# Solving Constraint Satisfaction Problems with Database Queries: An Overview

Alexander Felfernig<sup>1</sup>, Viet-Man Le<sup>1</sup>, Albert Haag<sup>2</sup> and Sebastian Lubos<sup>1</sup>

<sup>1</sup>Institute of Software Technology, Graz University of Technology, Graz, Austria <sup>2</sup>Product Management Haag GmbH, Bad Dürkheim, Germany

#### Abstract

Knowledge-based configuration tasks are often solved on the basis of constraint programming. Using constraint programming requires technical expertise regarding problem specification and - to some extent - also solution search, for example, in terms of being confronted with the definition of search heuristics. In this paper, we show how to apply database queries to solve knowledge-based configuration tasks. Using this approach, configuration tasks can be defined and solved without the need of integrating a potentially new technology, but rather stick with technical infrastructures (i.e., relational databases) already existing in the company.

#### Keywords

Constraint Solving, Knowledge-based Configuration, Database Queries

### 1. Introduction

Constraint programming (CP) [1] is based on the idea of defining a set of problem variables, variable domains, and related restrictions (constraints) and solving the problem using on a constraint solver. As such, this technology is often used for solving configuration tasks [2, 3]. There are further approaches used for configuration knowledge representation. For example, SAT solving [4] is based on the idea of representing a configuration task by set of Boolean variables where each variable represents a variable domain value in the general constraint satisfaction problem, for example, car color *red* is a domain value of the variable color. In the SAT context, red would be regarded as variable with the domain {true, false}. In addition, answer set programming (ASP) is based on a more object-oriented view on configuration knowledge representation [5] where on the reasoning level, ASP programs are solved using SAT solvers.

All these types of knowledge representation require additional expertise in at least one of the areas of constraint programming or SAT solving. Furthermore, additional investments are needed to increase CP-related knowledge of employees which is a major precondition for making underlying technologies applicable for configuration knowledge representation and reasoning. On the other hand, relational database technologies and related

D 0000-0003-0108-3146 (A. Felfernig); 0000-0001-5778-975X (V. Le); 0000-0002-1388-4904 (A. Haag); 0000-0002-5024-3786 (S. Lubos) 0 203 Copyright for this paper by its authors. Use permitted under Creative Commons I 0 203 Copyright for this paper by its authors. Use permitted under Creative Commons I CEUR Workshop Proceedings (CEUR-WS.org)



query languages are wide-spread in industrial software development projects. Our idea is to exploit the same technologies in a different form for the representation and solving of constraint-based configuration tasks. In this paper, we provide an overview of different types of knowledge representations and corresponding database queries that can be used to support configuration tasks. We focus on specific database queries which can be regarded as a conjunction of the individual constraints of a corresponding constraint satisfaction problem.

The contributions of this paper are the following: (1) we introduce the idea of a configuration task and a corresponding configuration defined and determined on the basis of the concepts of database queries. (2) we discuss the results of performance evaluations with existing configuration benchmark knowledge bases.

The remainder of this paper is organized as follows. In Section 2, we introduce an example of a car configuration task defined in terms of a constraint satisfaction problem (CSP). In Section 3, we introduce a database query based definition of a configuration task and discuss possibilities of table-based configuration knowledge representations. Thereafter, in Section 4 we provide a performance comparison between database query based and constraint solving based configuration. Threats to validity are discussed in Section 5. Finally, the paper is concluded with a summary of open research issues in Section 6.

# 2. Example Configuration Task

Following the concepts of constraint programming [1], a configuration task can be defined in terms of (1) finite domain variables  $v_i \in V = \{v_1..v_n\}$  (including the corresponding domain definitions  $dom(v_i)$ ) describing product properties and user preferences and (2) constraints

ConfWS'23: 25th International Workshop on Configuration, Sep 6-7, 2023, Málaga, Spain

vietman.le@ist.tugraz.at (V. Le):

albert@product-management-haag.de (A. Haag);

slubos@ist.tugraz.at (S. Lubos)

#### Table 1

Implicit configuration space description where each (CSP) variable  $v_i \in V$  is represented by a single table with one attribute *val* and the table entries derived from  $dom(v_i)$ , for example, variable *type* is represented by the corresponding table *type*. The corresponding database query has to take into account all constraints in *C*.

table	type	fuel	skibag	4wheel	pdc
attribute	val	val	val	val	val
domain	city,limo,combi,xdrive	41,61,101	yes,no	yes,no	yes,no

 $C = CF \cup CR$  representing product domain-specific constraints *CF* and customer requirements *CR* [2].

