
GMD Report
93

GMD –
Forschungszentrum
Informationstechnik
GmbH

Markus Hannebauer

# A Formalization of Autonomous Dynamic Reconfiguration in Distributed Constraint Satisfaction

February 2000

GMD –
Forschungszentrum Informationstechnik GmbH
Schloß Birlinghoven
D-53754 Sankt Augustin
Germany
Telefon +49 -2241 -14 -0
Telefax +49 -2241 -14 -2618
http://www.gmd.de



**Anschrift des Verfassers/Address of the author:**
Markus Hannebauer
Institut für Rechnerarchitektur und Softwaretechnik (FIRST)
GMD – Forschungszentrum Informationstechnik GmbH
Kekuléstraße 7
D-12489 Berlin-Adlershof
E-mail: hannebau@first.gmd.de

ISSN 1435-2702

**Abstract**
Several interesting practical problems in process control, planning and scheduling can be expressed and solved using the model of *constraint satisfaction problems*. At least four drawbacks of this classical model directly relate to areas of distribution: complexity, scalability, privacy and robustness. Hence, research on distributed constraint satisfaction problems is a new direction in this area. A typical engineering task in distributed constraint satisfaction is the design of the distribution itself. A careful look at this task reveals that the design of distribution is critical to the quality and efficiency of the problem solving process and is itself an optimization problem.

In this report we formalize different variants of this configuration problem and prove them to be all at least $NP$-complete. For solving these problems, we present two local operators, *agent melting* and *agent splitting*, that can be combined to allow for an autonomous and dynamic reconfiguration of the organizational structure of the problem solvers. We prove sequences of these operators to be sufficient for solving any given configuration problem. We also briefly describe what practical steps are necessary to exploit the rather theoretical result of the proof in realistic applications.

**Keywords**
distributed constraint satisfaction, autonomous dynamic reconfiguration, partitioning, agent melting, agent splitting

**Zusammenfassung**
Viele interessante praktische Probleme in der Prozeßkontrolle, in Aktions- und Einsatzplanung können als *Constraint-Satisfaction-Probleme* modelliert und mit entsprechenden Verfahren gelöst werden. Mindestens vier der Nachteile dieses klassischen Modells hängen direkt mit Themen der Verteilung zusammen: Komplexität, Skalierbarkeit, Datenschutz und Robustheit. Aus diesem Grunde hat sich in letzter Zeit die Untersuchung von *Distributed-Constraint-Satisfaction-Problem* als Forschungsrichtung etabliert. Eine typische technische Aufgabe in diesem Zusammenhang ist der Entwurf der Verteilung an sich. Eine genaue Untersuchung dieser Aufgabe zeigt, daß der Verteilungsentwurf eine kritische Größe für die Qualität und die Effizienz des Problemlösungsprozesses ist und auch selbst ein Optimierungsproblem darstellt.

In diesem Bericht formalisieren wir verschiedene Varianten dieses Konfigurationsproblems and weisen nach, daß sie alle zumindest $NP$-vollständig sind. Zur Lösung dieser Probleme schlagen wir zwei lokale Operatoren vor, die *Agentenverschmelzung* und die *Agentenzerteilung*, die beide kombiniert werden können, um die autonome und dynamische Rekonfiguration der Organisationsstruktur zwischen den Problemlösern zu ermöglichen. Wir beweisen, daß Folgen dieser Operatoren hinreichend für die Lösung jedes Konfigurationsproblems sind. Außerdem beschreiben wir kurz, welche praktischen Schritte notwendig sind, um die eher theoretischen Ergebnisses dieses Beweises in realistischen Anwendungen umzusetzen.

**Stichworte**
Distributed-Constraint-Satisfaction, autonome dynamische Rekonfiguration, Partitionierung, Agentenverschmelzung, Agentenzerteilung

# 1 Introduction

Several interesting practical problems in process control, planning and scheduling can be expressed and solved using the model of *constraint satisfaction problems* (CSPs, [28]) and the algorithms of constraint programming. Though the CSP framework is very generic and its algorithms are often quite successful in solving $NP$-hard optimization problems near to the optimum very efficiently, there are also some major drawbacks in this problem solving approach. At least four of these drawbacks directly relate to areas of *distribution*: complexity, scalability, privacy and robustness. After having briefly introduced the standard CSP model, we will hence motivate the need for a distributed variant of constraint satisfaction and will propose a slightly more detailed definition of a *distributed constraint satisfaction problem* (DCSP, [40]) in section 2.

Since the distribution of problem solvers is often not fully specified by the natural distribution of the problem, a typical engineering task in distributed constraint satisfaction is the design of the distribution itself. As soon as major parts of the problem space are under control of a single realm of authority (as it usually is in information or enterprise resource planning systems) it has to be decided what part of the problem should be assigned to a certain problem solver. More global problem solving promises the highest quality solutions to the common problem but may be infeasible due to technical, social or security restrictions. Distributed problem solving is able to obey social aspects and may diminish problem complexity by partitioning but also entails suboptimal solutions and vast communication overhead. Therefore, the design of distribution (called *configuration*) and the task allocation mechanism is critical to the quality and efficiency of the problem solving process and is itself an optimization problem. We formalize different variants of this problem in section 3 and prove them to be all at least $NP$-complete.

The traditional approach to distributed problem solving is to design the distribution aspects off-line by statically assigning certain roles and competences to specific problem solvers. Thus, the problem space is statically distributed among a more or less fixed set of problem solvers. In section 4 we will present two local operators, *agent melting* and *agent splitting*, that can be combined to allow for an autonomous and dynamic reconfiguration of the organizational structure of the problem solvers. Together with measures for internal problem complexity and communicational effort, the system shall autonomously adapt to the current problem structure by melting and splitting problem solving knowledge, tasks and skills. In this section we will also briefly describe our steps towards the realization of these operators in autonomous multi-agent problem solving. A section on related work (5) and remarks on open questions for future research (6) conclude our report.

# 2 Distributed Constraint Satisfaction

## 2.1 Common CSP Model

Before introducing the distributed model of constraint satisfaction we will present a formal definition of a constraint satisfaction problem which can be found more or less in all CSP articles.

**Definition 1 (Constraint Satisfaction Problem)** *A (general)* constraint satisfaction problem *(CSP) is specified by a triple* $\Pi_{cs} = (X, D, C)$.
$X = \{x_1, \ldots, x_n\}$ *is a set of* domain variables $x_i$ *each ranging over its* domain $D_i$ *from the set* $D = \{D_1, \ldots, D_n\}$. *A* labelling $\lambda = (v_1, \ldots, v_n) \in D_1 \times \ldots \times D_n$ *assigns a value* $v_i \in D_i$ *to each domain variable* $x_i$. $C = \{C_1, \ldots, C_m\}$, $C_i = \{(v_1, \ldots, v_n) \in D_1 \times \ldots \times D_n \mid p(v_{i1}, \ldots, v_{ik})\}$ *is a set of* constraints *each restricting the set of possible labellings by applying a boolean predicate*

$p : (D_{i1} \times \ldots \times D_{ik}) \longrightarrow \{\texttt{true}, \texttt{false}\}$. *Given this specification, the problem is to find a labelling* $\lambda \in \Lambda = C_1 \cap \ldots \cap C_m$ *from the* solution space $\Lambda$.
*A constraint satisfaction problem is called* binary *(*$\Pi_{bcs}$*), iff all constraints* $C_i \in C$ *have the form* $C_i = \{(v_1, \ldots, v_n) \in D_1 \times \ldots \times D_n \mid p(x_j, x_k)\}$, *with* $p : (D_j \times D_k) \longrightarrow \{\texttt{true}, \texttt{false}\}$.

**Definition 2 (Solution Equivalence)** *Two constraint satisfaction problems* $\Pi^1_*$ *and* $\Pi^2_*$ *are called* solution equivalent *(*$\Pi^1_* \equiv \Pi^2_*$*), iff* $\exists(f : \Lambda^1 \longrightarrow \Lambda^2) : f$ *bijective.*

**Remark 1** a) In the definition of binary CSPs $\Pi_{\text{bcs}}$, we do not consider unary constraints since they only restrict the domain of a single domain variable and can hence be satisfied directly by reducing the proper domain. No further processing is needed.
b) Every general CSP $\Pi_{\text{cs}}$ can be transformed into a solution equivalent binary CSP $\Pi_{\text{bcs}}$ by introducing further domain variables [3].

Sometimes it is necessary to illustrate constraint satisfaction problems graphically. Especially binary CSPs are easily represented by ordinary graphs as given by the following definition and example.

