Dynamic Constraint Satisfaction in Configuration

Timo Soininen
TAI Research Centre, Helsinki University of Technology
P.O.B. 9555, FIN-02015 HUT
Finland
Timo.Soininen@hut.fi

Esther Gelle
ABB Corporate Research AG
CHCRC.C2 Segelhof, CH-5405 Baden
Switzerland
Esther.Gelle@chcrc.abb.ch

Abstract

Configuration tasks exhibit dynamic aspects which require extending the basic constraint satisfaction framework. In this paper we give a new, well-founded and relatively simple definition of such dynamic constraint satisfaction problems (DCSP). On the basis of the definition, we show that the decision problem for DCSP is **NP**-complete. We also show that although the complexity of DCSP is the same as for CSP, DCSP is strictly more expressive in a knowledge representation sense. However, DCSP has its limitations in representing configuration knowledge. We generalise the activity constraints of DCSP with disjunctions and default negation, and show that the decision problem remains **NP**-complete with this generalization. We finally describe two approaches to implementing DCSP and provide test results for both.

Introduction

Configurable products are important in domains where standardized components are combined into customized products, such as computers, elevators and industrial mixers. A configuration task takes as input a model which describes the components that can be included in the product and a set of constraints that define how components can be combined, and requirements that specify properties of the product to be configured. The output is a description of a product to be manufactured, a configuration. It consists of a set of components as well as a specification of how they interact to form the working product. The configuration has to satisfy the constraints in the model and the requirements.

The combinatorial nature of a configuration problem is well captured by constraint satisfaction problems (CSP). A CSP consists of a set of variables with domains on which allowed value combinations are specified as constraints. Powerful search algorithms have been developed for solving CSPs (Tsang 1993). A CSP can be used to express *compatibility* knowledge in configuration tasks, and the algorithms to find a solution.

In some configuration tasks, however, *optional* components may be added or some components may *require* the existence of another component. This type of task leads to a constraint problem in which the set of variables that must be assigned a value may change dynamically in response to choices made in the course of problem solving. The solutions to such a problem differ in the sets of variables that are assigned values. When configuring a mixer, for example, a condenser is a typical optional component which does not have to be present in every solution. It is only necessary if the vessel volume is large and chemical reactions are expected to occur. This is difficult to capture in a standard CSP in which all variables are assigned values in every solution. Much effort has therefore been put to include dynamic aspects in CSPs (Mittal & Falkenhainer 1990; Sabin & Freuder 1996; Stumptner, Friedrich, & Haselböck 1998; Gelle 1998).

In this paper, we continue this line of research by giving a new definition of *dynamic CSP* that is equivalent to the original definition in (Mittal & Falkenhainer 1990), mathematically well-founded, and simpler than the characterization by (Bowen & Bahler 1991). The definition utilizes a fix point condition inspired by the semantics of logic programs (Lloyd 1987; Gelfond & Lifschitz 1988) instead of the minimality condition in the original definition. Using the new definition, we show that DCSP is **NP**-complete. It also remains **NP**-complete when suitably generalized with default negations on constraints and disjunctions on the activity of variables in activity constraints, which are useful for representing configuration knowledge. Although there is no difference between DCSP and CSP regarding computational complexity, we show that DSCP is more expressive than CSP in a knowledge representation sense. Finally, we sketch and compare two novel implementations of DCSP based on a variant of the original algorithm (Mittal & Falkenhainer 1990) and on mapping DCSP to problem solving with propositional logic programs (Soininen & Niemelä 1999).

The DCSP Formalism

We first recall the original definition of a dynamic constraint satisfaction problem. An instance $\mathcal{P}$ of DCSP is of form $\langle \mathcal{V}, \mathcal{D}, \mathcal{V}_I, \mathcal{C}_C, \mathcal{C}_A \rangle$, where $\mathcal{V} = \{v_1, \ldots, v_n\}$ is the set of *variables*, $\mathcal{D} = \{D_1, \ldots, D_n\}$ is the set of *domains* of the variables, and $D_i = \{d_{i1}, \ldots, d_{ij_i}\}$ are the *values*

in the domains. The set of *initial* variables is denoted by $\mathcal{V}_I$, $\mathcal{V}_I \subseteq \mathcal{V}$, the set of *compatibility constraints* by $\mathcal{C}_C$ and the set of *activity constraints* by $\mathcal{C}_A$. We assume that all the sets in a DCSP are finite. We define a *legal assignment* as follows:

