EPiC Series in Computing

Volume 50, 2017, Pages 51–63

GCAI 2017. 3rd Global Conference on Artificial Intelligence


# Replaceability and the Substitutability Hierarchy for Constraint Satisfaction Problems

Eugene C. Freuder and Richard J. Wallace

Insight Centre for Data Analytics
Department of Computer Science, University College Cork, Cork, Ireland
{eugene.freuder,richard.wallace}@insight-centre.org

**Abstract**

Problem simplification is a topic of continuing interest in the field of constraint satisfaction. In this paper we examine properties associated with the basic idea of substitutability and show how certain forms of substitutability can be organized into a strict hierarchy. One of these properties, here called *replaceability*, has been identified by other authors as being of special interest. In this work we confirm these earlier claims and show that replaceability is significant because it is the most general property in the hierarchy that allows inferences from local to global versions of this property. To make use of this discovery, we introduce two algorithms for establishing "neighbourhood replaceability", and we present an initial experimental examination of these algorithms including ways to improve their performance through ordering heuristics and various kinds of inference.

## 1 Introduction

Simplifying problems by using inference to remove values or combinations of values has been a primary approach to combinatorial complexity in constraint satisfaction problems. The removal of inconsistencies, which will not eliminate any solutions, has been most thoroughly studied. However, often a single solution suffices, and methods such as interchangeability have been introduced, which remove some, but not all, solutions.

In practice, it can be very useful to remove values that we can identify as dispensable in some tractable or at least reasonably efficient manner, either in preprocessing before search, or dynamically during search. This is, for example, what the basic process of ensuring arc consistency does. (In that case, removing an inconsistent value will not remove all solutions for the simple reason that the value will not participate in any solution.) But we can also remove values if we know that other values can serve as replacements in at least some solutions. This is the idea of substitution, which is the topic of the present paper. A special case occurs when either value can be substituted for the other; however, the important idea for purposes of simplification is substitution and removal.

A number of properties based on equivalence or subsumption relations with respect to solutions have been identified. The first paper on this subject defined two basic properties involving either equivalence or subsumption of single values; these were called *interchangeability* and *substitutability*, respectively [2]. Since then a large collection of related properties have been proposed, and subsequent to this various dominance and incomparability relations have been established; these are summarized in [7]

Some years ago Bordeaux, Cadoli, and Mancini proposed a general framework for these and other properties [1]. They identified what they considered a key concept that they called "removeability" and that we will refer to as *replaceability*, which is a generalized form of substitutability. More recently, Freuder attempted to enlarge this framework even further, using the idea of "dispensability" [3], which simply means that removing elements of a problem will not remove all solutions.

In the present work, which can be seen as an adjunct to the work of both [1] and [3], we focus on the basic property of substitutability and its generalizations, all of which involve relaxing the conditions under which we may remove a given value (or set of values). The reason for considering this special subset of properties, which forms a relatively small subgraph in the network of properties discussed by [7], is that this set of properties has a well-formed set of interrelations so that they can be arranged in a single series (i.e. a total order, or more simply a hierarchy) in which the substitutability property pertains to progressively more values. In addition, this work extends the characterization of the replaceability property highlighted by [1], and shows that it has certain 'maximal' features within the series with regard to computability. This confirms the contention of these authors that this is, indeed, a property of special significance.

In the following sections we first review earlier work that presented generalizations of substitutability. Then we present our basic framework, centered on the concept of replaceability, but attempting to situate it within existing work. Then we introduce local forms of this property that allow us to devise working algorithms that can achieve these forms. This is followed by some experimental tests. We then discuss how replaceability can sometimes be inferred to reduce overall effort. We end with some conclusions.

## 2 Generalizations of Substitutability

A constraint satisfaction problem (CSP) involves choosing values for a set of variables, $V$, which satisfy a set of constraints, $C$, which specifies permissible combinations of values for subsets of the variables. Each value selected for variable $V_i$ must come from a domain of values $D_i$ associated with that variable. A choice of values for a subset, $S$, of the variables is an instantiation of $S$. If the instantiation is consistent with the constraints involving $S$, then we say that it *satisfies* those constraints. An instantiation of all the variables, $V$, which satisfies all the constraints is a solution.

The basic property of substitutability in CSPs was first characterised in [2]. Here, we will use an equivalent definition that corresponds to later definitions in which the focus is on the value discarded.

**Definition 1.** Given a value $v$ in domain $D_i$ of variable $V_i$, if for any solution in which $v$ appears, we can substitute $u$ and still have a solution, then $v$ has the property of *substitutability* and value $u$ is substitutable for $v$.