**Definition 3 (Constraint Graph)** *A* constraint graph *of a binary CSP* $\Pi_{bcs}$ *is a triple* $\Gamma_{bcs} = (N, E, \mu))$.
$N = \{n_1, \ldots, n_n\}$ *is a set of* nodes *each associated with a domain variable* $x_i$ *from* $\Pi_{bcs}$. $E \subseteq \{\{n_j, n_k\} \mid n_j, n_k \in N\}$ *is a set of* edges *with* $\{n_j, n_k\} \in E \Longleftrightarrow \exists C_l \in C : C_l = \{\lambda \mid p(v_j, v_k)\}$. $\mu : N \cup E \longrightarrow T$ *marks nodes and edges with terms from* $T$ *defined by the predicate calculus underlying* $\Pi_{bcs}$; $\mu(n_i) = (x_i, D_i)$, $\mu(\{n_j, n_k\}) = \bigwedge_{\{\lambda \mid p(v_j, v_k)\} \in C} p(x_j, x_k)$.

**Example 1 (Simple Constraint Satisfaction Problem)**

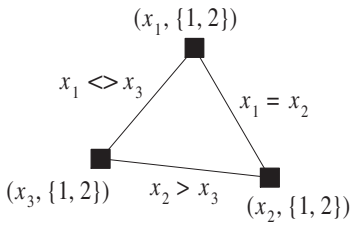


**Figure 1:** Simple CSP

The simple binary constraint satisfaction problem $\Pi_{\text{bcs}} = (X, D, C)$ with $X = \{x_1, x_2, x_3\}$, $D = \{\{1,2\}, \{1,2\}, \{1,2\}\}$ and $C = \{C_1, C_2, C_3\}$, $C_1 = \{(v_1, v_2, v_3) \in \{1,2\} \times \{1,2\} \times \{1,2\} \mid v_1 = v_2\} = \{(1,1,1), (1,1,2), (2,2,1), (2,2,2)\}$, $C_2 = \{(v_1, v_2, v_3) \in \{1,2\} \times \{1,2\} \times \{1,2\} \mid v_2 > v_3\} = \{(1,2,1), (2,2,1)\}$, $C_3 = \{(v_1, v_2, v_3) \in \{1,2\} \times \{1,2\} \times \{1,2\} \mid v_1 \neq v_3\} = \{(1,1,2), (1,2,2), (2,1,1), (2,2,1)\}$ can be illustrated by the constraint graph given by figure 1. The only possible feasible labelling $\lambda$ is determined by $\Lambda = C_1 \cap C_2 \cap C_3 = \{(2,2,1)\} \ni (2,2,1) = \lambda$.

Though it is theoretically possible to reduce every general constraint satisfaction problem to a binary one (as already mentioned), this is not very convenient and the constraint graphs tend to explode. To illustrate general CSPs graphically, we introduce an extended form of constraint graphs that can encode any type of constraints.

**Definition 4 (Extended Constraint Graph)** *An* extended constraint graph *of a general CSP* $\Pi_{cs}$ *is a 4-tuple* $\Gamma_{cs} = (VN, CN, E, \mu)$.
$VN = \{vn_1, \ldots, vn_n\}$ *is a set of* variable nodes *each associated with a domain variable* $x_i$ *from* $\Pi_{cs}$. $CN = \{cn_1, \ldots, cn_m\}$ *is a set of* constraint nodes *each associated with a constraint* $C_i$ *from* $\Pi_{cs}$. $E \subseteq \{\{vn_j, cn_k\} \mid vn_j \in VN \wedge cn_k \in CN\}$ *is a set of* edges *with* $\{vn_j, cn_k\} \in E \Longleftrightarrow C_k = \{\lambda \mid p(\ldots, vn_j, \ldots)\}$. $\mu : VN \cup CN \longrightarrow T$ *marks variable and constraint nodes with terms from* $T$ *defined by the predicate calculus underlying* $\Pi_{cs}$; $\mu(vn_i) = (x_i, D_i)$, $\mu(cn_i) = p(x_{i1}, \ldots, x_{ik}) \Longleftrightarrow C_i = \{\lambda \mid p(v_{i1}, \ldots, v_{ik})\}$.

## 2.2 Motivation for Distribution

Problems of real-world size tend to be too complex to be solved by a monolithic constraint solver. Additionally, monolithic systems often scale poorly in the size of variables and constraints. Partitioning the problem and searching for a near optimal solution composed from the solutions of detached subproblems is a typical approach of parallel computing and more recent distributed computing. But this efficiency and scalability argument is by far not the only one in favor of distributed problem solving. The information on variables and constraints is very often spatially distributed among different natural organizational units. Even in case of a monolithic solver one would have to collect all the information from its several sources, transfer it to the solver, solve the problem and again distribute the results of optimization among the different users. Hence, even in case of a central optimizer one has to cope with communication and information consistency problems. At this point, distribution is a should-have.

Another argument against a central solver is privacy. Social structures, like within enterprises or in supply chains of different enterprises, create heterogeneous fields of competencies and influences. Few executives of organizational units accept transferring all their process data to other organizational units for global optimization. Even less they accept automatic control over their unit's processes by a central instance. For acceptance, there have to be secure interfaces between realms of competency that only let pass authorized and restricted information. Decisions on processes have to be done at the same locations of competency where these processes are enacted in reality. At this point, distribution is already a must-have.

A last argument for distribution is redundancy and robustness. A crash of a central solver or missing connectivity would influence the whole enterprise connected to the solver leading to chaos. Master/slave concepts raise the amount of communication overhead by caching and mirroring. In contrast to that, the crash of a single problem solver in one organizational unit would influence only that unit and its neighbors.

## 2.3 DCSP Model and a Real-World Example

The formal model of distributed constraint satisfaction problems differs from that of ordinary CSPs only by adding a distribution function $\phi$ assigning each variable and constraint to a certain problem-solving agent out of a set of agents $A$.

**Definition 5 (Distributed Constraint Satisfaction Problem)** *A* distributed constraint satisfaction problem *(DCSP) is specified by a 5-tuple* $\Pi_{dcs} = (X, D, C, A, \phi)$.
*$X$, $D$ and $C$ are defined as given by Def. 1.* $A = \{a_1, \ldots, a_p\}$ *is a set of* agents. $\phi : X \cup C \longrightarrow A$ *assigns an agent to each domain variable and each constraint. Again, the problem is to find a labelling* $\lambda \in \Lambda = C_1 \cap \ldots \cap C_m$.

**Remark 2** a) From a declarative view point, a DCSP does not differ from its associated CSP. The difference can only be seen from an operational view point.
b) Since domains are always directly associated with their domain variables, the distribution of domains depends on the distribution of variables and is not discussed further. Hence, according to [27] our approach to distribution can be classified as *variable-based*.
c) $\phi$ defines a partition on the union of domain variables and constraints that can be denoted by $\phi^{-1} = \{(a_1, \phi^{-1}(a_1)), \ldots, (a_p, \phi^{-1}(a_p))\}$ with $\phi^{-1}(a_i) = \{xc \in X \cup C \,|\, \phi(xc) = a_i\}$. Hence, $\bigcup_{a \in A} \phi^{-1}(a) = X \cup C \wedge \forall a_i, a_j \in A, i \neq j : \phi^{-1}(a_i) \cap \phi^{-1}(a_j) = \emptyset$.
d) DSCPs are denoted graphically by using extended constraint graphs together with VENN-diagram-like hulls illustrating the partition. Edges connecting nodes from different partition sets,

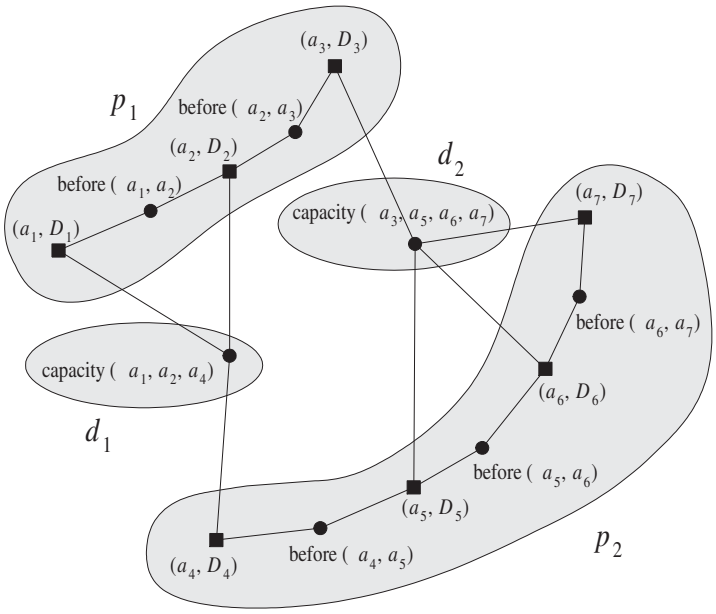


Figure 2: Small Sample of a Medical Appointment Scheduling Problem

i. e. $e = \{vn_i, cn_j\} \wedge \phi(vn_i) \neq \phi(cn_j)$, are called *external edges*. All other edges are called *internal edges*.