A simplified example of a car configuration task is the following where *type* represents the car type, *fuel* represents the average fuel consumption, *skibag* indicates the availability of a skibag, and *pdc* represents a parc distance control feature. In this example, the product domain specific constraints are  $CF = \{c_1..c_5\}$  and the customer requirements are  $CR = \{c_6..c_9\}$  which can be specified in a complete (all variables in *V* have a value) or incomplete fashion.

- V = {type, fuel, skibag, 4wheel, pdc}
- dom(type) = {city, limo, combi, xdrive}. dom(fuel)
   {4l, 6l, 10l}. dom(skibag) = {yes, no}. dom(4wheel) = {yes, no}. dom(pdc) = {yes, no}.
- $CF = \{c_1 : 4wheel = yes \rightarrow type = xdrive, c_2 : skibag = yes \rightarrow type \neq city, c_3 : fuel = 4l \rightarrow type = city, c_4 : fuel = 6l \rightarrow type \neq xdrive, c_5 : type = city \rightarrow fuel \neq 10l\}$
- $CR = \{c_6 : 4wheel = yes, c_7 : fuel = 6l, c_8 : type = city, c_9 : skibag = yes\}$

Based on this example CP-based configuration task representation, we will now discuss in more detail different options to represent and solve a configuration task on the basis of a corresponding database query definition.

# 3. Database Query Based Configuration

Representing a configuration task on the basis of a database query allows for the application of relational database technologies for determining corresponding configurations (solutions). We now introduce a definition of a configuration task on the basis of a database query setting  $P_{[C]}S$  (see Definition 1). In this context, P can be (1) a set of tables where each table represents one  $v_i \in V$  ("one table per variable" representation), (2) one table including "all possible configurations" (solutions), and (3) a set of tables representing tuples consistent with individual constraints in  $c_i \in C$  ("local consistency"). Furthermore, [C] is the set of constraints representing the selection criteria of the database query. Finally, S includes those variables  $v_i \in V$  representing the projection criteria

of the query, i.e., those variable values that should be shown as result of the configuration task.

**Definiton 1 (Configuration Task).** A configuration task can be defined as database query  $P_{[C]}S$ . In this context, *P* represents all possible configurations in tabular form (explicitly or implicitly) and [*C*] represents the selection criteria of the query in conjunctive form, i.e.,  $\bigwedge_{(c_i \in C)} c_i$ . Furthermore, *S* represents the projection attributes (variables). In this context,  $C = CR \cup CF$  with *CR* representing the given customer requirements and *CF* representing product domain-specific constraints.

Given such a definition of a configuration task, we are now able to introduce the definition of a configuration (see Definition 2).

**Definiton 2 (Configuration).** A *configuration* for configuration task is one tuple of a result of executing a database query  $P_{[C]}S$  using the selection criteria in *C* and the projection attributes (variables) in *S*.

Based on Definitions 1–2, we now discuss different ways of representing configuration knowledge in a tabular fashion. The chosen type of knowledge representation has an impact on the way the corresponding database query has to be formulated – for demonstration purposes, we will include related examples.

(1) "One Table per Variable". Using this representation, configuration knowledge is expressed in terms of tables (representing individual CSP variables) – see Table 1. For example, the CSP variable *type* is represented by table *type* with the attribute *val* having the CSP variable domain expressed by individual tuples {(*city*), (*limo*), (*combi*), (*xdrive*)}. Following this type of representation, the database (SQL) query  $P_{[C=CF\cup CR]}S$  for our car configuration task is the following (see Query 1).

1. SELECT \* FROM type,fuel,skibag,4wheel,pdc
WHERE (not 4-wheel.val=yes or type.val=xdrive)
and .. and (skibag.val=yes).

In this example, we assume that S includes all variables in *V* and *P* is regarded as implicit representation of the Cartesian product  $type \times fuel \times skibag \times 4wheel \times pdc.^1$  Table 2 shows one configuration returned by Query 1. The

<sup>&</sup>lt;sup>1</sup>We want to assure that at most one tuple is returned by a query – to support this, we assume a query setting such as *LIMIT=1* (this is database-specific).

configuration includes the *pdc* feature, i.e., *pdc.val* = *yes* (this attribute has not been specified by the user).

