The (Minimal) Specialization CSP: A basis for Generalized Interactive Constraint Processing

James Bowen

Department of Computer Science
UCC, Cork, Ireland
J.Bowen@cs.ucc.ie

Abstract. The most obvious form of user interaction in constraint satisfaction is the scenario where the user specifies parameter-to-value instantiations while the machine maintains consistency. This paper argues that, in fact, real world applications require a much more general form of user input: users should be able to assert and retract arbitrary constraints, in addition to those in the basic network, while the machine analyses the evolving network for consistency. While the usual form of interactive processing is directed at finding an example satisfactory instantiation of all the parameters (the Exemplification CSP), this paper argues that many real world applications can only be modelled by a more general form of CSP, of which the Exemplification CSP is merely a special case; the more general problem is called the Specialization CSP. The paper then considers some aspects of the computational complexity of the Specialization CSP, and of a special case called the Minimal Specialization CSP. Attention is also paid to the type of notation which should be provided to facilitate user assertion of arbitrary constraints.

1 Basic Constraint Processing

In the literature, several different definitions are given for constraint networks and constraint problems, with varying degrees of formality. In some cases, unfortunately, insufficient formality has led to confusion. Some authors, for example, have failed to distinguish between the properties of networks and the properties of problems which can be defined on networks. We shall try to avoid this confusion here.

The various types of network definition may all be regarded as variations of the following theme:

Definition 1, Constraint Network:
A constraint network is a triple $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$. $\mathbf{D}$ is a finite set of $p > 0$ domains, the union of whose members forms a universe of discourse, $\mathcal{U}$. $\mathbf{X}$ is a finite tuple of $q > 0$ non-recurring parameters. $\mathbf{C}$ is a finite set of $r \geq q$ constraints. In each constraint $C_k(T_k) \in \mathbf{C}$, T_k is a sub-tuple of $\mathbf{X}$, of arity a_k and $C_k(T_k)$ is a subset of $\mathcal{U}^{a_k}$. In $\mathbf{C}$ there are q unary

constraints of the form $C_k(X_j) = D_i$, one for each parameters X_j in $\mathbf{X}$, restricting it to range over some domain $D_i \in \mathbf{D}$ which is called the *domain of* X_j.

The following example is a network containing three parameters, X_1, X_2 and X_3, the first two parameters taking their values from the domain D_1 and the third parameter taking its values from D_2:

$$\begin{aligned}
&\mathbf{D} = \{D_1, D_2\}, \text{ with } D_1 = \{a, b, c\} \text{ and } D_2 = \{d, e, f\} \\
&\mathbf{X} = \{X_1, X_2, X_3\} \\
&\mathbf{C} = \{C_1(X_1), C_2(X_2), C_3(X_3), C_4(X_1, X_2), C_5(X_2, X_3)\}, \text{ with} \\
&\qquad C_1(X_1) = D_1,\ C_2(X_2) = D_1,\ C_3(X_3) = D_2 \\
&\qquad C_4(X_1, X_2) = \{\langle a, b\rangle, \langle b, c\rangle\} \\
&\qquad C_5(X_2, X_3) = \{\langle a, d\rangle, \langle b, e\rangle, \langle c, e\rangle, \langle c, f\rangle\}
\end{aligned}$$

A network constitutes an intensional specification of a joint possibility distribution for the values of the parameters in the network. The network above, for example, implicitly specifies the following possibility distribution for $\langle X_1, X_2, X_3\rangle$: $\{\langle a, b, e\rangle, \langle b, c, e\rangle, \langle b, c, f\rangle\}$ The joint possibility distribution implicitly defined by a network is a q-ary relation on $\mathcal{U}^q$, called the *intent*[1] of the network:

Definition 2, The Intent of a Constraint Network:
The intent of a constraint network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C}\rangle$ is

$$\Pi_{\mathbf{D},\mathbf{X},\mathbf{C}} = E_1(\mathbf{X}) \cap ... \cap E_r(\mathbf{X}),$$

where, for each constraint $C_k(T_k) \in \mathbf{C}$, $E_k(\mathbf{X})$ is its cylindrical extension[2] [8] in $\mathcal{U}^q$.