**Example 2 (Medical Appointment Scheduling)** In principle, medical appointment scheduling is a slightly extended variant of classical job-shop-scheduling. Primarily, it involves a set of patients each demanding appointments (decision variables) for a set of medical examinations and a set of diagnostic providers each offering a certain set of examination types. Constraints restricting the choice of appointments include partial orders among examinations of patients, capacity constraints defined by the calendars of the providing units and others, e. g. maximum number of examinations each day, setup times and so on. Figure 2 shows an extended constraint graph for a small sample of such a medical appointment scheduling problem. Only partial order and capacity constraints are shown. Two patients $\{a_1, a_2, a_3\}$ and $\{a_4, a_5, a_6, a_7\}$ compete for two common diagnostic resources. The appointments are sequentialized by before-relations. The figure also illustrates the distribution function $\phi$. There are four agents, two patient agents $p_1$ and $p_2$ and two diagnostic unit agents $d_1$ and $d_2$.

## 3 DCS Configuration Problems

Despite the advantages of distribution, such systems have also major disadvantages. The complexity that has been saved within the several solvers is transfered to the coordination process. This is strikingly shown by figure 3 in which ten patients compete for four diagnostic resources (even this example is not of real-life size). Due to this fact, investigations on todays distributed solver systems often report poor optimization results or vast communication overhead. Since global problem solving promises the higher quality solutions to the common problem but may be infeasible due to technical, social or security restrictions, we are facing a well-known trade-off between a global and a local design. Therefore, the design of the distribution (called *configuration*) is itself an

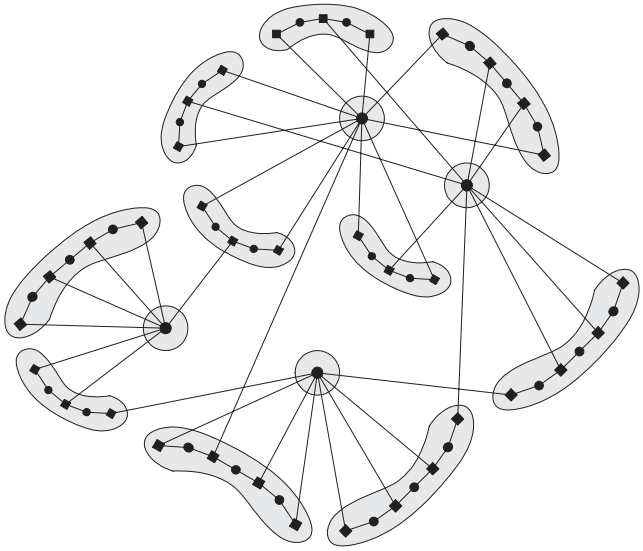

Figure 3: Complex Sample of a Medical Appointment Scheduling Problem

optimization problem. In this section we will formalize the problem of finding a good partition for a given constraint satisfaction problem.

### 3.1 Partitions and Quality

A first and important step towards a formalization of the configuration problem is to define the term "good". To do this, we assume that the quality of a partition is mainly determined by two criteria that are complementary: internal problem complexity and external communication cost. In DCSP solving, internal problem complexity depends on the structure of the sub-CSP to solve, i e. the number of variables, the number of constraints and the complexity of variable-constraint relations. Research in *phase transitions* [20, 11, 16] defines a notion of *constrainedness* that promises to be a good measure for identifying the complexity of a given sub-CSP. In this report, we simply assume the existence of a function $\kappa : 2^{X \cup C} \longrightarrow \mathbb{N}$ that assigns a complexity value to a subset of all variables and constraints. By external communication cost we do not denote the physical time for transmitting a message from one problem solver to another, but we denote the semantic effort of a whole consensus finding protocol run as presented in [12] to ensure consensus between two autonomous problem solvers. Nevertheless, we assume this effort to be directly related to a very simplistic measure: the number of external edges in the extended constraint graph of the DCSP.

The target of a good partition is to increase internal complexity as far as possible to utilize the full power of conventional CSP solving within the agent and such to achieve high-quality solutions. Additionally, a good partition has minimal external communication costs. All this is formalized by the following definition.

**Definition 6 (Quality of a Partition)** *Given a partition* $\phi^{-1} = \{(a_1, X_1 \cup C_1), \ldots, (a_k, X_k \cup C_k)\}$ *and an internal complexity function* $\kappa : 2^{X \cup C} \longrightarrow \mathbb{N}$ $(X = \bigcup_{i=1,\ldots,k} X_i, C = \bigcup_{i=1,\ldots,k} C_i)$*, the*

quality *of* $\phi^{-1}$ *is defined by*

$$q_{\phi^{-1}} = \sum_{i=1}^{k} \left[ \overbrace{\kappa\left(X_i \cup C_i\right)}^{\textit{internal complexity}} - \overbrace{\sum_{C' \in C_i} \left| \left\{ x_j \in X \mid C' = \{\lambda | p\left(\ldots, v_j, \ldots\right)\} \wedge \phi\left(x_j\right) \neq \phi\left(C'\right) \right\} \right|}^{\textit{external communication cost}} \right]. \quad (1)$$

The best partition in this sense is the one containing only one set including the whole initial constraint satisfaction problem. It has the highest possible internal complexity and no external communication costs at all. That directly corresponds to our intuition to solve a CSP problem centrally if only possible. Alas, following our argumentation in favor of distributed problem solving this is not always possible. Therefore, we have to encode the restrictions we used to argue in favor of DCSPs as it is done in the next subsection.

### 3.2 Core Problem and Variants

The main restrictions that do not allow us to solve many CSP centrally are the boundedness of tractable complexity and social boundaries. We will encode these restrictions by an upper bound $\kappa_u$ on the internal complexity perhaps found by empirical studies on the given CSP class and the careful attention to the social competency boundaries. Both are formalized in the following definition.

**Definition 7 (**DCS Configuration **Problem)** *A static* DCS Configuration problem *is specified by a 5-tuple* $(\Pi_{cs}, \kappa, \kappa_u, \kappa_l, \phi_s)$.
$\Pi_{cs} = (X, D, C)$ *is a constraint satisfaction problem,* $\kappa : 2^{X \cup C} \longrightarrow \mathbb{N}$ *is an internal complexity function,* $\kappa_u$ *is an internal complexity upper bound,* $\kappa_l$ *is an internal complexity lower bound and* $\phi_s$ *defines a social partition. Additionally, let*

$$Q = \Big\{ q_{\phi^{-1}} \;\Big|\; \Pi_{cs} \equiv \Pi_{dcs} = (X, D, C, A, \phi) \quad \wedge \quad (2)$$

$$\forall (a_i, X_i \cup C_i) \in \phi^{-1} : (\kappa_l \leq \kappa\left(X_i \cup C_i\right) \leq \kappa_u) \quad \wedge \quad (3)$$

$$\forall xc_i, xc_j \in X \cup C : \phi_s\left(xc_i\right) \neq \phi_s\left(xc_j\right) \Longrightarrow \phi\left(xc_i\right) \neq \phi\left(xc_j\right) \Big\}. \quad (4)$$

*Then, three variants of the static* DCS Configuration problem *are*

1. *Is* $q^* = \max Q \geq k$ *for a given* $k \in \mathbb{Z}$*? (decision problem* $\Pi_{\text{CON}}$*)*
2. *Find* $q^*$*! (value optimization problem* $\omega_{\text{CON}}$*)*
3. *Find* $\Pi^*_{dcs} = arg\ max Q$*! (solution optimization problem* $\Omega_{\text{CON}}$*)*

The set $Q$ includes the quality measures of all possible DCSPs that are solution equivalent to the given CSP. The following restrictions are put onto these DCSPs: (2) ensures that the DCSPs preserve the set of variables $X$, the set of domains $D$ and the set of constraints $C$. Hence, the only free variables of the DCS Configuration problem are the set of agents $A$ used to solve the given CSP and the distribution function $\phi$. (3) ensures that the complexity of every set of the partition defined by $\phi$ lies within the bounds given by $\kappa_u$ and $\kappa_l$. This property is called *complexity feasibility* and is denoted by the predicate $feasible_c$. Finally, (4) enforces the given social partition defined by $\phi_s$ to be preserved by the found partition defined by $\phi$, such that no two variables or constraints that are desired to be in different social partition sets are in the same partition set. This property is called *social feasibility* and is denoted by the predicate $feasible_s$.

**Theorem 1** *The decision variant of* DCS CONFIGURATION $\Pi_{\text{CON}}$ *is NP-complete in the size of* $|X \cup C|$.