The same work introduced a local form of substitutability, called *neighbourhood substitutability*.

**Definition 2.** (From [2]) For two values $u$ and $v$ belonging to the domain of variable $X_i$, $u$ is *neighbourhood substitutable* for $v$ iff for every constraint on $X_i$, if $v$ satisfies the constraint then $u$ also satisfies it.

This latter property had two important features: (i) neighbourhood substitutability is sufficient (but not necessary) for full substitutability, (ii) neighbourhood substitutability can be determined in polynomial time. Proofs of these features can be found in [2].

Given this interesting property, one important question is whether we can isolate its general features. In addition to elucidating the original property itself, this may allow us to identify more general properties that may allow even greater degrees of problem simplification.

In the past, there have been a few efforts in this direction. The earliest is the work by Jeavons et al., who defined what they called "generalized substitutability" [5]. Later Bordeaux et al. defined a similar

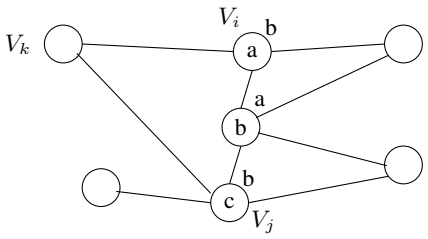


Figure 1: Example showing 3-tuple substitutability without substitutability with respect to single values of the tuple. In this figure, circles represent variables, lines constraints between variables. (So, for example, constraint $C_{ik}$ holds between variables $V_i$ and $V_k$.) Values to be substituted are inside the circles, substituted values above and to the right of each circle.

but simpler concept that they called "removability" [1]. In both cases, the basic idea is that, given value $v$ that can appear in some solution, then for any solution that $v$ appears in there is some other value that can be substituted for $v$, giving another valid solution. (A more formal definition is given in the next section and later the relation between the two approaches to this idea is discussed at length.)

In [3], the more general concept of dispensability was introduced.

**Definition 3.** An instantiation is *dispensable* if removing it will not remove all solutions to the problem. A set of instantiations is dispensable if removing them all will still not remove all solutions to the problem.

However, since this idea goes beyond the requirement of substition in any form (an instantiation can be dispensable without being related to any substitutable values), it is tangential to the main concerns of the present paper. Of more immediate relevance is the terminological problem raised by the introduction of this new concept. In this paper, Freuder pointed out that, since the ordinary meaning of removability is synonymous with dispensability, the former term is somewhat misleading in its original context. He suggested the term "onto-substitutability" instead. Here, we will use what we think is a more perspicuous term: "replaceability", which seems to have exactly the right connotations.

More recently [8] also discussed the concept of replaceability. They also introduced a generalization that they called "joint interchangeability".

**Definition 4.** A set of values $S \subseteq D_i$ is *joint-interchangeable* with a set of values $T \subseteq D_i$ iff $S$ is substitutable by $T$ and $T$ is substitutable by $S$.

Note that this is a restriction on the Jeavons et al. concept in that it deals with values in one domain.

In this paper we will not make use of generalizations of substitutability that involve $k$-tuples of values with $k > 1$ and which can involve sets of values or tuples to be replaced. This is because they entail complexities in computation and representation that will make it difficult to use them in practice. Here, we illustrate some issues when $k$-tuples are being considered for replacement.

To begin with, we can show that substitutability in terms of $k$-tuples over a variable set $R$ does not imply that individual values in the set are substitutable. This can be shown by example (see Figure 1).

Here, all domains have values $a, b, c$. Suppose that the 3-tuple $(b, a, b)$ can be substituted for $(a, b, c)$, where the assignments are made as shown in the figure. Also, suppose that constraint $C_{ik}$ includes these tuples: $\{(a, b), (a, c), (b, b)\}$, while constraint $C_{jk}$ includes these tuples: $\{(c, a), (c, b), (b, b), (b, c)\}$. An examination of the tuples will show that the only value in $D_k$ that supports both $a$ in $D_i$ and $c$ in $D_j$ is $b$. In addition, the listed tuples are consistent with the substitutability of $(b, a, b)$. However, if we consider $a \in D_i$ by itself, then we can ascertain that it is *not* neighbourhood substitutable (and hence not substitutable) with respect to $b$. Since these arguments can be extended to include each variable in the

set $R$, we can have a situation where the $k$-tuple can be replaced while this is not true for any individual value. In other words, the property of substitutability when applied a $k$-tuple is not decomposable.

One result of this is that ascertaining that a set of $k$-tuples is substitutable for another set is likely to involve some fairly unpleasant combinatorics. Consequently, it is reasonable to focus on substitutability properties with respect to single values, which is what we will do in the remainder of the paper.

