COREDIAG: Eliminating Redundancy in Constraint Sets

Alexander Felfernig 1, Christoph Zehentner 1, Paul Blazek 2

1 Graz University of Technology, Inffeldgasse 16b, 8010 Graz, Austria
alexander.felfernig@ist.tugraz.at
christoph.zehentner@ist.tugraz.at
2 cyLEDGE Media GmbH, Schottenfeldgasse 60, 1070 Vienna, Austria
p.blazek@cyledge.com

ABSTRACT
Constraint-based environments such as configuration systems, recommender systems, and scheduling systems support users in different decision making scenarios. These environments exploit a knowledge base for determining solutions of interest for the user. The development and maintenance of such knowledge bases is an extremely time-consuming and error-prone task. Users often specify constraints which do not reflect the real-world. For example, redundant constraints are specified which often increase both, the effort for calculating a solution and efforts related to knowledge base development and maintenance. In this paper we present a new algorithm (COREDIAG) which can be exploited for the determination of minimal cores (minimal non-redundant constraint sets). The algorithm is especially useful for distributed knowledge engineering scenarios where the degree of redundancy can become high. In order to show the applicability of our approach, we present an empirical study conducted with commercial configuration knowledge bases.

Keywords: redundant constraints, minimal cores.

1 INTRODUCTION
The central element of a constraint-based application is a knowledge base (constraint set). When developing and maintaining constraint sets, users are often defining faulty constraints (the system calculates solutions which are not allowed or – in the worst case – no solution can be found) (Bakker et al., 1993, Felfernig et al., 2004) or redundant constraints which are not needed to express the domain knowledge in a complete fashion (Sabin and Freuder, 1999, Piette, 2008, Fahad and Qadir, 2008, Levy and Sagiv, 1992). In this paper we focus on situations where users are defining redundant constraints which – when deleted from the constraint set (knowledge base) – do not change the semantics of the remaining constraint set. More formally, if $C = \{c_1, c_2, ..., c_n\}$ is the initial set of constraints defined for the knowledge base and one constraint $c_i$ is redundant, then $(C - \{c_i\}) \cup \bar{C}$ is inconsistent ($\bar{C}$ is the negation of $C$).

Redundancy elimination in knowledge bases is a topic extensively investigated by AI research. The identification of redundant constraints plays a major role, for example, in the development and maintenance of configuration knowledge bases (see, e.g., (Sabin and Freuder, 1999)). The authors introduce concepts for the detection of redundant constraints in conditional constraint satisfaction problems (CC-SPs). The approach is based on the idea of analyzing the solution space of the given problem (on the level of individual solutions) in order to detect different types of redundant constraints. (Piette, 2008) provide an in-depth discussion of the role of redundancy elimination in SAT solving. They introduce an (incomplete) algorithm for the elimination of redundant clauses and show its applicability on the basis of an empirical study. The role of redundancies in ontology development is analyzed by (Fahad and Qadir, 2008). The authors point out the importance of redundancy elimination and discuss typical modeling errors that occur during ontology development and maintenance. (Grimm and Wissmann, 2011) introduce algorithms for redundancy elimination in OWL ontologies. The authors propose an algorithm that computes redundant axioms by exploiting prior knowledge of the dispensability of axioms. (Levy and Sagiv, 1992) analyze two types of redundancies in Datalog programs. First, they interpret redundancy in terms of reachability, i.e., rules and predicates are eliminated that are not part of any derivation tree. Second, redundancy is defined on the basis of the concepts of minimal derivation trees which do not include any pair of identical atoms where one is the predecessor of the other one.

