Structural Constraint-Based Modeling and Reasoning with Basic Configuration Cells

Rafael M. Gasca, Juan A. Ortega, and Miguel Toro

Department of Languages and Computer Systems
University of Sevilla
Avda. Reina Mercedes s/n 41012 Sevilla (Spain)
{gasca, ortega, miguel.toro}@lsi.us.es

Abstract. Configuration tasks are an important application area in engineering design. The proposed solving techniques use either a constraint-based framework or a logic-based approach. We propose a methodology to obtains desired configuration using basic configuration cells(*BCC*). They are built by means of the predefined components and connections of the given configuration problem.
In practical applications of configuration tasks the *BCCs* and configuration goals are represented according to object-oriented programming paradigm. They are mapped into a numeric constraint satisfaction problem. The transformation of a basic configuration cell into a new one generates a sequence of numeric constraint satisfaction problems. We propose an algorithm that solves this sequence of problems in order to obtain a configuration solution according to the desired requirements or that detects inconsistencies in the requirements. The integration of object-oriented and constraint programming paradigms allows us to achieve a synergy that produces results that could not be obtained if each one were working individually.

1 Introduction

The goal of a lot of scientific and engineering activities has long been regarded the discovery of structural configuration. The design tasks in engineering sometimes need to combine the predefined components in order to obtain a desired configuration in a realistic time. A predefined component is described by a set of properties, by a set of ports for connecting it to other components and by structural constraints . The configuration tasks select and arrange combinations of predefined components that satisfy all the requirements.

The configuration problems have been studied in Artificial Intelligence area for the last years. A survey about the configuration frameworks has been recently published[13]. The proposed solving techniques use a constraint-based framework [12], [8] or a logic-based approach [6][10]. We propose a new structural constraint-based methodology. The modeling and search of possible configurations take into account object-oriented and constraint programming paradigms. The main motivation for integrating both paradigms is to achieve a synergy that produces results that could not be obtained if each mode were working individually.

Our work allows us to describe easily how to select and arrange components in a configuration problem. These tasks can be carried out through the series or parallel connection of components such as capacitors, resistors, chemical reactors, etc. in order to satisfy given goals. The properties of these components may be constrained between real upper and lower bounds. It takes us to consider a configuration task as a numeric constraint satisfaction problem($NCSP$). These problems can be efficiently solved by combining local consistency methods, such as approximations of arc-consistency, together with a backtracking search. Different techniques have been proposed in the bibliography [7], [1], [9],[14],[4]. The search space in numeric constraint problems is too wide and a lot of these techniques have a major drawback since they introduce choice points and they are exponential. The efficiency of some previous algorithms has been analyzed in a recent work [2].

We use abstractions named basic configuration cells(BCC) for solving previous configuration problems. These $BCCs$ allow us to model the configuration task as a set of structural constraints expressed in the form of equations, inequalities over integer or/and real variables. The model is enriched by means of the addition of symmetry-breaking constraints to avoid the inherent symmetry of the different configurations and to reduce its complexity. It is the major issue in CSP, specially when there is a large number of constraints and/or wide domains of the variables. In the last years, it has been an active research area [11], [3], [5].

The article is organized as follows: We show modeling of configuration problems in Section 2. Section 3 presents the structural constraint-based reasoning. In the last Section our conclusions and future works are presented.

2 Modeling of Configuration Problems

In the same way as the animal tissue contains cells, we consider that the solution of a configuration problem is to build something similar to a set of cells put on a determined way. These cells are the $BCCs$ and they must cover all the possible combinations of the components and their connections.