# 3 A Substitutability Hierarchy

We now define the other substitutability properties in our hierarchy. First, we define the central concept:

**Definition 5.** (From [1]) An instantiation $v$ is *replaceable* if for any solution involving $v$, there is some other instantiation that can be substituted for $v$ and still have a solution.

In other words, a value is replaceable if any value can be found to substitute for it in any solution, but it is substitutable only if a single other value can be substituted for it in every solution. Obviously, this is weaker than the original substitutability property in that the latter implies replaceability, while replaceability does not imply substitutability. At the same time, it is more general since any values that can be removed on the basis of substitutability are also replaceable, but not vice versa.

Still weaker (and more general) forms of substitutability can be defined.

**Definition 6.** An instantiation $v$ has the property of *$k$-restricted substitutability* (or simply restricted substitutability) iff $v$ is in some solution, and for some subset of $k \geq 2$ solutions it is in there is another instantiation that can be substituted for it.

**Definition 7.** An instantiation $v$ is *minimally substitutable* iff if $v$ is in any solution there is some other instantiation that can be substituted in at least one solution to make a valid solution.

Minimal substitutability is a special limiting case of restricted substitutability, where $k = 1$.

The hierarchy formed by these properties is depicted in Figure 2. Inconsistency is included as a degenerate case of substitutability, since if value $v$ is inconsistent then any value in any solution can be substituted for it without losing any solution.

Note that the weakening and generality mentioned above holds over the entire hierarchy. That is, each property in the hierarchy is implied by all those below it, and it in turn implies those above it in the hierarchy. And as we proceed up the hierarchy, we can remove more instantiations. But it also becomes harder to recover the solutions lost by discarding values.

**Proposition 1.** If a value can be discarded based on a given property in the substitutability hierarchy, then it can be discarded based on any property higher in the hierarchy.

There is an important division in the series. Up to and including replaceability, the property in question is defined with respect to *all* solutions in which instantiation $v$ appears, i.e. all solutions supported by a replaceable value $v$ are still supported.

**Proposition 2.** Replaceability is the most general form of substitutability for which all solutions consistent with a discarded value are consistent with some remaining value.

**Proof.** By definition, if a value is replaceable then all its associated solutions are still supported by at least one other value in the same domain. If even one such solution becomes unsupported after discarding the value, then the associated property is restricted substitutability, which does not guarantee that all solutions are still supported. □

It can therefore be said that replaceability is a "maximal property" with respect to this feature.

Now we will show that the "generalized substitutability" property of [5] is essentially the concept of replaceability in a somewhat different form. These authors give their definition of generalized substitutability in terms of two functions $\sigma$ and $\pi$, which are selection and projection functions, respectively.

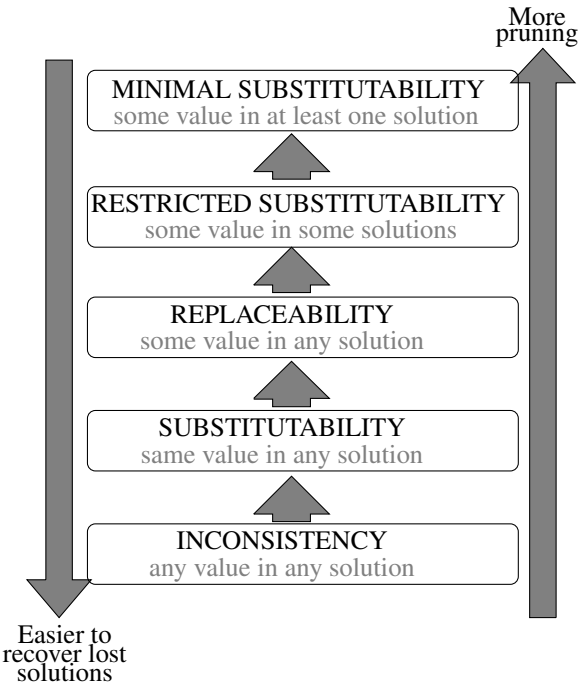


Figure 2: A substitutability hierarchy.

We are given a problem $P$ and two sets of variables $X$ and $R$, where $R \subseteq X$. If we consider a set of instantiations $S$ of $X$ and a set of instantiations $T$ of $R$, then $\sigma_T(S)$ selects the set of labels in $S$ whose projection on $R$ matches some instantiation in $T$, and $\pi_R$ applied to the output of $\sigma$ restricts the instantiations to those of the variables in $R$. Their definition is as follows:

**Definition 8.** Given a pair of instantiations $T_1$ and $T_2$ of $R$, $T_2$ is substitutable for $T_1$ (in their generalized sense) if

$$\pi_{X-R}(\sigma_{T_1}(\mathrm{Sol}(P))) \subseteq \pi_{X-R}(\sigma_{T_2}(\mathrm{Sol}(P)))$$

In words, $T_2$ can be substituted for $T_1$ if the labels for the variables other than those in $R$ (namely, $X - R$) that are associated with the selection based on $T_2$ form a superset of the labels for $X - R$ associated with the selection based on $T_1$.