All the mentioned approaches focus on the identification of redundant constraints in centralized scenarios where a knowledge engineer is interested in identifying redundant constraints in the given knowledge base. In such scenarios it is assumed that only a small subset of the given constraints is redundant (this assumption is also denoted as low redundancy assumption.
(Grimm and Wissmann, 2011). Existing algorithms are focusing on such centralized scenarios. In this paper we go one step further and propose an algorithm which is especially useful in distributed knowledge engineering scenarios where we can expect a larger number of redundant constraints due to the fact that different contributors add constraints which are related to the same topic (see, e.g., (Chklovski and Gil, 2005) [Richardson and Domingos, 2003] – we denote the assumption of larger sets of redundant constraints the high redundancy assumption. For example, we envision a scenario where a large number of users propose constraints to be applied by a constraint-based configuration or recommendation engine (Felfernig and Burke, 2008) and the task of an underlying diagnosis algorithm is to identify minimal sets of constraints which retain the semantics of the original constraint set – we denote such constraint sets as minimal cores. Note that the following discussions are based on the assumption of consistent constraint sets. Methods for consistency restoration are discussed in (Bakker et al., 1993), (Felfernig et al., 2004), (Friedrich and Shchekotykhin, 2005), (Felfernig et al., 2011).

The major contributions of our paper are the following. First, we introduce a new algorithm which allows for a more efficient determination of redundant constraints especially in the context of distributed (community-based) knowledge engineering scenarios. Second, we present the results of a performance analysis of our algorithm conducted with real-world configuration knowledge bases.

The remainder of this paper is organized as follows. In Section 2 we introduce a simple example configuration knowledge base from the automotive domain. In Section 3 we introduce a basic algorithm for the determination of redundant constraints in centralized settings (SEQUENTIAL). In Section 4 we introduce the COREDIAg algorithm. Thereafter we report the results of a performance evaluation conducted with real-world configuration knowledge bases (Section 5). The paper is concluded with Section 6.

2 WORKING EXAMPLE

For illustration purposes we use a car configuration knowledge base throughout this paper. A configuration task can be defined as a basic constraint satisfaction problem (CSP) (Tsang, 1993) (see the following definition).

Definition (Configuration Task) : A configuration task can be defined as a CSP (V, D, C). \( V = \{ v_1, v_2, \ldots, v_n \} \) is a set of finite domain variables. \( D = \{ dom(v_1), dom(v_2), \ldots, dom(v_n) \} \) is a set of corresponding domain definitions where \( dom(v_k) \) is the domain of the variable \( v_k \). \( C = C_{KB} \cup C_R \) where \( C_{KB} = \{ c_1, c_2, \ldots, c_{n_k} \} \) is a set of domain-specific constraints (the configuration knowledge base) and \( C_R = \{ c_{n_k+1}, c_{n_k+2}, \ldots, c_{n} \} \) is a set of customer requirements (as well represented as constraints).\(^1\)

\(^1\)Note that the presented concepts are as well applicable to other types of knowledge representations such as SAT or description logics.

The following configuration task will be used as a working example throughout the paper. The variable \( type \) represents the type of the car, \( pdc \) is the park distance control feature, \( fuel \) represents the average fuel consumption per 100 kilometers, a \( skibag \) allows convenient ski storage inside the car, and \( 4\)-\( wheel \) represents the actuation type (4-wheel supported or not supported). These variables represent the possible combinations of customer requirements. The set \( C_{KB} = \{ c_1, c_2, c_3, c_4, c_5 \} \) defines additional restrictions on the set of possible customer requirements \( C_R = \{ c_6, c_7, c_8, c_9, c_{10} \} \).

- \( V = \{ type, fuel, skibag, 4\text{-}wheel, pdc \} \)
- \( D = \{ \)
  - \( \)dom\( (type) = \{ \text{city, limo, combi, xdrive} \} \)
  - \( \)dom\( (fuel) = \{ 4l, 6l, 10l \} \)
  - \( \)dom\( (skibag) = \{ \text{yes, no} \} \)
  - \( \)dom\( (4\text{-}wheel) = \{ \text{yes, no} \} \)
  - \( \)dom\( (pdc) = \{ \text{yes, no} \} \)
- \( C_{KB} = \{ \)
  - \( c_1: 4\text{-}wheel = \text{yes} \rightarrow type = \text{xdrive} \)
  - \( c_2: skibag = \text{yes} \rightarrow type \neq \text{city} \)
  - \( c_3: fuel = 4l \rightarrow type = \text{city} \)
  - \( c_4: fuel = 6l \rightarrow type \neq \text{xdrive} \)
  - \( c_5: type = \text{city} \rightarrow fuel \neq 10l \)
- \( C_R = \{ \)
  - \( c_6: 4\text{-}wheel = \text{no} \)
  - \( c_7: fuel = 4l \)
  - \( c_8: type = \text{city} \)
  - \( c_9: skibag = \text{no} \)
  - \( c_{10}: pdc = \text{yes} \)

On the basis of this example configuration task we now give a definition of a corresponding configuration (solution).

