

**Laurent Granvilliers and Barry O'Sullivan**  
**(Editors)**

---

# **Constraints and Design**

**First International Workshop**  
**Sitges, Spain, 1st October 2005**  
**Proceedings**

Held in conjunction with the  
Eleventh International Conference on  
Principles and Practice of  
Constraint Programming (CP 2005)



# Preface

Constraint processing has emerged as an extremely expressive and powerful paradigm in which to model, solve and reason about many complex problems. Over the past several decades advances in both the fundamental aspects of constraint processing and practical applications of constraints have contributed to making it one of the most promising of Artificial Intelligence technologies.

In product development and design, constraints arise in many forms. The functional description of an artifact defines a set of constraints, as does the physical realization of that functionality. The production processes that will be used to manufacture the artifact can constrain the materials and dimensions that the designer can select. Preferences can be represented as constraints so that optimization techniques can be employed, as well as forming a basis for negotiation. Of course, in many situations constraints emerge during design. Therefore, techniques for supporting the acquisition and discovery of constraints are important. Finally, designers often wish to have explained to them why some design option is not available to them, or how to overcome blind-alleys. Techniques from the fields of diagnosis, as well as approaches to visualization and explanation are critical.

While the study of constraints has been maturing over the past several decades, there are many opportunities to hybridize constraint processing with other technologies from the fields of both Artificial Intelligence and Cognitive Science to develop sophisticated tools for supporting design. The objective of this workshop is to collect papers that primarily exploit developments in constraint processing in the domain of engineering design.

We wish to thank all the authors who submitted papers, the members of the programme committee, and the CP-2005 Tutorial and Workshop Chairs, Alan Frisch and Ian Miguel.

August 2005

Laurent Granvilliers and Barry O'Sullivan  
Programme Chairs

## **Organising Committee**

Laurent Granvilliers – LINA, France

Barry O’Sullivan – Cork Constraint Computation Centre, Ireland

## **Programme Committee**

Pedro Barahona – New University of Lisbon

Frederic Benhamou – LINA, France

James Bowen – University College Cork, Ireland

Boi Faltings – EPFL, Switzerland

Xavier Fischer – LIPSI ESTIA, France

John Gero – University of Sydney, Australia

Ulrich Junker – ILOG, France

Krzysztof Kuchcinski – Lund University, Sweden

Patrick Sebastian – TREFLE, France

Radoslaw Szymanek – Cork Constraint Computation Centre, Ireland

Laurent Zimmer – Dassault Aviation, France

## Table of Contents

Acquiring an Incomplete Specification as a Partially Defined Constraint . . .	1
<i>Arnaud Lallouet, Andreï Legtchenko</i>	
Knowledge Modeling for Decision Support Systems in Mechanical Embodiment Design . . . . .	17
<i>Patrick Sebastian, R. Chenouard and Jean-Pierre Nadeau</i>	
A Specificity of CSP in Design: Controlling the Relevance of the Variables in the Problem . . . . .	34
<i>Thomas van Oudenhove de Saint Gry, Paul Gaborit, and Michel Aldanondo</i>	



# Acquiring an Incomplete Specification as a Partially Defined Constraint

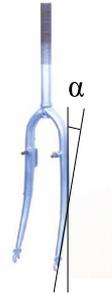
Arnaud Lallouet, Andrei Legtchenko

Université d'Orléans — LIFO  
BP6759, F-45067 Orléans  
lallouet|legtchen@lifo.univ-orleans.fr

**Abstract.** Partially defined Constraints can be used to model the incomplete knowledge of a concept or a relation. Instead of only computing with the known part of the constraint, we propose to complete its definition by using Machine Learning techniques. Since constraints are actively used during solving for pruning domains, building a classifier for instances is not enough: we need a solver able to reduce variable domains. Our technique is composed of two steps: first we learn a classifier for the constraint's projections and then we transform the classifier into a propagator. We show that our technique not only has good learning performances but also yields a very efficient solver for the learned constraint. In Constraint Aided Design, this technology could be used when some constraints are difficult to represent, like comfort, user satisfaction. . .

## 1 Introduction

The success of Constraint Programming takes its roots in its unique combination of modeling facilities and solving efficiency. However, the use of Constraint Programming is often limited by the knowledge of the constraints which may be appropriate to represent a given problem. It can happen that a model involves a constraint which is only *partially known* like for example if the constraint represents a concept we cannot, we do not know or we do not want to define in extension. It can be the set of mammals in a description of animals, solar systems inside astronomical data, a preference between possibilities in a configuration problem, the notion of “good” wine or an habit like the usually free time slots in somebody's diary. It may also happen that the user does not know which constraint can be used to model the problem because of lack of knowledge in Constraint Programming, but on the other hand can easily express examples or counter-examples for it. This situation appears in design [8, 18] when a requirement is difficult or is impossible to model. For example, when designing a bicycle, the angle  $\alpha$  of the fork (see figure 1) has an impact on measurable concepts like the turning circle but also on less well-defined concepts like the ability to go straight when driving hands up or the feeling of comfort of the user. Such a constraint can be modeled by examples and counter-examples and impose requirements on the shape of the item when designing a new fork. Also, other



**Fig. 1.** Angle of a bicycle fork

concepts like consumer satisfaction can be modeled alike by giving examples for already built objects.

In this paper, we propose to use partially defined finite domain constraints. In an partially defined constraint, some tuples are known to be true, some other are known to be false and some are just *unknown*. We make use of this partial knowledge for *learning* the concept which is behind the partially defined constraint. Given positive and negative examples of the concept, the task consists in completing the definition in such a way that new examples never met by the system will be correctly classified. This framework has been extensively studied in Machine Learning [16] under the name of *supervised classification*. Partially defined constraints were introduced in [11] in the context of distributed reasoning but with the goal of minimizing the number of requests needed to complete the definition. In contrast, we assume here that the full constraint is not available even by asking other agents. Also there is not a single way to complete a partially defined constraint since different people just agree on examples but may have different extensions in mind. In addition, they may revise its definition when they get more experience or knowledge. In this paper, we are only concerned by the acquisition of a single constraint. However, its arity may be large.

Let us take an example in which partially defined constraints occur naturally: in a large company, the canteen serves a large number of meals a day. One day, the Chef is asked to prepare as first course a salad which should be good (to respect the company's high standards) but also the cheapest possible (because of the company's low profits last year). The Chef owns a cookbook composed of 53 recipes of salads and has various ingredients such as tomatoes, mayonnaise, shrimps... All are given with price and available quantity. A first idea would be to select from the cookbook the cheapest recipe possible given the available ingredients. But, since not all knowledge about salads is contained in the cookbook, the invention of a new salad is also an interesting option. The concept of "good salad" can be modeled as an partially defined constraint whose solution tuples are *good* salads and non-solutions are *bad* ones. The cookbook is then viewed as a set of examples for the partially defined constraint (for the sake of learning, we should also give examples of bad salads).

Partially defined constraints can be learned whenever examples and counter-examples of the relation are available. For example, in the context of a distributed appointment system using diaries stored on PDAs, each agent may learn a representation of the other agents' schedule in order to minimize future conflicts when searching for a common appointment. Then a correct learning can be used as an heuristic which proposes first the slots which are more likely to be free. Examples are here provided by the history of interactions between agents.

The idea of the technique we use for learning comes directly from the classical model of solvers computing a chaotic iteration of reduction operators [3]. We begin by learning the constraint. But instead of learning it by a classifier which takes as input all its variables and answers "yes" if the tuple belongs to the constraint and "no" otherwise, we choose to *learn the support function* of the constraint for each value of its variables' domains. A tuple is part of the constraint if accepted by all classifiers for each of its values and rejected as soon as it gets rejected by one. This method is non-standard in Machine Learning but we show in section 4 that it can achieve a low error ratio — comparable to well-established learning methods — when new tuples are submitted, which proves experimentally its validity.

As is, a classifier is only able to perform satisfiability checks for an partially defined constraint. If put in a CSP, this constraint would not contribute to the reduction of variables domains and it would yield a "generate and test" behavior that could quickly ruin the performances of the system. Hence, it is needed that partially defined constraints should own a *solver* and not only a satisfiability test in order to meet the standards of efficiency of Constraint Programming. The classifiers we learn are expressed by functions and we turn them into propagators by taking their extension to intervals. This formal transformation does not involve any more learning technique, thus preserving the properties of the first part. Then the classifiers can be used with variable domains as input. We also show that the consistency they enforce, while weaker than arc-consistency, is nevertheless interesting and yields a strong pruning along the search space.

## 2 Preliminaries: building consistencies

We first recall the basic notion of consistency in order to present the approximation scheme we use for learning. For a set  $E$ , we denote by  $\mathcal{P}(E)$  its powerset and by  $|E|$  its cardinal. Let  $V$  be a set of variables and  $D = (D_X)_{X \in V}$  be their family of (finite) domains. For  $W \subseteq V$ , we denote by  $D^W$  the set of tuples on  $W$ , namely  $\prod_{X \in W} D_X$ . Projection of a tuple or a set of tuples on a set of variables is denoted by  $|$ . A *constraint*  $c$  is a couple  $(W, T)$  where  $W \subseteq V$  are the variables of  $c$  (denoted by  $var(c)$ ) and  $T \subseteq D^W$  is the set of solutions of  $c$  (denoted by  $sol(c)$ ). A *Constraint Satisfaction Problem* (or CSP) is a set of constraints. A solution is a tuple which satisfy all constraints. In this paper, we use the common framework combining *search* and domain reduction by a *consistency*.

A *search state* is a set of yet possible values for each variable: for  $W \subseteq V$ , it is a family  $s = (s_X)_{X \in W}$  such that  $\forall X \in W, s_X \subseteq D_X$ . The corresponding

search space is  $S_W = \prod_{X \in W} \mathcal{P}(D_X)$ . The set  $S_W$ , ordered by pointwise inclusion  $\subseteq$  is a complete lattice. Some search states we call *singletonic* represent a single tuple and play a special role as representant of possible solutions. A singletonic search state  $s$  is such that  $|\Pi s| = 1$ .

A consistency can be modeled as the greatest fixpoint of a set of so-called *propagators* and is computed by a chaotic iteration [3]. For a constraint  $c = (W, T)$ , a *propagator* is an operator  $f$  on  $S_W^1$  having the following properties:

- *monotony*:  $\forall s, s' \in S_W, s \subseteq s' \Rightarrow f(s) \subseteq f(s')$ .
- *contractance*:  $\forall s \in S_W, f(s) \subseteq s$ .
- *correctness*:  $\forall s \in S_W, \Pi s \cap \text{sol}(c) \subseteq \Pi f(s) \cap \text{sol}(c)$ .
- *singleton completeness*: let  $s$  be a singletonic state, then  $\Pi s \in \text{sol}(c) \Leftrightarrow f(s) = s$ .

Correctness means that a solution tuple never gets rejected across the search space while singleton completeness means that the operator is also a satisfiability test for a single tuple.

Let us now define some consistencies associated to a constraint  $c = (W, T)$ . The well-known arc-consistency operator  $ac_c$  is defined by:

$$\forall s \in S_W, ac_c(s) = s' \text{ with } \forall X \in W, s'_X = (\Pi s \cap T)|_X$$

If we suppose that each variable domain  $D_X$  is equipped with a total ordering  $\leq$ , we denote by  $[a..b]$  the interval  $\{e \in D_X \mid a \leq e \leq b\}$ . For  $A \subseteq D_X$ , we denote by  $[A]$  the set  $[\min(A).. \max(A)]$ . By extension to Cartesian products, for  $s = (s_X)_{X \in W} \in S_W$ , we denote by  $[s]$  the family  $([s_X])_{X \in W}$  in which each variable domain is extended to its smallest enclosing interval. The bound-consistency operator  $bc_c$  is defined by:

$$\forall s \in S_W, bc_c(s) = s' \text{ with } \forall X \in W, s'_X = s_X \cap [( \Pi [s] \cap T )|_X]$$

Bound-consistency only enforces consistency for the bounds of the domain by shifting them up to the next consistent value in the suitable direction. Consistencies are partially ordered according to their pruning power and we have  $f \subseteq f'$  if  $\forall s \in S_W, f(s) \subseteq f'(s)$ .

Since only variables domains are reduced, a consistency operator  $f$  for a constraint  $c = (W, T)$  can be splitted in  $|W|$  projection operators  $(f_X)_{X \in W}$  according each variable of the constraint. By confluence of chaotic iterations [3], these operators can be scheduled independently as long as they follow the three first properties of consistency operators. In order to represent the same constraint, they have to be singleton complete collectively. It is worth to notice that there is a disymetry between reject and acceptance and that for satisfiability, a non-solution tuple must be rejected (at least) by *one* of these operators while correctness imposes that a solution tuple is accepted by *all* operators.

<sup>1</sup> When iterating operators for constraints on different sets of variables, a classical cylindrification on  $V$  is applied.

The role of a consistency operator  $f_X$  is to eliminate from the domain of its target variable  $X$  some values which are unsupported by the constraint. Arc-consistency eliminates all inconsistent values. Thus, it has to find a *support* for each considered value  $a$  (a solution tuple whose projection on the target variable is  $a$ ) in order to allow the value to remain in the variable's domain. This task has been proved to be NP-complete in general [6] for  $n$ -ary constraints. While many useful constraints have polynomial-time arc-consistency propagators, there exists some for which this task is intractable. Since we are dealing in this paper with constraints expressed by examples, this case must be taken into account seriously and motivates an approximation scheme.

At a finer level of granularity, we can decompose an arc-consistency operator  $f_X$  according each value of  $X$ 's domain. We call an *Elementary Reduction Function* (or ERF) a boolean function  $f_{X=a}$  checking if a value  $a$  in  $X$ 's domain has a support. In order to achieve this check, this function uses as input the domain of the other variables of the constraint. By combining ERFs for each domain value, we can reconstitute the arc-consistency operator. Bound-consistency can be obtained if the function reduces the bounds of its target variable and only makes use of the bounds of the other variables' domains.