**Proof:**
a) ($\Pi_{\text{CON}} \in NP$) $\Pi_{\text{CON}}$ is in $NP$ because it can be solved by enumerating every possible partition $\phi^{-1}$ of $X \cup C$ (which are $B_{n=|X\cup C|} = \sum_{k=1}^{n} \left[\frac{1}{k!}\sum_{i=0}^{k-1}(-1)^i \binom{k}{i}(k-1)^n\right]$ many[1]) and testing $q_{\phi^{-1}} \geq k$ (which can be done in polynomial time as long as $\kappa$ is computable in polynomial time).
b) ($NP$-hardness of $\Pi_{\text{CON}}$) We reduce the MINIMUM BISECTION graph problem to $\Pi_{\text{CON}}$. The MINIMUM BISECTION graph problem is proven to be $NP$-complete [10] and is defined as follows. Given a graph $\Gamma = (N, E), |N| = 2 \cdot i, i \in \mathbb{N}^+$. Let

$$M = \left\{ m = |\{\{i,j\} \in E \mid n_i \in N_1 \wedge n_j \in N_2\}| \;\middle|\; N_1 \cup N_2 = N \wedge N_1 \cap N_2 = \emptyset \wedge |N_1| = |N_2| \right\}. \quad (5)$$

Is $m^* = \min M \leq k$ for a given $k \in \mathbb{Z}$?
We will do the reduction in a three step process.

1. (Input) We transform the input of the MINIMUM BISECTION problem to an input of $\Pi_{\text{CON}}$ by $X = N$ and $C = \{\{(v_i, v_j) \mid p(v_i, v_j)\} \mid \{i,j\} \in E\}$. Hence, there are only binary constraints in $C$. The domain set $D$ is without any importance to this reduction and can be left empty.

2. (Restrictions) According to remark 2b) $\phi$ defines a partition. Additionally, defining $\kappa(X_i \cup C_i) = |X_i|$ and $\kappa_u = \frac{|N|}{2} = \kappa_l$ enforces $\phi^{-1}$ to be a bisection with equally sized parts, since

$$\begin{aligned} & \forall (a_i, X_i \cup C_i) \in \phi^{-1} : \left(\kappa(X_i \cup C_i) \leq \tfrac{|N|}{2} \wedge \kappa(X_i \cup C_i) \geq \tfrac{|N|}{2}\right) \\ \Longrightarrow\ & \forall (a_i, X_i \cup C_i) \in \phi^{-1} : |X_i| = \tfrac{|N|}{2} \\ \Longrightarrow\ & |\phi^{-1}| = 2 \ \wedge\ |X_i| = |X_j| = \tfrac{|N|}{2} \qquad \text{(since } |N| = \textstyle\sum_{(a_i, X_i \cup C_i) \in \phi^{-1}} |X_i|\text{)} \end{aligned}$$

   Finally, disabling the social partition by $\forall xc \in X \cup C : \phi_s(xc) = 1$ leads to a full reduction of the set restrictions of $M$ [(5)] to the set restrictions of $Q$ [(2)–(4)].

3. (Measure) The last thing to do is to reduce the measure $m$ of the MINIMUM BISECTION problem [(5)] to the measure of $\Pi_{\text{CON}}$ $q_{\phi^{-1}}$ [(1)].

$$\begin{aligned} m &= |\{\{j,k\} \in E \mid n_j \in N_1 \wedge n_k \in N_2\}| \\ &= |\{\{j,k\} \in E \mid \phi(x_j) \neq \phi(x_k)\}| && \text{(see restrictions)} \\ &= |\{\{\lambda | p(v_j, v_k)\} \in C \mid \phi(x_j) \neq \phi(x_k)\}| && \text{(see def. of } C\text{)} \\ &= \textstyle\sum_{(a_i, X_i \cup C_i) \in \phi^{-1}} \sum_{C' \in C_i} |\{C' = \{\lambda | p(v_j, v_k)\} \mid \phi(x_j) \neq \phi(x_k)\}| && \text{(altern. counting)} \end{aligned}$$

   What we are doing here is to check every constraint $C' \in C$ whether it restricts variables that are in different sets of the partition $\phi^{-1}$. Since we have only two sets in the partition and constraints are only binary, there are three possible cases for the relation among a constraint $C'$ and its restricted variables $x_j$ and $x_k$: $\phi(C') = \phi(x_j) \neq \phi(x_k)$, $\phi(C') = \phi(x_k) \neq \phi(x_j)$ or $\phi(C') = \phi(x_j) = \phi(x_k)$. In the first two cases, $m$ is increased by 1. Alternatively, we can count the number of nodes connected to a constraint $C'$ that are not in the same partition

[1] $B_n$ is the so-called *Bell*-number.

set as the constraint, because (looking at the three cases) there is exactly one such node or none, leading also to an increase of $m$ by 1 in the first two cases. Hence, we can derive

$$\begin{aligned} m &= \sum_{(a_i, X_i \cup C_i) \in \phi^{-1}} \sum_{C' \in C_i} |\{ x_j \in X \mid C' = \{\lambda | p(\ldots, v_j, \ldots)\} \wedge \phi(x_j) \neq \phi(C')\}| \\ &= \sum_{(a_i, X_i \cup C_i) \in \phi^{-1}} \kappa(X_i \cup C_i) - q_{\phi^{-1}} \\ &= \sum_{(a_i, X_i \cup C_i) \in \phi^{-1}} |X_i| - q_{\phi^{-1}} \\ &= |N| - q_{\phi^{-1}} \end{aligned}$$

Therefore, $m$ has been successfully reduced to an equivalent term containing a constant and the inverse of $q_{\phi^{-1}}$. The constant can be removed by adjusting $k \in \mathbb{Z}$ properly, the negative sign is indeed correct and needed because $\Pi_{\text{CON}}$ maximizes $Q$ while MINIMUM BISECTION minimizes $M$.

This polynomial three step procedure reduces MINIMUM BISECTION to $\Pi_{\text{CON}}$ and hence proves our proposition. □

**Corollary 1** *The optimization variants of* DCS CONFIGURATION *are NP-hard.*

Though the DCS CONFIGURATION problem is itself interesting, it is too static and centralized in nature to be fully suitable for what we are behind in multi-agent systems research. Hence, we propose an extended version of this problem that describes the task to dynamically adapt to the given CSP structure.

**Definition 8 (**DCS RECONFIGURATION **Problem)** *A dynamic* DCS RECONFIGURATION problem *is specified by a 5-tuple* $(\Pi^o_{dcs}, \kappa, \kappa_u, \kappa_l, \phi_s)$.
$\Pi^o_{dcs} = (X, D, C, A^o, \phi^o)$ *is a distributed constraint satisfaction problem,* $\kappa : 2^{X \cup C} \longrightarrow \mathbb{N}$ *is an internal complexity function,* $\kappa_u$ *is an internal complexity upper bound,* $\kappa_l$ *is an internal complexity lower bound and* $\phi_s$ *defines a social partition. Additionally, let* $\Pi^o_{dcs}$ *be complexity and socially feasible and*

$$Q = \Big\{ q_{\phi^{-1}} \;\Big|\; \Pi^o_{dcs} \equiv \Pi^n_{dcs} = (X, D, C, A^n, \phi^n) \;\wedge\; \mathit{feasible}_c(\Pi^n_{dcs}, \kappa, \kappa_u, \kappa_l) \;\wedge\; \mathit{feasible}_s(\Pi^n_{dcs}, \phi_s) \Big\}.$$

*Then, three variants of the dynamic* DCS RECONFIGURATION problem *are*

1. *Is* $q^* = \max Q \geq k$ *for a given* $k \in \mathbb{Z}$*? (decision problem* $\Pi_{\text{RECON}}$*)*
2. *Find* $q^*$*! (value optimization problem* $\omega_{\text{RECON}}$*)*
3. *Find* $\Pi^*_{dcs} = arg\ max Q$*! (solution optimization problem* $\Omega_{\text{RECON}}$*)*

**Remark 3** a) An interesting special case of the decision variant $\Pi_{\text{RECON}}$ is $q^* = \max Q \geq k = q_{\phi^{o-1}}$?, i. e. whether the optimal partition quality is greater than the partition quality defined by the given DCSP $\Pi^o_{\text{dcs}}$.
b) Of course, the decision variant of this problem is again $NP$-complete. This can be proven similarly to theorem 1 by reducing MINIMUM BISECTION to this problem with the help of a trivial initial bisection (for example $X_1 \cup C_1 = \{x_1, \ldots, x_{\frac{|V|}{2}}\} \cup C$ and $X_2 \cup C_2 = \{n_{\frac{|V|}{2}}, \ldots, n_{|V|}\}$).