Definition 1 *An assignment of a value d_{ij} to a variable v_i is of the form $v_i = d_{ij}$, where $d_{ij} \in D_i$. A legal assignment $\mathcal{A}$ to a DCSP $\langle \mathcal{V}, \mathcal{D}, \mathcal{V}_I, \mathcal{C}_C, \mathcal{C}_A \rangle$ is a set of such assignments with at most one assignment for each variable. Formally: $\mathcal{A} \subseteq \{v_i = d_{ik} \mid v_i \in \mathcal{V}, d_{ik} \in D_i,$ and for all $i, \mid \{v_i = d_{ik} \mid v_i = d_{ik} \subseteq \mathcal{A}\} \mid \leq 1\}$*

In contrast to CSP, an assignment to DCSP does not necessarily assign a value to each variable. A variable can thus be in one of two states: active or not active. A variable is said to be *active* iff it is assigned a value in an assignment.

A compatibility constraint c_i with arity j specifies the set of allowed combinations of values for a set of variables $v_1, \ldots, v_j$ as a subset of the Cartesian product of the domains of the variables, i.e. $c_i \subseteq D_1 \times \ldots \times D_j$. We say that a compatibility constraint is *active* iff all the variables it constrains are active. An activity constraint is of form $c_j \stackrel{ACT}{\rightarrow} v$, where c_j is defined equivalently to a compatibility constraint and v is the variable that must be active if c_j is active and satisfied. Note that the original definition required that the activated variable is distinct from the variables that the constraint on the left hand side refers to. Our definition will work for such constraints as well. *Always require*, *require not* and *always require not* activity constraints (Mittal & Falkenhainer 1990) are treated as shorthand notation for simplicity. [1]

A solution to a DCSP is a subset minimal legal assignment of values to the variables that satisfies all compatibility and activity constraints.

Definition 2 *A legal assignment $\mathcal{A}$ for a DCSP $\mathcal{P} = \langle \mathcal{V}, \mathcal{D}, \mathcal{V}_I, \mathcal{C}_C, \mathcal{C}_A \rangle$ satisfies the constraints in $\mathcal{P}$ iff the following conditions hold:*

1. *$\mathcal{A}$ satisfies the compatibility constraints $\mathcal{C}_C$, i.e. for all $c \in \mathcal{C}_C$ s.t. c constrains variables $v_1, \ldots, v_j$ it holds that if c is active, i.e. $\{v_1 = d_{1k_1}, \ldots, v_j = d_{jk_j}\} \subseteq \mathcal{A}$ for some $d_{ik_i}, 1 \leq i \leq j$, then $(d_{1k_1}, \ldots, d_{jk_j}) \in c_i$.*
2. *$\mathcal{A}$ satisfies the activity constraints $\mathcal{C}_A$, i.e. for all $a \in \mathcal{C}_A, a = (c \stackrel{ACT}{\rightarrow} v_k)$ it holds that if c is active in and satisfied by $\mathcal{A}$ then v_k is active, i.e. $(v_k = d_{kj}) \in \mathcal{A}$ s.t. $d_{kj} \in D_k$.*

Definition 3 *A legal assignment $\mathcal{A}$ is a solution to a DCSP iff $\mathcal{A}$*

1. *satisfies the constraints,*
2. *contains assignments of values to initial variables,*
3. *is subset minimal, i.e. there is no assignment $\mathcal{A}_1$ that satisfies 1. and 2. s.t. $\mathcal{A}_1 \subset \mathcal{A}$.*

[1] For example, an always require constraint of the form $v_1, \ldots, v_i \stackrel{ACT}{\rightarrow} v_j$ is defined as $c \stackrel{ACT}{\rightarrow} v_j$ where c allows all value combinations on the variables $v_1, \ldots, v_i$, i.e $c = D_1 \times \ldots \times D_i$.