That this form of substitutability is a generalized form of replaceability in our terms can be seen by setting $T_1$ in the definition to a single instantiation of $R$. In this case, $T_2$ becomes a set of instantiations of $R$ that support all of the instantiations elsewhere in the problem that $T_1$ supports. This is the same as saying that one can replace $T_1$ in any solution with some other instantiation. Thus we have the following equivalence.

**Proposition 3.** If $T_1$ is a singleton, then the instantiation in $T_1$ is substitutable by $T_2$ in the sense of [5] iff it is replaceable.

**Proof.** If the instantiation in $T_1$ is replaceable, then all of the projections on the remaining variables in the set of solutions in which this instantiation occurs can be supported by some set of instantiations of $R$. But if this latter set is chosen as $T_2$ then Definition 8 is satisfied. Conversely, any $T_2$ that is substitutable in the sense of [5] for the instantiation in $T_1$ must contain a set of instantiations that can support any

projection on $X - R$ consistent with the instantiation in $T_1$ by definition. But this is equivalent to the statement that these instantiations can replace the instantiation in $T_1$ in any solution in which it appears. □

The concept in [5] generalizes the present concept of replaceability in two ways. In the first place, $T_1$ is no longer necessarily a single instantiation, but a set, as already noted. Secondly, the formulation of [5] is more general in some ways with respect to the possible $T_2$ sets associated with a given $T_1$. For example, taking single values, consider the case where $b$ can be substituted for $a$ in all solutions, i.e. $T_1 = a$ and $T_2 = b$. Suppose further that another value $c$ in the same domain has no solutions in common with $a$. Then [5] would allow the set $b, c$ to substitute for $a$. While this situation is not ruled out by our formulation, it does involve a value that is 'irrelevant' to the definition of replaceability given above (and by [1]). This leads to the following observation.

There is a difference in perspective between [5] and the present work that is potentially significant. They view the replaceability property from the point of view of the instantiations that can substitute for a given set of instantiations. We (and [1]) on the other hand, view replaceability in terms of the instantiations that can be discarded. Despite the elegance and greater generality of the former perspective, in some ways the latter perspective is more perspicuous. It may also lead to more focused (and therefore effective) algorithms as will be seen later in this paper.

Removing instantiations based on these properties, either in preprocessing or dynamically during search, can reduce search effort. This has been amply demonstrated in practice. In general, however, determining various forms of substitutability can be as intractable as solving the problem. One way to address this is to consider restricted classes of problems, where the computation of forms of dispensability is tractable [1]. We focus here instead on situations in which a substitutability property with respect to a subproblem implies this property for the full problem, and thus the complexity is limited by the size of the subproblems we consider.

# 4 Substitutability Properties and Local Reasoning

Studies of inconsistency and substitutability have shown that tractable forms of these properties exist based on reasoning about subproblems that involve only a subset of the variables in the original problem. Here, we show that some of these ideas can be extended to more general forms of substitutability. For brevity, we restrict the discussion to subproblems based on closure (defined below), although it is also possible to reason about induced subproblems in a way that is analogous to $k$-consistency and $k$-interchangeability. However, it is not yet clear that the latter ideas can yield effective algorithms.

Bordeaux et al. [1] claim that replaceability cannot be "detected efficiently, but incompletely, through local reasoning"; this, they say, justifies the extensive use of properties like inconsistency and substitutability, which can. The paper also raises "an interesting open issue: do there exist new (i.e., other than substitutability and inconsistency) properties for which local reasoning is sound and which imply removability". However, they use a narrow definition of "local reasoning". Thus, their definition of soundness requires one to consider "all subsets of constraints $C_1 \subseteq C, ..., C_k \subseteq C$ such that $\bigcup_{i \in 1...k} C_i = C$". Soundness itself requires that for all such collections of subsets, if the property holds for all (or for some properties for at least one of them), then it also holds for the complete problem. This definition leaves open the possibility that there may be particular collections of subsets for which local reasoning is sound. Here we show that for "closure subproblems" we can define sound methods that can be applied to properties as general as replaceability.