If we give each domain value an ERF but if we assume that this ERF takes as input only the bounds of the other variables' domains, we get a new intermediate consistency, we call  $ac^-$ :

$$\forall s \in S_W, ac_c^-(s) = s' \text{ with } \forall X \in W, s'_X = s_X \cap (\Pi[s] \cap T)|_X$$

It does not have the full power of arc-consistency since it make use of less input information but may reduce more than bound-consistency since not only the bounds can be reduced. The counterpart, called  $bc^+$  is when bounds can be reduced by a function taking as input all information available in the whole domain of the other variables:

$$\forall s \in S_W, bc_c^+(s) = s' \text{ with } \forall X \in W, s'_X = s_X \cap [(\Pi s \cap T)|_X]$$

**Proposition 1.**  $bc \subseteq bc^+ \subseteq ac$  and  $bc \subseteq ac^- \subseteq ac$ .

**Proposition 2.**  $ac^-$  and  $bc^+$  are uncomparable.

### 3 Partially Defined Constraints

In this section, we give the definition of partially defined constraints and we introduce the notion of *extension* which provides a closure of the constraint.

A classical constraint  $c = (W, T)$  is supposed to be known in totality. The underlying *Closed World Assumption* (or CWA) states that what is not explicitly declared as true is false. Hence the complementary  $\bar{T}$  is the set of tuples which do not belong to  $c$ . In the following, we call ordinary constraints under CWA *closed* or *classical* constraints. When dealing with incomplete information, it may happen that some parts of the constraint are unknown:

**Definition 3 (Partially defined constraint).**

An partially defined constraint is a triple  $c = (W, c^+, c^-)$  where  $c^+ \subseteq D^W$ ,  $c^- \subseteq D^W$  and  $c^+ \cap c^- = \emptyset$ .

In a partially defined constraint  $c = (W, c^+, c^-)$ ,  $c^+$  represents the allowed tuples and  $c^-$  the forbidden ones. The remaining tuples, i.e.  $\overline{c^+ \cup c^-}$  are simply unknown. Note that a classical constraint  $c = (W, T)$  is a particular partially defined constraint  $c = (W, T, \overline{T})$  for which the negative part is the complementary of the positive part.

Partially defined constraints need a special treatment in order to be used in a CSP since few propagation can be done without knowing the integrality of the constraint. Hence a partially defined constraint needs to be *closed* to be usable in a constraint solving environment. The closure of a partially defined constraint  $c$  is done by choosing a class (it belongs or not to the constraint) for all unknown tuples. We call the resulting classical constraint an *extension* of the partially defined constraint:

**Definition 4 (Extension).**

Let  $c = (W, c^+, c^-)$  be a partially defined constraint. A (classical) constraint  $c' = (W, T)$  is an extension of  $c$  if  $c^+ \subseteq T$  and  $c^- \subseteq \overline{T}$ .

In other terms, an extension is a classical constraint compatible with the known part (positive and negative) of the partially defined constraint. A partially defined constraint allows to catch a glimpse of a hidden reality and one of its extensions corresponds to the genuine relation of the world. In most cases, the knowledge of this constraint is impossible to get and all that can be done is computing an approximation of it. In general, many extensions can be considered. We are interested in the extension in which the unknown part is completed by a learning algorithm  $\mathcal{A} : D^W \rightarrow \{0, 1\}$  such that  $t \in c^+ \Rightarrow \mathcal{A}(t) = 1$  and  $t \in c^- \Rightarrow \mathcal{A}(t) = 0$ .

This kind of extension is obtained by supervised classification: it consists in the induction of a function which associates to all tuples a class from a set of examples given with their respective class. Machine Learning puts strong requirements on what is called a good algorithmic extension. First and perhaps the most important, the correct class for unknown tuples should be forecast with the highest possible accuracy. The ratio between the number of correctly classified tuples and the number of presented tuples defines the *correctness ratio* of the generalization. Most techniques used in Machine Learning provide much better performances than random classification and more than 90% of success in prediction is not unusual. In order to achieve this, we assume that the known part of the partially defined constraint should be *representative* of the whole underlying constraint. Then, the representation of the classification function is searched in a space of possible functions called *hypothesis space*. A *learning algorithm* finds the best possible function in this space by optimizing some criterions, like correctness, accuracy, simplicity or generalization. . .

## 4 Constraint Acquisition

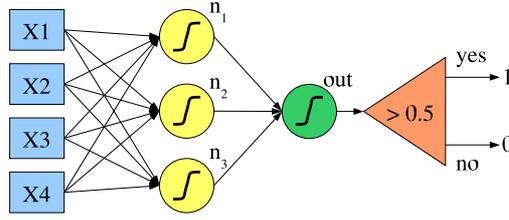
At first, we address the problem of constructing a good extension for a partially defined constraint. In order to represent a relation, the first idea is to build a classifier taking as input an instantiation of all variables of the relation and returning a boolean stating if the tuple belongs or not to the relation. But unfortunately, while learning is effective with this technique (see [19]), it would be difficult to extract a solver from this representation. Motivated by the equivalence between a constraint and a correct and singleton complete solver, we propose to acquire a partially defined constraint  $c = (W, c^+, c^-)$  by learning the support function for all value of domain variables. More precisely, we propose to build an independent classifier for each value  $a$  of the domain of each variable  $X \in W$  in the spirit of ERFs introduced in section 2. This classifier computes a boolean function stating if the value  $a$  should remain in the current domain (output value 1) or if it can be removed (value 0). We call it an *elementary classifier*. It takes as input the value of every other variable in  $W - \{X\}$ .

We propose to use as representation for learning an Artificial Neural Network (ANN) with an intermediate hidden layer. This representation has been chosen for its good properties in learning and the possibility of a further transformation into a solver. Other kinds of classifiers can also be used but we cannot describe them for lack of space. For  $W \subseteq V$ , a *neuron* is a function  $n(W) : \mathbb{R}^{|W|} \rightarrow \mathbb{R}$  computing the weighted sum of its inputs followed by a threshold unit. A dummy input set at 1 is added to tune the threshold. The sigmoid function is often chosen for the threshold unit since derivability is an important property for the learning algorithm. Let  $(w_X)_{X \in W}$  be the weights associated to each input variable and  $w_0$  be the adjustment weight for the dummy input. Hence, the function computed by a neuron taking as input  $a = (a_X)_{X \in W}$  is:

$$n(a) = \frac{1}{1 + e^{w_0 - \sum_{X \in W} w_X \cdot a_X}}$$

For a constraint  $c = (W, c^+, c^-)$ , the classifier we build for  $X = a$  is a tree of neurons with one hidden layer as depicted in figure 2. Let  $(n_i)_{i \in I}$  be the intermediary nodes and *out* be the output node. All neurons of the hidden layer have as input a value for each variable in  $W - \{X\}$  and are connected to the output node. Let us call  $n_{\langle X=a \rangle}$  the network concerning  $X = a$ . Since neurons are continuous by nature, we use an analog coding of the domains. Let  $D$  be a finite domain and  $<$  a total order on  $D$  (natural or arbitrary), then we can write  $D$  as  $\{a_0, \dots, a_n\}$  with  $\forall i \in [1..n], a_{i-1} < a_i$ . According to this order, we can map  $D$  on  $[0..1]$  by coding  $a_i$  by  $i/n$ . In a similar way, the output is in the interval  $[0..1]$  and we choose as convention that the value  $a$  should be removed from the domain of  $X$  if  $out \leq 0.5$ . This threshold is the last level of the network depicted in figure 2.

The global classifier for the partially defined constraint is composed of all of these elementary classifiers for all values in the domain of all variables  $\{n_{\langle X=a \rangle} \mid X \in W, a \in D_X\}$ . Following the intuition of ERFs for solving, we can use



**Fig. 2.** Structure of the ANN

these elementary classifiers to decide if a tuple belongs to the extension of the constraint or not by checking if the tuple gets rejected or not by one of the classifiers. Let  $t \in D^W$  be a candidate tuple and let  $(n_{\langle X=t|_X \rangle}(t|_{W-\{X\}}))_{X \in W}$  be the family of 0/1 answers of the elementary classifiers for all values. We can interpret the answers according two points of view:

- *vote with veto*: the tuple is accepted if and only if it is accepted by all elementary classifiers.
- *majority vote*: the tuple is accepted if accepted by a majority of elementary classifiers.

In order to produce the extension of the partially defined constraint, these classifiers are trained on examples and counter-examples selected from the projection of the known part of the constraint on the sub-space orthogonal to a variable's value. For  $E \subseteq D^W$ ,  $X \in W$  and  $a \in D_X$ , we denote by  $E_{\langle X=a \rangle}$  the selection of tuples of  $D^W$  having  $a$  as value on  $X$ :  $E_{\langle X=a \rangle} = \{t \in E \mid t|_X = a\}$ . Thus, in order to build the classifier  $n_{\langle X=a \rangle}$ , we take the following sets of examples and counter-examples:

$$e_{\langle X=a \rangle}^+ = c_{\langle X=a \rangle}^+|_{W-\{X\}}$$

$$e_{\langle X=a \rangle}^- = c_{\langle X=a \rangle}^-|_{W-\{X\}}$$

For example, for a partially defined constraint defined by  $W = \{X, Y, Z\}$ ,  $c^+ = \{(1, 1, 0), (1, 0, 1)\}$  and  $c^- = \{(1, 1, 1)\}$ , we get:

- $e_{\langle X=1 \rangle}^+ = \{(1, 0), (0, 1)\}$ .
- $e_{\langle X=1 \rangle}^- = \{(1, 1)\}$ .

The networks are trained by the classical backpropagation algorithm [21] which finds a value for the weights using gradient descent. The algorithm is stopped when all examples and counter-examples are correctly classified. This requirement comes from the need of correctness of constraint programming but it may be adjusted according to the application and to how noisy the training set is. In order to do this, some changes to the structure of the network may have to be done. ANN with an hidden layer have a sufficient expressivity to represent any boolean function, but at the price of an exponential space complexity. In

Database	salad	mush.	cancer	votes-84
<i>Arity</i>	22	22	9	16
<i>Size of DB</i>	334	8124	699	435
<i>Domain sz</i>	2-4	2-12	10	3
<i># neurons in HL</i>	3	3	5	5
<i># classifiers</i>	64	116	90	48
<i>Learning time</i>	55''	2'30''	8'30''	4'30''
<i>gen ratio Solar veto (SD)</i>	88.08 (3.66)	93.06 (1.91)	95.36 (1.80)	74.07 (3.82)
<i>gen ratio Solar maj (SD)</i>	<b>96.36</b> (3.51)	99.29 (0.99)	<b>96.52</b> (2.15)	96.23 (3.26)
<i>gen ratio C5.0 (SD)</i>	90.14 (6.87)	99.19 (1.21)	94.54 (2.44)	<b>96.29</b> (3.39)
<i>gen ratio C5.0 boost (SD)</i>	95.17 (3.42)	<b>99.80</b> (0.61)	96.33 (2.26)	95.63 (3.63)

**Table 1.** Learning results.

many case, it is better to keep a small network size in order to preserve its generalization capabilities.

Since this technique for learning relations is not classical in Machine Learning, we present validation results to show that the concept lying behind the partially defined constraint is actually learned. This framework has been implemented in a system called SOLAR and tested on the *salad* example presented in introduction and on various Machine Learning databases<sup>2</sup> used as constraint descriptions. This makes a contrast with classical Constraint Programming experiments, for example on random CSPs, because we need to be sure that there is an actual concept behind the partially defined constraint for the learning to make sense. With random constraints, no learning is possible. For the *salad* example, we have added to the cookbook a set of 281 recipes of bad salads as counter-examples. The results are summarized in Table 1. The database *mushroom* gives attributes to recognize edible candidates, *breast-cancer-wisconsin* to recognize benign and malignant forms of disease and *house-votes-84* to classify democrats and republicans. For the *salad* and *mushroom* constraints, we have 3 neurons in the hidden layer while we have 5 for *breast-cancer-wisconsin* and *house-votes-84*.

We compare the generalization performance of our technique to the popular decision tree learning system *C5.0* [20]. For each base, we have performed a cross-validation by using the following method: we cut the base in ten parts, we use nine of them for learning and the tenth for validation. This process is repeated 10 times, each part of the base being used in turn for validation. The cut off is identical for the test with all methods. Then the whole test is repeated on 5 sessions with different cuts off, yielding 50 tests for each technique. The generalization ratio is the percentage of correctly classified tuples.

Table 1 contains three parts. The first one contains a description of the data: database name, arity, size and size of the variables' domains. Then follow

<sup>2</sup> The databases are taken from the UCI Machine Learning repository (<http://www.ics.uci.edu/~mllearn>).

some informations about the learned classifiers: the number of neurons in the hidden layer, the number of individual classifiers learned, and the learning time. In comparison, the learning time for *C5.0* is very low, typically a few seconds, but the interest of our technique is not only for classification, as described in the next section. In the last part are presented the generalization results: the generalization ratio and standard deviation (SD) for SOLAR with veto vote, for SOLAR with majority vote, for *C5.0* and for *C5.0* with boosting (with number of trials equal to the arity of the constraint in order to balance our number of classifiers). The *mushroom* database is very easy to learn, hence we only used 500 examples out of 8124 for all techniques, otherwise they all would have reached 100%.

The technique we propose challenges powerful techniques such as boosting [13], both in generalization performance and scattering of results as measured by standard deviation and error. Nevertheless, the vote of elementary classifiers cannot be viewed as a variant of boosting. An important difference is that we partition the set of examples. In veto mode, the learned concept is more focused on the core of the real concept as we impose more elementary classifiers to agree on a tuple. Thus it is not surprising that veto mode performs less satisfactorily than majority mode. The tuples which are accepted at unanimity are in some sense the most "pure" ones with respect to the real concept and the error depicted in Table 1 corresponds to rejected positive tuples and never to accepted negative ones. For optimization purposes, this could even be an advantage since the solution space is smaller and the correctness of the answer is better preserved.