Variable	Domain	
Mt	{disp,entr,susp,blend}	$\in \mathcal{V}_I$
Mi	{mixer,reactor,tank}	$\in \mathcal{V}_I$
V	{spheric,cyl,ellip}	
Coo	{coo1,coo2}	
Con	{con1,con2}	
Mt.press	{high,low}	
Mt.trans	{true,false}	
V.vol	{large,small}	

Figure 1: *Variables of the mixer configuration problem.*

Example 1 *We use a typical configurable product, an industrial mixer (van Velzen 1993) to demonstrate DCSP. The components of the mixer such as vessel, cooler, condenser and their properties, for example the vessel volume, are represented as variables. The cooler and condenser are optional. The type of mixing task is represented as the variable mixing task with properties pressure and heat transfer. The components have different types, e.g. the mixer is of type reactor, storage tank, or simple mixer. These as well as the different mixing tasks are represented as domains of the variables. To designate variables in the problem, we use the first letters of their names. The variables and domains of the mixer configuration problem are shown in Figure 1 and the constraints in Figure 2. A cooler and condenser need only be included in the solution if other variables have specific values. Therefore, they are introduced by activity constraints. Activity constraints are also used to introduce the properties of components.*

Two solutions to the mixer problem are $\{Mt = disp, Mi = mixer, Mt.trans = false, Mt.press = high, V = spheric, V.vol = large, Con = con1\}$ and $\{Mt = blend, Mi = reactor, Mt.trans = true, Mt.press = high, V = spheric, V.vol = small, Coo = coo1\}$. The first solution includes a condenser whereas the second has a cooler. There are also solutions without cooler and condenser but no solutions exist containing both. In the latter case, the compatibility constraints C_3 and C_4 require different values for the vessel volume.

A Fixpoint Definition of DCSP

In this section we give a new definition of a solution to a DCSP that utilizes a fixpoint condition instead of the minimality condition on the solutions. We then show that the definition is equivalent to the original definition. This allows a straightforward analysis and generalization of DCSP in latter sections.

The fixpoint condition is another way of ensuring that active variables are justified by initial variables and activity constraints. To capture this, we first define a *reduction* of a set of activity constraints and the initial variables with respect to an assignment $\mathcal{A}$. The intuition behind the reduction is that the reduct contains

$a_1 \quad Mi \overset{ACT}{\rightarrow} V$
$a_2 \quad V \overset{ACT}{\rightarrow} V.vol$
$a_3 \quad Mt \overset{ACT}{\rightarrow} Mt.trans$
$a_4 \quad Mt \overset{ACT}{\rightarrow} Mt.press$
$a_5 \quad Mi = reactor \overset{ACT}{\rightarrow} Coo$
$a_6 \quad Mt = disp \overset{ACT}{\rightarrow} Con$

$c_1(Mt.press, V) =$
$\{(high, spheric), (high, ellip), (high, cyl), (low, cyl)\}$
$c_2(Mt.trans, Mi) =$
$\{(true, reactor), (false, mixer), (false, tank)\}$
$c_3(Mi, V.vol) =$
$\{(reactor, small), (tank, small), (tank, large)$
$(mixer, small), (mixer, large)\}$
$c_4(Con, V.vol) =$
$\{(con1, large), (con2, large)\}$

Figure 2: *Activity and compatibility constraints in the mixer configuration problem.*

the possible justifications for the assignments of values to variables in $\mathcal{A}$ in the form of a set of *instantiated activity constraints*. These are obtained by instantiating the activity constraints to reflect $\mathcal{A}$ in the sense that the variable in the right hand side of an activity constraint is replaced by its assignment in $\mathcal{A}$. An activity constraint whose activated variable is not active in $\mathcal{A}$ is not included in the reduct. In addition, each initial variable is given a special activity constraint form, the left hand side of which is satisfied by every assignment.

Definition 4 *We define the reduct $(\mathcal{C}_A, \mathcal{V}_I)^{\mathcal{A}}$ of a set of activity constraints, $\mathcal{C}_A$, and a set of initial variables, $\mathcal{V}_I$, w.r.t an assignment $\mathcal{A}$ as follows:* $(\mathcal{C}_A, \mathcal{V}_I)^{\mathcal{A}} =$
$\{c \overset{ACT}{\rightarrow} v_j = d_{jk} \mid (c \overset{ACT}{\rightarrow} v_j) \in \mathcal{C}_A, (v_j = d_{jk}) \in \mathcal{A}\} \cup$
$\{\overset{ACT}{\rightarrow} v_j = d_{jk} \mid v_j \in \mathcal{V}_I, (v_j = d_{jk}) \in \mathcal{A}\}$

We denote the reduct $(\mathcal{C}_A, \mathcal{V}_I)^{\mathcal{A}}$ by the shorthand notation $\mathcal{C}_I$. We now define an operator on the lattice formed by the sets of all possible assignments and the subset relation on these. The operator intuitively captures how the instantiated activity constraints introduce new active variables to an assignment when their left hand side constraints are active and satisfied.