Definition (Configuration) : A configuration (solution) for a configuration task is an instantiation \( I = \{ v_1 = ins_{v_1}, v_2 = ins_{v_2}, \ldots, v_n = ins_{v_n} \} \) where \( ins_{v_k} \) is an element of the domain of \( v_k \). A configuration is consistent if the assignments in \( I \) are consistent with the constraints in \( C \). A complete solution is one in which all the variables are instantiated. Finally, a configuration is valid if it is both, consistent and complete.

A configuration for our example configuration task would be \( I = \{ 4\text{-}wheel = \text{no}, fuel = 4l, type = \text{city}, skibag = \text{no}, pdc = \text{yes} \} \).

3 DETERMINING REDUNDANT CONSTRAINTS

Let us now consider a simple adaptation of the original set of constraints \( C_{KB} \) which we denote with \( C_{KB}' \). \( C_{KB}' \) includes an additional constraint \( c_a \) which has been added by a knowledge engineer.

\( C_{KB}' = \{ \)
  - \( c_a: skibag \neq \text{no} \rightarrow type = \text{limo} \lor type = \text{xdrive} \)
  - \( c_1: 4\text{-}wheel = \text{yes} \rightarrow type = \text{xdrive} \)
  - \( c_2: skibag = \text{yes} \rightarrow type \neq \text{city} \)
  - \( c_3: fuel = 4l \rightarrow type = \text{city} \)
  - \( c_4: fuel = 6l \rightarrow type \neq \text{xdrive} \)
  - \( c_5: type = \text{city} \rightarrow fuel \neq 10l \)

It is obvious that \( c_a \) is redundant since it does not further restrict the solution space defined by the constraints \( C_{KB} = \{c_1, c_2, c_3, c_4, c_5\} \). In order to discuss constraint redundancy on a more formal level, we introduce the following definitions.

**Definition (Redundant Constraint)**: Let \( c_a \) be a constraint element of the configuration knowledge base \( C_{KB} \). \( c_a \) is called redundant if \( C_{KB} - \{c_a\} \models c_a \). If this condition is not fulfilled, \( c_a \) is said to be non-redundant. Redundancy can also be analyzed by checking \( C_{KB} - \{c_a\} \cup \overline{C_{KB}} \) for consistency - if consistency is given, \( c_a \) is non-redundant.

Iterating over each constraint of \( C_{KB} \), executing the non-redundancy check \( C_{KB} - \{c_a\} \cup \overline{C_{KB}} \), and deleting redundant constraints from \( C_{KB} \) results in a set of non-redundant constraints (the minimal core). If the non-redundancy check fails (no solution can be found), the constraint \( c_a \) is redundant and can be deleted from \( C_{KB} \). Otherwise (the non-redundancy check is successful), \( c_a \) is non-redundant.

**Definition (Minimal Core)**: Let \( C_{KB} \) be a configuration knowledge base. \( C_{KB} \) is denoted as minimal core if \( \forall c_i \in C_{KB} : C_{KB} - \{c_i\} \cup \overline{C_{KB}} \) is consistent. Obviously, \( C_{KB} \cup \overline{C_{KB}} \models \bot \).

The principle of the following algorithm (SEQUENTIAL - Algorithm 1) is often used for determining such redundancies (see, e.g., [Piette, 2008] Grimm and Wissmann, 2011).