**Definition 9.** Let S be a subset of the variables of problem $P$, and $PS$ be the subproblem induced by those variables. The *closure* of $PS$, $CPS$, is the subproblem induced by S and all the variables that share a constraint with a variable in $S$. Call the variables of $S$ the *core variables*, and the others in $CPS$ the *frontier variables.*

**Definition 10.** An instantiation of variables $S$ is *closure-replaceable* if it is replaceable with respect to the closure of the subproblem induced by $S$.

Closure-replaceability is related to the concept of local substitutability defined in [5], which again is a replaceability concept. Since our variable-based concept of closure includes any set of variables that together are the variables associated with a given constraint, the present concept is in some respects more general. On the other hand, since the [5] concept considers sets of instantiations that might be replaced, in this respect it is the more general concept.

**Proposition 4.** Closure-replaceability implies replaceability.

**Proof.** The basic idea is that it is necessary and sufficient to have 'support' in the core for every instantiation of the frontier that can appear in a solution of the closure. Any solution of the entire problem must contain a solution of the closure, and the rest of the problem only interacts directly with the closure through the frontier. Given a problem $P$, and an instantiation $v$ for variables $S$ that is closure-replaceable, where $CPS$ is the closure. Suppose $sp$ is a solution of $P$ that contains $v$; $sp$ must also contain a solution, $scps$, to $CPS$ that includes $v$. Since $v$ is replaceable with respect to $CPS$, there is some other instantiation $u$ of $S$ that can be substituted for $v$ in $scps$ yielding another solution for $CPS$. When we substitute $u$ for $v$ in $sp$ we obtain another solution for $P$, since the constraints involving $S$ are all in $CPS$. □

Of course, in general the closure can be the entire problem, but often that will not be the case. Consider, for example, binary CSP's where $S$ consists of a single variable. The closure will be the subproblem induced by that variable and all the neighbouring variables in the constraint graph. If the degree of the constraint graph is bounded by some constant $k$, or if we restrict our attention to variables with $k$ or fewer neighbours, then the complexity of the effort required to look for closure-replaceable values is correspondingly bounded.

**Definition 11.** An instantiation of a single variable $V$ is *neighbourhood replaceable* if it is replaceable with respect to the closure of $V$.

The next proposition suggests that relations analogous to that of Proposition 4 will not be possible for properties that are more general than replaceability.

**Proposition 5.** If a closure-property of an instantiation does not imply that it has a viable substitute instantiation in all valid tuples within the closure, then the closure-property cannot be extended to the entire problem.

**Proof.** If there is one valid tuple in which no substitution can be made, then if this is the only tuple that can be extended to a full solution, the property will not hold for the full problem. □

**Corollary.** Replaceability is the most general property of an instantiation that can be inferred from a closure to the entire problem. In other words, it is the maximal property for which this feature holds.

A form of singleton arc consistency (SAC) was proposed recently that is also based on the notion of closures [9]. In this form of SAC, arc consistency is established with respect to the subgraph formed by a variable and its neighbours. Here again $S$ is a single variable, $X_i$, and each of its values is considered singly. If, for a given value $a$ in the domain of $X_i$, arc consistency processing removes all values from a domain, then this value is neighbourhood inconsistent and can be removed. But being singleton neighbourhood inconsistent, it is also neighbourhood-replaceable.

**Proposition 6.** Neighbourhood replaceability implies neighbourhood singleton arc consistency.

Although space does not permit further discussion, these ideas can be extended to $n$-ary constraints, with some adjustments or exclusions for cases where a subset of neighbouring values forms a subset of the scope of a constraint that does not include the variable with the replaceable value.

```
1   Repeat
2       Set no-change to true
3       Set Q to list of all variables
4       While not empty Q
5           Remove variable V from Q; set S to neighbours of V
6           For each value v in domain of V
7               Set domain of V to {v}
8               If arc-inconsistent({v} ∪ S) or replaceable(v,S)
9                   Remove v from domain
10                  Set no-change to false
11  Until failure or no-change
```

Figure 3: CNR-1 algorithm for neighbourhood replaceability.

# 5 Neighbourhood Replaceability Algorithms

Replaceability can be computed locally in a manner analogous to the local computation of neighbourhood inverse consistency [4]. As shown in the previous section, effective reasoning on this basis is possible even with a property as general as replaceability; hence, the strategy we will focus on in this section involves using closure subproblems to deduce replaceability.

Although other proposals have been made for computing replaceability, these are either meant to find the generalized form specified by [5], or they repeatedly compose neighbourhood solutions using join operations [8], which requires more elaborate data structures and is equivalent to the all-solutions search that our algorithms perform. It is therefore unlikely that these approaches will be more efficient than the depth-first algorithms that we present. Since to our knowledge these algorithms have not been implemented and tested, the algorithms described here are the first for which this has been done.