Many authors assume that the domains in $\mathbf{D}$ are all finite. It is important, however, to note that this does not have to be the case. The definitions given above admit infinite domains, implying that the universe of discourse $\mathcal{U}$ may be infinite and that the constraint relations may be infinite, thereby making it possible that the intent of a network may be an infinite relation. In finite-domain networks, the domains and constraints are usually specified extensionally, as in the example network above. In networks which contain infinite domains and constraint relations, these domains and relations must, necessarily, be specified intensionally. An intentional constraint specification is of the form $C_k(T_k) =$

[1] One reviewer of this paper asked why the intent is not called the "solution set". A network's intent is a property of the *network itself*. A solution set is a property of a *problem*. Given a network, it is possible to define several problems based on the network, each of which has its own solution set. Some of these problems may have solution sets which are reminiscent of the network's intent, but the intent is still a distinct concept which, for the sake of precision, needs its own term.

[2] The cylindrical extension of a g-ary relation into an h-ary Euclidean space, $h > g$, is an h-ary relation. Each tuple in the g-ary relation results in many tuples in the extension, one for each of the tuples in the $(h-g)$-ary space. In the network above, for example, the cylindrical extension of $C_1(X_1)$ in $\mathcal{U}^q$ is $\{$ $\langle a, a, d\rangle$, $\langle a, a, e\rangle$, $\langle a, a, f\rangle$, $\langle a, b, d\rangle$, $\langle a, b, e\rangle$, $\langle a, b, f\rangle$, $\langle a, c, d\rangle$, $\langle a, c, e\rangle$, $\langle a, c, f\rangle$, $\langle b, a, d\rangle$, ... $\langle b, c, f\rangle$, $\langle c, a, d\rangle$, ... $\langle c, c, f\rangle$ $\}$

$\{T_k | wff(T_k)\}$, where T_k is a tuple containing some of the network's parameters and $wff(T_k)$ is some well-formed formula in which these parameters are free variables.

Many different forms of constraint satisfaction problem (CSP) have been distinguished. The forms of CSP encountered most frequently in the literature can be defined in terms of a network intent as follows:

Definition 3, The Decision CSP:
Given a network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$, decide whether $\Pi_{\mathbf{D},\mathbf{X},\mathbf{C}}$ is non-empty.
Definition 4, The Exemplification CSP:
Given a network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$: return nil if $\Pi_{\mathbf{D},\mathbf{X},\mathbf{C}}$ is empty; otherwise, return some tuple from $\Pi_{\mathbf{D},\mathbf{X},\mathbf{C}}$.
Definition 5, The Enumeration CSP:
Given a network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$, return $\Pi_{\mathbf{D},\mathbf{X},\mathbf{C}}$.

The Exemplification CSP is the one most commonly addressed by algorithm researchers. (Indeed, in those cases where authors fail to be sufficiently formal and expect their readers to know what is meant by "the CSP", the Exemplification CSP is usually what is intended.) Solving the Exemplification CSP is frequently used as a surrogate for solving the Decision CSP. The Enumeration CSP is rarely addressed; it is sometimes solved analytically (in the case of infinite-domain problems) or, in the case of finite-domain problems, by finding all solutions to the Exemplification CSP.

In the case of networks which contain only finite domains, the Exemplification CSP is usually solved by search-based techniques. While there are other, so-called repair, techniques, the most commonly used search techniques are variants of tree search or some hybrid of tree search with an arc consistency algorithm, such as MAC.

Even in networks which contain infinite domains, it is usually, in practice, the case that there are also some finite domains; if not, it is usually the case that some of the constraint relations are finite, which means that consistency processing can filter the domains of the parameters that are subject to these constraints down to finite subsets of their originals. The Exemplification CSP for a network containing infinite domains can be solved by search-based techniques provided: first, there are some finite domains or some finite constraint relations; second, the constraints in the network are such that those parameters which neither have finite domains nor are subject to finite constraints are subject to functional dependence on the parameters which do.

In other infinite-domain networks, recourse must be had to analytic techniques; for certain classes of infinite-domain network, analytic techniques adequate to solve the Exemplification CSP are available; for others, they are not.

2 Motivation for Interactive Constraint Processing

CSPs, as usually posed, are intended for batch processing; the machine is intended to solve the problem autonomously. Many real-world applications, however, require interactive decision support rather than autonomous automated problem-solving. The motivations for human-guided processing vary from application to application. In some applications, the task is believed to be too hard to be completed without the provision, by the user, of additional information.

In other applications, the task admits many different solutions; in these cases, human guidance is needed to select some solution which is best, where the meaning of "best" depends on some criteria which are unknown to the machine. These additional criteria cannot be stated in the original problem specification, as additional constraints, because they are not yet known – the user can only make tentative progress towards defining these criteria as and when consequences that are implicit in the original body of constraints are revealed to him.