Definition 5 *Given a set of instantiated activity constraints $\mathcal{C}_I$, the operator $T_{\mathcal{C}_I}()$ on an assignment $\mathcal{A}$ is defined as follows:* $T_{\mathcal{C}_I}(\mathcal{A}) =$
$\{v_j = d_{jk} \mid (c \overset{ACT}{\rightarrow} v_j = d_{jk}) \in \mathcal{C}_I$, *$c$ is active in and satisfied by $\mathcal{A}$*$\}$.

Intuitively, a solution is a legal assignment that satisfies the constraints in the DCSP and contains all and only the active variables that are justified by the set of initial variables and activity constraints. Since active variables may be justified recursively, a solution to a DCSP is defined as a fixpoint of the above operator for a given reduct. A *fixpoint* q of an operator $\tau(.)$ is such that $\tau(q) = q$. This ensures that every variable with a justification is active in the solution.

The operator T is monotonic, i.e. for assignments $\mathcal{A}_i$ and $\mathcal{A}_j$, $\mathcal{A}_i \subseteq \mathcal{A}_j$, it holds that $T_{\mathcal{C}_I}(\mathcal{A}_i) \subseteq T_{\mathcal{C}_I}(\mathcal{A}_j)$. This can be seen by noting that if the constraint part of an activity constraint is satisfied by an assignment, it is satisfied by all its supersets as well. A monotonic operator has a unique *least fixpoint* that can be computed by iteratively applying the operator, starting from the empty set (Lloyd 1987). As a solution must not contain unjustified active variables, it should be such a least fixpoint, denoted by $lfp(.)$.

Definition 6 *A legal assignment $\mathcal{A}$ is a solution to a DCSP $\mathcal{P} = \langle \mathcal{V}, \mathcal{D}, \mathcal{V}_I, \mathcal{C}_C, \mathcal{C}_A \rangle$ iff i) $\mathcal{A}$ satisfies the constraints in $\mathcal{P}$, ii) the initial variables are active in $\mathcal{A}$, and iii)* $\mathcal{A} = \mathrm{lfp}(T_{(\mathcal{C}_A, \mathcal{V}_I)^{\mathcal{A}}})$

Example 2 *Consider the following simplified version of the mixer DCSP. Let* $\mathcal{V} = \{Mi, Mt, V.vol, Con\}$, $D_{Mi} = \{r, m, t\}$, $D_{Mt} = \{d, b\}$, $D_{V.vol} = \{s, l\}$, $D_{Con} = \{c1, c2\}$, $\mathcal{V}_I = \{Mi, Mt\}$. *Further, let* $\mathcal{C}_C = \{c_1\}$, *where* $c_1 = \{(c1, l), (c2, l)\}$ *is a constraint on Con and $V.vol$. The activity constraints* $\mathcal{C}_A = \{a_1, a_2\}$ *are as follows:* $a_1 = (Mi \overset{ACT}{\rightarrow} V.vol)$ *and* $a_2 = (Mt = d \overset{ACT}{\rightarrow} Con)$. *The assignment* $\mathcal{A}_1 = \{Mi = r, Mt = b, V.vol = l\}$ *is a solution since it clearly satisfies the constraints,* $\mathcal{C}_I = (\mathcal{C}_A, \mathcal{V}_I)^{\mathcal{A}_1} = \{\overset{ACT}{\rightarrow} Mi = r, \overset{ACT}{\rightarrow} Mt = b, Mi \overset{ACT}{\rightarrow} V.vol = l\}$, *and* $\mathrm{lfp}(T_{\mathcal{C}_I}) = \mathcal{A}_1$. *The assignment* $\mathcal{A}_2 = \{Mi = r, Mt = b, V.vol = l, Con = c1\}$ *is not a solution. It does satisfy the constraints, but* $\mathcal{C}_I = (\mathcal{C}_A, \mathcal{V}_I)^{\mathcal{A}_2} = \{\overset{ACT}{\rightarrow} Mi = r, \overset{ACT}{\rightarrow} Mt = b, Mi \overset{ACT}{\rightarrow} V.vol = l, Mt = d \overset{ACT}{\rightarrow} Con = c1\}$, *and* $\mathrm{lfp}(T_{\mathcal{C}_I}) = \{Mi = r, Mt = b, V.vol = l\} \neq \mathcal{A}_2$.

We now show that our definition is equivalent to the original definition.

