CS_1 \cap \emptyset = CS_3 \cap \emptyset$.
- $CS_3$ is preferred over $CS_2$ (denoted as $CS_3 >_{\text{lex}} CS_2$) since $\exists c_{13} \in CS_2 \setminus CS_3$ with $CS_2 \cap \{c_{11}, c_{12}\} = CS_3 \cap \{c_{11}, c_{12}\}$.
- The preferred conflict set is $CS_3$.

In Example 3, the preferred conflict set is $CS_3$. Following the transitive properties of such lexicographical orderings (Brewka 1989; Junker 2004), no further comparisons are needed for $CS_1$ and $CS_2$.

**Preference formula.** Following Definition 5, we now introduce a numerical evaluation function (based on bitmap indexing (O’Neil 1987)) which is used in the following to numerically evaluate conflict preference (see Formulae 1–3).

$$\text{preference}(CS) = \text{MAX} - \sum_{c_i \in CS} \text{importance}(c_i) \quad (1)$$

$$\text{importance}(c_i) = 2^{|C_R| - i} \quad (2)$$

$$\text{MAX} = 2^{|C_R|} - 1 \quad (3)$$

The preference of a conflict set $CS$ (preference($CS$)) can be determined by evaluating the sum of the individual user preferences regarding constraints in $CS$ (see Definition 4). In this context, we apply a so-called bitmap indexing (O’Neil 1987) following the idea of evaluating individual constraint rankings following a binary system $(2^{|C_R| - 1}, 2^{|C_R| - 2} \ldots 1)$, where $|C_R|$ denotes the $C_R$’s cardinality. In our working example, preference($CS_1 = \{c_{11}, c_{13}\}$) = 191 assuming importance($c_{11}$) = 256 and importance($c_{13}$) = 64. A complete evaluation of the conflict sets of our working example is depicted in Table 2.

The higher the sum over the importance values of constraints in $CS$, the lower the probability that $CS$ is the preferred conflict set. In this context, MAX (see Formula 3) refers to the least preferred conflict set including all elements of $C_R$ with a related theoretical preference value of 0. Consequently, the lower the total importance of constraints in $CS$, the higher the preference for $CS$. The theoretical upper bound of preference($CS$) is $\text{MAX} - 1$ assuming that the
The core algorithm is implemented in the function \( \text{QX} \) (Algorithm 2) that determines a minimal conflict set in a divide-and-conquer fashion. An execution trace of \( \text{QUICKXPLAIN} \) on the basis of our working example is depicted in Figure 1. The algorithm \( \text{QX} \) adds additional constraints (from \( C_{RB} \)) to \( C_{KB} \) as long as the resulting constraint set remains consistent. If it is inconsistent, then the algorithm leaves out the remaining constraints. For example, in the activation [3], the set \( C_{KB} \cup \{c_{14} \ldots c_{19}\} \) is inconsistent and thus, the remaining constraints \( \{c_{11} \ldots c_{13}\} \) can be removed.

On the other hand, if the background knowledge is consistent and only one constraint remains that induces the inconsistency, this constraint must be part of the conflict set. For example, in the activation [9], the background knowledge consists of the constraints \( C_{KB} \cup \{c_{14}, c_{17} \ldots c_{19}\} \) and \( c_{16} \) remains as a single constraint. It is clear that \( c_{16} \) is part of the conflict set since \( C_{KB} \cup \{c_{14}, c_{17} \ldots c_{19}\} \) is consistent but \( C_{KB} \cup \{c_{14}, c_{17}, c_{19}\} \cup \{c_{16}\} \) is inconsistent.

**INFORMEDQX**

**General idea.** \( \text{QUICKXPLAIN} \) works efficiently in many scenarios, however, it shows performance issues in the context of large and complex knowledge bases (Vidal et al. 2021). To tackle this challenge, we introduce INFORMEDQX (as a \( \text{QUICKXPLAIN} \) variant) which exploits known conflicts (e.g., from previous configuration sessions) to efficiently narrow down the conflict analysis space.