Applications requiring interactive constraint processing can be found in many areas. However, we will mention just one: engineering design. A design problem can be expressed as follows: given a set of requirements (ranging from the laws of physics through to customer preferences), return a description for an artifact which satisfies these requirements. In a constraint-based formulation, the requirements in a design problem would be expressed as the constraints in a network, while the space of satisfactory artifacts is implicit in the intent of the network. In real-world design applications, there are many parameters and many of these have large domains – many parameters, indeed, take their values from the real numbers, $\Re$, although, given the use of tolerances in manufacturing, $\Re$ can be approximated by the rationals Q. In such an application the search space, the cross-product of the parameter domains, is much larger than in those CSPs where autonomous search has been found practical. Thus, on the one hand, these applications are too hard for autonomous processing; external help is required. On the other hand, however, design requirements are usually very loose – they admit many satisfactory artifacts. External advice is needed to choose the "best" artifact: this external advice cannot be provided as *a priori* constraints because "best" is a subjective notion which can only be dimly perceived by an interactive user as and when the implications of the design requirments are discovered.

Both rationales for interactive constraint processing (simplifying the design problem and choosing the "best" artifact) motivate reduction of the space of possible artifacts by accepting, from the user, new constraints, additional to those in the initial network. However, the user will frequently make mistakes – he will specify constraints which, when added to those in the initial network, result in a new network whose intent is the empty set. Thus, interactive constraint processing must also enable the user to retract constraints he has previously asserted.

3 Motivation for Generalized Interactive Processing

Since the Exemplification CSP is the most commonly addressed of the basic CSPs, a reasonable first approach to interactive constraint processing might be based on this model. Various scenarios are possible. At each node of a search tree, an interactive user might tell a search program which network parameter should be instantiated next or, even, specify which value should be assigned to the selected parameter. However, this sort of "micro-management" seems to defeat the whole purpose of computerized search – the "added value" being contributed by the machine amounts to no more than recording the choices made by the user.

A more reasonable approach would use the machine to compute the consequences of the user's choices and make these available to the user, so that his subsequent choices would be better informed. In other words, the interactive process would be based on one of the hybrid search-consistency algorithms; the user could direct the search but, after each choice made by the user, the system would perform a consistency analysis of the network, so that the user's next choice would be based on a better understanding of the consequences of his previous choice(s) – he could view the newly filtered parameter domains before making his next choice of parameter and/or value. The MAC algorithm, for example, could be modified to support this style of interaction: the user would direct the search while the machine would maintain arc-consistency.

However, in many cases, this form of interaction would not allow the user to take advantage of the full range of contextual knowledge in his possession. In essence, by making search choices, the user is asserting equational constraints; when he backtracks, he is retracting a previously asserted equation. In practical applications, the user's understanding of the problem area is much richer than can be expressed as a set of equations. This is particularly true in engineering design applications. Consider, for example, the following statement, which expresses knowledge that would be used by a sophisticated designer of printed circuit boards [2]:

> Either the oscillation frequency of every crystal on the board must lie within the range testable by the available test equipment or the board must contain a divider sub-circuit.

Although a detailed examination of this statement is beyond the scope of this paper, even a cursory glance shows that it involves disjunction, something which cannot be expressed as a set of equations.

A good interactive system would allow the user to make as much of his information available to the machine as possible. This suggests that, in an interactive environment based on constraints, the user should be able to assert arbitrary constraints, rather than merely equations. As we shall see, this means that new forms of CSP are needed.

4 The Specialization and Minimal Specialization CSPs

Consider, again, the the task posed by the Exemplification CSP: given a network, return nil if the intent is empty; otherwise, return a single member, any member, of the intent. Each member of a network intent is a tuple, representing one consistent valuation for the parameters in the network; that is, such a tuple is shorthand for a set of equations, each equating one parameter to a value from the domain of the parameter. Thus, the Exemplification CSP can be restated (equivalently) as follows:

Definition 6, (A recasting of the) Exemplification CSP:
Given a network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$: return nil if $\Pi_{\mathbf{D},\mathbf{X},\mathbf{C}}$ is empty; otherwise return a set, $\mathbf{A}$, containing q equational constraints (q being the number of parameters in $\mathbf{X}$), each of these constraints equating one parameter in $\mathbf{X}$ to a value from its domain, such that $\Pi_{\mathbf{D},\mathbf{X},\mathbf{C}\cup\mathbf{A}}$ contains exactly one member.