#### Table 2

Configuration determined by Query 1.

table	type	fuel	skibag	4wheel	pdc
val	xdrive	61	yes	yes	yes

Note that if we are interested only in specific attribute values, the query projection has to specify those attributes, for example, since *4wheel*, *fuel*, *type*, and *skibag* have already been specified as customer requirements, only *pdc* needs to be included (see Query 2).

2. SELECT DISTINCT pdc.val FROM type,fuel,skibag,4wheel,pdc WHERE (not 4-wheel.val=yes or type.val=xdrive) and .. and (skibag.val=yes).

*Example Query Optimization.* Possibilities of improving the performance of such queries are (1) to reduce variable domains in terms of assuring node consistency (e.g., each value of the domain of the variable *type* must be consistent with each unary constraint referring to this variable). (2) queries can be "enriched" by including so-called no-goods (conflict sets) [6] in negated form – the determination of possible conflicts must also be performed in a pre-calculation step. (3) It is also possible to further restrict variable (attribute) domains by establishing arc consistency within a pre-calculation step.

(2) "All Possible Configurations". Specifically for small configuration problems with a limited configuration space size there is also the possibility of just enumerating all possible configurations and storing those configurations in a corresponding table (see, e.g., Table 3). Such an enumeration can be performed on the basis of a database query  $P_{[CF]}S$  where *P* is a table that includes all possible configurations and *CF* represents the set of domain-specific constraints.

#### Table 3

Explicit configuration space description in one table *P* including the CSP variables  $v_i \in V$  as table attributes. The corresponding database query has to take into account the constraints in *CR* (*CF* is already taken into account in *P*).

type	fuel	skibag	4wheel	pdc
xdrive	61	yes	yes	yes
xdrive	61	yes	yes	no
city	41	no	no	no

Following this knowledge representation, the database query in the context of our car configuration task is the following (see Query 3). In this example, we again assume that *S* includes all CSP variables. Furthermore, *P* 

represents a table that includes all (pre-generated) possible configurations.

3. SELECT \* FROM P

WHERE (4-wheel=yes) and .. and (skibag=yes).

Table 4 shows one configuration returned by Query 3.

#### Table 4

type	fuel	skibag	4wheel	pdc
xdrive	61	yes	yes	yes

Example Query Optimization. A basic approach to increase query efficiency is to reduce the number of table entries in *P*. For example, instead of having one centralized table, we could introduce one table per car *type* which is reasonable if the user is sure about the car type selection and just wants to configure the remaining parameters. If we want to generate a table just for the car type *city*, this could be performed on the basis of the query  $P_{[CF\cup{type=city}]}S$ .

(3) "Local Consistency". An alternative to the previously discussed knowledge representations is to use tables that represent local consistency properties. For example, the constraint  $c_1 : 4wheel = yes \rightarrow type = xdrive$  can be represented by a corresponding *consistency table* (*variant table* [7]) *c1* expressing all possible combinations of variable values of 4wheel and type as specified by the corresponding constraint  $c_1$  (see Table 5).

#### Table 5

Implicit configuration space description representing locally consistent variable value combinations (Table c1) – in this case, combinations specified by  $c_1$ :  $4wheel = yes \rightarrow type = xdrive$ .

4wheel	type
yes	xdrive
no	xdrive
no	city
no	limo
no	combi

In a similar fashion, we can define a consistency table c2 expressing the possible variable value combinations as defined by constraint  $c_2$  (see Table 6). This procedure needs to be performed for each constraint  $c_i \in CF$ .

This way, we are able to specify tables fulfilling the property of arc consistency since only variable values are included which are part of at least one tuple included in the corresponding consistency table. Following this knowledge representation, the database query in our car configuration task is the following (see Query 4).