This exploitation (conflict reuse) can take place in two basic settings. First, a direct reuse (without activating \( \text{QX} \)) is possible if a specific predetermined conflict is the preferred conflict of the user requirements \( C_R \). Second, when some predetermined conflicts match with the constraints of the user requirements \( C_R \) but its preferred conflict is not available yet, the currently preferred conflict of the predetermined conflicts should be identified. In this setting, we can reduce the size of \( C_R \) by pruning certain constraints that would not fit well with the more preferred conflict (the conflict with a higher preference value as described in Formula 1). This pruned set, \( C_P \), can then be passed to \( \text{QX} \), making the process of finding conflicts faster.

We assume that \( N \) denotes the set of predetermined minimal conflicts. It is not necessarily complete (i.e., not all conflicts of \( C_R \) are in \( N \)), only those that have already been identified. The determination of the preferred conflict based on \( N \) can be formally described in the following scenarios:

<table>
<thead>
<tr>
<th>( C_R )</th>
<th>( c_{11} )</th>
<th>( c_{12} )</th>
<th>( c_{13} )</th>
<th>( c_{14} )</th>
<th>( c_{15} )</th>
<th>( c_{16} )</th>
<th>( c_{17} )</th>
<th>( c_{18} )</th>
<th>( c_{19} )</th>
<th>( \text{preference}(C_S) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i )</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>511 - (256 + 64) = 191</td>
</tr>
<tr>
<td>( \text{importance}(c_i) )</td>
<td>256</td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>511 - (64 + 2) = 445</td>
</tr>
<tr>
<td>( C_{S1} )</td>
<td>( x )</td>
<td>( - )</td>
<td>( x )</td>
<td>( - )</td>
<td>( - )</td>
<td>( - )</td>
<td>( - )</td>
<td>( - )</td>
<td>( - )</td>
<td>511 - (32 + 8) = 471</td>
</tr>
<tr>
<td>( C_{S2} )</td>
<td>( - )</td>
<td>( - )</td>
<td>( x )</td>
<td>( - )</td>
<td>( - )</td>
<td>( - )</td>
<td>( - )</td>
<td>( - )</td>
<td>( - )</td>
<td>511 - (64 + 2) = 445</td>
</tr>
<tr>
<td>( C_{S3} )</td>
<td>( - )</td>
<td>( - )</td>
<td>( - )</td>
<td>( x )</td>
<td>( - )</td>
<td>( - )</td>
<td>( - )</td>
<td>( - )</td>
<td>( - )</td>
<td>511 - (256 + 64) = 191</td>
</tr>
</tbody>
</table>

Table 2: Preference values for the conflict sets \( C_{S1}, C_{S2} \), and \( C_{S3} \) given \( \{c_{11} < c_{12} < \ldots < c_{19}\} \) as the strict total ordering of the user requirements in \( C_R \). In this setting, \( C_{S3} \) is regarded as preferred minimal conflict set.
1. Scenario 1 - If there are no conflicts in \( N \) or if none of the conflicts in \( N \) match the constraints of \( C_R \), then the original QUICKXPLAIN (QX) algorithm must be used.

2. Scenario 2 - If there are conflicts in \( N \) that belong to \( C_R \), then the currently preferred conflict \( N_{cp} \) out of these conflicts can be determined. Additionally, we can check if there are any conflicts in \( C_R \) that are even more preferred than \( N_{cp} \). If so, there are two possible sub-scenarios:

(a) Scenario 2.1 - There do not exist any more preferred conflicts: We do not need to activate QX, i.e., the currently preferred conflict is the preferred conflict. For instance, given \( C_R = \{ c_{11} \ldots c_{19} \} \), \( N = \{ N_1 = \{ c_{11}, c_{13} \}, N_2 = \{ c_{13}, c_{18} \}, N_3 = \{ c_{14}, c_{16} \} \} \), \( N_{cp} = \{ c_{14}, c_{16} \} \) (see Example 3), and no conflicts in \( C_R \) that are more preferred over \( N_{cp} \), \( N_{cp} \) then becomes the preferred conflict of \( C_R \) and the activation of QX is not needed (see Figure 2).