That is, for a network with a non-empty intent, the Exemplification CSP involves finding a set of equational constraints such that, when these additional constraints are added to the network, the intent of the resultant network contains exactly one tuple.

When the Exemplification CSP is stated like this, an obvious generalization presents itself. This is: for a network with a non-empty intent, find a set of arbitrary constraints such that, when these additional constraints are added to the network, the intent of the resultant network contains exactly one tuple. This generalized form of CSP can be formalized as follows:

Definition 7, The Specialization CSP:
Given a network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$: return nil if $\Pi_{\mathbf{D},\mathbf{X},\mathbf{C}}$ is empty; otherwise return a set $\mathbf{A}$ of additional constraints such that $\Pi_{\mathbf{D},\mathbf{X},\mathbf{C}\cup\mathbf{A}}$ contains exactly one member.

This new form of CSP is called the Specialization CSP because, when the given network has a non-empty intent, the task posed by the problem is to specialize the network, to restrict it so that it admits only one consistent valuation for the network parameters. (Of course, if the intent of the given network already contains exactly one tuple, the empty set could be returned, because the given network is already specialized.)

It is important to note that a set of arbitrary constraints can contain redundancy: one constraint can imply another – the tuples which one constraint eliminates from the intent of the network can be a superset of those eliminated by another constraint; more generally, one subset of constraints can imply another subset – the tuples which one subset of constraints eliminates from the network intent can be a superset of those eliminated by another subset. It seems, sensible, therefore, to restrict our generalization of the Exemplification CSP, so as to prohibit answers which contain redundancy; that is, the answer for a network with a non-empty intent is a minimal set of constraints which, when added to the network, results in a new network whose intent contains exactly one tuple. This restricted generalization can be formalized as follows:

Definition 8, The Minimal Specialization CSP:
Given a network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$: return nil if $\Pi_{\mathbf{D},\mathbf{X},\mathbf{C}}$ is empty; otherwise return a minimal set $\mathbf{A}$ of additional constraints such that $\Pi_{\mathbf{D},\mathbf{X},\mathbf{C}\cup\mathbf{A}}$ contains exactly one member. (A set $\mathbf{A}$ is minimal if there exists no subset $\mathbf{B}$ of $\mathbf{A}$ such that $\Pi_{\mathbf{D},\mathbf{X},\mathbf{C}\cup\mathbf{B}}$ contains exactly one member.)

As in the case of the Exemplification CSP, the Minimal Specialization CSP could be solved by interactive co-operation between a user and a machine and this interactive process could be based on one of the hybrid search-consistency algorithms. However, the user is not restricted to asserting equational constraints. Instead, he can assert any constraint which is relevant to the domain of discourse – any constraint which is a subset of $\mathcal{U}^q$.

After each such assertion, the machine would perform a consistency analysis and inform the user of the result. If the intent of the newly extended network is found to contain exactly one tuple, the problem is solved. If the intent of the newly extended network is found to be the empty set, the user would retract (some subset of) his assertion(s) and assert a new constraint. If the intent of the newly extended network is found to contain more than one tuple, the user would assert a further constraint. The interaction of user-specified assertions/retractions with machine-performed consistency analysis would continue until an extended network is found whose intent contains exactly one tuple.

5 Limitations imposed by the Computational Complexity of Consistency Analysis

There is one drawback to the scenario just described: it requires very powerful consistency processing. The requirement imposed by the Minimal Specialization CSP, that a *minimal* set of additional constraints be returned, means that a complete q-consistency analysis must be performed after each user assertion (q being the number of parameters in the network). By contrast, when MAC is used in automated solution of the Exemplification CSP, only an arc-consistency analysis is performed after each search choice.

Of course, automated processing is different from interactive processing. In automated processing, all work is done by the machine and efficiency considerations are directed at minimizing the overall amount of work (all of which is done by the machine). In interactive processing, however, different considerations are relevant: on the one hand, it is important to reduce the amount of work done by the user; on the other hand, responses to user assertions/retractions must be timely, which implies that attention must still be paid to the computational complexity of the consistency processing to be performed by the machine.

This latter consideration means that we cannot expect the machine to perform a full q-consistency analysis after each assertion. This, in turn, has several implications.