4. SELECT \* FROM c1, c2, ...
WHERE c1.type = c2.type AND ...

#### Table 6

Implicit configuration space description representing locally consistent variable value combinations (Table c2) – in this case, combinations specified by  $c_2$ :  $skibag = yes \rightarrow type \neq city$ .

skibag	type	
yes	xdrive	
yes	limo	
yes	combi	
no	xdrive	
no	limo	
no	combi	
no	city	

In this setting, we again assume that *S* includes all variables. Furthermore, *P* can be regarded as the table related to the *equi-join* of all generated consistency tables, i.e.,  $c1 \bowtie c2 \bowtie c3 \bowtie c4 \bowtie c5$  in our case where table *ci* represents the corresponding constraint  $c_i \in CF$ . In this context, join conditions have to be integrated in the query for every combination of consistency tables where there is an overlap in terms of the included attributes. For example, consistency tables c1 and c2 both include the *type* attribute. Consequently, 4wheel.type = skibag.type has to be included as join condition into the query. Table 7 shows the complete set of (partial) configurations returned by Query 4 if we assume that only Tables c1 and c2 have been defined and included into the query.

Table 7

Partial configurations determined by Query 4.

4wheel	type	skibag
yes	xdrive	yes
yes	xdrive	no
no	xdrive	yes
no	xdrive	no
no	city	no
no	limo	yes
no	limo	no
no	combi	yes
no	combi	no

*Example Query Optimization.* Following the idea of *k*consistency in constraint-based reasoning [1], the number of tuples in a consistency table can be further reduced. For example, if the car type *city* is included in Table *c*1 but there does not exist a tuple in *c*2 with type = city, all tuples of *c*1 including *city* can be removed as well. Formulated differently, we could check each entry of each consistency table for the existence of a solution which can lead to a further reduction of the number of tuples in the existing consistency tables. Following this idea, we are able to guarantee global consistency [1] meaning that each consistency table only includes tuples which are part of at least one configuration. Furthermore, as a result of related work [8], the inclusion of negative consistency tables and/or tables representing conflicts could make sense to further improve database query efficiency.

### 4. Initial Performance Analysis

We compared the performance of the three discussed approaches for representing configuration tasks as database query with constraint solving on the basis of five realworld feature models [9, 10] selected from the S.P.L.O.T. feature model repository [11]. Table 8 provides an overview of selected feature models. Due to space complexity, not all configurations could be determined for *TTax* and *FQAs* within reasonable time limits.

For each feature model, we randomly synthesized<sup>2</sup> and collected 25,000 user requirements that cover 40% of the leaf features in the feature model. We applied the systematic sampling technique [12] to select 10 *no-solution* user requirements and 10 user requirements with at least one solution. In Table 9, each setting shows the average runtime of the corresponding approach after executing the queries on the basis of these 20 user requirements. We used Choco Solver<sup>3</sup> and HSQLDB<sup>4</sup> as an in-memory relational database management system. All experiments were run with an Apple M1 Pro (8 cores) with 16-GB RAM, and an HSQLDB maximum cache size of 4GB.

Table 9 shows the results of this evaluation of selected feature models represented as (1) an explicit enumeration of *all possible configurations*, (2) an implicit representation of the feature model configuration space (*one table per variable*), (3) an implicit representation where individual tables represent *local consistency*, and (4) constraint satisfaction problem (CSP). Corresponding evaluation results show similar runtimes for small models and significantly longer runtimes for more complex models. Basically, the results of our performance evaluation show the applicability of database query based configuration approaches.

#### Table 8

Feature models used for evaluation purposes (IDE=IDE product line, DVS=digital video system, DELL=DELL Laptops, MTT=Model transformation taxonomy, FQAs=Functional Quality Attributes, FM=feature model, F= features, LF=leaf features, HC=hierarchical constraints, CC = cross-tree constraints, and CONFS = configurations).

FM	IDE	DVS	DELL	MTT	FQAs
#F	14	26	47	88	178
#LF	9	16	38	55	124
#HC	11	25	16	54	92
#CC	2	3	105	0	9
#CONFS	80	22,680	2,319	-	-

<sup>2</sup>To ensure the reproducibility of the results, we used the seed value of 141982L for the random number generator. <sup>3</sup>choco-solver.org

<sup>4</sup>hsqldb.org

#### Table 9

The average runtime (*msec*) of database query and constraintbased configuration (FM = feature model, ALLC = all configurations (no optimization), OTV = one table/variable (with node consistency), OTC = one table/constraint (with arc consistency), and CSP = constraint satisfaction problem).