Theorem 1 *Definition 6 is equivalent to Definition 3.*

Proof: The minimality condition of Definition 3 is replaced by the fixpoint condition in Definition 6. In order to prove the equivalence it remains to show that the fixpoint condition is equivalent to subset minimality. This can be done using the following observations. A solution according to Definition 3 is a minimal set satisfying the constraints and making the initial variables active. Hence, it is contained in a solution according to Definition 6 which also satisfies the constraints and makes the initial variables active. On the other hand, a solution according to Definition 3 is contained in a solution according to Definition 6 as it can be constructed iteratively by starting from the empty assignment and by applying the operator T. It can be shown inductively that the result of this iteration is contained in a solution according to Definition 3.

Properties of DCSP

In this section we show that the decision task of DCSP is **NP**-complete and therefore the same as for CSP. However, we also show that DCSP is strictly more expressive than CSP in a knowledge representation sense.

Definition 7 *$DCSP(D)$: Given a DCSP $\mathcal{P}$, is there a solution for $\mathcal{P}$?*

Theorem 2 *$DCSP(D)$ is **NP**-complete.*

Proof: The task is **NP**-hard since CSP, which is known to be **NP**-complete, is a special case of DCSP with $\mathcal{V}_I = \mathcal{V}$ and no activity constraints. Further, $DCSP(D)$ is **NP**-complete because it is in **NP**. The containment in **NP** is due to the fact that whether an assignment is a solution can be checked in polynomial time. This result can be shown by noting that whether an assignment satisfies a set of constraints and whether it is the least fix-point of the operator on the reduced activity constraints can be both decided in polynomial time. The latter property holds since the reduct can obviously be computed in polynomial time, by processing one rule at a time, and the least fix-point of the operator can be computed in polynomial time.

Thus, from the computational complexity point of view, there is no difference between DCSP and CSP. However, we next show that DCSP is strictly more expressive than CSP by using the concept of *modularity* (Soininen & Niemelä 1999). Intuitively, a modular representation of knowledge is such that a small change in the knowledge leads to a small change in the representation. This property is important for maintaining the knowledge in a configurator. For example, if knowledge represented as a DCSP were mapped to a CSP, a simple update like adding a variable to the set of initial variables should result in a local change in the CSP.

More precisely, we say that DCSP is representable by CSP iff there is a mapping for every DCSP $\mathcal{P}$ to a CSP $\mathrm{T}(\mathcal{P})$ such that the solutions to $\mathrm{T}(\mathcal{P})$ agree with the solutions to $\mathcal{P}$. The solutions *agree* iff the assignments of values to the variables of $\mathrm{T}(\mathcal{P})$ are the same as for $\mathcal{P}$ if the variables are active, and NULL otherwise. Note that this requirement is not a real restriction, as $\mathrm{T}(\mathcal{P})$ may contain additionally arbitrary variables, domains and constraints. It is made to ensure that the solution to $\mathcal{P}$ can be easily recovered from the solution to $\mathrm{T}(\mathcal{P})$.

Further, we say that such a mapping is *modular* if the mapping of the initial variables is independent of the other parts. This means that a change in the initial variables of $\mathcal{P}$ leads only to *simple updates* to $\mathrm{T}(\mathcal{P})$. A simple update consists of arbitrary number of either i) additions of constraints, removals of allowed tuples from constraints, removals of values from domains, and removals of variables with their domains, or ii) removals of constraints, additions of allowed tuples to constraints, additions of values to domains and additions of variables with their domains. In other words, in a simple update it is not allowed to mix changes in the first and second sets. In the case of removing the variables, the constraints and assignments are projected to the remaining variables appropriately. We note that this concept of modularity and the following result can be extended to involve more complex changes to DCSP, such as additions and removals of compatibility and activity constraints and their allowed value tuples.

Theorem 3 *DCSP is not modularly representable by CSP.*

Proof: Consider the DCSP $\mathcal{P} = \langle \mathcal{V}, \mathcal{D}, \mathcal{V}_I, \mathcal{C}_C, \mathcal{C}_A \rangle$ with $\mathcal{V} = \{v_1, v_2\}, \mathcal{V}_I = \{v_1\}, D_1 = \{a\}, D_2 = \{b\}, \mathcal{C}_C = \mathcal{C}_A = \emptyset$, and assume it can be modularly represented by a CSP. Hence, there is a CSP $\mathrm{T}(\mathcal{P})$ such that in all the solutions of $\mathrm{T}(\mathcal{P})$ $v_1 = a$ and $v_2 = NULL$, as that is the only solution to $\mathcal{P}$. Consider now adding v_2 to $\mathcal{V}_I$. The resulting DCSP has only one solution where $v_1 = a$ and $v_2 = b$. This means that $\mathrm{T}(\mathcal{P})$ updated with v_2 must not have a solution in which $v_1 = a$ and $v_2 = NULL$. In addition, $\mathrm{T}(\mathcal{P})$ updated with v_2 must have at least one solution in which $v_1 = a$ and $v_2 = b$. It can be shown that simple updates cannot both add solutions and remove them, which is a contradiction and hence the assumption is false.