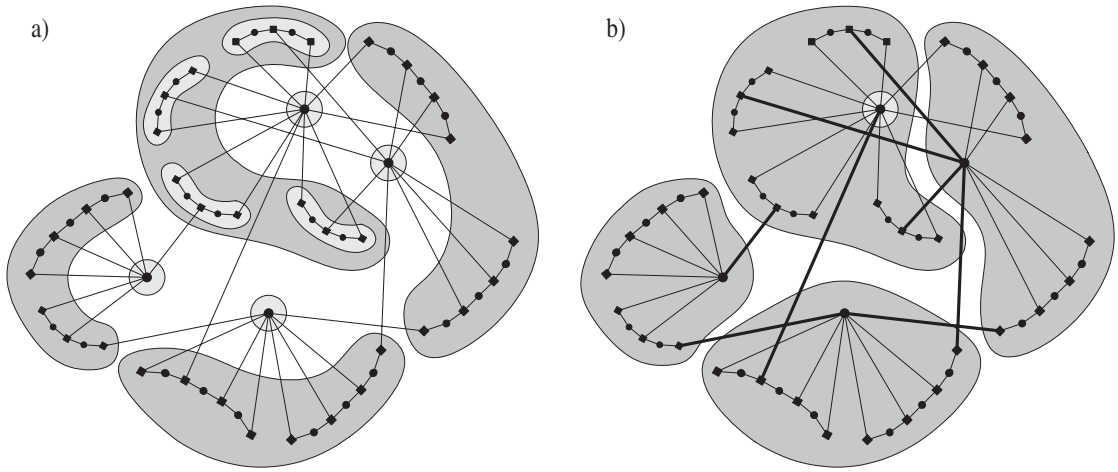


Figure 4: Results of Agent Melting

# 4 Solving DCS Configuration Problems

In subsection 2.2 we have argued in favor of distribution to solve realistic constraint satisfaction problems. The same arguments hold for solving a given DCS Configuration problem and its variants. For example, if our system of problem solvers underlies a certain social competence structure, the configuration of the problem solvers won't be influencable by a central instance. Due to this, we need reconfiguration operators that allow for local and potentially autonomous adaption to the problem structure.

Please take another look at figure 3. Ten patient agents have to communicate with four resource agents to reach consensus. Now, figure 4 shows the results of a local operator, called *agent melting*. In part a) patient agents that are strongly connected with a single resource agent have been unified to a single patient group agent. Though this has the advantage that requests of several patients can be bundled and sent collectively to the resource agent, external communication has not decreased substantially. Part b) shows a further step of local reconfiguration in which resource agents have been melt with patient group agents that often use their resource. As can be seen by the thick lines, this step significantly reduces the external communication overhead. In the next subsections we will formalize a minimal set of local operators that will prove to be sufficient to solve any given DCS Reconfiguration problem and will briefly present steps towards getting this theoretical result to practice.

## 4.1 Local Operators and their Sufficiency

The local operator introduced by the aforementioned example is formalized by the following definition.

**Definition 9 (Agent Melting)** *Given a set of agents* $A = \{a_1, \ldots, a_p\} : \mathcal{A}$*, a partition* $\phi^{-1} = \{(a_1, X_1 \cup C_1), \ldots, (a_p, X_p \cup C_p)\} : \Phi$ *of a domain variable and constraint set* $X \cup C$ *and two indices* $i, j : \mathbb{N}$*,* agent melting *is a function* $\mu : (\mathcal{A} \times \Phi) \times \mathbb{N} \times \mathbb{N} \longrightarrow \mathcal{A} \times \Phi$ *defined by*

$$\mu\left((A, \phi^{-1}), i, j\right) = \begin{cases} \Big(A \setminus \{a_j\}, (\phi^{-1} \setminus \{(a_i, XC_i), (a_j, XC_j)\}) \cup \{(a_i, XC_i \cup XC_j)\}\Big), & a_i, a_j \in A \\ (A, \phi^{-1}), & \textit{else} \end{cases}$$

Of course, to fulfill all demands on complexity and social feasibility of a problem solver configuration, agent melting alone is not sufficient. As soon as one agent emerges to have a too high internal complexity it has to be split into two (or more) smaller agents. The complementary local operator to agent melting is called *agent splitting* and is defined as follows.

**Definition 10 (Agent Splitting)** *Given a set of agents* $A = \{a_1, \ldots, a_p\} : \mathcal{A}$*, a partition* $\phi^{-1} = \{(a_1, X_1 \cup C_1), \ldots, (a_p, X_p \cup C_p)\} : \Phi$ *of a domain variable and constraint set* $X \cup C$*, an index* $i : \mathbb{N}$ *and a subset* $XC' : 2^{X \cup C}$ *of domain variables and constraints,* agent splitting *is a function* $\sigma : (\mathcal{A} \times \Phi) \times \mathbb{N} \times 2^{X \cup C} \longrightarrow \mathcal{A} \times \Phi$ *defined by*

$$\sigma\left((A, \phi^{-1}), i, XC'\right) = \begin{cases} \Big(A \cup \{a_{|A|+1}\}, (\phi^{-1} \setminus \{(a_i, XC_i)\}) \cup \{(a_i, XC_i \setminus XC'), (a_{|A|+1}, XC')\}\Big), & a_i \in A \wedge \\ & XC' \subset XC_i \\ (A, \phi^{-1}), & else \end{cases}$$

**Remark 4** To simplify the notation of consecutive application of these local operators to the same initial set of agents $A$ and partition $\phi^{-1}$, we will use the following abbreviations. Applying agent melting of agents $i$ and $j$ to $(A, \phi^{-1})$ is denoted by $(i, j)(A, \phi^{-1})$. Applying agent splitting of agent $i$ with subset $XC$ to $(A, \phi^{-1})$ is denoted by $(i, XC)(A, \phi^{-1})$. For further abbreviation, we write $(\vec{\imath}, \vec{\jmath})^m(A, \phi^{-1})$ for $(i_1, j_1) \ldots (i_m, j_m)(A, \phi^{-1})$. Hence, $((\vec{\imath}, \vec{\jmath})^2(\vec{k}, \vec{XC}))(A, \phi^{-1})$ denotes $\sigma\left(\mu\left(\mu\left((A, \phi^{-1}), i_1, j_1\right), i_2, j_2\right), k_1, XC_1\right)$.

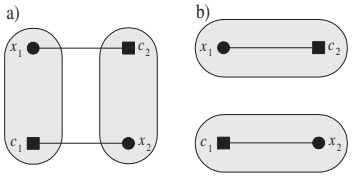


**Figure 5:** Counter-example

The given definitions of the local operators do not obey complexity and social feasibility. These operators are not directly applicable to DCS Reconfiguration problems, since they may only lead to configurations that are not feasible. This can easily be seen by the example given by figure 5. Let's assume that $\kappa(X \cup C) = |X \cup C|, \kappa_u = 2, \kappa_l = 2$. Though this may seem very restrictive, we have used a similar restriction in our $NP$-completeness proof. Part a) of the figure shows the given configuration and part b) shows the optimal one. Neither agent melting nor agent splitting is solely applicable to the configuration given by a), because both operators violate the demanded complexity restrictions. Therefore, we cannot build our reconfiguration strategy directly on the local operators. We need an extended atomic operator that is able to obey complexity and social feasibility. We call this atomic operator *reconfiguration transaction* and specify it as follows.

**Definition 11 (Reconfiguration Transaction)** *Given a complexity and social feasible dynamic* DCS Reconfiguration *problem* $*_{\text{ReCon}} = ((X, D, C, A, \phi), \kappa, \kappa_u, \kappa_l, \phi_s), A : \mathcal{A}, \phi : \Phi$*, a* reconfiguration transaction *is a function* $\rho : \mathcal{A} \times \Phi \longrightarrow \mathcal{A} \times \Phi$ *specified by*

$$(A', \phi^{-1\prime}) = \rho(A, \phi^{-1}) = (\vec{\imath}, \vec{\jmath})^m(\vec{k}, \vec{XC})^n(A, \phi^{-1}),$$
$$\vec{\imath}, \vec{\jmath} \in (\{1, \ldots, |X \cup C|\})^m, \vec{k} \in (\{1, \ldots, |X \cup C|\})^n, \vec{XC} \in (2^{X \cup C})^n, m, n \in \mathbb{N}_0$$

*such that*

$$\mathit{feasible}_c\left((X, D, C, A', \phi'), \kappa, \kappa_u, \kappa_l\right) \wedge \mathit{feasible}_s\left((X, D, C, A', \phi'), \phi_s\right)$$

To solve the example illustrated by figure 5, only a kind of exchange operator would be sufficient. Fortunately, with the help of the above specified reconfiguration transaction we can emulate any

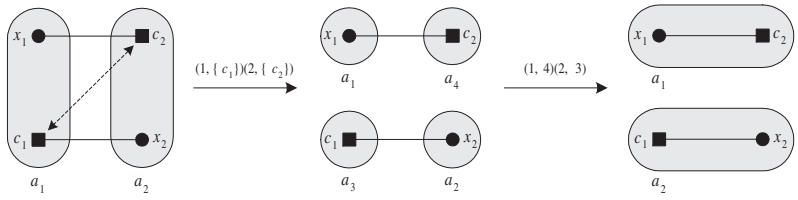


Figure 6: An Exchange Transaction composed from Local Operators

other feasible local operator, like exchange and transfer. Figure 6 shows how a reconfiguration transaction composed of four local splits and melts can be used to emulate an exchange and solve the given example.