**Algorithm 1 SEQUENTIAL(C\(_{KB}\))**

\[
\{C_{KB} : \text{configuration knowledge base}\}
\{C_{KB} : \text{the complement of } C_{KB}\}
\{\Delta : \text{set of redundant constraints}\}
\]

\[
C_{KBt} \leftarrow C_{KB};
\]

for all \( c_i \) in \( C_{KBt} \) do

\[
\text{if inconsistent}( (\overline{C_{KBt}} - c_i) \cup C_{KB} ) \text{ then}
C_{KBt} \leftarrow C_{KBt} - \{c_i\};
\]

end if

end for

\[
\Delta \leftarrow C_{KB} - C_{KBt};
\]

return \( \Delta \);

The approach of SEQUENTIAL is straightforward: each individual constraint \( c_i \) is evaluated w.r.t. redundancy by checking whether \( C_{KBt} - c_i \) is still inconsistent with \( C_{KB} \). If this is the case, \( c_i \) can be considered as redundant. If \( C_{KBt} - c_i \) is consistent with \( C_{KB} \), \( c_i \) is a non-redundant constraint since its deletion induces consistency with \( C_{KB} \). Applying the algorithm SEQUENTIAL to our example \( C_{KB} \) results in \( \Delta = \{c_a\} \) since \( C_{KB} - \{c_a\} \cup \overline{C_{KB}} \) is inconsistent and no further constraint \( c_i \) can be deleted from \( C_{KB} \) such that \( C_{KB} - \{c_a\} \cup \overline{C_{KB}} \) is still inconsistent.

The problem of checking whether a given constraint can be inferred from the remaining part of a constraint set has shown to be Co-NP-complete in the general case [Piette, 2008]. The major goal of our work was to figure out whether there exist alternative algorithms that have a better runtime performance compared to SEQUENTIAL in situations with a large amount of redundant constraints in \( C_{KB} \). Large amounts of redundant constraints typically occur in distributed knowledge engineering scenarios where a large number of users specify rules that in the following have to be aggregated into one consistent constraint set (see, e.g., [Chlkovský and Gil, 2009]).

In the following section we introduce the COREDIAG algorithm which is a valuable alternative to SEQUENTIAL in situations with a large number of redundant constraints. After having introduced COREDIAG we will analyze the performance of both algorithms (SEQUENTIAL and COREDIAG) on the basis of real-world configuration knowledge bases (Section 5).

## 4 COREDIAG

The COREDIAG algorithm (together with CORED) is based on the principle of divide-and-conquer: whenever a set \( S \) which is a subset of \( C_{KB} \) is inconsistent with \( C_{KB} \), it is or contains a minimal core, i.e., a set of constraints which preserve the semantics of \( C_{KB} \).

In our implementation CORED is responsible for determining such minimal cores, COREDIAG returns the complement of a minimal core which is a maximal set of redundant constraints in \( C_{KB} \). CORED is based on the principle of QuickXPlain [Junker, 2004] - as a consequence a minimal core (minimal set of constraints that preserve the semantics of \( C_{KB} \)) can be interpreted as a minimal conflict, i.e., a minimal set of constraints that are inconsistent with \( C_{KB} \).

CORED allows the determination of preferred minimal cores since the algorithm is based on the assumption of a strict lexicographical ordering of the constraints in \( C_{KB} \). On an informal level a preferred minimal core can be characterized as follows: if we have different options for choosing a minimal core, we would select the one with the most agreed-upon constraints. For more details on the role of strict lexicographical orderings of constraints we refer the reader to the work of [Junker, 2004] and [Fellermü et al., 2011].

The COREDIAG algorithm generates \( C_{KB} \) from \( C_{KB} \). It then activates CORED (see Algorithm 3) which determines a minimal core on the basis of a divide-and-conquer strategy that divides the constraints in \( C \) into two subsets (\( C_1 \) and \( C_2 \)) with the goal to figure out whether one of those subsets already contains a minimal core. If \( C_2 \) contains a minimal core, \( C_1 \) is not further taken into account. If \( C \) contains only one element (\( c_a \)) and \( B \) is still consistent, then \( c_a \) is part of the minimal core.