The fact that there is no modular representation of DCSP in the CSP formalism is caused by the requirement that activity constraints or initial variables justify the activity of a variable in a solution. We note that there obviously is a modular mapping from a CSP to DCSP, since CSP is a special case of DCSP.

Generalizing DCSP

In this section we generalize the activity constraints of DCSP. The aim is to extend the expressiveness of activity constraints while keeping the complexity of the corresponding decisions problems in **NP**. We show that this indeed is the case for our generalization.

The activity constraints of DCSP are not very expressive. For instance, in some cases a functional requirement can be satisfied by any one of a given set of components, which would require disjunctive activity constraints. Another case that cannot be represented in a straightforward manner is that a variable is active if some constraints are not active or satisfied in an assignment, i.e. complements of activity or constraints.

To enable succinct representation of such knowledge, we allow activity constraints of the following form:

$$c_1, \ldots, c_j, not(c_{j+1}), \ldots, not(c_k) \stackrel{ACT}{\rightarrow} m\{v_1 \mid \ldots \mid v_l\}n \quad (1)$$

where $0 \leq m \leq n$. Intuitively, this activity constraint states that if constraints $c_1, \ldots, c_j$ are satisfied by an assignment $\mathcal{A}$ and the constraints $c_{j+1}, \ldots, c_k$ are not satisfied by or not active in $\mathcal{A}$, then a subset of the variables $v_1, \ldots, v_l$ is active in $\mathcal{A}$, such that the cardinality of the subset is between m and n. If $m = 1$ and n is the number of variables in the right hand side, this becomes an inclusive disjunction of the variables. On the

other hand, if $m = n = 1$, the right hand side becomes an exclusive disjunction. We allow activity constraints with $m = 0$ for representing *optional* variables.

We next define when an assignment satisfies the new form of activity constraints and the reduction of activity constraints w.r.t. an assignment. The reduct handles default negations by pruning from the instantiated activity constraints those for which one of the constraints within default negation is active and satisfied, and by removing the default negations from the rest of the instantiated activity constraints. This is similar to the stable model semantics of logic programs (Gelfond & Lifschitz 1988). The operator on assignments is extended to handle the remaining activity constraints with several constraints in their left hand side in the obvious manner, i.e. all of the constraints need to be active and satisfied. The definition of when a compatibility constraint is satisfied by an assignment and the definition of a solution remain the same as previously.

Definition 8 *A legal assignment $\mathcal{A}$ for a DCSP $\mathcal{P} = \langle \mathcal{V}, \mathcal{D}, \mathcal{V}_I, \mathcal{C}_C, \mathcal{C}_A \rangle$ satisfies the activity constraints in $\mathcal{P}$ iff for each activity constraint in $\mathcal{C}_A$ of the form (1) the following holds: if $c_1 \ldots c_j$ are active and satisfied, and $c_{j+1}, \ldots, c_k$ are not satisfied by or not active in $\mathcal{A}$, then some set $V \subseteq \{v_1, \ldots, v_l\}$ is active in $\mathcal{A}$, and $m \leq \mid V \mid \leq n$.*

Definition 9 *We define the reduct $(\mathcal{C}_A, \mathcal{V}_I)^{\mathcal{A}}$ of a set of activity constraints of the form (1), $\mathcal{C}_A$, and a set of initial variables, $\mathcal{V}_I$, w.r.t an assignment $\mathcal{A}$ as follows:* $(\mathcal{C}_A, \mathcal{V}_I)^{\mathcal{A}} = \{c_1, \ldots, c_i \stackrel{ACT}{\rightarrow} v_j = d_{jm} \mid (c_1, \ldots, c_i, not(c_{i+1}), \ldots, not(c_k) \stackrel{ACT}{\rightarrow} m\{v_1 \mid \ldots \mid v_l\}n) \in \mathcal{C}_A, (v_j = d_{jm}) \in \mathcal{A}$ *for some j where $1 \leq j \leq l$, $c_{i+1}, \ldots, c_k$ are not satisfied by or not active in $\mathcal{A}$}$\}$
$\cup$
$\{\stackrel{ACT}{\rightarrow} v_j = d_{jk} \mid v_j \in \mathcal{V}_I, (v_j = d_{jk}) \in \mathcal{A}\}$