## 5 From classifiers to solvers

When put in a CSP, a constraint shall have an active behavior, it should contribute to the domain reduction. Hence the "generate and test" behavior induced by classifiers is not powerful enough. Another idea could be to first generate offline the solutions of the extension of the constraint and use them for solving with a standard but efficient arc-consistency propagation algorithm like GAC-schema [7]. But unfortunately, this method suffers from two major drawbacks. First the generation time is prohibitive. For example, 3 hours of "generate and test" computation on the *mushroom* database could hardly produce 76835 solutions out of  $1.5 \cdot 10^7$  tries. A second problem comes from the representation size of the relation. The extension of the 22-ary constraint *mushroom* contains more than  $4E6$  solutions and would thus need more than 88 Mb of memory to be stored. In contrast, the representation we have is rather economic. For a constraint of arity  $n$ , if we assume that the hidden layer contains  $m$  neurons and the size of the domains is  $d$ , it requires  $n(n+1)dm+1$  floats (10 bytes) to store the weights. For the *salad* constraint ( $n = 22, m = 3, d = 4$ ), we get a size of 60Kb, for *mushroom* ( $n = 22, m = 3, d = 12$ ), 180 Kb...

We propose to use the learned classifiers also for solving. In order to do this, let us recall some notions on interval analysis [17]. We call  $Int_{\mathbb{R}}$  the interval

lattice built on the set  $\mathbb{R}$  of real numbers. First, all functions have extensions to intervals:

**Definition 5 (Extension to intervals).**

Let  $f : \mathbb{R} \rightarrow \mathbb{R}$  be a function. A function  $F : Int_{\mathbb{R}} \rightarrow Int_{\mathbb{R}}$  is an extension to intervals of  $f$  if  $\forall I \in Int_{\mathbb{R}}, \forall x \in I, f(x) \in F(I)$ .

An extension  $F$  is monotonic if  $A \subseteq B \Rightarrow F(A) \subseteq F(B)$ . Between all extensions to intervals of  $f$ , there is a smallest one, called *canonical extension to intervals*:  $\hat{f}(I) = [\{f(x) \mid x \in I\}]$ . The canonical extension is monotonic. Here are the canonical extensions to intervals of the operators we use in classifiers:

$$\begin{aligned} [a, b] + [c, d] &= [a + c, b + d] \\ [a, b] - [c, d] &= [a - d, b - c] \\ [a, b] \times [c, d] &= [\min(P), \max(P)] \\ &\quad \text{where } P = \{ac, ad, bc, bd\} \\ \exp([a, b]) &= [\exp(a), \exp(b)] \end{aligned}$$

Division is not a problem in our setting since no interval contains 0 (see the sigmoid denominator). If  $e$  is an expression using these operators and  $E$  the same expression obtained by replacing each operator by a monotonic extension, then  $\forall I \in Int_{\mathbb{R}}, \forall x \in I, e(x) \in E(I)$ . This property of monotonic extensions is called "The Fundamental Theorem of Interval Arithmetic" [17]. It also holds when domains are replaced by cartesian products of intervals. By taking the canonical extension of all basic operators in an expression  $e$ , we do not always obtain an extension  $E$  which is canonical. We instead call it the *natural* extension. Multiplication is only sub-distributive in Interval Arithmetic [17], i.e.  $A \times (B + C) \subseteq (A \times B) + (A \times C)$ . Hence the natural extension is canonical only for expressions with single variable occurrences ("Single Occurrence Theorem", [17]).

An elementary classifier  $n_{\langle X=a \rangle}$  defines naturally a boolean function of its input variables. Let  $N_{\langle X=a \rangle}$  be its natural interval extension, defined by taking the canonical extension of each basic operator  $+, -, \times, /, \exp$ . Then, by using as input the current domain of the variables, we can obtain a range for its output. In order to do this, we compute the interval range of every neuron of the hidden layer and we use these results to feed the output neuron and compute its domain. Since we put a 0.5 threshold after the output neuron, we can reject the value  $a$  for  $X$  if the maximum of the output range is less than 0.5, which means that all tuples are rejected in the current domain intervals. Otherwise, the value remains in the domain.

**Proposition 6.**  $N_{\langle X=a \rangle}$  is an ERF.

*Proof.* It is only needed to check the correctness of the function applied to a search state  $s$  with respect to the partially defined constraint's accepted tuples. If a tuple  $t$  such that  $t_X = a$  belongs to the solutions of the learned constraint, then  $n_{\langle X=a \rangle}(\{t_Y\}_{Y \in W - \{X\}}) = 1$ . Hence if  $t \in \Pi s$ , we have  $\max N_{\langle X=a \rangle} = 1$  because  $N$  is a monotonic extension.

By doing this for every value of every variable’s domain, we are able to define a consistency operator  $f_X$  which gathers the results of the ERFs. For  $s \in S_W$ ,  $f_X(s) = s'$  where  $s'_X = s_X \cap \{a \in D_X \mid \max(N_{\langle X=a \rangle}(s|_{W-\{X\}})) = 1\}$  and  $s'_Y = s_Y$  for  $Y \neq X$ .

**Proposition 7.** *The operators  $(f_X)_{X \in W}$  define a consistency for  $c$ .*

*Proof.* Each operator  $f_X$  is monotonic, contractant and correct (by the fundamental theorem of Interval Arithmetic). They are together singleton complete (because the extension of the partially defined constraint is defined by them).

We call  $lc$  (for *learned consistency*) the consistency defined by the learned propagators. Because we use multiple occurrences of the same variable, the consistency  $lc$  computes an approximation of  $ac^-$ :

**Proposition 8.**  $lc \subseteq ac^-$

*Proof.* Since multiple occurrences of variables yield a less precise interval, it follows that the maximum of the output interval of the last neuron of an ERF  $N_{\langle X=a \rangle}$  may exceed 0.5 even if there is no support for  $X = a$ . Thus the value is not suppressed as it would be by  $ac^-$ .

Note that if we had used single layer perceptrons, the extension would have been exact and we would have got  $ac^-$ . But this technique has severe limitations in learning [16]. The propagators for each variable are independent, thus the generalization obtained when using the solver is the one obtained with *veto* vote. This is due to the independent scheduling of the consistency operators for each variable in the fixpoint computed by chaotic iteration [3].

Database	salad	mushroom	cancer	votes-84
<i>#Sol</i>	7.4E5	$\geq 4.1E6$	1.27E5	1.27E5
<i>#Fail lc</i>	1.34E5	$\geq 3.1E6$	1.28E5	3.47E5
<i>ratio lc</i>	1.8	0.75	0.99	2.86
<i>Time lc</i>	5'15"	$\geq 2$ hours	3'00"	7'30"

**Table 2.** Results for solutions and failure.

The SOLAR system takes a partially defined constraint as input and outputs a set of consistency operators which can be adapted to any solver. In our experiments, we used a custom made solver. We made two sets of experiments in order to evaluate the learned consistency. The first one is summarized in table 2 and describes the search for all solutions using the learned consistency  $lc$ . It is done by taking a CSP containing the partially defined constraint alone. For every partially defined constraint, we use our system to count the number of solutions (*#Sol*). Since we do not have arc-consistency, we record the number of failures  $lc$  makes while finding these solutions (*#Fail lc*). Then we compute

Salads:

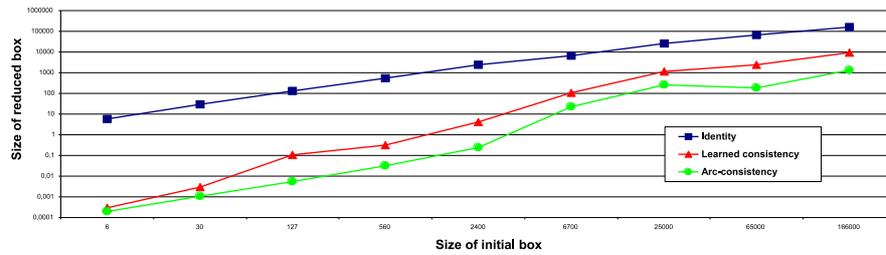
$ s $	29.881	127.406	561.79	2381.654	6721.499	24929.11	64928.5	166563.9
$lc$	0.003	0.106	0.31	4.094	104.220	1138.65	2440.9	9037.4
$bc$	0.001	0.005	0.04	0.256	31.889	473.99	280.5	2315.9
$bc^+$	0.001	0.005	0.04	0.260	31.888	473.15	275.8	1998.2
$ac^-$	0.001	0.005	0.03	0.237	22.210	257.37	186.3	1533.7
$ac$	0.001	0.005	0.03	0.233	22.209	256.63	185.0	1326.5

Mushroom:

$ s $	24.53	500.2	3711.2	23249.4
$lc$	1.57	181.6	1067.2	10226.1
$bc$	0.74	194.2	451.6	4078.4
$bc^+$	0.69	147.8	430.1	3933.2
$ac^-$	0.55	126.4	275.6	2148.4
$ac$	0.52	96.0	261.4	2084.1

**Table 3.** Consistency tests.

the ratio  $lc = \#Fail\ lc / \#Sol$ . If we had arc-consistency, there would not be any backtrack. The purpose of this experiment is to compare  $lc$  to what  $ac$  could have done if  $ac$  was available for partially defined constraints. In terms of failure, the average ratio on all experiments is of 1.6 failures per solution. This result should be put into balance with the huge number of failures "generate and test" would have done. We also report the time  $lc$  needs to find these solutions. The *mushroom* constraint has a very large space and a medium tightness and we could not obtain its full extension. But for the other constraints, this is the only method to get all solutions since the Cartesian products of the domains are so large that this prevents the use of "generate and test" with the classifier.



**Fig. 3.** Compared reduction of  $lc$  and  $ac$ .

Our second set of experiment is a random sampling of the reductions made by the different consistencies on random search states (the domain of each variable are arbitrary sets, not intervals). These results are depicted in Table 3 and 4. For each constraint, we give the number of tuples of the initial search state ( $|I|s|$ ) and the number of tuples after an application of each of the operators  $lc$ ,  $bc$ ,

Breast cancer:

$ s $	11.663	54.70	366.6	997.5	3224.7	8003.6	32014.0
$lc$	1.166	8.31	104.6	351.5	1483.4	4350.8	22142.7
$bc$	0.111	1.17	50.5	127.8	1012.6	1907.2	9423.7
$bc^+$	0.108	0.93	37.1	101.3	955.5	1550.0	7727.8
$ac^-$	0.085	0.78	33.6	84.9	838.0	1528.7	6500.2
$ac$	0.083	0.72	22.0	69.4	686.4	1260.7	4998.0

Votes-84:

$ s $	4.5207	18.408	68.62	229.5	766.9	2096.6	4159.5	13752.9	25147.4
$lc$	0.1256	1.352	11.98	53.6	231.4	858.9	1670.5	7787.2	12597.4
$bc$	0.0334	0.225	3.45	14.0	82.0	456.0	850.6	3990.5	6792.2
$bc^+$	0.0330	0.217	3.28	12.7	77.4	441.1	796.9	3450.0	6038.1
$ac^-$	0.0326	0.219	3.43	13.9	81.9	453.3	850.1	3984.5	6777.4
$ac$	0.0322	0.211	3.26	12.6	77.3	438.5	796.4	3444.0	6027.4

**Table 4.** Consistency tests.

$bc^+$ ,  $ac^-$  and  $ac$ . The data are the average on 1000 experiments with the same average size of search state. In order to compute exactly the consistencies  $bc$ ,  $bc^+$ ,  $ac^-$  and  $ac$ , we have first computed all solutions included in  $s$  in a table with the help of  $lc$ . In a second step, we have computed all needed projections from the solution table. For example, the last column of table 3 for the *salad* example, shows that, starting from a search space containing 166563 tuple (in its Cartesian product), the learned consistency reduces it to a search space of 9037 tuple while for example arc-consistency allows to reduce to 1326 tuples (all these values are average). This shows that the learned consistency is weaker than more classical consistencies but still reduces notably the search space. We recall that  $lc$  is the only available consistency for partially defined constraints, thus this test is only made to give an hint of  $lc$ 's pruning power. In Figure 3 is depicted a graphical view of the consistencies  $lc$  and  $ac$  for the *salad* example with the data of Table 3.

In addition, the partially defined constraint *salad* has been used in the optimization problem described in introduction and, as expected, the best solution found is a recipe *invented* by the system and which is not in the cookbook.

## 6 Related work and Conclusion

Partially defined constraints were introduced in [15]. In [12], the comparable concept of Open Constraint is proposed in the context of distributed reasoning but with the goal of minimizing the number of requests needed to complete the definition. They are similarly used in the framework of Interactive Constraint Satisfaction [2]. Solver learning has been introduced in [4] with a special rule system but the generation algorithm was a bottleneck to handle large constraints.

This work has been extended by [1] and [14] but still in the context of closed constraints. None of these methods can combine generalization and solver efficiency. Partially defined constraints are also related to uncertainty since an uncertain constraint [22] can be viewed as a limited form of partially defined constraint for which it is assumed that only a few tuples are missing. The idea of learning constraints, extended to the learning of a preference instead of just a boolean for a tuple has been used in [19] in the context of soft constraints. They use an ad-hoc neural network to represent the constraint. While the learning is effective, the problem of building a solver for the constraint is not tackled in this work. In [9] and [5], a CSP composed of predefined constraints like  $=$  or  $\leq$  is learned. The constraints are discovered by a version-space algorithm which reduces the possible constraints during the learning process. Artificial Neural Networks have been considered for solving CSPs in the GENET system [10] but with a completely different approach.

### Summary

Partially defined constraints allow the use of constraints partially defined by examples and counter-examples in decision and optimization problems. In this work, we propose a new technique for learning partially defined constraints by using classifiers. Not only the generalization we obtain has remarkable properties from a Machine Learning point of view, but it can also be turned into a very efficient solver which gives an active behavior to the learned constraint. In a design perspective, partially defined constraints can be used to represent complex requirements for which a precise definition is either too complex or impossible to get.