5.1 Consequences of q-consistency being too hard

Minimal Specialization CSP is too hard One consequence of full q-consistency being too expensive to be performed after each assertion is that, even though the intent of the network which has resulted from a sequence of user assertions contains only one tuple, the user may not be informed of this fact by the machine. He would continue to make further assertions and, provided the later assertions do not result in an empty intent, the set of user assertions will not be minimal by the time that it is discovered that the intent does contain only one tuple. In other words, if the machine performs less than a full q-consistency analysis, the interactive process cannot guarantee to solve the Minimal Specialization CSP; instead, it can only solve the (non-minimal) Specialization CSP.

Blind-alleys A second consequence of the machine not performing a full q-consistency analysis is that the user may travel down "blind-alleys" That is, even though the network which has resulted from a sequence of user assertions has an empty intent, this fact may not be discovered by the machine and the user will continue to make further assertions before it is discovered that (some further specialization of) the network has an empty intent. The user will then have to backtrack, retracting some subset of his assertions.

Imperfect Explanations From quite wide experience of using interactive constraints-based reasoning to implement intelligent design support systems [**?**,4], this author has found that it is essential that the system be capable of explaining inconsistencies discovered during interactive processing. In other words, when an empty intent is discovered, the machine should be able to explain to the user which conjunction of constraints (some of them user-assertions and some of them part of the original network) caused the empty intent, so that the user's choice of retraction is well-informed. One approach to generating these explanations is described in [5]. Recently, other authors have also developed an interest in explanation for interactive constraint processing [6, 7, 10, 9].

It should be noted, however, that if an empty intent is not spotted immediately and, instead, is only discovered after several subsequent assertions, then the machine's explanation may not include the constraint whose assertion first caused the empty intent. This is a consequence of the fact that, when a network has an empty intent, this may have several independent causes. A network has an empty intent if some subset of the set of constraints in the network are mutually inconsistent. Different such subsets may exist. If the level of consistency processing performed by the machine is not sufficient to guarantee finding the empty intent immediately that the augmented network contains an inconsistent subset of constraints, then it is possible that, by the time, several assertions later, the empty intent finally is discovered, the network may contain several independently-inconsistent subsets of constraints. When the machine reports finding an empty intent, its explanation will inform the user about only one of these inconsistent subsets of constraints.

Persistent Inconsistency A fourth consequence of the machine not performing full q-consistency analysis arises when, after some sequence of user assertions has been added to a network, the machine computes that the intent of the augmented network contains exactly one tuple, the machine could be wrong; that is, because a full q-consistency analysis has not been performed, the augmented network may, in fact, have an empty intent.

In finite-domain networks, however, this deficiency can always be remedied by a fairly simple analysis, involving only $r^{'}$ constraint checks, where $r^{'}$ is the number of constraints in the augmented network: the parameter valuations implied by the sole remaining tuple in the computed network intent can be checked against each of the constraints in the augmented network. The same analysis will also be an adequate remedy for most infinite-domain networks.

However, it is not possible to *guarantee* that the deficiency can be remedied for infinite-domain networks: such a network could contain one or more infinite-relation constraints and such a constraint must be specified intensionally, which raises the spectre of Gödel's Incompleteness Theorem. In an intentional constraint specification of the form $C_k(T_k) = \{T_k | wff(T_k)\}$, $wff(T_k)$ can be any well-formed formula; thus, it could have embedded within it a formulation of some undecidable problem (the halting problem, for example) which means that the task of checking whether a given valuation for the parameters in T_k satisfies $wff(T_k)$ is undecidable.

Full q-consistency analysis could truthfully be promised by a system supporting interactive processing of finite-domain constraint networks. However, the harsh realities of computational complexity mean that, in the worst case on large problems, the time taken to respond to user assertions would be unacceptably long. The situation is even worse for infinite-domain networks: one consequence of Gödel's Theorem is that, in general, consistency analysis is B undecidable.

A system to support generalized interactive constraint processing cannot, therefore, claim to provide full q-consistency analysis. (Indeed, for networks which contain infinite domains, such a system cannot guarantee to provide even arc consistency.) This means that generalized interactive constraint processing can only solve the Specialization CSP rather than the more desirable Minimal Specialization CSP. Furthermore, users of generalized interactive constraint processing systems have to accept that they will encounter blind-alleys.

5.2 In practice, however

Despite the caveats expressed above, in practical applications the lack of full q-consistency analysis does not seem to cause serious difficulty.

This author has extensive experience of applying generalized interactive constraint processing to electronics design [1, 2, 4] and, less extensively, to mechanical design. Even though most such applications involved some parameters with infinite domains, the lack of full q-consistency did not cause much difficulty.