Two algorithms have been devised for establishing neighbourhood replaceability. Both are designated as "consistent neighbourhood replaceability" (CNR) algorithms because they remove all values that are locally replaceable including arc-inconsistent values. In its full form CNR thereby incorporates neighbourhood SAC.

The first CNR algorithm, called CNR-1, uses an AC-1 style procedure in which all values in the problem are repeatedly tested for replaceability until no value is discarded. In the implementation of this algorithm, the replaceable procedure uses a MAC-style search (MAC=maintained arc consistency) to find all solutions for the subproblem, and for each solution it seeks a value from the current domain that can replace value $v$. Pseudocode for this algorithm is shown in Figure 3.

**Proposition 7.** The CNR-1 algorithm is sound in that it is correct, complete, and always terminates.
**Proof.** *Correctness.* Since neighbourhood replaceability is tested directly, the algorithm will only discard values that have this property.
*Completeness.* Since CNR-1 checks all values in the problem, any values that are replaceable in the original problem will be discovered. In addition, since CNR-1 checks all values in the problem subsequent to any deletion (i.e. in the next repeat), any values that become replaceable because of deletions will be discovered in the next iteration of the repeat loop.
*Termination.* Since by definition the problem has only a finite number of values that might be replaced, the algorithm will always terminate. □

A second algorithm, called CNRQ, uses a queue updating mechanism in place of the repeat loop used by CNR-1. Pseudocode for this algorithm is shown in Figure 4.

**Proposition 8.** The CNRQ procedure is sound and achieves the same results as CNR-1.

```
1   Set Q to list of all variables
2   While not empty Q and not failure
3       Remove variable V from Q; set S to neighbours of V
4       For each value v in domain of V
5           Set domain of V to {v}
6           If arc-inconsistent({v} ∪ S) or replaceable(v,S)
7               Remove v from domain
8               Put neighbours of V in Q if not there already
```

Figure 4: CNRQ algorithm for neighbourhood replaceability.

**Proof.** *Correctness.* Since neighbourhood replaceability is tested directly, the algorithm will only discard values that have this property.
*Completeness.* If values are replaceable on a given pass, since this property is tested for each value (line 6), such values will always be discovered. The only way in which a formerly irreplaceable value could become replaceable is for neighbouring solutions in which it participates to be lost, so that for the remaining solutions it is replaceable. This can only happen if values in neighbouring variables are discarded. But if this happens the variable with this value in its domain is put back on the queue; hence the alteration in status of the value will be detected.
*Termination.* Since all neighbourhood replaceable values will be detected the algorithm will terminate when this is accomplished and the queue is emptied or when a domain is wiped out. □

Space does not permit further discussion, but it may be noted that replaceability algorithms do not have unique fixpoints.

# 6 Some Experimental Results

The main purpose of this paper was to present a new framework for thinking about problem simplification, as well as some means of achieving it. From a practical perspective, we expect that higher-order properties such as replaceability will become more useful as the field moves away from one-shot problem solving to more long-term venues involving problem compilation and/or repeated solving in evolving situations.

However, we can learn something about the effects of testing for replaceability using single problems and one-shot search. This will allow us to determine how many replaceable values can be found in comparison with inconsistent values in various kinds of problems and determine the effect on search of removing these values.

For these tests, we used three types of problems:

1. heterogeneous random problems with "geometric" constraint graphs in which the probability of support was varied [1];
2. random distance problems, i.e. problems with constraints of the form $|X_i - X_j| > k$.
3. open shop scheduling problems with relatively small domains

Geometric problems [6] had 120 variables, domain size 20, and two levels of support: for 80% of the values tightness (obverse of support) was 0.3; for 20% it was 0.7. The (Euclidean) distance parameter was 0.17 and a target was set for 540 constraints ($\pm$ 3), giving a graph density of 0.076.

[1] Straightforward probabilistic arguments show that homogeneous random problems are unlikely to have replaceable values except in domains of variables of very low degree.

Three hundred problems were generated of which 169 had solutions; results for this subset are reported here. Distance problems had 50 variables, domain size 10, constraint graph density 0.10, and fixed $k = 3$. Scheduling problems were based on the os-taillard-4-100 series, where constraints are of the form $X_i + k_1 \leq X_j \bigvee X_j + k_2 \leq X_i$. Domains were reduced to one quarter of their original size, i.e. from about 160 to 40.

In these experiments search was done with MAC using the domain/forward degree variable ordering heuristic. This weaker-than-SOA heuristic was used in order to demonstrate problem simplification more clearly with smaller problems.