## References

1. Slim Abdennadher and Christophe Rigotti. Automatic generation of rule-based constraint solvers over finite domains. *Transaction on Computational Logic*, 5(2), 2004.
2. M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and M. Milano. A chr-based implementation of known arc-consistency. *Theory and Practice of Logic Programming*, to appear.
3. K.R. Apt. The essence of constraint propagation. *Theoretical Computer Science*, 221(1-2):179–210, 1999.
4. K.R. Apt and E. Monfroy. Automatic generation of constraint propagation algorithms for small finite domains. In Joxan Jaffar, editor, *International Conference on Principles and Practice of Constraint Programming*, volume 1713 of *LNCS*, pages 58–72, Alexandria, Virginia, USA, 1999. Springer.
5. Christian Bessière, Rémi Coletta, Eugene C. Freuder, and Barry O’Sullivan. Leveraging the learning power of examples in automated constraint acquisition. In Mark Wallace, editor, *Principles and Practice of Constraint Programming*, volume 3258 of *LNCS*, pages 123–137, Toronto, Canada, 2004. Springer.
6. Christian Bessière, Emmanuel Hebrard, Brahim Hnich, and Toby Walsh. The complexity of global constraints. In Deborah L. McGuinness and Georges Ferguson, editors, *National Conference on Artificial Intelligence*, pages 112–117, San Jose, CA, USA, July, 25-29 2004. AAAI Press / MIT Press.

7. Christian Bessière and Jean-Charles Régin. Arc-consistency for general constraint networks: preliminary results. In *IJCAI*, pages 398–404, Nagoya, Japan, 1997. Morgan Kaufmann.
8. B. Chandrasekaran. Design problem solving: A task analysis. *AI Magazine*, 11(4):59–71, 1990.
9. R. Coletta, C. Bessière, B. O’Sullivan, E. C. Freuder, S. O’Connell, and J. Quinque-ton. Semi-automatic modeling by constraint acquisition. In Francesca Rossi, editor, *International Conference on Principles and Practice of Constraint Programming*, number 2833 in LNCS, pages 812–816, Kinsale, Ireland, 2003. Springer.
10. A. Davenport, E. Tsang, C. Wang, and K. Zhu. GENET: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. In *National Conference on Artificial Intelligence*, pages 325–330, Seattle, WA, USA, 1994. AAAI Press.
11. Boi Faltings and Santiago Macho-Gonzalez. Open constraint satisfaction. In *CP*, volume 2470 of LNCS, pages 356–370. Springer, 2002.
12. Boi Faltings and Santiago Macho-Gonzalez. Open constraint satisfaction. In Pascal van Hentenryck, editor, *International Conference on Principles and Practice of Constraint Programming*, volume 2470 of LNCS, pages 356–370, Ithaca, NY, USA, Sept. 7 - 13 2002. Springer.
13. Y. Freund and R. Shapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771–780, 1999.
14. Arnaud Lallouet, Thi-Bich-Hanh Dao, Andreï Legtchenko, and AbdelAli Ed-Dbali. Finite domain constraint solver learning. In Georg Gottlob, editor, *International Joint Conference on Artificial Intelligence*, pages 1379–1380, Acapulco, Mexico, 2003. AAAI Press.
15. Arnaud Lallouet, Andreï Legtchenko, Eric Monfroy, and AbdelAli Ed-Dbali. Solver learning for predicting changes in dynamic constraint satisfaction problems. In Ken Brown Chris Beck and Gérard Verfaillie, editors, *Changes’04, International Workshop on Constraint Solving under Change and Uncertainty*, Toronto, CA, 2004.
16. Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
17. Ramon E. Moore. *Interval Analysis*. Prentice Hall, 1966.
18. Barry O’Sullivan. *Constraint-Aided Conceptual Design*. Professional Engineering Publishing, 2002.
19. F. Rossi and A. Sperduti. Acquiring both constraint and solution preferences in interactive constraint system. *Constraints*, 9(4), 2004.
20. RuleQuest Research. See5: An informal tutorial, 2004. <http://www.rulequest.com/see5-win.html>.
21. D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. *Parallel Distributed Processing*, vol 1:318–362, 1986.
22. Neil Yorke-Smith and Carmen Gervet. Certainty closure: A framework for reliable constraint reasoning with uncertainty. In Francesca Rossi, editor, *9th International Conference on Principles and Practice of Constraint Programming*, number 2833 in LNCS, pages 769–783, Cork, Ireland, 2003. Springer.

# Knowledge modeling for decision support systems in mechanical embodiment design

Patrick SEBASTIAN (1), R. CHENOUEARD (1), Jean-Pierre NADEAU (1)

(1) TREFLE UMR CNRS 8508, Bordeaux-Talence, France

(2) LIPSI- ESTIA, Izarbel Centre Technopôle, Bidart, France

## ABSTRACT

There has been growing interest in modeling complex mechanical or energetic systems for decision support in embodiment design. However, despite recent progresses in the development of Constraint Satisfaction Problem (CSP) solvers devoted to mechanical embodiment design, knowledge modeling appears to be the main bottleneck of the development of decision support systems in industrial mechanical applications. A general approach of knowledge modeling dedicated to this problem still lacks. In this paper, some fundamental aspects of knowledge modeling for decision support in embodiment design are investigated. This approach is based on the management of the intrinsic precision, exactness, specialization and parsimony of models, including functional and physics models. Models are adapted to decision support applications by managing the degrees of freedom related to these intrinsic parameters and by using model adaptation methods. Model adaptation methods investigated here are analytical integration or numerical approximation methods used to improve the parsimony of models. As an illustration, the design of an aircraft air conditioning system has been performed using a Constraint Satisfaction Problem solver based on interval analysis.

**Keywords:** Knowledge modeling, Decision support, Embodiment design, Constraint Satisfaction Problem.

## 1. INTRODUCTION

In the framework of innovative processes, mechanical system design is facing with a major difficulty related to the embodiment phase of design [1]. Embodiment design aims to establish the performances of mechanical concepts before performing their detail design. This design phase is equivalent to a feasibility investigation stage for testing new mechanical concepts. Despite recent progresses in the domain of Constraint Satisfaction Problem (CSP) solvers devoted to mechanical embodiment design, decision support tools remains limited by the knowledge pretreatment required by such tools, which appear to be little used or ineffective in most cases. This difficulty is due to the fact that CSP models frequently evolve during the embodiment design phase and the development of new CSP models remains tricky. The evolution of CSP models may be imputed to two aspects:

- CSP models are representative of the functioning of mechanical system whereas designer reasoning is based on the system functionalities
- CSP models corresponds to the translation of knowledge in a particular decisional state (intrinsic

capability of a model to support decision) whereas designer reasoning is directly based on knowledge

Thus, embodiment design platform development is limited by the management of model complexity within these platforms. Model complexity must be managed from preliminary stages of design processes as it depends on the conceptual, functional and physics aspects of system design. During the embodiment phase of design [2, 3], the main characteristics of the mechanical systems have to be determined including the global structure and the main dimensions of the systems. As embodiment design is performed whereas the main characteristics of the designed system are ill-known, the structure of the system is evolving while some parameters of the system environment are not fixed [4]. As design delays are short and as the model must evolve quickly, designers are limited by the complexity of complex physics phenomena. They often manage this difficulty by using simplified and decoupled models based on their design experience. There is a lack of numerical tools devoted to embodiment design due to the difficulties inherent to this design phase [5, 6]. The approach proposed in this paper aims to define a general method for guiding knowledge modeling in the development of embodiment design platforms.

In the framework of this approach, knowledge modeling is perceived as a definition process of models related to intrinsic precisions, exactnesses, parsimonies and specializations. The complexity of a model is adapted to real time simulation and decision support by increasing the model parsimony and by using model adaptation methods. These adaptation methods are based on the characterization of the model parameters which are related to precision, exactness, parsimony and specialization and by using the degrees of freedom offered by these parameters.

Two model adaptation methods are investigated and compared in this paper based on analytical integration or numerical approximation. The Transfer Units method is used in the energy exchanger modeling domain [7, 8]. In the framework of this method, two types of analytical integration approaches are combined by analyzing the topologies of fluid circulations inside the exchangers and the physics phenomena inside elementary models. Integration is performed by fuzzifying some physics parameters and considering different fluid circulation topologies. The scope of application of neural network approximation methods [9] is more extended than the previous method. This adaptation method leads to the definition of grey box models which appear to be highly specialized and parsimonious.

As an illustration, the embodiment design of an Air Conditioning System (ACS) is investigated in this paper. The global performances of the ACS are greatly influenced by the

behavior and the inner shapes of two heat exchangers. These inner shapes have been optimized by developing a knowledge base taking into account the physical behavior of the components of the system. The numerical treatment of the knowledge base is performed using a CSP solver. This solver takes into account discrete or continuous variables and is based on interval analysis.

## 2. KNOWLEDGE MODELING

### Knowledge modeling for embodiment design

Design process is classically developed in sequenced phases starting from need analysis, mechanical concept investigation and leading to detail design of mechanical systems. Embodiment design phase is taking place between concept analysis and detailed definition. This phase is a feasibility analysis stage and aims to define the main characteristics of mechanical systems, namely, the structure, main dimensions and performances of these systems. It is currently of major importance to support design during this phase as it is a strategic stage for innovation processes. However, embodiment design support remains challenging and there is a lack of computing tools for the embodiment of mechanical systems.

This lack is due to specific difficulties encountered during this phase. Embodiment design aims to define the structure of mechanical systems and their geometry remains ill-known at this stage. Thus, the structural model of the system is evolving and the system functioning can't be simulated using classical simulation tools. However, physics modeling must be taken into account during the embodiment design of complex mechanical systems as it determines the system performances. Designers are therefore facing with the difficulty of handling complex knowledge in a varying context and to take care of several models corresponding to different system structures.

Embodiment design platforms (see figure 1) are based on knowledge modeling and numerical treatment units. The numerical treatment unit is involved in decision support, real time and virtual environment simulations of the system. The knowledge modeling unit is concerned with physics, functional and system environment modeling. Due to the complexity of the models resulting from the knowledge modeling process, knowledge integration and qualification for the numerical treatment remains challenging. Models have to be adapted to numerical treatment requirements and this adaptation process is the bottleneck of the development of embodiment design platforms.

Model adaptation process still requires expert competences and can't be computed as some fundamentals of the process are ill-known. Figure 2 presents the main design alternatives taken into account during the knowledge modeling process and the numerical treatment of the design problem. The knowledge modeling phase aims to translate functions into functioning and modeling alternatives. Due to the high number of alternatives corresponding to the mechanical system functions, designers have to perform choices to narrow down the complexity of the models considered during the numerical treatment. The numerical treatment phase is performed to determine

dimensional and structural characteristics of the system being designed.

Table 1 presents, as an illustration, some functioning alternatives related to the main function of heat exchangers, namely "Transfer heat between two fluids". These functioning alternatives are restricted by use limitations which guide designers in their choices. The correspondence between functions and functioning alternatives is currently investigated by several research teams [] and this paper is more concerned with knowledge modeling alternatives.

The approach of knowledge modeling process being proposed is based on intrinsic adaptation parameters. These parameters are the intrinsic precision, exactness, parsimony and specialization related to any knowledge. Model adaptation is performed by adjusting these parameters in consistency with decision support or real time simulation objectives. By modifying these intrinsic parameters, the model expression is evolving whereas the corresponding knowledge remains constant.

### Decision support applications

Real time simulation and decision support for the design of complex mechanical systems are facing with similar difficulties deriving from model complexity. Few variables are used by decision support or real time applications in design whereas models resulting from design knowledge entail many variables and relations. These variables derive from model discretization process as complex models of mechanical systems take into account several scales inside the system.

System scales derive from the technical organization chart of the mechanical systems (see figure 3). These systems are organized by assembling functional blocs deriving from the system definition at a conceptual level (level 1) and beyond this level (2 to n) technical choices are performed, which tends to complexify the system. Model complexity of a mechanical system derives from interactions between the levels of the system organization. The scale of elements in level "n" sometimes correspond to representative elementary volumes derives from the discretization of differential models, which describe physics phenomena at very small scales. In such cases, the number of elements or components contributing to the system functioning tends to be high and the corresponding model might involve a high number of relations and variables.

Real time simulation and decision support applications are concerned with a few numbers of variables within complex models. In this case, most of the variables appear to be intermediary variables used to link observation variables, which concern the design process and the choices performed by designers. Observation variables may be divided in two categories. The first category consists of design evaluation variables used to give an assessment of the mechanical system performances being designed. For instance, thermal efficiencies are design evaluation variables used to determine energetic system performances. The second category contains the design variables, namely, variables used to define the system characteristics. In particular, these definition variables are used by real time virtual applications to represent and visualize the system in its virtual environment. More generally, these variables are related to the system definition from the design point of view.

The amount of intermediary variables of a model has to be restricted to limit the model complexity and the number of operations required by its numerical treatment. Throughout design processes, this is performed by adapting knowledge modeling to design requirements and numerical treatment necessities. Model complexity is managed by adjusting the model precision and exactness but also the extent of the range of application of the model. In the following paragraphs these intrinsic parameters of models are called Precision, Exactness, Parsimony and Specialization (PEPS parameters). These intrinsic parameters are defined as follows:

- Parsimony is an inverse measurement of complexity. The parsimony of a model is often roughly estimated as the inverse number of relations and coupled variables involved in this model. Parsimony may also be more precisely calculated using the order of magnitude of the number of operations required to solve the model.
- The Specialization of a model consists of the whole hypothesis and data restricting its application scope.
- The Precision of a model is related to the vagueness of the variables and parameters of the model. In the later paragraphs, variable precision is taken into account through the width of interval values.
- The Exactness of a model is related to its realism, namely, the deviation between the model and reference models such as models based on experimental analysis.