**Algorithm 2 COREDIAG (C\(_{KB}\))**

\[
\{C_{KB} = \{c_1, c_2, ..., c_n\}\}
\{C_{KB} : \text{the complement of } C_{KB}\}
\{\Delta : \text{set of redundant constraints}\}
\]

\[
C_{KBt} \leftarrow \{\neg c_1 \lor \neg c_2 \lor ... \lor \neg c_n\};
\]

return \( C_{KB} - \text{CORED}(C_{KBt}, C_{KB} \cup C_{KBt}) \);
5 EVALUATION

We now compare the performance of COREDIAG with the SEQUENTIAL algorithm discussed in Section 3. The worst case complexity (and best case complexity) of SEQUENTIAL in terms of the number of needed consistency checks is \( n \) (the number of constraints in \( C_{KB} \)). Worst case and best case complexity are identical since SEQUENTIAL checks the redundancy of each individual constraint \( c_i \) with respect to \( C_{KB} \).

In contrast, the worst case complexity of COREDIAG depends on the number of redundant constraints in \( C_{KB} \). The worst case complexity of COREDIAG in terms of the number of needed consistency checks is \( 2c \log_2(2c) + 2c \) where \( n \) is the number of constraints in \( C_{KB} \) and \( c \) is the minimal core size. The best case complexity in terms of the number of needed consistency checks can be achieved if all constraints element of the minimal core are positioned in one branch of the CORED search tree: \( \log_2(2c) + 2c \). Consequently, the performance of COREDIAG heavily relies on the number of constraints contained in the minimal core (the lower the number of constraints in the minimal core, the better the performance of COREDIAG).

Table 1 reflects the results of our analysis conducted with the knowledge bases of www.itu.dk/research/cla/externals/clib. "Bikex": bicycles; "esvs": corporate networks; "fs": financial services (insurances); "hypo": financial services (investments); "large2": electronic circuits. Evaluation data: (#IP-calls / runtime (ms) / #redundant constraints).

![Table 1: Application of SEQUENTIAL (S) and COREDIAG (C) (Algorithm 2) to configuration knowledge bases of www.itu.dk/research/cla/externals/clib. "Bikex": bicycles; "esvs": corporate networks; "fs": financial services (insurances); "hypo": financial services (investments); "large2": electronic circuits. Evaluation data: (#IP-calls / runtime (ms) / #redundant constraints).](image)

2 www.itu.dk/research/cla/externals/clib.
\begin{align*}
R(C_{KB}) &= \frac{\text{redundant constraints in } C_{KB}}{\text{constraints in } C_{KB}} \quad (1)
\end{align*}

In addition to the original version (redundancy rate = \text{0-10\%}) we generated three knowledge bases with the redundancy rates 50\%, 75\%, and 87.5\%. For example, a knowledge base with redundancy rate 50\% can be generated by simply duplicating each constraint of the original knowledge base. Starting with a redundancy rate of 50\% we can observe a transition in the runtime performance (COREDIAG starts to perform better than SEQUENTIAL) due to the increased number of redundant constraints (see the large2 configuration knowledge base in Figure 1). Another outcome of our analysis is that nearly each of the investigated configuration knowledge bases contains redundant constraints (see Table 1). The average runtime for determining configurations without the redundant constraints is lower compared to the runtime with the redundant constraints included (see Table 2) – for this evaluation as well the number of iterations per setting was set to 10; for each iteration we applied a randomized constraint ordering.

\section{6 CONCLUSIONS}

The detection of redundant constraints plays a major role in the context of (configuration) knowledge base development and maintenance. In this paper we have proposed two algorithms which can be applied for the identification of minimal cores, i.e., minimal sets of constraints that preserve the semantics of the original knowledge base. The SEQUENTIAL algorithm can be applied in settings where the number of redundant constraints in the knowledge base is low. The second algorithm (COREDIAG) is more efficient but restricted in its application to knowledge bases that contain a large number of redundant constraints.