(b) Scenario 2.2 - There exists at least one more preferred conflict: We activate QX with the pruned set \( C_P \). Based on Definitions 4 and 5, the constraints of \( C_R \) with a lower lexicographical order than the currently preferred conflict can be omitted. For instance, given \( C_R = \{ c_{11} \ldots c_{19} \} \), and \( N = \{ N_1 = \{ c_{11}, c_{13} \}, N_2 = \{ c_{13}, c_{18} \}, N_3 = \{ c_{14}, c_{16} \} \} \), the currently preferred conflict set is \( N_{cp} = \{ c_{13}, c_{18} \} \), the pruned set of \( C_R \) should be \( C_P = \{ c_{13} \ldots c_{19} \} \). The reason is that the constraints \( \{ c_{11}, c_{12} \} \) cannot be part of the more preferred conflict and therefore are omitted. Consequently, QX is activated with \( C_P \), whose size is much smaller than this of the original set \( C_R \) (see Figure 3).

Construction of \( N \). Identifying a \( N_{cp} \) out of \( N \) is an expensive computational task. To tackle this issue, \( N \) is constructed on the basis of the Binary Decision Diagram (BDD) (Cheng and Yap 2005). In this context, the approach requires an offline process where a BDD is built to compress all identified conflict sets.

INFORMEDQX algorithm. The INFORMEDQX algorithm (see Algorithm 3) comes into play when the existing background knowledge \( C_{KB} \) is inconsistent with \( C_R \). The initial step of Algorithm 3 identifies the currently preferred conflict \( N_{cp} \) from the set \( N \) (line 1). The FINDPREFERREDCONFLICT function determines this preferred conflict based on Definition 5 and Formula 1. Initially, this function extracts from \( N \) all the conflict sets associated with \( C_R \) utilizing the findAll operation of the BDD diagram (Cheng and Yap 2005). It then calculates the preferences for each of these identified conflict sets. Eventually, the function returns the conflict set with the highest preference value. Moreover, if \( N \) is empty, the conflict extraction process from \( N \) is ignored, and the function returns an empty set. Additionally, if no conflicts of \( C_R \) are present in \( N \), the function returns an empty set.
Algorithm 3: INFORMEDQX($C_R, C_{KB}, N) : CS$

1: $N_{cp} \leftarrow \text{FindPreferredConflict}(N, C_{R})$
2: if $N_{cp} = \emptyset$ then
3:   return(QX($\emptyset, C_{R}, C_{KB}$))
4: else
5:   $C_P \leftarrow \text{Prune}(C_{R}, N_{cp})$
6:   \{begin - examination for a further preferred conflict\}
7:   $\text{prev}_c \leftarrow \emptyset$
8:   for all $c \in N_{cp}$ do
9:     $\text{idx}_c \leftarrow \text{Index}(c, C_P)$
10:    $C'_p \leftarrow \text{prev}_c \cup (C_P \setminus \{C_P[i] : \forall i \in [0 \ldots \text{idx}_c]\})$
11:    if $\text{Inconsistent}(C_{KB} \cup C'_p)$ then
12:      return(QX($\emptyset, C'_p, C_{KB}$))
13: end if
14: end for
15: \{end - examination for a further preferred conflict\}
16: return(N_{cp})

<table>
<thead>
<tr>
<th>loop</th>
<th>$c \in N_{cp}$</th>
<th>$C'_p$</th>
<th>$\text{Inconsistent}(C_{KB} \cup C'_p)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{c_{14} \ldots c_{19}}</td>
<td>{c_{15} \ldots c_{19}}</td>
<td>false</td>
</tr>
<tr>
<td>2</td>
<td>{c_{14}, c_{17} \ldots c_{19}}</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: An illustration of lines 6 – 14 in Algorithm 3 for checking whether there exists further conflicts in $C_P = \{c_{14} \ldots c_{19}\}$, where $N_{cp} = \{c_{14}, c_{16}\}$.