FM	IDE	DVS	DELL	MTT	FQAs
ALLC	0.05	3.66	1.44	-	-
OTV	0.49	0.45	2.53	1,448	301,541
OTC	0.51	0.78	3.32	704	220,922
CSP	0.73	0.78	1.09	1.19	2.43

# 5. Threats to Validity

We have shown how to apply database queries to the identification of configurations. In a performance analysis, we compared the runtimes of database queries with the Choco constraint solver. Related results show the basic applicability of our approach, however, further evaluations and optimizations are needed - specifically with industrial datasets. Our focus in this paper is a discussion of basic alternative knowledge representation approaches that can be used as a basis for defining database queries. We are aware of related work focusing on compression aspects when supporting configuration with variant tables - see, for example, Haag [7]. A major issue for our future work will be to understand the possibilities of knowledge compression depending on the used knowledge representation. Finally, integrating machine learning with constraint solving is a relevant topic [13] - a major goal for future work is to analyze related application potentials in the context of database queries [14].

# 6. Conclusions and Future Work

We have introduced a database query based approach to constraint-based configuration. With this, we provide an alternative to approaches such as SAT solving and constraint solving. For sure, further evaluations need to be performed and the proposed queries have to be further optimized for more complex industrial scenarios.

Open issues for future related work are the following: (1) further evaluations on the basis of industrial configuration benchmarks, (2) comparison with other knowledge representation and reasoning approaches such as answer set programming (ASP) and SAT solving, (3) performance improvements through parallelization approaches (e.g., [15]), (4) understanding in more detail how constraintbased reasoning and database queries can profit from each other, for example, in which way could forward checking be applied in database queries and in which way can techniques from relational databases be useful in the context of SAT and constraint solving, and (5) we are interested in which way machine learning and knowledge compression can be used and combined to increase database query efficiency.

# References

- [1] F. Rossi, P. van Beek, T. Walsh, Handbook of Constraint Programming, Elsevier, 2006.
- [2] A. Felfernig, L. Hotz, C. Bagley, J. Tiihonen, Knowledge-based Configuration - From Research to Business Cases, Elsevier, 2014.
- [3] U. Junker, Configuration, in: F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Elsevier, 2006, pp. 837–873.
- [4] J. Gu, P. W. Purdom, J. Franco, B. W. Wah, Algorithms for the Satisfiability (SAT) Problem: A Survey, in: DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1996, pp. 19–152.
- [5] V. Myllärniemi, J. Tiihonen, M. Raatikainen, A. Felfernig, Using Answer Set Programming for Feature Model Representation and Configuration, in: ConfWS'14, Novi Sad, Serbia, 2014, pp. 1–8.
- [6] U. Junker, QuickXPlain: Preferred Explanations and Relaxations for Over-constrained Problems, in: AAAI 2004, 2004, pp. 167–172.
- [7] A. Haag, Managing Variants of a Personalized Product, JIIS 49 (2017) 59–86.
- [8] A. Haag, Arc Consistency with Negative Variant Tables, in: ConfWS'15, Vienna, Austria, 1015, pp. 81–87.
- [9] D. Benavides, S. Segura, A. Ruiz-Cortes, Automated Analysis of Feature Models 20 Years Later: A Literature Review, Inf. Sys. 35 (2010) 615–636.
- [10] K. Kang, S. Cohen, J. Hess, W. Novak, S. Peterson, Feature-oriented Domain Analysis (FODA) – Feasibility Study, Technical Report SEI-90-TR-21 (1990).
- [11] M. Mendonca, M. Branco, D. Cowan, S.P.L.O.T.: Software Product Lines Online Tools, in: OOPSLA '09, ACM, New York, NY, USA, 2009, pp. 761–762.
- [12] S. A. Mostafa, I. A. Ahmad, Recent Developments in Systematic Sampling: A Review, Journal of Statistical Theory and Practice 12 (2018) 290–310.
- [13] A. Popescu, S. Polat-Erdeniz, A. Felfernig, M. Uta, M. Atas, V. Le, K. Pilsl, M. Enzelsberger, T. Tran, An Overview of Machine Learning Techniques in Constraint Solving, JIIS 58 (2022) 91–118.
- [14] R. Guo, K. Daudjee, Research Challenges in Deep Reinforcement Learning-Based Join Query Optimization, in: aiDM'20, ACM, New York, USA, 2020.
- [15] V. Le and C. Vidal Silva and A. Felfernig and D. Benavides and J. Galindo and T.N.T. Tran, FastDiagP: An Algorithm for Parallelized Direct Diagnosis, in: AAAI-23, 2023, pp. 6442–6449.