The consistency analysis performed in these applications might be termed "2.x-consistency" after each user assertion, parameter domains were filtered to remove all arc-inconsistent values and, in regions of the network which satisfied certain topological criteria, constraint relations were filtered to remove path-inconsistent tuples, with consequent filtering of parameter domains to remove all values which were arc-inconsistent with the newly more-restrictive constraint relations.

The theoretical difficulty of undecidability does not seem to arise in practical applications. None of the logic formulae used in the intensional domain/relation specifications needed for the applications mentioned above was such as to render undecidable the question of whether a given value or tuple was a member of such a domain or relation. The apparent rarity of undecidability in practical applications is probably not unrelated to the fact that, as recently as 1900, when David Hilbert posed his famous problems at the International Congress of Mathematicians in Paris, undecidability was not suspected, despite the fact that mathematics had been applied to progressivley more sophisticated engineering problems over a period of several thousand years. Gödel's 1930 Incompleteness Theorem probably should not worry engineers very much.

It is easy to contrive situations which would cause an interactive user to progress far into blind-alleys – one merely has to construct networks containing inconsistency which is just beyond the horizon of the consistency analysis performed by the machine. Such inconsistency could be in finite-domain regions of the network, in infinite-domain regions, or in regions which include both finite- and infinite-domain parameters. In the practical applications encountered by this author, however, inconsistent situations beyond the horizon of the available consistency analysis were rare. Those situations that did arise rarely involved inconsistency of degree 4 or higher. Instead, they involved arc- or path-inconsistency in network regions where intensional formulae used to specify the constraint relations involved non-linear polynomials, the exponential, logarithmic or trigonometric functions, and so on.

Indeed, even in these cases, the problem was not insurmountable. Arc- and path-consistency analysis of intensionally-specified constraints was based on the use of a library of rewrite rules to simplify intensional formulae. This library was extended in the light of experience: rewrite rules were added to overcome newly-encountered problem situations. The main problem was that, as the rewrite library was extended, the time taken to respond to user assertions lengthened.

A further feature of the design applications encountered by this author is that, despite the lack of full q-consistency analysis, the sets of additional constraints which were computed as solutions to the Specialization CSP were nearly always minimal. That is, in practice, solutions found to the Specialization CSP were nearly always also solutions to the Minimal Specialization CSP. The networks where this was not true involved, once again, regions where intensional formulae that were used to specify the constraint relations involved non-linear polynomials, the exponential, logarithmic or trigonometric functions, and so on. Once again, the difficulty was easily overcome: if appropriate additions were made to the library of rewrite rules, minimal solutions were computed.

Based on this experience, it seems possible that, with sufficient research, rewrite libraries could be compiled which would cope adequately with the forms of intentional expressions that are used in specific application areas. Although performance would degenerate as more libraries were loaded into the machine, adequate performance might be achieved if libraries were of appropriate granularity and if care were taken in selecting which libraries to load.

6 Computational Complexity

A full discussion of the computational complexity of generalized interactive constraint processing is beyond the scope of this paper. Nevertheless, a useful impression of the greatly increased complexity of generalized interactive processing can be gained by contrasting the Specialization and Minimal Specialization CSPs with the Exemplification CSP.

When a given network has an empty intent, all three of these CSPs based on the network have exactly one answer; indeed, they all have the same answer, nil. When the intent of the given network contains exactly one valuation for the network parameters, the Exemplification and Minimal Specification CSPs still have exactly one answer: the answer to the Exemplification CSP is precisely this valuation, while the answer to the Minimal Specialization CSP is the empty set. The Specialization CSP, however, has many answers: any subset of the set of all those constraints which could be redundantly added to the given network is an acceptable answer, redundancy meaning that each such constraint would not alter the intent of the network.

When the intent of the given network contains more than one valuation for the network parameters, even the Minimal Specialization CSP has far more solutions than has the Exemplification CSP – the (non-minimal) Specialization CSP, of course, has even more again. In general, given a network whose intent contains more than one valuation for the network parameters, there are far more solutions to the Minimal Specialization CSP than to the Exemplification CSP; in fact, for each solution to the Exemplification CSP there exist many corresponding solutions to the Minimal Specialization CSP.