We will first show some simple configuration problem. We would like to know the series-parallel combination of predefined resistors to obtain a new resistor that has a given real resistance value R_{goal} and a set of constraints related to its cost and volume. The construction of $BCCs$ must take into account the domain knowledge and the constructs of the configurations(Components, Connections and Goals). $BBCs$ are the entities that reflect in a minimal way the possible connections of the basic components. They must allow the pure connection of the components. It will indicate the absence of some component in the $BCCs$. In the proposed problem we may add a null Resistor instance whose attributes are $R_{null}.r = 0, R_{null}.Vol = 0, R_{null}.Cost = 0$. A grammatical description of a BCC can be :

$$\begin{aligned} BCC_0 &: BCC_1 \| (BCC_2; BCC_3) \\ &\mid R \end{aligned}$$

The associated attributes and constraints of these $BCCs$ are the following:

BCC
Attributes:$\{Name, R, Cost, Vol, \{Components : BCC_1, BCC_2, BCC_3\}$
Constraints: $\{BCC_1.Cost + BCC_2.Cost + BCC_3.Cost = Cost$,
$BCC_1.Vol + BCC_2.Vol + BCC_3.Vol = Vol$,
$BCC_3.R > 0 \Rightarrow$
$(BCC_1.R + BCC_2.R) * BCC_3.R = (BCC_1.R + BCC_2.R + BCC_3.R) * R$,
$BCC_3.R = 0 \Rightarrow (BCC_1.R + BCC_2.R) = R\}$

In these $BCCs$, the modeler can add redundant symmetry-breaking constraints to remove the symmetries of the configuration problems. The object-oriented paradigm allows us to specify a BCC easily. In a similar way we specify the goals of a configuration problem as

Goals
Attributes: $\{Name\}$
Constraints: $\{BCC.Cost \leq MaxCost, BCC.Vol \leq MaxVol$,
$(BCC).R = R_{Goal}\}$

3 Methodology of Structural Constraint-Based Reasoning

This reasoning must search the *structural constraints* that satisfy the requirements of the configuration problem. Some attributes of Components Objects are numeric variables. It forces the structural constraint-based reasoning to treat numeric constraint satisfaction problems.These variables, their continuous domains and the constraints determine a $NCSP$. A $NCSP$ instance is defined by a triple (X, D, C), where $X = \{x_1, ..., x_n\}$ is a set of variables, $D = \{d_1, ..., d_n\}$ is a set of continuous domains for the variables, and C is a set of constraints. A constraint is defined by a subset of variables $X_c \subseteq X$ on which it holds, and a numeric relation linking them [7]. A solution of an instance is an assignment of values for all constrained variables, which satisfy all constraints.

A solving algorithm obtains the desired configuration. The attributes and constrainst of the BCC Objects can be mapped into Variables, Domains and Constraints Objects and their conjunction generates a $NCSP$. We have named the methodology of reasoning in these configuration problems as *Structural Constraint-based Reasoning*. The solution of the reasoning is a set of "*structural constraints*" that satisfies all the specified goals. The abstraction of the $BCCs$ allows us the performance of this task. A structural constraint-based reasoning is modeled by means of the following steps: Generation of the $NCSP$ by means of BCC Objects and application of a constraint solver with the corresponding heuristic. In this point, we want to highlight the sharp separation between the configuration problem specification and solving method. It allows the easy modification of the configuration problem specification.

3.1 Generation of the Numeric Constraint Satisfaction Problems

In this step, we generate a $NCSP$ using the BCC Objects. The variables are the attributes related to the constraints of the goals. The domains are the types of these variables and the constraints are $C_{goals} \cup C_{BCC}$. These problems may have multiple solutions or may have none. Our goal is to obtain the "*structural constraint*", such that a $NCSP$ is satisfied. If the desired requirements of the configuration problem are not satisfied in a certain $NCSP$, then we must consider a new $NCSP$.

3.2 Configuration Problem Solving

Every $NCSP$ is solved according to the exact topology of the previous $BCCs$. If they do not hold the requirements of the desired configuration, then we will build a new $NCSP$. It will have equivalent constraints, variables and the domains of the $BCCs$ Objects will increase. Then we will have a configuration problem mapped into a sequence of $NCSP$. The mechanism that obtains this sequence is a recurrent task.