If $N_{cp} = \emptyset$, i.e., $N$ is empty or no conflicts of $C_R$ stay in $N$ (line 2), then the traditional QUICKXPLAIN is activated for $C_R$ (line 3). Otherwise, in line with Scenario 2.2, certain unnecessary constraints in $C_R$, which will not be part of the more preferred conflict than the currently preferred conflict $N_{cp}$, are omitted (line 5). After pruning $C_R$, the algorithm examines whether there exist further preferred conflicts in the pruned set $C_P$ (lines 6 – 14). This is addressed by evaluating if the inconsistency of $C_P$ with $C_{KB}$ remains unchanged even when removing a constraint $c \in N_{cp}$ from $C_P$ (lines 9 – 10). In our approach, for each $c \in N_{cp}$, not only $c$ itself is removed, but also constraints not present in $N_{cp}$ yet confirmed as consistent with $C_{KB}$ in the previous checks (line 9). For instance, since $C'_p = \{c_{15} \ldots c_{19}\}$ is consistent with $C_{KB}$ (see loop 1 in Table 3), i.e., no conflict between $c_{15}$ and other constraints in $C'_p$, $c_{15}$ is omitted from $C_P$ in the next iteration (see loop 2 in Table 3). Besides, $c_{14}$ remains untouched because it belongs to $N_{cp} = \{c_{14}, c_{16}\}$.

During the examination, if a constraint $c \in N_{cp}$ satisfies the check (line 10), the algorithm triggers QX for the current constructed subset $C'_p$ (line 11). Should the examination fail for all constraints of the conflict $N_{cp}$, the algorithm returns $N_{cp}$ without activating QX (line 15). In other words, $N_{cp}$ represents the preferred conflict of $C_R$. Figure 4 depicts the INFORMEDQX execution trace for our working example, highlighting the reduction in the number of required consistency checks to 6, in contrast to the 9 checks in QUICKXPLAIN (as shown in Figure 1).

<table>
<thead>
<tr>
<th>method</th>
<th>best case</th>
<th>worst case</th>
</tr>
</thead>
<tbody>
<tr>
<td>IQX</td>
<td>$m + 1$</td>
<td>$2c \times \log_2(\frac{m}{c}) + 2c + m$</td>
</tr>
<tr>
<td>QX</td>
<td>$\log_2(\frac{m}{c}) + 2c$</td>
<td>$2c \times \log_2(\frac{m}{c}) + 2c$</td>
</tr>
</tbody>
</table>

Table 4: The complexity of INFORMEDQX (IQX) and QUICKXPLAIN (QX).

Analysis of INFORMEDQX

We will now delve into a theoretical analysis of INFORMEDQX and proceed to evaluate its performance in comparison with QUICKXPLAIN (Junker 2004).

QUICKXPLAIN Complexity. The worst-case complexity of QX in terms of the number of needed consistency checks for determining one minimal unsatisfiable subset $CS$ is $2c \times \log_2(\frac{m}{c}) + 2c$, where $c$ is the size of the minimal conflict set, $m$ is the number of constraints in $C_R$, and $2c$ represents the branching factor and the number of leaf-node consistency checks (Junker 2004). The best-case complexity is $\log_2(\frac{m}{c}) + 2c$. In the worst case, each faulty element is located in a different path of the search tree. The factor $\log_2(\frac{m}{c})$ represents the depth of a path of the QX search tree. Under the best circumstance, every constraint belonging to a conflict is included in a single path of the search tree.