Though we may already have an intuition on the strength of reconfiguration transactions, the following theorem lays a theoretical foundation for the sufficiency of this concept to solve DCS RECONFIGURATION problems.

**Theorem 2 (Sufficiency of Reconfiguration Transactions)** *Given a complexity and social feasible* DCS RECONFIGURATION *problem* $*_{\text{RECON}} = ((X, D, C, A, \phi), \kappa, \kappa_u, \kappa_k, \phi_s)$. *Assuming that* $A$ *and* $\phi$ *do not influence the solution equivalence between two different DCSPs, i. e.* $\forall A_1, A_2 \in \mathcal{A}, \phi_1, \phi_2 \in \Phi : (X, D, C, A_1, \phi_1) \equiv (X, D, C, A_2, \phi_2)$, $*_{\text{RECON}}$ *can be solved by finding a certain sequence*

$$((\vec{i}_1, \vec{j}_1)^{m_1}(\vec{k}_1, \vec{XC}_1)^{n_1}) \dots ((\vec{i}_p, \vec{j}_p)^{m_p}(\vec{k}_p, \vec{XC}_p)^{n_p})$$

*of reconfiguration transactions applied to* $(A, \phi^{-1})$.

**Proof:**
Since we are assuming that the solution equivalence between any pair of DCSPs does not depend on $(A, \phi)$, it is enough to show that we can systematically enumerate all complexity and social feasible partitions of the underlying CSP with the help of a sequence of reconfiguration transactions.
a) (Correctness) In our context, correctness means that only complexity and social feasible partitions are created by a sequence of reconfiguration transactions. This is true by definition 11, since every single reconfiguration transaction of the sequence takes a complexity and social feasible partition as input and computes a partition that is forced to be complexity and social feasible.
b) (Completeness) In our context, completeness means that (1.) every given complexity and social feasible partition can be produced and that (2.) all partitions can be found by systematically enumerating sequences of reconfiguration transactions.

1. Assume that we want to produce the (complexity and social feasible) partition $\phi_2^{-1} = ((a_1, XC_{2,1}), \dots, (a_n, XC_{2,n}))$ given the (complexity and social feasible) partition $\phi_1^{-1} = ((a_1, XC_{1,1}), \dots, (a_m, XC_{1,m}))$. This can be done by the following reconfiguration transaction

$$(A_2, \phi_2^{-1}) = \underbrace{(1,2)(1,3)\dots(1,m)}_{\text{melt all agents}} \underbrace{(1, XC_{2,1})(1, XC_{2,2})\dots(1, XC_{2,n})}_{\text{split agent } a_1 \text{ according to } \phi_2^{-1}} (A_1, \phi_1^{-1})$$

   Note that this may be only one (and not very clever) reconfiguration transaction of several possible. Nevertheless, this reconfiguration transaction is invariant to $(A_1, \phi_1^{-1})$ as long as $a_1 \in A_1$, i. e. $(A_2, \phi_2^{-1})$ can be produced from every given partition.

2. Given any initial partition $(A, \phi^{-1})$, the following algorithm realizes a systematic enumeration of all possible reconfiguration transactions and incrementally builds the sequence of successful reconfiguration transactions (*sequence*) and the set of feasible partitions ($partitions_f$) and non-feasible partitions ($partitions_{\neg f}$).

{initialization}
$sequence \leftarrow ()$; (1)
$(A_c, \phi_c^{-1}) \leftarrow (A, \phi^{-1})$; (2)
$partitions_f \leftarrow \{(A_c, \phi_c^{-1})\}$; (3)
$partitions_{\neg f} \leftarrow \emptyset$; (4)

{enumeration scheme}
**while** $|partitions_f| + |partitions_{\neg f}| < B_{|X \cup C|}$ **do**
{compute new partition}
$(\vec{i}, \vec{j}, \vec{k}, \vec{XC}, m, n) \leftarrow (\perp, \perp, \perp, \perp, 0, 0)$; (5)
**while** $(A, \phi^{-1}) \in partitions_f \vee (A, \phi^{-1}) \in partitions_{\neg f}$ **do**
{compute and apply new local operator sequence from the previous one}
$(\vec{i}, \vec{j}, \vec{k}, \vec{XC}, m, n) \leftarrow$ `generate_new_parameters` $(\vec{i}, \vec{j}, \vec{k}, \vec{XC}, m, n)$; (6)
$(A, \phi^{-1}) \leftarrow (\vec{i}, \vec{j})^m (\vec{k}, \vec{XC})^n (A_c, \phi_c^{-1})$; (7)
**end while**

{test feasibility of new partition}
**if** $feasible_c((X, D, C, A, \phi), \kappa, \kappa_u, \kappa_l) \wedge feasible_s((X, D, C, A, \phi), \phi_s)$ **then**
{local operator sequence is a valid reconfiguration transaction}
$sequence \leftarrow sequence : (\vec{i}, \vec{j}, \vec{k}, \vec{XC}, m, n)$; (8)
$(A_c, \phi_c^{-1}) \leftarrow (A, \phi^{-1})$; (9)
$partitions_f \leftarrow partitions_f \cup \{(A, \phi^{-1})\}$; (10)
**else**
$partitions_{\neg f} \leftarrow partitions_{\neg f} \cup \{(A, \phi^{-1})\}$; (11)
**end if**
**end while**

After initializing the reconfiguration transaction sequence, the current configuration and the partition sets (lines 1 to 4), the algorithm passes two loops. The outer loop checks whether all possible partitions have been enumerated (this is done by checking the cardinality of the two partition sets against the *Bell*-number). If not, we have to compute a new partition by initializing the search for a new local operator sequence (line 5) and running the inner loop. Using the previous parameter set, we compute a new parameter set (line 6) and apply the local operator sequence to the current partition (line 7).

So why can this inner loop produce all possible partitions? This is because for every partition there is at least one local operator sequence producing the partition regardless of the given recent partition (see 1.). Hence, if we are able to enumerate all possible local operator sequences every partition will eventually appear. It is the responsibility of the function `generate_new_parameters` to ensure that all possible local operator sequences are enumerated. The parameters of a local operator sequence $i_r, j_r, k_r, XC_r$ are from finite sets, because they are all restricted by the given fixed number of domain variables ($X$) and constraints ($C$). Even $m$ and $n$ are restricted to be from a finite set because we cannot apply an arbitrary number of effective agent melting operator to a set of agents as well as we cannot split agents

infinitely. Therefore, the set of possible local operator sequences is finite and well-structured, i. e. we can enumerate it by trying one melt with all parameter combinations without any splits, one split with all parameter combinations without any melts, one melt and one split and so on.

After having found a new partition, it is checked whether we have found a valid reconfiguration transaction and such a complexity and social feasible partition. If yes, our sequence of reconfiguration transactions is augmented, the current configuration is set to the found one and the partition is stored in the set of feasible partitions (lines 8 to 10). If not, the partition is stored in the set of non-feasible partitions (line 11) and the sequence of transactions and the current configuration remain unchanged.

Following the described algorithm, we can derive all possible feasible partitions from the given partition by consecutively applying the reconfiguration transactions from the computed sequence. Hence, we can easily solve the decision variant, the value optimization variant and the solution optimization of the given DCS Reconfiguration problem. □

Of course, the aforementioned theorem is only a theoretical result. The presented algorithm to prove sufficiency of reconfiguration transactions is neither realistic nor efficient. In fact, it is much less efficient than simply enumerating all possible partitions, since there are many reconfiguration transactions leading to partitions already stored and there are even more local operator sequences that are no valid reconfiguration transactions. Nevertheless, the theorem is the theoretical foundation for allowing us to seek in the space of reconfiguration transactions rather than in the space of partitions to solve a DCS Reconfiguration problem. This can be exploited to build more realistic algorithms as briefly introduced in the next subsection.