Many constraint solvers are designed to tackle $NCSP$. Constraint solvers are systems that implement data structures and algorithms to perform consistency and entailment checks very efficiently. In our case we must tackle a sequence of $NCSP$ that only varies the domains of the constrained variables. We propose an algorithm that begins the construction of the initial $NCSP$. If the requirements are satisfied, then it will return the solutions that are the associated "*structural constraints*" to the BCC, but if the requirements are not satisfied, then we will build a new $NCSP$ with different BCC Objects instances. The pseudocode of the algorithm is as follows:

```
program Configuration Solving (in p:Configuration Problem)
                  out Sol:BCC
        begin
           Sol := ∅
           ncsp:= Generate an initial NCSP of p
           while ( time < MaxTime and DepthCell <= MaxDepth)
              Sol := Solve(ncsp)
              if Sol ≠ ∅
                  Return Sol
              else
                 ncsp:= Generate a new NCSP
              endif
           endwhile
        end
```

The *Solve* function is a numeric constraint solver that searches possible solutions, it creates new $BCCs$ in order to build a new $NCSP$ and it avoids the recalculation of nogood $BCCs$.The condition of termination of this algorithm may depend on the maximum Time($MaxTime$) of computation or the depth($MaxDepth$) of the $BCCs$ used.

4 Conclusions and Future Works

We have defined a novel methodology for configuration tasks that use *structural constraint-based reasoning*. Our methodology uses the advantages of object-orientation paradigm for the abstraction of components and the constraint programming paradigm that allows us an easy declaration of the constraints and a search of the structural constraints. The main entities of this methodology are basic configuration cells that permit a recursive effect in the configuration tasks. A constraint solver treats the sequence of generated $NCSPs$ efficiently.

A future work will consider other more complex configuration problems, where the components have more ports. Other interesting problems are the obtaining of configuration with different types of connections and components at the same time.

References

1. Benhamou F. and Granvilliers L.: Combining Local Consistency, Symbolic Rewriting and Interval Methods in Proceedings of AISMC-3 volume 1138 of LNCS, Springer-Verlag. (1996) 44-159
2. Collavizza H., Delobel F. and Rueher M.: Extending consistent domains of numeric CSP Proceecings of Sixteenth IJCAI'99, Stockholm, (1999) 406-411
3. Fox M. and Long D.: The Detection and Explotation of Symmetry in Planning Problems IJCAI'99 (1999), 956-961
4. Gasca, R. M. Razonamiento y simulación en sistemas que integran conocimiento cualitativo y cuantitativo Ph. D. dissertation Seville University, (1998)
5. Gent I.P. and Barbara M. Smith Symmetry Breaking in Constraint Programming. In ECAI'2000 14th European Conference on Artificial Intelligence. (2000)
6. Klein R., Buchheit M. and Nutt W. Configuration as model construction: The constructive problem solving approach. In Proc. of the 3rd Int. Conference on AI in Design (1994)
7. Lhomme O.: Consistency Techniques for Numeric CSPs Proceedings IJCAI'93, (1993) 232-238
8. D. Mailharro, A Classification and constraint-based framework for configuration. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 12(4) (1998)
9. Marti P., Rueher M.: Concurrent Cooperating Solvers over Reals. Reliable Computing 3 (1997) 325-333
10. McGuiness D.L. and Wright J. R.: Conceptual Modelling for Configuration: A description Logic-based Approach. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 12(4) (1998)
11. Meseguer P. and Torras C. Solving Strategies for Highly Symmetric CSPs. In IJCAI'99 (1999) 400-411
12. Sabin, D. and Freuder F. Configuration as Composite Constraint Satisfaction. In Workshop Notes of AAAI Fall Symposium on Configuration AAAI Press. Menlo Park, CA (1996) 28-36
13. Sabin D. and Weigel R. Product Configuration Frameworks-A survey. In IEEE Intelligent Systems (1998) 42-49
14. Van Hentenryck P., (1998) 42-49 Michel L. and Deville Y. Numerica. A modeling language for global optimization The MIT Press (1997)