\section{7 ACKNOWLEDGEMENTS}

The work presented in this paper has been conducted within the scope of the research project ICONE (Intelligent Assistance for Configuration Knowledge Base Development and Maintenance) funded by the Austrian Research Promotion Agency (827587).

\begin{table}
\centering
\begin{tabular}{|c|c|c|c|c|c|}
\hline
KB & Redundancy Rate & 0-10\% & 50\% & 75\% & 87.5\% \\
\hline
BikeA & 9.9 & 10.2 & 14.2 & 33.7 & 43.0 \\
BikeB & 9.3 & 9.5 & 29.0 & 24.6 & 41.9 \\
BikeC & 8.4 & 11.0 & 18.3 & 32.3 & 44.0 \\
BikeD & 7.8 & 8.9 & 25.6 & 28.0 & 42.6 \\
BikeE & 6.3 & 13.0 & 19.0 & 24.2 & 41.5 \\
BikeF & 7.7 & 12.8 & 19.3 & 22.3 & 41.4 \\
BikeG & 7.3 & 10.3 & 16.9 & 26.0 & 44.8 \\
BikeH & 10.5 & 11.0 & 13.6 & 24.1 & 37.2 \\
BikeI & 8.0 & 9.0 & 15.1 & 31.3 & 45.4 \\
BikeJ & 9.1 & 19.5 & 24.8 & 26.6 & 49.2 \\
BikeK & 7.7 & 11.4 & 17.3 & 23.8 & 45.5 \\
BikeL & 9.3 & 13.4 & 25.8 & 26.7 & 45.5 \\
BikeM & 44.8 & 48.5 & 84.1 & 162.0 & 323.4 \\
esvs & 22.7 & 26.0 & 45.1 & 79.5 & 157.8 \\
is & 22.6 & 24.2 & 44.0 & 76.5 & 149.7 \\
hypo & 8.3 & 8.4 & 19.1 & 26.0 & 47.6 \\
large2 & 15.5 & 16.6 & 22.0 & 25.5 & 36.4 \\
\hline
\end{tabular}
\caption{Time in ms needed for calculating a solution for a given configuration knowledge base version.}
\end{table}
(Fahad and Qadir, 2008) M. Fahad and M. Qadir. A framework for ontology evaluation. In *16th International Conference on Conceptual Structures (ICCS 2008)*, pages 149–158, Toulouse, France, 2008.

(Felfernig and Burke, 2008) A. Felfernig and R. Burke. Constraint-based recommender systems: Technologies and research issues. In *ACM International Conference on Electronic Commerce (ICEC’08)*, pages 17–26, Innsbruck, Austria, 2008.

(Felfernig *et al.*, 2004) A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner. Consistency-based diagnosis of configuration knowledge bases. *Artificial Intelligence*, 152(2):213–234, 2004.

(Felfernig *et al.*, 2011) A. Felfernig, M. Schubert, and C. Zehentner. An efficient diagnosis algorithm for inconsistent constraint sets. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing (AIEDAM)*, 25(2):175–184, 2011.

(Friedrich and Shchekotykhin, 2005) G. Friedrich and K. Shchekotykhin. A general diagnosis method for ontologies. In *4th Intl. Semantic Web Conference (ISWC05)*, number 3729 in Lecture Notes in Computer Science, pages 232–246, Galway, Ireland, 2005. Springer.

(Grimm and Wissmann, 2011) S. Grimm and J. Wissmann. Elimination of redundancy in ontologies. In *Extended Semantik Web Conference (ESWC2011)*, pages 260–274, Heraklion, Greece, 2011.

(Junker, 2004) U. Junker. Quickxplain: Preferred explanations and relaxations for over-constrained problems. In *19th National Conference on Artificial Intelligence (AAAI04)*, pages 167–172, San Jose, CA, 2004.

(Levy and Sagiv, 1992) A. Levy and Y. Sagiv. Constraints and redundancy in datalog. In *11th Conference on the Principles of Database Systems*, pages 67–80, San Diego, CA, 1992.

(Piette, 2008) C. Piette. Let the solver deal with redundancy. In *20th IEEE International Conference on Tools with Artificial Intelligence*, pages 67–73, Dayton, OH, 2008.

(Richardson and Domingos, 2003) M. Richardson and P. Domingos. Building large knowledge bases by mass collaboration. In *2nd International Conference on Knowledge Capture (K-CAP 2003)*, pages 129–137, Sanibel Island, FL, 2003.

(Sabin and Freuder, 1999) M. Sabin and E. Freuder. Detecting and resolving inconsistency and redundancy in conditional constraint satisfaction problems. In *AAI 1999 Workshop on Configuration*, pages 90–94, Orlando, FL, 1999.

(Tsang, 1993) E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.