Example 3 *Consider a DCSP with variables $Mi, Mt, V.vol, Coo, Con$ with domains as given in Figure 1, a set of initial variables $\{Mi, Mt, V.vol\}$, an activity constraint $c_1 \stackrel{ACT}{\rightarrow} 1\{Coo \mid Con\}1$ with c_1 a constraint $\{(reactor, disp)\}$ defined on Mi and Mt, and a compatibility constraint $c_2 = \{(con1, large)(con2, large)(con2, small)\}$ defined on Con and $V.vol$. The assignment $\mathcal{A}_1 = \{Mi = reactor, Mt = disp, V.vol = small, Con = con2\}$ is a solution since it clearly satisfies the constraints and for the reduct $(\mathcal{C}_A, \mathcal{V}_I)^{\mathcal{A}_1} = \{\stackrel{ACT}{\rightarrow} Mi = reactor, \stackrel{ACT}{\rightarrow} Mt = disp, \stackrel{ACT}{\rightarrow} V.vol = small, c_1 \stackrel{ACT}{\rightarrow} Con = con2\}$, $\mathrm{lfp}(T_{(\mathcal{C}_A, \mathcal{V}_I)^{\mathcal{A}_1}}) = \mathcal{A}_1$. On the other hand, $\mathcal{A}_2 = \{Mi = reactor, Mt = disp, V.vol = small, Con = con2, Coo = coo1\}$ is not a solution since it does not satisfy the constraint that only one of the variables Coo and Con should be active.*

A similar argument as for Theorem 2 can be used to establish the following result.

Theorem 4 *$DCSP(D)$ is **NP**-complete when extended with activity constraints of the form (1).*

In order to retain the same complexity for the generalized DCSP we in fact relax the subset minimality condition. Each active variable in a solution is justified by the activity constraints or initial variables, but the solutions may not be minimal, since an activity constraint may justify more variables than a minimal solution would contain. We believe that in configuration problems minimality is more related to optimality of a solution rather than correctness of a solution, and therefore consider our generalization a reasonable one.

Implementation

In this section, we briefly discuss two novel implementations of DSCP. The first is similar to the original algorithm described in (Mittal & Falkenhainer 1990) whereas the second is based on mapping a DCSP to a type of propositional logic programs (Soininen & Niemelä 1999). To test the performance of both algorithms, we use a set of examples from the configuration domain, CAR (Mittal & Falkenhainer 1990), CARx2 (Soininen & Niemelä 1999), a simplified form of a hospital monitor problem (Soininen *et al.* 1998) and the mixer problem in Example 1.

The first implementation differs from the one described in (Mittal & Falkenhainer 1990) in that it does not use an ATMS. The algorithm is based on a simple backtracking algorithm used to solve standard CSPs. As long as variables are not yet assiged a value the algorithm chooses the next variable and a value to assign to that variable such that the value is still consistent with the compatibility constraints. Then the algorithm checks if some activity constraint has become relevant, i.e. if the condition in an activity constraint is consistent with the already assigned values. In that case, new variables are activated and added to the list of not yet assigned variables. If all values of a variable have been considered, the algorithm backtracks to the last variable that still has values left. Variables that have been activated by an activity constraint based on the value of a variable deassigned in the backtracking step have to be deactivated as well. The algorithm continues until no more activity constraints can be activated and all currently active variables have been assigned a value. The implementation was written in Java (Sun's JDK) and run on a Pentium II, 266 MHz, with 96MB of memory, Windows NT operating system. The implementation and the test problems are available at `www.cs.hut.fi/~pdmg/CP99/`.