In some tests we also included neighbourhood inverse consistency (NIC) preprocessing [4]. Here, an important question is whether CNR is able to delete values above and beyond those removed by NIC. Of particular interest was the possibility that NIC-preprocessing would result in more values that are fully replaceable. CNRQ was used since it proved to be more efficient than CNR-1.

For random geometric problems, when values that are fully replaceable are discarded before search, this can have a significant effect on subsequent search effort (Table 1). In some cases, the effect exceeds that of preprocessing with arc consistency. In addition, although NIC was also quite effective, when CNR was combined with NIC, even more values were removed during preprocessing.

**Table 1. Search Efficiency and Values Removed for Geometric and Distance Problems**

| | geometric | | | distance | | |
|---|---|---|---|---|---|---|
| | rem | nodes | time | rem | nodes | time |
| AC | 38 | 55,896 | 408 | 0 | 154 | .1 |
| NSAC | 189 | 30,602 | 154 | 101 | 189 | .2 |
| CNR | 269 | 30,277 | 214 | 351 | 54 | 19 |
| NIC | 310 | 10,211 | 97 | 101 | 189 | .5 |
| NIC+CNR | 383 | 8,788 | 122 | 351 | 54 | 18 |

Means per problem: search nodes, total time (sec), values removed during preprocessing (rem).

For distance problems, CNR also simplified problems effectively, as indicated by the number of search nodes, although because these problems were easy, the overall processing time was greater. For scheduling problems, due to time constraints, only the basic AC-based algorithms were run together with CNR. In this case, AC removed 29 values per problem, and mean search nodes was 256. NSAC removed 208 values, and mean search nodes was 156. CNR removed 348 values, and mean search nodes was 92. However, because of the larger domains runtime was 30,000 sec as opposed to 0.4 and 1.8 for the other two algorithms. These tests show, however, that such problems are amenable to simplification by removing replaceable values. They also show that unless stronger forms of inference can be brought to bear, even local forms of replaceability will remain impractical. This was also found with the distance problems, where increasing domain size to 15 or 20 made it impossible for CNR to finish in a reasonable time. Hence, it is important to determine whether there are conditions where we can in fact infer replaceability or non-replaceability.

# 7 Inferring Replaceability

In fact, many kinds of intensional constraints do afford some basis for inferring whether or not a given value is replaceable. For example, if there is a binary *equality* constraint in the problem, say between $X_i$ and $X_j$, then any value $a$ of variable $X_i$ that satisfies this constraint will be associated with a unique

value $b$ in the domain of $X_j$, and any other value in the domain of $X_i$ will not support $b$. Hence, no other value can replace it. A similar argument can be made in cases where the difference between two values must satisfy an equality constraint, as with radio link frequency assignment problems (RLFAPs). For example, if $|a - b| = k$ satisfies a difference constraint between $X_i$ and $X_j$, then depending on whether $a$ is greater or smaller than $b$, only $b - k$ or $b + k$ may be substituted for $a$, respectively (assuming that all values are nonnegative).

For problems with distance constraints like the ones above, which do not involve equalities, there are several ways to avoid checking individual values. First, depending on the minimal distance, it may be possible to infer substitutability for values close to the extremes, since support for extreme values will always include the support for less extreme values. Secondly, there are symmetry relations with respect to support for values in the lower and upper halves of the range such that if a value in one half-range can be shown to be replaceable then its complement in the other half-range is also replaceable. (For brevity the proof of this is omitted.)

For problems based on relational operators such as $>, \leq, \neq$, we can also make potentially significant inferences, as shown by the following proposition.

**Proposition 9.** For problems based on relational operators whose original domains are complete integer ranges, the set of non-replaceable values in a domain is always a complete subrange.

**Proof Sketch.** Very briefly. For any variable $X_i$, the $<$ and $>$ (or $\leq$ and $\geq$) constraints establish a subrange of viable values determined by the tightest $<$ and $>$ constraints. Values in this range can be replaced by either more extreme or more central values depending on whether only one or both kinds of constraints apply. In either case, this results in a subsubrange of adjacent values. In addition, inequality constraints ensure that if the ranges of adjacent variables overlap, then the set of irreplaceable values cannot be reduced to one value. □

From this we can infer a significant corollary:

**Corollary.** For problems of the type just described, if a value $k$ in the domain of $X_i$ is replaceable, then either every value greater than $k$ is replaceable or every value less than $k$ is replaceable.

Preliminary results have been obtained with somewhat larger distance problems that do show improvement due to inference. These problems had 30 variables, domain size 15, graph density 0.25. All constraints were of the form $|X_i - X_j| > 3$. A sample of the 50 hardest problems was culled from a set of 500.