### Knowledge modeling

The PEPS parameters of models are the intrinsic parameters handled by designers during the knowledge modeling phase in embodiment design. Designer structures his knowledge by defining models (variables linked to relations and domains) and by looking for a suitable balance between PEPS intrinsic parameters of the models being defined. As an illustration, table 2 displays four models related to energy balance in heat exchangers. The four models are concerned with the same objective as they aim to determine the outlet temperatures of the fluids flowing through heat exchangers. Model 1 is a very imprecise but very parsimonious model as it consists of only two relations. Model 2 is based on an analytical integration method (NTU-Efficiency method) and appears to be fairly precise and parsimonious. Nevertheless, the analytical integration exploits some hypothesis related to the topology of fluid circulation inside the heat exchanger, limiting the range of application of the model and thus increasing the model specialization. The imprecision of the NTU-Efficiency model is due to a heat exchange global parameter ( $\kappa^*$ ) related to complex physics phenomena occurring at small scale inside the exchanger. The third model is based on the numerical approximation of partial differential equations taking into account complex fluid mechanics and heat transfer phenomena. This type of model may integrate thousands of equations or intermediary variables and has a very low parsimony. Despite the fact that the third model is based on equations having a large application range, it has been highly specialized during the PDE approximation process as it is linked to the geometry of the heat exchanger and to the fluid properties. The last model is an experimental model. Its range of application is highly restricted by the experimental conditions (fixed topology, geometry, fluids, mass flows, temperatures, etc.). Thus, it is highly specialized but, on the other hand, appears to be very parsimonious.

The previous illustration shows off that knowledge of a designer may be interpreted in different manners leading to different models. Knowledge interpretation is related to PEPS parameters which are interdependent. The designer manages the interpretation of its knowledge by increasing the precision, exactness and parsimony of the model being defined and by decreasing the model specialization. However, PEPS parameters can't be individually improved without affecting the other parameters. PEPS parameters are confined inside a convex domain preventing the knowledge modeling to converge towards an ideal model extremely precise, exact, parsimonious and lowly specialized. This can be summarized by the following

- PEPS conjecture: "The intrinsic exploitability of a model only depends on four parameters characterizing the model parsimony, exactness, precision and specialization (PEPS parameters). The more, parsimonious, exact, precise and lowly specialized is the model, the more it is exploitable".
- PEPS evolution law: "Knowledge modeling process aims simultaneously to express knowledge and manage the intrinsic exploitability of the models being defined. The exploitability of models is managed by adjusting the PEPS parameters related to these models. Nevertheless, PEPS parameters are confined inside a convex domain called intrinsic PEPS domain of models. None of the PEPS parameters related to a model can be individually and indefinitely improved without altering some of the other parameters".

Table 3 displays extreme values reached by the PEPS parameters, namely the extreme limits of the definition intrinsic domain of a model.

The intrinsic PEPS domain of models is completed by an extrinsic domain related to the environment of the model. This environment takes into account the numerical treatment of the model (algorithms, computing performances, etc.) and the design environment linked to the model (delays, design requirements, etc.). The extrinsic domain characteristics may be investigated as the intrinsic domain ones. This domain includes the point of infinite parsimony, exactness and precision and of nil specialization, seeing that a model characterized by such PEPS parameters is obviously exploitable. Starting from a model already exploitable, it is always possible to use a model related to better PEPS parameters. By increasing the model parsimony, its numerical treatment is facilitated; by increasing its exactness the model leads to more exploitable results; by increasing its precision, the probability that these results support designer decisions increases. By decreasing the model specialization, the model range of application enlarges and the model is more exploitable. Moreover, the extrinsic exploitability domain of models does not include the point of nil parsimony, exactness and precision and of infinite specialization, seeing that models related to such PEPS parameters are not exploitable.

Figure 4 outlines the extrinsic and intrinsic exploitability domains of models. Knowledge modeling process performed by designers aims to express their knowledge through models and to find a satisfactory compromise between PEPS parameters of the model being built. This compromise search endeavors to assess a model related to PEPS parameters at the intersection between the extrinsic and intrinsic exploitation domains.

The development of embodiment design platforms for decision support and real time simulation is faced with the difficulties of intrinsic exploitability management of models in the knowledge modeling process. This process is still performed by designers, which are able to take into account the parsimony, precision exactness and specialization of models and meet an exploitable model.

### 3. MODELING PROCESS

#### Functional modeling

The management of the intrinsic exploitability of models for the embodiment design of mechanical systems has to be performed from functional modeling to physics modeling of systems. Functional modeling leads to the definition of functional blocks and components (technical organization chart) characterizing a hierarchy between the system elements and therefore, between the variables involved in the system modeling. This functional organization is completed by service or constraint functions of the mechanical system in its environment. These functions are related to variables at the different level of the system organization.

The main variables of a design problem are those which have been defined at the inlet or outlet of the functional blocks or components at the upper levels of the system technical organization chart (see figure 3). These main variables also come out of the definition of the system functions and some of them may be related to the lowest levels of the system organization. Main variables of design problems appear to be design evaluation or design definition variables (see § 2.2). The rest of the variables are intermediary variables that should be eliminated to improve the model parsimony.

Due to the PEPS evolution law, the parsimony of models can't be improved indefinitely without altering the model precision, exactness or specialization. This adaptation process supposes a transformation of the model structure, which appears to be a complex and difficult operation based on mathematical, physical and technical considerations.

#### Physics modeling and model adaptation methods

Most of the model adaptation methods are devoted to physics modeling. More to the point, many of them are dedicated to specific domains of physics as they are based on mathematical transformations linked to physics phenomena. However, model adaptation methods can be classified in four main categories:

- Parsimonious approximation methods are based on analytical or numerical integration approaches. These mathematical methods tend to increase the parsimony of models by increasing the model specialization. The precision and exactness of models are lowly altered throughout the adaptation process,
- Mixed methods increase the parsimony of models by adapting both model specialization and precision. These methods may be based on mathematical, physics and technological considerations to manage the precision of some of the model parameters,
- Analogical methods are based on the analogy between different physics models. Model exactness is altered

- by the physics analogy but may be improved by limiting the application range of the model or by introducing imprecise parameters within the model,
- Standardization methods are based on the restriction of the model range of application to punctual domains. These punctual domains correspond to standard configurations of the system being investigated. The standardization process highly limits the application range of the model to application cases and therefore highly increases the model parsimony and specialization.

In the following paragraphs two different adaptation methods are considered based on parsimonious approximation methods (neural network based approximation) or mixed methods (transfer units).

#### Transfer Units adaptation method

The Transfer Units method is used in the energy transfer modeling domain to investigate the behavior of complex heat exchangers. This method is based on the analytical integration of equations derived from fluid mechanics and energy transfers. It also based on the fuzzification of heat exchange parameters. Table 2 presents some heat exchanger models derived from this method.

In the framework of this method, the heat transfer coefficient “ $\kappa$ ” is regarded as an imprecise parameter taking into account the complex physical phenomena out of the scope of the model. Due to this fuzzification process, the differential equations which describe the energy transfers inside the heat exchanger may be analytically integrated. However, this integration entails the specialization of the models. Transfer Units models consist of two types of models relative to the interactions between the system elements (interaction models) and to the behavior of these elements (elementary models). The topologies of fluid circulations inside the exchanger lead to the specialization of the interaction models. Some physics phenomena such as fluid mixing inside the model elements lead to the specialization of the elementary models.

The models derived from this method are currently used by designers to dimension and define the structure of heat exchangers. Due to the parsimony of Transfer Units models they are involved in the embodiment design of most of these systems. However, this method is highly dedicated to transfer phenomena analysis because of the analytical integration process which is dependent on the structure of the differential equations that describe the transfer phenomena. On the other hand, Transfer Units methods may be regarded as network based adaptation methods and the corresponding models could be related to other types of models such as Bond Graphs for instance.

#### Neural network based approximation method

Neural network based approximation technique is a very general method used in many application domains of mechanical design. Due to some specific properties of neural networks [10], the models derived from this approach appear to be highly parsimonious. However, despite the generic character of the method, these models also appear to be highly specialized because of the approximation process.

Figure 6 presents the different phases of the method. Starting from physics and functional models, the method aims to converge towards a grey box model of the mechanical system being investigated. Grey box models integrate the physics model related to the system components inside a structure derived from the functional model. The parsimonies of the physics models of the components are adapted to balance the global parsimony of the grey box model. Thus, elementary models which appear to be lowly parsimonious are approximated by using a numerical approximation technique. Numerical approximation is based on numerical results derived from the mechanical system modeling and on neural networks fitting to these results. Elementary models generally concern the behavior of some system components or functional units; however, they may also concern physics phenomena occurring inside the components.

The numerical approximation process narrows the application range of the elementary models and thus increases the specialization of the grey box model. This approximation is limited by the extent of the domain mapped by the numerical values resulting from the numerical treatment of the elementary models. Thus, grey box models are generally related to narrow validity domains and appear to be highly specialized.

#### 4. DECISION SUPPORT FOR AIR CONDITIONING IN AERONAUTICS

A knowledge base dedicated to the embodiment design of an Air Conditioning System (ACS) for aircrafts has been developed. This base takes into account the physics and functional modeling of this mechanical system. The knowledge base is coupled to a Constraint Satisfaction Problem solver called "*Constraint explorer*" (CE) based on interval analysis and devoted to problems mixing continuous and discrete variables. This solver has been developed in the framework of the CO2 project (French RNTL project Conception par Contraintes) managed by the society DASSAULT-Aviation. The constraints propagation and domain restrictions are based on the HC3 algorithm (Hull Consistency 3) developed within the LINA [11]. The TREFLE laboratory is involved in the project at the interface between design and the numerical aspects of the solver.

The knowledge base aims to optimize component choices and dimensions inside of the air conditioning system. These systems are based on an open Joule-Brayton cycle[12]. Figure 7 presents the functional bloc diagram of this mechanical system. It is mainly constituted by:

- Two cross current heat exchangers
- A turbine and a compressor coupled by a transmission shaft
- Diffusers, nozzles, gates and pipes that bring air from the outside of the aircraft and from the aircraft turboreactor into the system

The ACS works in varying conditions corresponding to the different flight points of the aircraft. In the following application, we consider an aircraft flying at an altitude of 3000 feet and flying at a speed of 0.8 Mach.

The structure of the fins inside of the heat exchangers determines the performances of the system. The fin configuration optimization appears to be very difficult and classical design tools are ineffective to support designer. In this application the exchanger fins are chosen among six different

configurations. These configurations are used for the two exchange surface of each exchanger leading to 1296 exchanger configurations.

The knowledge base includes:

- 23 thermodynamical variables (pressures, temperatures, mass flows)
- 14 geometrical and structural variables (lengths, exchange surface configurations, number of passes)
- 8 variables (efficiencies, etc.) defining the system performances
- 24 intermediary variables linking the previous 45 variables. Some of these variables are defined through matricial functions due to the model adaptation process and to the approximations of some physics laws.

The constraints of the model have been classified as:

- 26 physics functional constraints
- 9 physics constraints concerning some phenomena induced by the physics functional phenomena
- 2 constraints relative to the system regulations
- 18 standardization constraints
- 12 constraints (12 inequalities) relative to the design requirements

As an illustration figure 8 presents some important results obtained using the knowledge base. Due to the heat exchangers, the Air Conditioning System induces a drag force tending to decrease the aircraft performances. This drag force is compared to the thrust force induced by the turboreactor linked to the ACS. Some combinations of exchange surfaces appear to highly increase the system performances.

#### 5. CONCLUSIONS

An approach of knowledge modeling for real time simulation and decision support in embodiment design of mechanical systems has been proposed in this paper. Knowledge modeling appears to be the bottleneck of the development of embodiment design platforms. The knowledge modeling method is based on the management of the intrinsic Parsimony, Exactness, Precision and Specialization (PEPS parameters) of models. PEPS intrinsic parameters determine the exploitability of models. They are submitted to an evolution law preventing the knowledge modeling process to converge towards an ideal model extremely precise, exact, parsimonious and lowly specialized. Model adaptation to real time and decision support applications may be regarded as PEPS regulation techniques of models used for specific applications. These applications require more parsimonious models than classical simulation applications.

Two different model adaptation techniques have been investigated based on Transfer Units method and on neural network numerical approximation technique. Transfer Units method is based on the fuzzification of some physical parameters and on analytical integration. The models derived from this method appear to be very parsimonious and specialized to particular topologies of fluid circulations inside energy transfer systems. Numerical network approximation techniques are used in many domains of mechanical design. However, the validity domain of the grey box models resulting from this adaptation method is highly limited by the numerical approximation process. Thus, these models are very precise and parsimonious, but in counterpart they are highly specialized by

the validity domains of some sub-models involved in the grey box model definition. An illustration has been presented based on the optimized design of an air conditioning system. This optimization process shows off the ability of constraint based decision support systems to optimize some internal parameters of the system functioning.