The second implementation is based on first translating the DCSP to a set of rules in Configuration Rule Language (**CRL**), a type of propositional logic programs (Soininen & Niemelä 1999), for which the solutions are found using an efficient C++ implementation of the stable model semantics of normal logic programs. The tests were run on a Pentium II 266 MHz with 128MB of memory, Linux 2.2.3 operating system, `smodels` version 1.12 and `lparse` version 0.9.25. Further details can be found in (Soininen & Niemelä 1999) and the implementation and the

Table 1: *Characteristics of the examples.*

Example	# solutions)	Search space
Car **CRL**	198	1296
Carx2 **CRL**	44456	331776
Monitor **CRL**	1320	196608
Mixer **CRL**	88	1152

Table 2: *Results for the examples.*

Problem, implem.	First solution	All solutions
Car, backtrack	0.04 s	0.07 s
Car, **CRL**	0.05 s	0.07 s
Carx2, backtrack	0.01 s	11.8 s
Carx2, **CRL**	0.08 s	5.1 s
Monitor, backtrack	< 1 ms	0.08 s
Monitor, **CRL**	0.10 s	0.39 s
Mixer, backtrack	< 1 ms	0.01 s
Mixer, **CRL**	0.05 s	0.06 s

test problems at `www.tcs.hut.fi/pub/smodels` and `www.tcs.hut.fi/pub/smodels/tests/cp99.tar.gz`.

The number of solutions to the problems found by the implementations and the potential search space for them are given in Table 1. The size of the potential search space is calculated by multiplying the sizes of the domains of the variables. The times to find the first and all solutions are given in Table 2. We note that for the first implementation, the problem inputs and outputs are handled in main memory, whereas the results for the second implementation include the time for reading the input from a file and parsing it. However, the execution time for the second implementation does not include the time for manually translating the DCSP to **CRL**.

Conclusions and Future Work

Dynamic constraint satisfaction problems were introduced to capture the dynamic aspect of, e.g., configuration (Mittal & Falkenhainer 1990). We present a relatively simple and mathematically well-founded definition of such problems which is equivalent to the original. We then show that the decision problem of DCSP is **NP**-complete. Thus, from the complexity point of view, DCSP and CSP are equivalent. However, we also show that DCSP is strictly more expressive than CSP in a knowledge representation sense and thus indeed seems a more feasible representation for dynamic problems.

The activity constraints of DCSP are limited with regard to configuration knowledge modeling. We generalize them with default negation on constraints and their activity, and a generalized disjunction of activity of variables. The decision problem for the generalized DCSP is shown to remain **NP**-complete. As further work, resource constraints and non-predefined sets of variables (Stumptner, Friedrich, & Haselböck 1998) should also be cleanly included in our definition to represent resource interactions and configurations whose size is not known in advance.

There are few reports on implementations of algorithms for the original DCSP. We sketch two implementations, one based on a modified CSP algorithm, the other based on the mapping from DCSP to propositional logic programs. The test results for both on a set of simple problems are close to each other and acceptable for practical applications. The implementations should be extended to handle the generalizations in this paper and outlined as future work and empirically tested on larger problems.

Acknowledgements

The work of the first author has been supported by the Helsinki Graduate School in Computer Science and Engineering (HeCSE) and the Technology Development Centre Finland. We thank Ilkka Niemelä and Tomi Männistö for their comments on the paper.

References

Bowen, J., and Bahler, D. 1991. Conditional existence of variables in generalized constraint networks. In *Proc. of the Ninth National Conf. on Artificial Intelligence*, 215–220.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proc. of the 5th International Conf. on Logic Programming*, 1070–1080.

Gelle, E. M. 1998. *On the generation of locally consistent solution spaces.* Phd thesis, Ecole Polytechnique Fédérale de Lausanne, Switzerland.

Lloyd, J. 1987. *Foundations of Logic Programming.* Springer-Verlag, second edition.

Mittal, S., and Falkenhainer, B. 1990. Dynamic constraint satisfaction problems. In *Proc. of the 8th National Conf. on Artificial Intelligence*, 25–32.

Sabin, D., and Freuder, E. C. 1996. Configuration as composite constraint satisfaction. In *Configuration – Papers from the 1996 Fall Symposium. AAAI Technical Report FS-96-03.*

Soininen, T., and Niemelä, I. 1999. Developing a declarative rule language for applications in product configuration. In *Practical Aspects of Declarative Languages (PADL99), LNCS 1551.* Springer-Verlag.

Soininen, T.; Tiihonen, J.; Männistö, T.; and Sulonen, R. 1998. Towards a general ontology of configuration. *AI EDAM* 12:357–372.

Stumptner, M.; Friedrich, G.; and Haselböck, A. 1998. Generative constraint-based configuration of large technical systems. *AI EDAM* 12:307–320.

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

van Velzen, M. 1993. A Piece of CAKE, Computer Aided Knowledge Engineering on KADSified Configuration Tasks. Master's thesis, Univeristy of Amsterdam, Social Science Informatics.