In addition to using inferences, in this case it was also possible to organize the queue so that values were removed more quickly. To this end, a heuristic was used that ordered the initial queue by ascending degree minus width (i.e. minus the constraints with variables already queued). This allowed shorter neighbourhood searches to be done at the beginning with rapid removal of some values.

Clearly, removing replaceable values greatly simplified these problems, and with variable ordering and inference methods the time was not excessive. In contrast, using the basic CNR algorithm on these problems is barely feasible.

**Table 2. Simplification and Search Efficiency for Problems with Distance Constraints**

| | rem | nodes | time |
|---|---|---|---|
| AC | 0 | 2480 | 1.2 |
| NSAC | 0 | 2480 | 1.3 |
| NIC | 32 | 1792 | 1.5 |
| CNR | $284^i$ | $45^i$ | $10{,}777^i$ |
| CNR-infer | 319 | 49 | 8.0 |

Notes. Means for 50 problems. Times in sec. "rem" is values removed.$^i$ incomplete run of 5 problems.

# 8 Conclusions

The goal of the present paper was to systematize and explore higher-level properties that can be used to simplify problems. The major theme is substitutability; here we have shown that there is a hierarchy that can be constructed around this theme. Because it is a well-behaved hierarchy, properties can be identified with certain levels that hold for all levels below but do not hold at higher levels.

We also verified earlier claims [5, 1] that the property we call replaceability has special significance. By locating this property within a hierarchy of substitutability properties, we have also clarified the nature of its significance, in that it is the most general property that can support local forms of substitutability than can be used to infer corresponding global properties. In this way, we have modified the more pessimistic assessment of [1] concerning the usefulness of this property.

Our experimental results show that, (i) algorithms for replaceability can in fact find numerous values with this property in various problem classes, (ii) this can in turn improve search efficiency in many cases. We have also shown how algorithms for finding replaceable values can sometimes be refined by inference techniques for problems with ordered domains.

However, we think that the real opportunities for using 'higher-level' properties such as replaceability remain to be discovered. For example, in cases where effort is made to compile CSPs into more compact forms such as MDDs, the task of establishing replaceability or other related properties may be obviated to a degree. (In fact, it should be possible to compile properties such as replaceability, so that instead of discarding such values, they are 'kept on hand' as potential substitutes.) Also, in dynamic situations where information from previous situations can be reused, it may be possible to amortize the cost of establishing high-level properties like replaceability, thus facilitating the process of finding new solutions to new problems. Finally, it may be possible to find approximations to full replaceability that are reasonably effective whilst being more efficient.

**Acknowledgements.** This work was supported by Science Foundation Ireland under Grant No. 05/IN/I886 and Grant No. SFI/12/RC/2289. We thank the reviewers for their perceptive comments, which definitely enhanced the quality of the paper.

# References

[1] L. Bordeaux, M. Cadoli, and T. Mancini. A unifying framework for structural properties of CSPs: Definitions, complexity, tractability. *Journal of Artificial Intelligence Research*, 32:607–629, 2008.

[2] E. C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Nineth National Conference on Artificial Intellgence – AAAI'91*, pages 227–233, 1991.

[3] E. C. Freuder. Dispensable instantiations in constraint satisfaction problems. In *Tenth International Workshop on Constraint Modelling and Reformulation – ModRef 2011*, 2011.

[4] E. C. Freuder and C. D. Elfe. Neighborhood inverse consistency preprocessing. In *Thirteenth National Conference on Artificial Intelligence – AAAAI'96. Vol. 1*, pages 202–208. AAAI/MIT, 1996.

[5] P. Jeavons, D. Cohen, and M. C. Cooper. A substitution operation for constraints. In A. Borning, editor, *Principles and Practice of Constraint Programming - PPCP'94*, number 874 in LNCS, pages 161–177. Springer, 1994.

[6] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Shevron. Optimization by simulated annealing: An experimental evaluation. part ii. graph coloring and number partitioning. *Operations Research*, 39:378–406, 1991.

[7] S. Karakashian, R. Woodward, B. Y. Choueiry, S. D. Prestwich, and E. C. Freuder. A partial taxonomy of substitutability & interchangeability. In *Tenth International Workshop on Symmetry in Constraint Satisfaction Problems - SymCon2010*, 2010.

[8] C. Likitvivatanavong and R. H. C. Yap. Eliminating redundancy in CSPs through merging and subsumption of domain values. *ACM SIGAPP Applied Computing Review*, 13:20–29, 2013.

[9] R. J. Wallace. SAC and neighbourhood SAC. *AI Communications*, 28:345–364, 2015.