Consider the simplest case, where the intent of the given network contains just two valuations. In this case, there are just two answers to the Exemplification CSP (each of the two valuations in the intent is an acceptable answer) but there are far more than two answers to the Minimal Specialization CSP. Consider a very simple example network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$ where

$$
\begin{aligned}
&\mathbf{D} = \{D_1\}, \text{ with} \\
&\qquad\qquad D_1 = \{a, b, c\} \\
&\mathbf{X} = \{X_1, X_2\} \\
&\mathbf{C} = \{C_1(X_1), C_2(X_2), C_3(X_1, X_2)\}, \text{ with} \\
&\qquad\qquad C_1(X_1) = D_1 \\
&\qquad\qquad C_2(X_2) = D_1 \\
&\qquad\qquad C_3(X_1, X_2) = \{\langle a, b\rangle, \langle b, c\rangle\}
\end{aligned}
$$

A simple inspection will show that the intent of this network is $\Pi_{\mathbf{D},\mathbf{X},\mathbf{C}} = \{\langle a, b\rangle, \langle b, c\rangle\}$. Thus, there are two solutions to the Exemplification CSP based on this network; these are the tuples $\langle a, b\rangle$ and $\langle b, c\rangle$. However, corresponding to each of these there are many solutions to the Minimal Specialization CSP based on this network. Consider, for example, the Exemplification CSP solution $\langle a, b\rangle$. Among the Minimal Specialization CSP solutions which correspond to this are: $\mathbf{A} = \{C_4(X_1)\}$, where $C_4(X_1) = \{a\}$; $\mathbf{A} = \{C_4(X_2)\}$, where $C_4(X_2) = \{b\}$; $\mathbf{A} = \{C_4(X_1, X_2)\}$, where $C_4(X_1, X_2) = \{\langle a, a\rangle, \langle a, b\rangle\}$; $\mathbf{A} = \{C_4(X_1, X_2)\}$, where

$C_4(X_1, X_2) = \{\langle a, b\rangle, \langle b, b\rangle\}$. There are many others, but the point has been made – given the same underlying constraint network, the Minimal Specialization CSP has far more solutions than the Exemplification CSP. (And, of course, the (non-minimal) Specialization CSP has far more solutions again.)

The fact that the Specialization and Minimal Specialization CSPs based on a constraint network have more solutions than the corresponding Exemplification CSP is related to another disparity in size – the search space for the specialization CSPs is much larger than the search space for the Exemplification CSP based on the same underlying constraint network. The number of points in the search space for an Exemplification CSP based on a given network is the product of the sizes of the domains of the parameters in the network. By contrast, the number of points in the search space for the specialization CSPs is the cardinality of the power-set of the set of all constraints that could be defined on the parameters in the network, minus the number of constraints specified in the given network.

Another way of seeing this is to consider the difference between what would happen, if systematic tree search were used to solve finite-domain instances of these problems, at a typical node in each search tree. In tree search for the Exemplification CSP, two choices have to be made at each node: which parameter to instantiate at this node and which value from the domain of this parameter to use for instantiating the parameter. Similarly, in tree search for the specialization CSPs, two choices also have to be made at each node in the tree; however, the two choices have much larger branching factors. For the first choice, instead of choosing which single parameter in the network to instantiate, it is necessary to choose which subset of the parameters in the network to constrain with the next new constraint; that is, instead of a branching factor of $|\mathbf{X}|$, we have a branching factor of $|P(\mathbf{X})|$, where $P()$ is the power-set function of a tuple and the cardinality function $||$ is overloaded to apply to both tuples and sets. For the second choice, instead of choosing one value from the domain of the parameter selected for instantiation, it is necessary to choose some subset of the cross-product of the domains for the parameters that have been selected for constraining with the next new constraint; that is, instead of a branching factor of $|C_i(X_i)|$ (where X_i was the variable chosen for instantiation), we have a branching factor of $|P(C_{i1}(X_{i1}) \times ... \times C_{ik}(X_{ik}))|$ (where X_{i1} ... X_{ik} are the variables chosen for constraining with the next new constraint). The presence of the power-set function, P, in each of these new branching factor expressions gives some impression of the dramatic increase in branching factor.

The fact that the specialization CSPs have very large search spaces might seem to indicate that they are not amenable to interactive processing. After all, users will not wish to have to search widely through the space of possible constraints that could be added to the given network.

However, the fact is that, in real-world applications (or, at least, in those design applications that have been encountered by this author), the quality of heuristic information available to experienced users is such that very little search is required. When this information is asserted as additional constraints, the result is that, even though the augmented networks are subjected merely to arc-and restricted path-consistency analysis, few blind-alleys are entered and little backtracking occurs.