$\Delta D$ : drag force (N)
$\varepsilon$ : efficiency (-)
$\kappa$ : heat transfer coefficient (W/m <sup>2</sup> /K)
$\psi$ : thrust force (N)

## 6. REFERENCES

- [1] Pahl G., Beitz W., **Engineering design: A systematic approach**, ISBN 3-540-19917-9, Springer-Verlag Berlin Heidelberg, 1996.
- [2] Hicks B.J., Culley S.J., An integrated modelling approach for the representation and embodiment of engineering systems with standard components, **Engineering with Computers**, Vol. 20, 1, p 96, 2004
- [3] Sehyun M., Soonhung H. Knowledge-based parametric design of mechanical products based on configuration design method, **Expert Systems with Applications**, Vol. 21, 2, pp 99-107, 2001
- [4] Scaravetti D., Nadeau J.-P., Sébastien P., Pailhès J., Aided decision-making for an embodiment design problem, proceedings of International IDMME, Bath, UK, 2004.
- [5] Thornton A., The Use of Constraint-based Design Knowledge to Improve the Search for Feasible Designs, **Engineering Application of Artificial Antelligence**, Vol. 9, No, 4, pp. 393-402. 1996
- [6] O'Sullivan, B., **Constraint-Aided Conceptual Design** PhD thesis, Professional Engineering Publishing, ISBN: 1-86058-335-0, 2001.
- [7] Incropera F. P., DeWitt D. P., **Fundamentals of Heat and Mass Transfer**, John Wiley & Sons, New York, p 644, 2002.
- [8] Sébastien P., Nadeau J.P., Aso S., "Numeric-CSP for air conditioning in aeronautics", Actes du congrès "proceedings of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics, SCI 2004", p 6, 2004.
- [9] Hugget A., Sébastien P., Nadeau J.P., Global Optimization of a Dryer by using Neural Networks and Genetic Algorithms, **AIChE Journal** , Vol 45, No 6, pp 1227-38, 1999.
- [10] Dreyfus G., Martinez J., Samuelides M., Gordon M., Badran F., Thiria S., Hérault L., **Réseaux de neurones - Méthodologie et applications**, Edit. Eyrolles, p 386, 2002
- [11] Granvilliers L., On the Combination of Interval Constraint Solvers, **Reliable Computing**, Vol 7, N°6, pp 467-483, 2001.
- [12] Pérez-Grande I., Leo T. J., Optimization of a commercial aircraft environmental control system, **Applied Thermal Engineering**, 2002; 22, pp 1885–1904.

## 7. NOTATIONS

A: exchange surface (m <sup>2</sup> )
Cp: heat capacity (J/kg/K)
NTU: number of transfer units (-)
q: mass flow (kg/s)
R: heat capacity ratio (-)
T: temperature (K)
<b>Greek letters:</b>

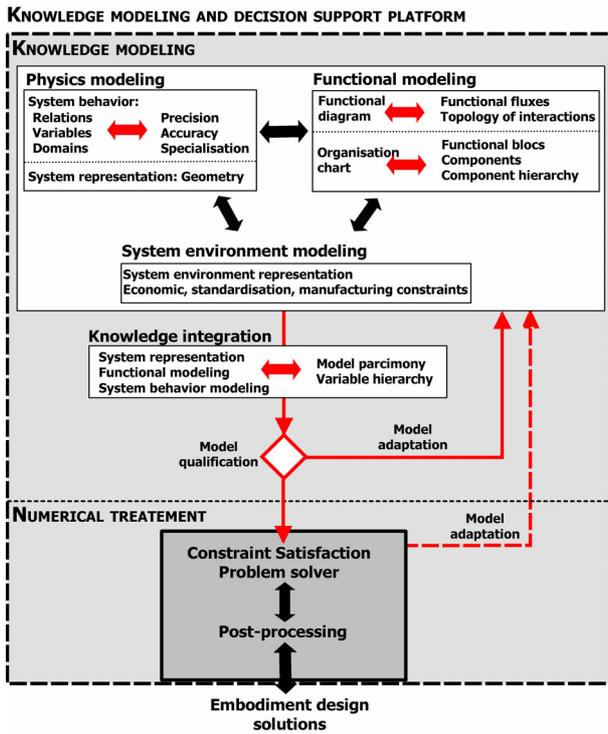


Figure 1: Embodiment design platform

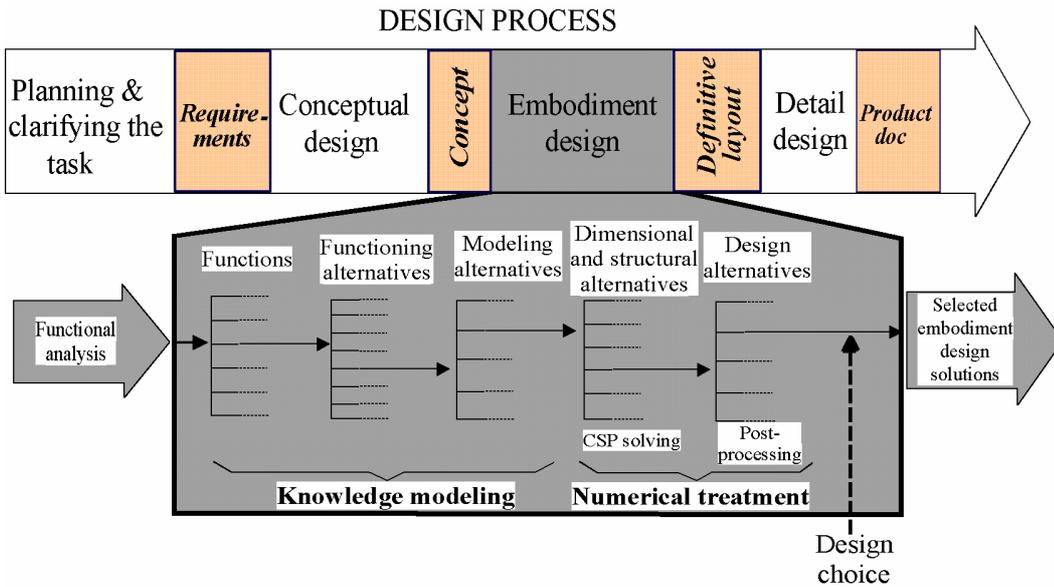


Figure 2: From product functions to selected solutions throughout embodiment design process

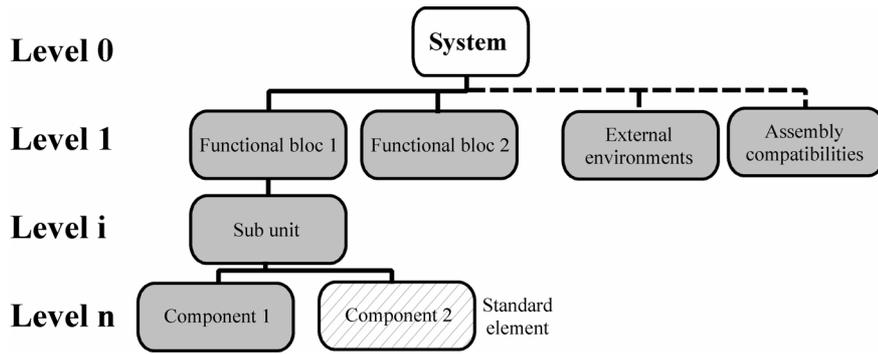
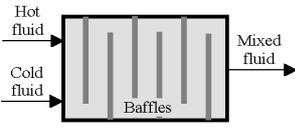
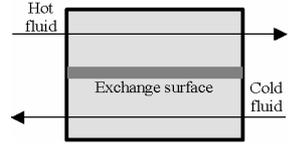
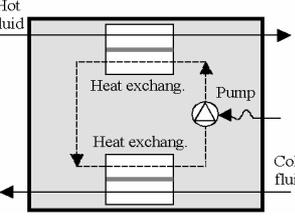
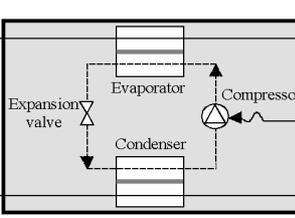
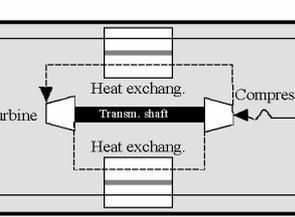


Figure 3: Technical organization chart of a mechanical system

Function	Functioning alternatives	Principle illustration	Main physics phenomena	Use limitations (*)
To transfer heat between two fluids	n°1 Fluid mixer		Fluid mixing Fluid flowing	$\Rightarrow T_{cf}^i \leq T_{cf}^o = T_{hf}^o \leq T_{hf}^i$ $\Rightarrow$ Fluids are mixed $\Rightarrow$ System power on weight is high $\Rightarrow$ Investment cost is low
	n°2 Heat exchanger		Heat convection Heat conduction Fluid flowing	$\Rightarrow T_{cf}^i \leq T_{cf}^o \leq T_{hf}^o \leq T_{hf}^i$ $\Rightarrow$ Hot and cold fluid circuits are close $\Rightarrow$ System power on weight is high $\Rightarrow$ Investment cost is low
	n°3 Indirect heat exchanger		Heat convection Heat conduction Fluid flowing	$\Rightarrow T_{cf}^i \leq T_{cf}^o \leq T_{hf}^o \leq T_{hf}^i$ $\Rightarrow$ Hot and cold fluid circuits may be distant $\Rightarrow$ System power on weight is average $\Rightarrow$ Investment cost is average
	n°4 Vapor cycle based heat pump		Heat convection Heat conduction Fluid flowing Phase change Fluid compression Fluid expansion	$\Rightarrow T_{cf}^o \leq T_{cf}^i \leq T_{hf}^i \leq T_{hf}^o$ $\Rightarrow$ Hot and cold fluid circuits may be distant $\Rightarrow$ System power on weight is low $\Rightarrow$ Investment cost is average
	n°5 Brayton cycle based heat pump		Heat convection Heat conduction Fluid flowing Fluid compression Fluid expansion Torque transmission	$\Rightarrow T_{cf}^o \leq T_{cf}^i \leq T_{hf}^i \leq T_{hf}^o$ $\Rightarrow$ Hot and cold fluid circuits may be distant $\Rightarrow$ System power on weight is high $\Rightarrow$ Investment cost is high

(\*) Inlet and outlet fluid temperatures: 

Table 1: Function “To transfer heat between two fluids” related to functioning alternatives and their corresponding use limitations.

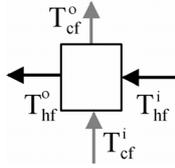
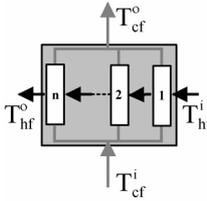
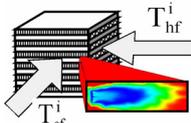
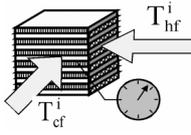
Knowledge modeling	Modeling alternatives	Model illustration	Model definition	Model qualification																													
Calculation method of the outlet temperatures of fluids flowing through an heat exchanger	n°1 Interval based modeling		$\exists T_{cf}^* \in [T_{cf}^i, T_{hf}^o] / T_{cf}^o = T_{cf}^*$ $\exists T_{hf}^* \in [T_{cf}^o, T_{hf}^i] / T_{hf}^o = T_{hf}^*$	$\Rightarrow$ Precision: low (due to $T^*$ ) $\Rightarrow$ Parsimony: high $\Rightarrow$ Specialization: low																													
	n°2 Transfer Units based modeling		$\varepsilon_1 = \frac{T_{cf}^o - T_{cf}^i}{T_{hf}^i - T_{cf}^i}; \varepsilon_2 = \frac{T_{hf}^i - T_{hf}^o}{T_{hf}^i - T_{cf}^i}$ $\exists \kappa^* \in [\kappa_{min}, \kappa_{max}] / \kappa = \kappa^*$ $R = \frac{(q \cdot Cp)_h}{(q \cdot Cp)_c}; NTU = \frac{\kappa \cdot A}{(q \cdot Cp)_h}$ $\varepsilon_1 = 1 - \prod_{i=1}^n \left( - \exp \left( \frac{\exp(-R \cdot NTU^{0.78}) - 1}{R \cdot NTU^{-0.22}} \right) \right)$ $\varepsilon_2 = R \cdot \varepsilon_1$	$\Rightarrow$ Precision: average (due to $\kappa^*$ ) $\Rightarrow$ Parsimony: average $\Rightarrow$ Specialization: average (fixed topologies)																													
	n°3 Navier-Stokes and Fourier based modeling		Discretized partial differential equ.: - Navier Stokes (fluid mechanics) - Fourier Law (heat conduction)	$\Rightarrow$ Precision: high $\Rightarrow$ Parsimony: low $\Rightarrow$ Specialization: high (fixed geometries)																													
	n°4 Experiments		Experimental measurements: <table border="1" data-bbox="786 1021 1129 1122"> <thead> <tr> <th></th> <th>323</th> <th>333</th> <th>...</th> <th>423</th> </tr> </thead> <tbody> <tr> <td>Tcf<sup>i</sup>/Thf(K)</td> <td>323</td> <td>333</td> <td>...</td> <td>423</td> </tr> <tr> <td>...</td> <td>293</td> <td>311.5</td> <td>315.5</td> <td>...</td> </tr> <tr> <td>...</td> <td>...</td> <td>...</td> <td>...</td> <td>353.3</td> </tr> <tr> <td>...</td> <td>353</td> <td>-</td> <td>-</td> <td>...</td> </tr> <tr> <td>...</td> <td>...</td> <td>...</td> <td>...</td> <td>403.6</td> </tr> </tbody> </table>		323	333	...	423	Tcf <sup>i</sup> /Thf(K)	323	333	...	423	...	293	311.5	315.5	...	...	...	...	...	353.3	...	353	-	-	...	...	...	...	...	403.6
	323	333	...	423																													
Tcf <sup>i</sup> /Thf(K)	323	333	...	423																													
...	293	311.5	315.5	...																													
...	...	...	...	353.3																													
...	353	-	-	...																													
...	...	...	...	403.6																													

Table 2: Knowledge modeling related to energy balance in a heat exchanger.

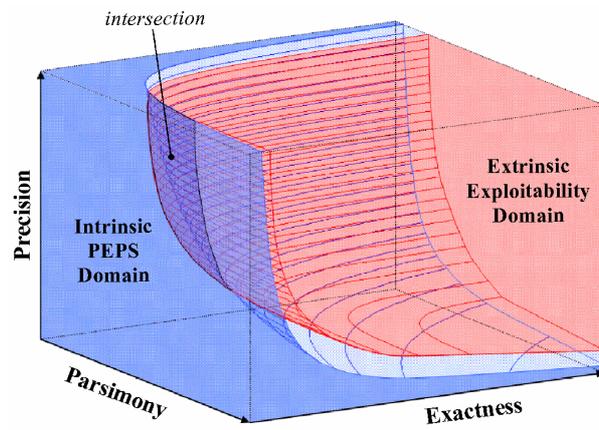


Figure 4: Extrinsic exploitability and intrinsic PEPS domains of models.