INFORMEDQX Complexity. The complexity of INFORMEDQX can be calculated according to the following factors: (1) the number of needed consistency checks for determining one minimal unsatisfiable subset $CS$ and (2) the complexity of BDD queries ($m$). In the worst-case, the complexity emerges as the cumulative sum of these two factors, which is $2c \times \log_2(\frac{m}{c}) + 2c + m$ where $m$ is the number of the nodes in $N$’s BDD. In this scenario, the only distinction in the complexity of the algorithms arises from the extra factor $m$ that is the complexity of BDD queries in the FindPreferredConflict. The corresponding best-case complexity is $m + 1$. In this scenario, since the preferred conflict of $C_R$ is already known, the complexity of INFORMEDQX is the sum of the complexity of BDD queries in the FindPreferredConflict and one consistency check that confirms the preferred conflict.

INFORMEDQX Runtime Performance. We have evaluated the performance of INFORMEDQX compared to QUICKXPLAIN on the basis of the Linux-2.6.33.3 configuration knowledge base taken from Diverso Lab’s benchmark\(^1\) (Heradio et al. 2022). The characteristics of this knowledge base are the following: \#features = 6,467; \#relationships = 6,322; and \#cross-tree constraints = 7,650. For this knowledge base, we used a genetic approach (Uran and Felfernig 2018) to synthesize and collect 136 minimal conflict sets, whose cardinality varies from 2 to 8.\(^2\)

All experiments have been conducted with an Apple M1 Pro (8 cores) computer with 16-GB RAM. For evaluation purposes, we used the CHOCO solver\(^3\) to perform consis-

---

\(^1\) https://github.com/diverso-lab/benchmarking

\(^2\) To ensure the reproducibility of the results, we used the seed value of 141982L for the random number generator.

\(^3\) choco-solver.org
Figure 4: INFORMEDQX execution trace for $C_R = \{c_{11} \ldots c_{19}\}$, $B = C_{KB}$.

Table 5: Number of solver calls / number of reused conflict sets / average runtime performance (in seconds) of INFORMEDQX versus QUICKXPLAIN needed for determining the preferred conflict set after 20 iterations. $|CS|$ denotes the cardinality of the preferred conflict set. $N_{cp} = \emptyset$, $N_{cp} \neq \emptyset$ and $N_{cp} \neq CS$ indicate three following cases of $N_{cp}$ returned by FINDPREFERREDCONFLICT: (1) there are no conflicts of $C_R$ in $N$, (2) $N_{cp}$ is not the preferred conflict set, and (3) $N_{cp}$ is the preferred conflict. Please note that, column 2 shows zero reused conflict sets for QUICKXPLAIN since this algorithm does not utilize any reuse mechanisms.

| $|CS|$ | QUICKXPLAIN | INFORMEDQX |
|------|--------------|-------------|
|      | $N_{cp} = \emptyset$ | $N_{cp} \neq \emptyset$ | $N_{cp} = CS$ |
| 2    | 8 / 0 / 3.58 | 8 / 0 / 3.64 | 4 / 3 / 2.57 |
| 4    | 12 / 0 / 7.47 | 12 / 0 / 7.59 | 10 / 3 / 6.21 |
| 8    | 22 / 0 / 15.92 | 22 / 0 / 18.91 | 18 / 4 / 12.60 |

Conclusions

In this paper, we have proposed an algorithm so-called INFORMEDQX as an improved version of the QUICKXPLAIN algorithm. The proposed algorithm resolves run-time performance issues in scenarios where the knowledge base is complex and exponentially large. With our algorithm, only conflicts that have been predetermined in previous conflict detection sessions are taken into account. This way, the algorithm helps to decrease the conflict analysis space and, hence, speeds up the conflict detection process. The evaluation results show that INFORMEDQX outperforms QUICKXPLAIN in most of the evaluation cases.

Acknowledgements

This work has been funded by the FFG-funded project PARXCEL (880657).
References