7 Notation for Expressing User-Asserted Constraints

In generalized interactive constraint processing, it is essential that the user's task of expressing his application-specific knowledge as additional constraints should be as painless as possible. This requires a rich notation for expressing constraints.

Consider, for example, a constraint such as the constraint $C_4(X_1, X_2) = \{\langle a,b\rangle, \langle b,b\rangle\}$, which we saw in the previous section. While this is a very simple constraint, this expression of it is in a notation which lacks perspicuity for users who are more comfortable with their own application areas than with set notation. A higher-level notation is needed, one which can easily be customized to provide a vocabulary of symbols that are familiar to practitioners in different application areas and where these custom symbols can easily be given application-specific semantics.

Predicate logic seems to offer a formal system with the necessary levels of expressiveness and customizability. The Galileo family of interactive constraint processing systems [1–3] use a notation based on the full first-order predicate calculus. Given a new application area, it is relatively easy to develop a new application-specific vocabulary of constant, function and predicate symbols and to define the semantics of these new symbols. Users can then express their application expertise and problem-solving decisions as arbitrary first-order sentences containing these new symbols. A full description of the notation system is beyond the scope of this paper but see [1].

8 Conclusions

Real world applications require a more powerful form of user interaction in constraint satisfaction than merely allowing the user to specify parameter-to-value instantiations while the machine maintains consistency. Real world applications require that users should be able to assert and retract arbitrary constraints, in addition to those in the basic network, while the machine analyses the evolving network for consistency.

Two forms of CSP have been formulated as a basis for this form of user interaction, the Specialization CSP and the Minimal Specialization CSP. The harsh realities of computational complexity mean that the Minimal Specialization CSP is, in general, beyond the capability of practical interactive processing; generalized interactive processing systems can only commit to solving the (non-minimal) Specialization CSP. Nevertheless, experience with a range of applications in the areas of electronic and mechanical design indicate that systems which interact with the user to solve the Specialization CSP provide useful functionality. Indeed, it has been found that the heuristic information possessed by experienced users is frequently so good that the solution computed to the Specialization CSP is, in fact, minimal.

Generalized interactive processing requires provision of a richly expressive notation for interactive user-specification of additional constraints. While a full argumentation is beyond the scope of this paper, it has been, briefly, suggested that a good candidate for such a notation is that provided by the full first-order predicate calculus.

References

1. Bowen J and Bahler D, 1992. "Frames, Quantification, Perspectives and Negotiation in Constraint Networks for Life-Cycle Engineering," *Int. J. of AI in Engineering*, **7**, 199-226.
2. Bowen J and Bahler D, 1991. "Supporting cooperation between multiple perspectives in a constraint-based approach to concurrent engineering," *J. of Design and Manufacturing*, **1**, 89-105.
3. Bowen J and Bahler D, 1991, "Conditional Variable Existence in Generalized Constraint Networks," *Proc. 9th Natl. Conf. on Artificial Intelligence (AAAI-91)*, 251-256.
4. Bahler D, Dupont C and Bowen J, 1994, "An axiomatic approach that supports negotiated resolution of design conflicts in concurrent engineering" *Proc. International Conference on Artificial Intelligence in Design (AID-94)*.
5. Bowen J, 1997, "Using dependency records to generate design coordination advice in a constraint-based approach to concurrent engineering", *Computers in Industry*, 191-199.
6. Freuder E, Likitvivatanavong C and Wallace R, 2000, "A Case Study in Explanation and Implication", CP-2000 Workshop on Analysis and Vizualization of Constraint Programs and Solvers, Singapore.
7. Freuder E, Likitvivatanavong C and Wallace R, 2001, "Explanation and Implication for Configuration Problems", IJCAI-2001 Workshop on Configuration, Seattle.
8. Friedman G and Leondes C, 1969, "Constraint Theory, Part I: Fundamentals," *IEEE Transactions on Systems Science and Cybernetics*, ssc-5, 1, 48-56.
9. Junker, U, 2001, "Quickxplain: Conflict detection for arbitrary constraint propagation algorithms", IJCAI-2001 Workshop on Modeling and Solving Problems with Constraints, Seattle.
10. Jussien, N. and Barichard, V, 2000, "The PaLM system: explanation-based constraint programming", CP-2000 Workshop on Techniques for Implementing Constraint Programming Systems, Singapore.