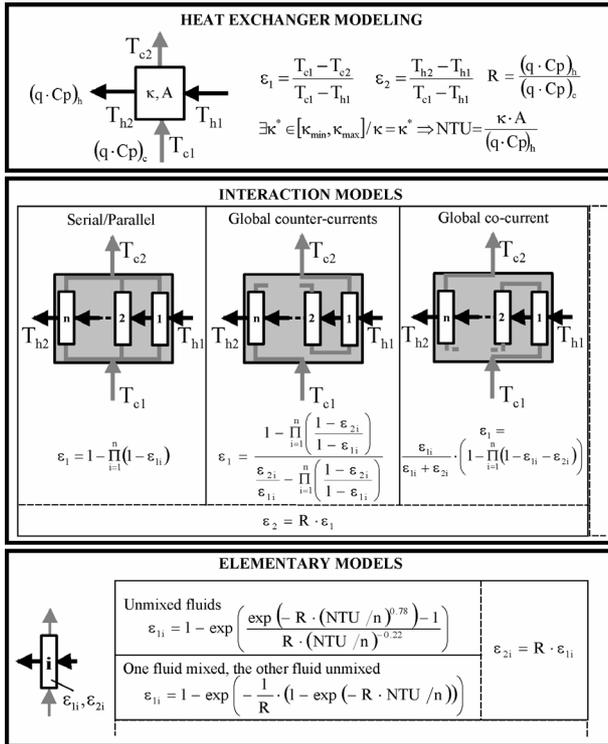


Figure 5: Heat transfer models derived from the Transfer Unit method.

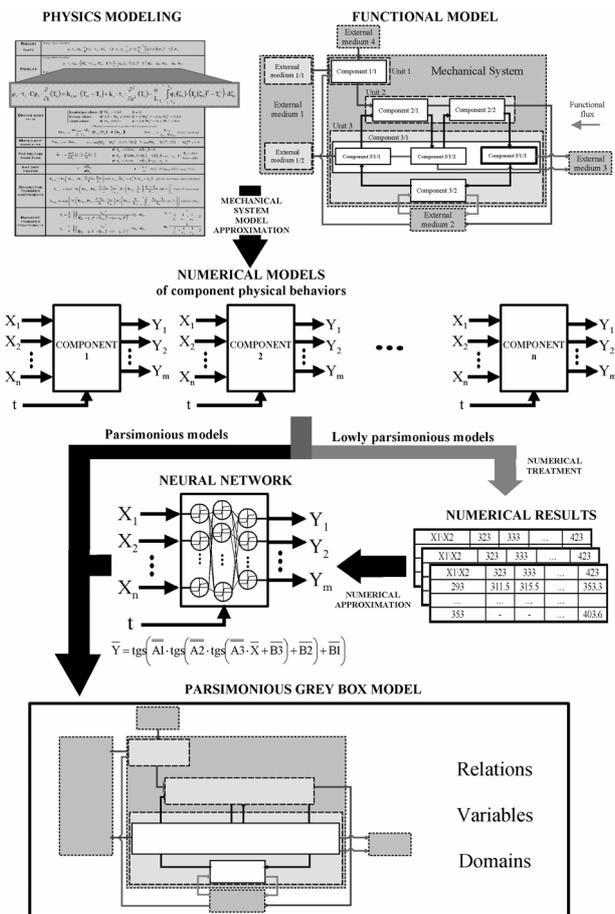


Figure 6: Parsimonious grey box models derived from the neural network approximation method.

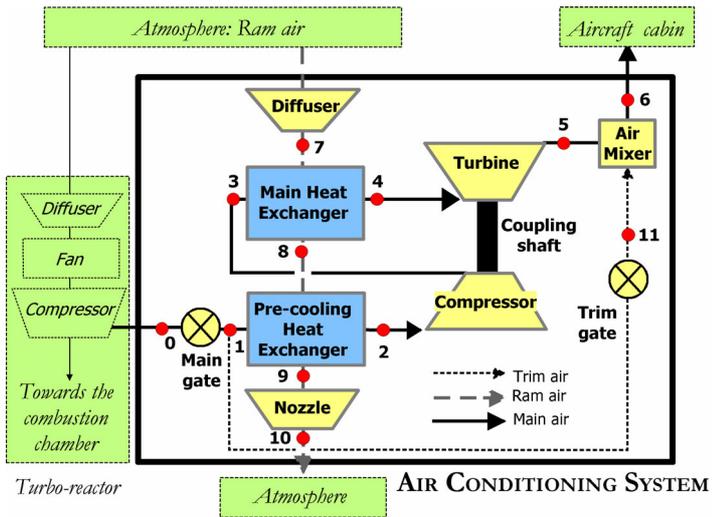


Figure 7: Functional block diagram related to an air conditioning system

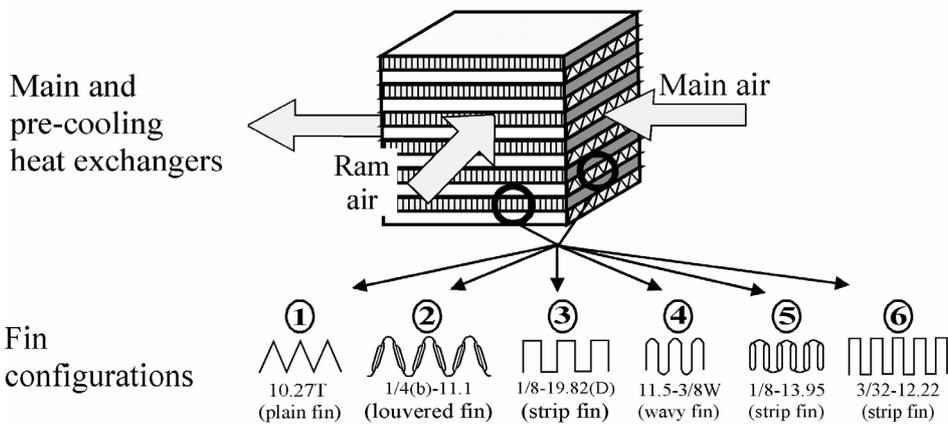
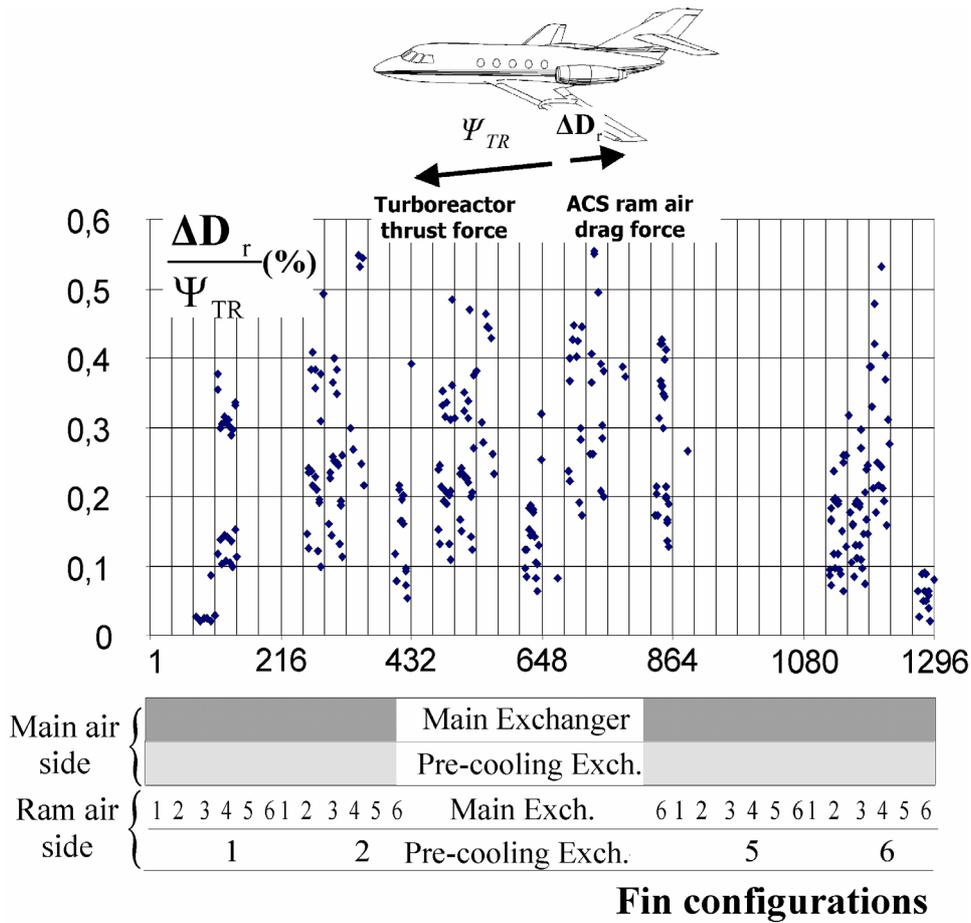


Figure 8: Drag forces due to the ram air to the turboreactor thrust versus the heat exchanger surface configurations.

<b>Parsim.</b>	<b>Exact.</b>	<b>Precis.</b>	<b>Special.</b>	<b>Interpretation</b>
Infinite	Nil	-	-	Very parsimonious models describe real behaviors with low exactness
Infinite	-	Nil	-	Very parsimonious models must be lowly precise to describe real behaviors
Infinite	-	-	Infinite	Very parsimonious models describe highly specialized real behaviors
Nil	Infinite	-	-	Very exact models must be lowly parsimonious to describe real behaviors
-	Infinite	Nil	-	Very exact models must be lowly precise to describe real behaviors
-	Infinite	-	Infinite	Very exact models describe highly specialized real behaviors
Nil	-	Infinite	-	Very precise models must be lowly parsimonious to describe real behaviors
-	Nil	Infinite	-	Very precise models describe real behaviors with low exactness
-	-	Infinite	Infinite	Very precise models describe highly specialized real behaviors
Nil	-	-	Nil	Lowly specialized models describing real behaviors are lowly parsimonious
-	Nil	-	Nil	Lowly specialized models describing real behaviors are lowly exact
-	-	Nil	Nil	Lowly specialized models describing real behaviors are lowly precise

Table 3: PEPS parameters at the limits of the exploitability intrinsic domain.

# A Specificity of CSP in Design: Controlling the Relevance of the Variables in the Problem

Thomas VAN OUDENHOVE DE SAINT GÉRY, Paul GABORIT, and Michel ALDANONDO

{vanouden, gaborit, aldanond}@enstimac.fr  
École des Mines d'Albi-Carmaux,  
Laboratoire de Génie Industriel,  
Campus Jarlard, Route de Teillet,  
81013 ALBI CT Cedex 09, FRANCE.

**Abstract.** Research in product configuration intends to provide tools for manufacturers to satisfy their clients' requirements. Among the constraint-based approaches, we study the StCSP model, which consists in associating a state attribute to each variable in order to control the relevance of the variables. After a brief presentation of our reasoning, we present our first results in resolution.

## 1 Introduction: domain and needs

Among design problems, configuration is a particular kind of design problem where the solution space is known in advance. For this kind of problem, Mittal and Frayman [1] have proved that CSP was a good candidate for modelling and solving.

The configuration process can be achieved in two different ways: autonomous or interactive. The autonomous way aims at finding at least one or all the solutions of a configuration problem. Therefore solving CSP techniques based on the BackTrack algorithm [2] are used. The interactive way consists in progressively reducing the solution space after each user' choice. In that case, filtering techniques mainly based on Arc-Consistency [3] are used most of the time.

However, almost all configurable products include optional components, meaning that this kind of component may or may not belong to the configured product. In order to take into account this kind of component in a CSP, we need a way to control that the variables of the CSP may or may not belong to the whole problem.

A very small number of studies have already tried to answer this need. The Dynamic Constraint Satisfaction Problems (DCSP, [4]) allow to activate or deactivate variables through different kinds of activity constraints. The State Constraint Satisfaction Problems (StCSP, cf. [5]) associate a state attribute to every problem variable; state attributes and variables are constrained. Another approach consists in adding a value meaning *inactive* to each domain of the CSP variables [6]. This approach is of interest but [7] have shown that StCSP can handle hierarchical models; therefore we focus upon this approach: StCSP.

These references proposed a general framework aiming to consider conditional variables in CSP. Nevertheless, they do not contain many details about the implementation of these models, and very few performance analyses. Therefore, our main goal consists in studying these models. As Veron [5] proved that a DCSP can be handled with the StCSP framework, this paper deals with a first level analysis of the StCSP propositions. Our ideas are (i) to reformulate the StCSP model in a CSP model and (ii) to modify classical CSP solving and propagation techniques thanks to the information that characterizes the StCSP model.

Our goal is to conduct an evaluation and a comparison of these two ideas. The next section describes the general idea of the evaluation. Then, we will present our first results in resolution (for autonomous configuration). The conclusion will move on to propagation techniques.

## 2 Evaluation approach

We first recall the definition of the CSP and StCSP models. Then a simple example allows to illustrate StCSP model interests. The final section describes the reformulation idea (i) and the propositions of modification (ii).

### 2.1 Definitions

The CSP [8] is a triplet  $(X, D, C)$ , where  $X$  is a set of variables,  $D$  a set of domains and  $C$  a set of compatibility constraints describing allowed combinations of variable values.

The StCSP can be defined as follows:

**Definition 1.** *A StCSP (State CSP) is a triplet  $(V, A, F)$  where:*

- $V$  is a set of variables, each one with a state attribute (whose value may be active or inactive);
- $A$  is the set of variables' domains;
- $F$  is a set of constraints over the variables, each one can be seen as a logical formula, involving at least one variable or its state attribute.