## 4.2 Using the Local Operators autonomously

The algorithm used to prove the potential completeness of sequences of reconfiguration transactions is in fact a global one, since it employs a centrally controlled function `generate_new_parameters` to systematically enumerate all possible reconfiguration transactions. This is bad for two reasons: first, we initially wanted to avoid a global algorithm to solve DCS Reconfiguration problems; second, the way of enumerating all possible transactions is far from being efficient. Nevertheless, the local character of reconfiguration transactions remains and we can modify their use to build a real autonomous approach to DCS Reconfiguration solving.

Real autonomy means in this context to allow arbitrary concurrent reconfiguration transactions to be initiated by the problem-solving agents themselves by using knowledge-based heuristics. This yields the potential to be much more efficient than centrally and sequentially scanning all possible transactions, but also entails the danger of instability, loops and the loss of completeness though not of correctness. We claim this to be a general rule in optimization: to gain significant speed-up in efficiency by exploiting parallelism or distribution you have usually to abandon completeness. This is, because to reach completeness in distributed solving of highly interconnected problems the problem solvers have to be coordinated to explore the full operator space. Coordination algorithms used to do this often contain some kind of hidden synchronization or even sequentialization (refer e. g. to the prioritization scheme of Yokoo et al. [40]), such that the speed-up of distribution is destroyed at once.

The same argumentation holds for solving DCSPs. In our opinion, it is suitable to use complete algorithms for solving intra-agent problems and incomplete cooperation protocols for solving inter-agent problems. This also emphasizes the need to minimize external communication demand

while maximizing internal complexity to find better solution to the given problem. Unfortunately, allowing inter-agent problem solving to be incomplete directly violates our assumption in theorem 2 that configuration does not influence solution equivalence. One way to cope with this structural problem is to resign solution equivalence in the definition of the DCS CONFIGURATION problem and its variants and replace it by a weaker notion of *solution reducibility*. In words, solution reducibility means to resign completeness but to keep correctness.

**Definition 12 (Solution Reducibility)** *A constraint satisfaction problem* $\Pi^2_*$ *is called* solution reducible *to another constraint satisfaction problem* $\Pi^1_*$ *(*$\Pi^2_* \leq \Pi^1_*$*), iff* $\exists(f : \Lambda^2 \longrightarrow \Lambda^1)$ : *f injective.*

The influence of using this weaker notion to the formal foundation of autonomous dynamic reconfiguration is subject to further research. Nevertheless, we are currently evaluating empirically the autonomous usage of reconfiguration transactions in the so-called AURECON project. This project involves the development of concepts and realizations in the fields of agent technology and constraint-based optimization. A central concept are *composable agents* that can freely exchange their knowledge, targets and skills to support reconfiguration (see [15] and [32]). Another important part of AURECON is the development of an incomplete but very efficient consensus finding protocol for inter-agent cooperation (see [12]). For intra-agent problem solving we are using *constraint logic programming* that is a very declarative form of programming languages and is hence well-suited for information exchange among agents ([14], [13]).

# 5 Related Work

## 5.1 DCSP Solving

An excellent, though a little out-dated overview to DCSP models and algorithms is given in [27] and [24]. They present an interesting classification of DCSP solving approaches by distinguishing variable-based approaches (in which every agent cares for a subset of variables), domain-based approaches (in which ever agent cares for a subset of values for a unique variable) and function-based approaches (in which costly computations in centralized CSP solving are distributed to speed them up). Following this classification our approach is variable-based. In [24, 26, 25] the authors propose different algorithms to solve DCSPs variable-based, domain-based and function-based. They all assume a binary DCSP and are hence based on simple constraint representations via no-good-sets.

An important contribution to DCSP solving has been given by Sycara, Roth, Sadeh and Fox in [38] in which they present *Distributed Constrained Heuristic Search*. They characterize the design trade-off for a proper level of distribution in a system for a given communication bandwidth, but do not address this problem in the paper. Sycara et al. characterize the effect of different decompositions and their characteristics to be a subject of future research.

Being another classical reference in DCSP, the work of Yokoo and Ishida introduces a DCSP model that simply assigns the variable nodes of a binary CSP graph to the different agents. Hence, this is a variable-based approach. Their main contribution lies in the development of distributed search algorithms, like *asynchronous backtracking* and *asynchronous weak-commitment search*. The earlier versions (collectively presented in [40] and [42]) relied on the assumption, that every agents cares for just one variable. Newer versions [41] overcome this restriction by allowing complex local problems. All these algorithms are correct and complete. To coordinate the different forms of

asynchronous backtracking, the algorithms establish a static or a dynamic order among agents that determines the cooperations patterns between agents. Unfortunately, the assumption of simple binary constraints restricts the applicability in real-world settings.

Extending Yokoo's asynchronous weak-commitment Mammen and Lesser [29] investigate the impact of problem structure to the solving process. To do this, they propose a parameterized problem generator for DCSPs and implement a MAS simulator. Based on simulated runs they draw conclusion on the interdependence of structure and solvability. Though providing useful insight in random problem generation and simulation their approach does not allow to assign constraints to agents.

Another approach to DCSP solving does not try to solve the DCSP with new distributed propagation or search methods but to facilitate existing CSP solvers to solve the problems local to an agent and then to combine the results of these solvers. An early reference on this approach is [4]. Berlandier and Neveu introduce the notion of *interface problems* by partitioning a DCSP along variable nodes and not as usual along constraint arcs. All variable nodes that belong to more than one agent form a new problem — the interface problem. The variable nodes not belonging to the interface problem can be labeled independently from other variable nodes. Such, solving the interface problem and then solving the independent problems eventually using backtracking solves the whole problem. A disadvantage is the need for a global instance for finding the solution to the interface problem and collecting the solutions of the independent problems. Solotorevsky and others [37] follows a similar strategy by defining *canonical DCSPs* which consists of a special constraint graph connecting all independent local constraint graphs. Similar to Berlandier and Neveu they use common CSP solvers to solve the partitioned problems. All these authors assume a given partitioning of the DCSP and facilitate a global instance for guiding the solving process.

## 5.2 Graph Partitioning

Our DCS Configuration problem and its variants are strongly related to the problem of partitioning graphs, as can be seen in the reduction of Minimum Bisection to our problem. It is not surprising that graph partitioning algorithms are quite often the basis for load balancing in parallel computing. Hendrickson and Leland [18] provide a good overview on static graph partitioning algorithms. They include greedy algorithms like the famous one by Kernighan and Lin [23] and extensions of it [6], so-called *spectral methods* [2, 5, 1] and other hybrid approaches. Several of these algorithms have been implemented and integrated into tools that can be used as off-line pre-processors to partitioning a central computing problem. Among these tools are Chaco [19], METIS [22] and JOSTLE [39]. Recent research also tackles the problem of dynamic load balancing. Hendrickson and Devine [17] assess different approaches to this problem.

Though the problem of load balancing in parallel computing is similar in its objectives to our research context, the given environmental situation is usually quite different. In parallel computing there is a central problem that can easily be decomposed into equally sized subproblems which can be solved nearly independently. Hence, the task of load balancing is to find a good distribution of these decomposed subproblems to processors such that communication is minimized. In our setting, decomposition itself is the problem. Subproblems are not easy to find and have to be constructed such that interrelations are minimized. These interrelations are not just communication lines but semantic connections, like shared resources or other social constraints. Nevertheless, results from load balancing, especially heuristics are promising to be applied to our autonomous dynamic reconfiguration approach.

### 5.3 Agent Technology for Reorganization

In [9] Durfee reports on various approaches to organizational structuring in distributed problem solving as a strategy for reducing communication. Durfee states that the design of organizational structures is a complex search problem in its own right. He reports on the work of Decker and Lesser [7] that allows structuring when the problem instance is initialized and the work of Ishida et al. [21] and Prasad et al. [33] that allow dynamic restructuring. Our approach to autonomous dynamic reconfiguration differs from these approaches as they do not reconfigure on the social structure level but on the individual structure level of agents. Additionally, the work of [21] relies on a production system-based agent architecture which entails the known problems of rule-based computing (e. g. rules are not combinable).

*Agent cloning* [8, 35] denotes the reproduction of a similar agent in place or remotely. This approach claims to subsume mobile agents and other similar approaches. It has been mainly developed to support load balancing by delegating tasks to other or new agents that are idle. Compared to our proposal, this approach is more on the technical side of reconfiguration. The concepts presented in this report include not only a quantitative reconfiguration but also a qualitative reconfiguration in the sense of forming new agent types.

In the project MetaMorph I [31, 30] Maturana and others have developed an architecture for distributed task decomposition in manufacturing and coordination in dynamically created clusters. The agents in this system are organized by mediators and contain templates and cloning mechanism to create new agent (sub)levels. The follow-up project, MetaMorph II [36] extends these concepts by taking into account manufacturing design issues, marketing, supply chain management and execution control. MetaMorph II has not been fully implemented. Again this approach for reconfiguration lies on the social level of agency and not on the individual like proposed in this report.

Sandholm et al. report research on *coalition structure generation* [34] that is quite near to ours. Alas, the optimization measure is simpler in this setting, since it only cumulates the value (in our approach complexity) of all partition subsets without taking into account connecting edges. Hence, they can use different kinds of algorithms. Additionally, no concept similar to our local operators can be found in this work.

## 6 Conclusions

After motivating the need for distribution in certain problems of constraint satisfaction, we have introduced a model of distributed constraint satisfaction problems that slightly differs from other models by assigning constraints to agents, too. Since there is often some freedom in assigning subproblems to agents, the main administrative problem in DCSP solving is the design of distribution. We have formally introduced the DCS Configuration problem and its variants and have proven them to be at least $NP$-complete. To solve this problem we have described two local operators, *agent melting* and *agent splitting*, that can be combined to sequences of *reconfiguration transaction* and such are sufficient to solve any given DCS Configuration problem.

As can be seen by our brief considerations on the autonomous usage of these operators, our main objective for future work is to extend the formalism to be applicable under weaker conditions of completeness in the reconfiguration and DSCP solving process. Another main point will be the realization of the named concepts and their empirical evaluation in the AuReCon project.

## Acknowledgments

Thanks to Armin Wolf and Ulrich Geske for fruitful discussion on this work.

## References

[1] S. T. Barnard and H. D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6:101–107, 1994.

[2] E. R. Barnes. An algorithm for partitioning the nodes of a graph. *SIAM Journal on Algebraic and Discrete Methods*, 3(4):541–550, 1982.

[3] R. Barták. Constraint programming: What is behind? In *Proceedings of the Workshop on Constraint Programming for Decision and Control (CPDC-99)*, Gliwice, Poland, 1999.

[4] P. Berlandier and B. Neveu. Problem partition and solvers coordination in distributed constraint satisfaction. In *Proceedings of the Workshop on Parallel Processing in Artificial Intelligence (PPAI-95)*, Montréal, Canada, 1995.

[5] R. B. Boppana. Eigenvalues and graph bisection: An average-case analysis. In *Proceedings of the twentyeightth Annual Symposium on Foundations of Computer Science*, pages 280–285, Los Angeles, USA, 1987.

[6] T. N. Bui. *Graph Bisection Algorithms*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1986.

[7] K. Decker and V. Lesser. A one-shot dynamic coordination algorithm for distributed sensor networks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 210–216, 1993.

[8] K. Decker, K. P. Sycara, and M. Williamson. Cloning for intelligent adaptive information agents. In C. Zhang and D. Lukose, editors, *Multi-Agent Systems: Methodologies and Applications*, volume 1286 of *LNAI*, pages 63–75. Springer, 1997.

[9] E. H. Durfee. Distributed problem solving and planning. In G. Weiss, editor, *Multiagent Systems — A Modern Approach to Distributed Artificial Intelligence*, pages 121–163. MIT Press, 1999.

[10] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[11] I. P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The constrainedness of search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 246–252. AAAI Press, 1996.

[12] M. Hannebauer. Multi-phase consensus communication in collaborative problem solving. In *Proceedings of the Third Workshop on Communication-based Systems (CBS-2000, to appear)*, pages 131–146. Kluwer, 2000.

[13] M. Hannebauer. Transforming object-oriented domain models into declarative CLP expressions. In *Proceedings of the 14th Workshop on Logic Programming (WLP-99)*, Würzburg, Germany, 2000.

[14] M. Hannebauer and U. Geske. Coordinating distributed CLP-solvers in medical appointment scheduling. In *Proceedings of the Twelfth International Conference on Applications of Prolog (INAP-99)*, pages 117–125, Tokyo, Japan, 1999.

[15] M. Hannebauer and R. Kühnel. Dynamic reconfiguration in collaborative problem solving. In H.-D. Burkhard, L. Czaja, H.-S. Nguyen, and P. Starke, editors, *Proceedings of the Eighth Workshop on Concurrency, Specification and Programming (CS&P-99)*, pages 71–82, Warsaw, Poland, 1999.

[16] B. Hayes. Can't get no satisfaction. *American Scientist*, 85(2):108–112, 1997.

[17] B. Hendrickson and K. Devine. Dynamic load balancing in computational mechanics. *Comp. Meth. Applied Mechanics & Engineering (to appear)*, 1999.

[18] B. Hendrickson and R. Leland. An empirical study of static load balancing algorithms. In *Proceedings of the Scalable High Performance Computing Conference*, pages 682–685. IEEE Press, 1994.

[19] B. Hendrickson and R. Leland. The chaco user's guide version 2.0. Technical Report SAND95-2344, Sandia National Laboratories, Albuquerque, USA, 1995.

[20] T. Hogg, B. A. Huberman, and C. P. Williams. Phase transitions and the search problem. *Artificial Intelligence*, 81:1–15, 1996.

[21] T. Ishida, L. Gasser, and M. Yokoo. Organization self-design of distributed production systems. *IEEE Transactions on Knowledge and Data Engineering*, 4(2):123–134, 1992.

[22] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report 95-035, Department of Computer Science, University of Minnesota, Minneapolis, USA, 1995.

[23] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

[24] Q. Y. Luo, P. G. Hendry, and J. T. Buchanan. Heuristic search for distributed constraint satisfaction problems. Research Report KEG-6-92, Department of Computer Science, University of Strathclyde, Glasgow G1 1XH, UK, 1992.

[25] Q. Y. Luo, P. G. Hendry, and J. T. Buchanan. A hybrid algorithm for distributed constraint satisfaction problems. Research Report RR-92-62, Department of Computer Science, University of Strathclyde, Glasgow G1 1XH, UK, 1992.

[26] Q. Y. Luo, P. G. Hendry, and J. T. Buchanan. A new algorithm for dynamic constraint satisfaction problems. Research Report RR-92-63, Department of Computer Science, University of Strathclyde, Glasgow G1 1XH, UK, 1992.

[27] Q. Y. Luo, P. G. Hendry, and J. T. Buchanan. Comparison of different approaches for solving distributed constraint satisfaction problems. Research Report RR-93-74, Department of Computer Science, University of Strathclyde, Glasgow G1 1XH, UK, 1993.

[28] A. K. Mackworth. Constraint satisfaction. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 285–293. Wiley-Interscience Publication, second edition, 1992.

[29] D. L. Mammen and V. R. Lesser. Problem structure and subproblem sharing in multi-agent systems. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, Paris, France, 1998.

[30] F. Maturana, W. Shen, and D. H. Norrie. Metamorph: An adaptive agent-based architecture for intelligent manufacturing. *International Journal of Production Research*, 1998.

[31] F. P. Maturna and D. H. Norrie. Multi-agent mediator architecture for distributed manufacturing. *Journal of Intelligent Manufacturing*, 7:257–270, 1996.

[32] H. Müller, T. Hilbrich, and R. Kühnel. An assistant agent. *Fundamenta Informaticae*, 39(3):327–336, 1999.

[33] M. V. N. Prasad, K. Decker, A. Garvey, and V. Lesser. Exploring organizational designs with TAEMS: A case study of distributed data processing. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*, pages 283–290, 1996.

[34] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Thomé. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1–2):209–238, 1999.

[35] O. Shehory, K. P. Sycara, P. Chalasani, and S. Jha. Increasing resource utilization and task performance by agent cloning. In *Proceedings of the Fifth International Workshop on Agent Theories, Architectures and Languages (ATAL-98)*, pages 305–318, Paris, France, 1998.

[36] W. Shen and D. H. Norrie. A hybrid agent-oriented infrastructure for modeling manufacturing enterprises. In *Proceedings of the Knowledge Acquisition Workshop (KAW-98)*, Banff, Canada, 1998.

[37] G. Solotorevsky, E. Gudes, and A. Meisels. Modeling and solving distributed constraint satisfaction problems (DCSPs). In *Proceedings of the Conference on Constraint-Processing (CP-96)*, 1996.

[38] K. P. Sycara, S. F. Roth, N. Sadeh, and M. S. Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1446–1461, 1991.

[39] C. Walshaw, M. Cross, and M. Everett. Mesh partitioning and load-balancing for distributed memory parallel systems. In *Proceedings of Parallel and Distributed Computing for Computational Mechanics*, Lochinver, Scotland, UK, 1998.

[40] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and DATA Engineering*, 10(5), 1998.

[41] M. Yokoo and K. Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, pages 372–379, Paris, France, 1998.

[42] M. Yokoo and T. Ishida. Search algorithms for agents. In G. Weiss, editor, *Multiagent Systems — A Modern Approach to Distributed Artificial Intelligence*, pages 165–199. MIT Press, 1999.