The idea of StCSP is to consider that when a state attribute is *inactive*, the variable is not of interest in the problem interpretation and is considered *inactive* (see the example of section 3.2). A consequence of StCSP is that some constraints may gather active and inactive variables. Then, it is necessary to decide about the way constraints are taken into account. According to [4]:

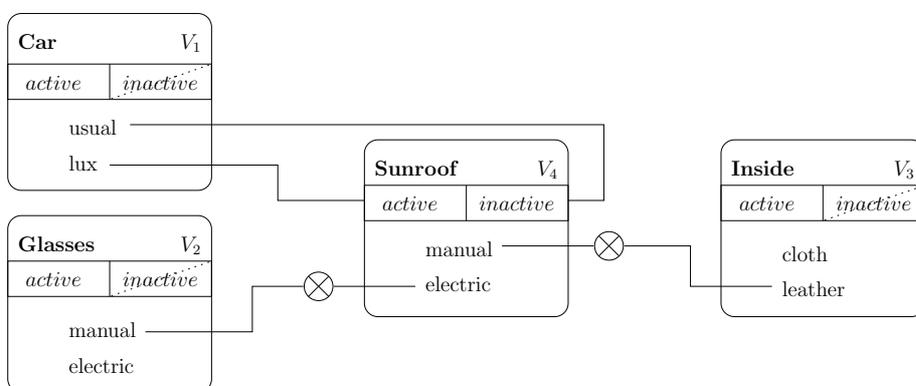
**Premise 1.** *A constraint will be taken as part of the problem if and only if every variable involved in it is active; in other cases, the constraint is considered satisfied.*

The definition of a solution of a StCSP is also necessary for a good understanding of the differences between a CSP and a StCSP.

**Definition 2.** *A solution of a StCSP is an instantiation of all the state attributes and of all the active variables (whose attribute is active) such as all constraints that must be considered are satisfied.*

## 2.2 Example

We show on figure 1 a simple example with an optional variable. The three variables “car”, “glasses” and “inside”, always belong to the problem. The variable “sunroof” belongs to the problem if and only if the variable “car” is valuated at “lux”.



**Fig. 1.** Example “simple car” with an optional variable

Thus, we have a StCSP modelled as follows:

- variables:  $\mathbf{V}$ : {Car ( $V_1$ ), Glasses ( $V_2$ ), Inside ( $V_3$ ), Sunroof ( $V_4$ )};
- domains:  $\mathbf{A}$ : {{lux, usual}, {manual, electric}, {cloth, leather}, {manual, electric}}};
- constraints:  $\mathbf{F}$ :

$$\begin{aligned}
 F_1 &: \forall i \in \{1, 2, 3\}, V_i \text{ active} \\
 F_2 &: (V_2 = \text{manual}) \Rightarrow (V_4 = \text{manual}) \\
 F_3 &: (V_3 = \text{leather}) \Rightarrow (V_4 = \text{electric}) \\
 F_4 &: (V_1 = \text{lux}) \Leftrightarrow (V_4 \text{ active})
 \end{aligned}$$

## 2.3 Reformulation idea

The first idea is to translate the StCSP model in a CSP, to solve the CSP problem or prune the domains, and then to come back to the StCSP formulation. In order to do so, we propose the following steps:

1. translate the StCSP in a CSP:
  - (a) each state attribute becomes a *state variable*, whose domain is  $\{active, inactive\}$ ,
  - (b) variables of the StCSP become *base variables* in the translated CSP,
  - (c) constraints are translated the way we show in premise 1 (cf. equation 1);
2. solve the translated CSP (or prune its domains), using known techniques (BackTrack, Arc-Consistency,...);
3. one of the following items, depending on the needs (resolution or propagation):
  - aggregate the CSP solutions in solutions of the StCSP,
  - modify the initial StCSP variables states and domains with the filtered domains obtained with the CSP propagation.

So, the example presented in previous section will be translated into this CSP:

- variables:  $\mathbf{X}$ : {Car ( $X_1^b$ ), Glasses ( $X_2^b$ ), Inside ( $X_3^b$ ), Sunroof ( $X_4^b$ ), Car State ( $X_1^s$ ), Glasses State ( $X_2^s$ ), Inside State ( $X_3^s$ ), Sunroof State ( $X_4^s$ )};
- domains:  $\mathbf{D}$ : {{lux, usual}, {manual, electric}, {cloth, leather}, {manual, electric}, {active, inactive}  $\forall X_i^s, i \in \{1, 2, 3, 4\}$ };
- constraints:  $\mathbf{C}$ :

$$\begin{aligned}
 C_1 : & \quad \forall i \in \{1, 2, 3\}, X_i^s = active \\
 C_2 : & \quad (X_2^s = inactive) \vee (X_4^s = inactive) \vee \\
 & \quad : \quad ((X_2^b = manual) \Rightarrow (X_4^b = manual)) \\
 C_3 : & \quad (X_3^s = inactive) \vee (X_4^s = inactive) \vee \\
 & \quad : \quad ((X_3^b = leather) \Rightarrow (X_4^b = electric)) \\
 C_4 : & \quad (X_1^s = inactive) \vee ((X_1^b = lux) \Leftrightarrow (X_4^s = active))
 \end{aligned} \tag{1}$$

## 2.4 Modification of CSP solving techniques

The second idea is similar to the first one, except that CSP solving and propagation techniques are modified in order to be able to use some information relevant to the fact that the CSP is the result of a translation of a StCSP.

In this communication we will only consider the solving problem and propose two modifications:

- m1:** BackTrack with specific heuristics on the instantiation order of variables for solving the translated CSP: the *state variables* are instantiated before *base variables*;
- m2:** modifying the BackTrack algorithm: we do not need any value for the variables whose state is *inactive*.

### 3 First results: resolution

#### 3.1 Evaluation criteria

The three following criteria are considered:

- number of solutions (it may vary if we try to reach the set of solutions of the CSP or the StCSP one);
- number of BackTracks;
- number of constraint verifications (which is the most important, as it is the longest operation).

Our proposals will be all the more efficient since the number of verifications will decrease.

#### 3.2 Evaluation examples

Two examples are considered. The first one has been presented in section 2.2. The second one is larger and has been introduced by Mittal and Falkenhainer [4] for the DCSP and used by Soinen and Gelle [9] (extended DCSP) and Veron and Aldanondo [7] (State CSP).

#### 3.3 Evaluation of the reformulation idea

Two variable ordering heuristics have been used :

**H0** — **random**: variables are instantiated in a random order.

**H1** — **CSP**: we first instantiate the most constrained variables (those which appear in most constraints), in case of equality, we choose the smallest domain (classical CSP heuristic “Most Constrained Variables”, cf. [10, 11]).

These results will be used as a basis in order to compare our proposals. The results are shown on table 1. All values in the following tables are the average results for 20 tests; this explains the decimal values.

**Table 1.** Comparison of H0 and H1 with the BackTrack algorithm

Criteria	Solutions: CSP (StCSP)	BackTracks	Constraint verifications
<b>Example “simple car”</b>			
H0 — random	12 (8)	110.2	198
H1 — CSP	12 (8)	32.8	62.4
<b>Example of the car (cf. [4, 5])</b>			
H0 — random	1072 (450)	61418.8	207335.2
H1 — CSP	1072 (450)	794.2	3824.6

### 3.4 Evaluation of the modification of CSP solving techniques

**Evaluation of the modification m1** The three following heuristics have been evaluated:

- H2** — **StCSP<sub>1</sub>**: we instantiate first the *state variables*; in the *state* or *base variable* subsets, the order is randomized.
- H3** — **StCSP<sub>2</sub>**: we use the principles of H2 and H1; the state variables and base variable are separated, and each subset is then ordered in accordance with the number of constraints their appear in (and the cardinal number of their domains if necessary).
- H4** — **StCSP<sub>3</sub>**: we use the principles of H1 and H2; we first instantiate the most constrained variables (those which appear in most constraints), in case of equality, we choose the smallest domain; if it remains equalities, we instantiate first the state variables.

The results are shown on table 2.

**Table 2.** Comparison of StCSP heuristics with the BackTrack algorithm (**m1**)

Criteria	Solutions: CSP (StCSP)	BackTracks	Constraint verifications
<b>Example “simple car”</b>			
H2 — StCSP <sub>1</sub>	12 (8)	64.8	50.4
H3 — StCSP <sub>2</sub>	12 (8)	25.0	54.1
H4 — StCSP <sub>3</sub>	12 (8)	24.2	52.5
<b>Example of the car (cf. [4, 5])</b>			
H2 — StCSP <sub>1</sub>	1072 (450)	680.8	6381.0
H3 — StCSP <sub>2</sub>	1072 (450)	334.3	2680.0
H4 — StCSP <sub>3</sub>	1072 (450)	869.7	4071.2

**Evaluation of the modification m2** In this case, the modification of the BackTrack is based on the fact that we look for StCSP solutions. So we do not need to verify the constraints that act on an inactive variable (associated with a *state variable*, whose value is *inactive*).

When the algorithm instantiates a variable, it verifies whether the associated *state variable* is active or not. If this *state variable* is *inactive*, the variable is instantiated with an arbitrary value and all the constraints involving this variable are considered satisfied. Thus, we do not verify the whole domain of the variable.

This modified BackTrack algorithm is evaluated with the five previous heuristics H0, H1, H2, H3, H4, as shown on table 3.

**Table 3.** Comparison of different heuristics and random order with a modified algorithm based on BackTrack (**m2**)

Criteria	Solutions: CSP (StCSP)	BackTracks	Constraint verifications
<b>Example “simple car”</b>			
H0 — random	10.0 (8)	82.0	142.0
H1 — CSP	8 (8)	25.6	45.6
H2 — StCSP <sub>1</sub>	8 (8)	18.0	34.8
H3 — StCSP <sub>2</sub>	8 (8)	21.9	41.8
H4 — StCSP <sub>3</sub>	8 (8)	22.5	43.0
<b>Example of the car (cf. [4, 5])</b>			
H0 — random	733.6 (450)	9061.2	24474.8
H1 — CSP	450 (450)	331.0	1541.8
H2 — StCSP <sub>1</sub>	450 (450)	404.4	3904.0
H3 — StCSP <sub>2</sub>	450 (450)	231.5	1291.0
H4 — StCSP <sub>3</sub>	450 (450)	342.0	1550.0

### 3.5 Evaluation discussion

We first notice in table 3 that the number of solutions is not the same without heuristics (H0 cases). This is due to the fact that if a *base variable* is instantiated before its *state variable*, the algorithm can lead to several solutions which are equivalent for the StCSP, if the *state variable* is inactive.

From the comparison of the two first tables (tables 1 and 2), we can deduce that StCSP heuristics are of interest, specially on small problems. The fact that CSP heuristic H1 may be better than some StCSP heuristics on bigger problems is a consequence of premise 1. With our process of translation, we will get, for each constraint on the *base variable*, a constraint on the *state variable* too. As a consequence, *state variables* will appear in at least as much constraints as their corresponding *base variable*; moreover, the state domains are binary, so they are most of the time smaller than base domains. Thus, even in the CSP heuristic (H1), *state variables* will be instantiated before their corresponding *base variable*, which allows a significant reduction of constraint verifications. As we can expect, the heuristic H3 is better on large problems, because it has all advantages of both heuristics H1 and H2. Heuristic H4 is quite equivalent to H1.

Comparing the classical BackTrack algorithm and our modified version (tables 1 and 2 and table 3), the number of verified constraints is obviously much lower using this modified algorithm (**m2**). Thus, this algorithm seems quite promising for the resolution of StCSP.

Finally, on table 3, we can notice that H1 remains a good heuristic for StCSP resolution and H2 is an interesting way to enhance H1 performances, as seen in H3. We will have to test all these heuristics on realistic problems to conclude which one is the best, depending on the size of problems (number of variables and constraints, size of domains,...).

## 4 Conclusion

In product configuration and design problems, it is often necessary to control the relevance of the variables. StCSP allows this possibility.

In order to prove StCSP efficiency, we build an approach based on a translation of StCSP into CSP. Then, we deal with this CSP with classical and adapted algorithms. We presented our first results in resolution, where we noticed that using StCSP specific information leads to better results.

As a perspective, we are working on propagation capabilities of StCSP. We intend to develop algorithms for StCSP filtering, similar to Arc-Consistency.

## References

1. Mittal, S., Frayman, F.: Toward a Generic Model of Configuration Tasks. In: IJ-CAI'89, International Joint Conference on Artificial Intelligence, Detroit, Michigan (1989).
2. Golomb, S.W., Baumert, L.D.: Backtrack Programming. *Journal of the ACM* **12** (1965) 516–524.
3. Mackworth, A.: Consistency in networks of relations. *Artificial intelligence* **8** (1977) 99–118.
4. Mittal, S., Falkenhainer, B.: Dynamic Constraint Satisfaction Problem. In: AAAI'90, American Association for Artificial Intelligence, Boston, Massachusetts (1990).
5. Veron, M.: Modélisation et résolution du problème de configuration industrielle : utilisation des techniques de satisfaction de contraintes. Thèse de doctorat, Institut National Polytechnique de Toulouse, ENI Tarbes (2001).
6. Prosser, P.: Domain filtering can degrade intelligent backtracking search. In: IJ-CAI'93, International Joint Conference on Artificial Intelligence, Chambéry, France (1993) 262–267.
7. Veron, M., Aldanondo, M.: Yet another approach to CCSP for configuration problem. In: ECAI'00, European Conference on Artificial Intelligence, Workshop on Configuration, Berlin, Allemagne (2000) 59–62.
8. Montanari, U.: Networks of constraints : fundamental properties and application to picture processing. *Information sciences* **7** (1974) 95–132
9. Soiminen, T., Gelle, E.: Dynamic Constraint Satisfaction in Configuration. In: AAAI'99, Workshop on Configuration, Orlando, Floride (1999) 95–100.
10. Haralick, R.M., Elliott, G.L.: Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence* **14** (1980) 263–313.
11. Smith, B.M.: A Tutorial on Constraint Programming. Technical report, University of Leeds (1995)