Interchangeability Supports Abstraction and Reformulation for Multi-Dimensional Constraint Satisfaction

Eugene C. Freuder and **Daniel Sabin**
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
e-mail: ecf, ds1@cs.unh.edu

Abstract

Interchangeability provides a principled approach to abstraction and reformulation of constraint satisfaction problems. Values are interchangeable if exchanging one for the other in any solution produces another solution. Abstracting a problem by simplifying the constraints can increase interchangeability. Multi-dimensional constraint satisfaction problems can provide natural opportunities for this abstraction process. Multi-dimensional problems may involve vectors of values, or conjunctive constraints. Utilizing the interchangeability can permit more efficient solutions of the abstracted problem. These solutions can be expanded into smaller reformulations of the original problem. Solving abstracted and then reformulated problems can be considerably more efficient than solving the original problems. We provide data that demonstrates the potential of this abstraction/reformulation process for multi-dimensional problems, and illuminates how its utility can depend on natural problem parameters.

Introduction

Constraint satisfaction problems (*CSPs*) involve finding values for problem variables subject to constraints on which combinations of values are allowed. A *solution* is an assignment of a value to each variable such that all the constraints are simultaneously satisfied. The set of potential values for each variable will be called its *domain*. We will focus here on binary CSPs, where constraints specify pairs of allowable values.

We present a new method for abstracting and reformulating CSPs using interchangeability. Values are *interchangeable* (Freuder 1991) if exchanging one for the other in any solution produces another solution. Interchangeable values form equivalence classes. The original problem can then be viewed in terms of these equivalence classes. A limited form of interchangeability, neighborhood interchangeability, can be determined in quadratic time. This is the form of interchangeability we will employ here.

Our method applies to CSPs generally, but we focus in this paper on problems that are represented in a multi-dimensional manner. These lend themselves to abstraction: we may isolate a subset of the dimensions for consideration. CSPs may have *multi-dimensional domains*; the domain values can be vectors or tuples. For example, a school scheduling problem might involve courses as variables, to which a value pair must be assigned, representing a choice for both instructor and classroom. Multi-dimensional domain CSPs have been applied to scheduling (Yoshikawa & Wada 1992) and to configuration (Darr & Birmingham 1996). CSPs may also have *multi-dimensional constraints*; a constraint may consist of a conjunction of constraints to be satisfied. For example, the scheduling problem may require that the instructors for classes that meet at the same time not be the same person (for obvious reasons) and not both teach with video projectors (because the school only has one).

Abstraction has a long history in AI; we restrict our references here to those that address constraint satisfaction specifically. An early example is Stefik's work on Molgen (Stefik 1981). Ellman provides a general framework for abstraction for constraint satisfaction using various forms of symmetry (Ellman 1993). In more specifically CSP terms, Schrag and Miranker abstract domains for determining unsatisfiability of CSPs (Schrag & Miranker 1996). Choueiry and Faltings relate interchangeability to abstraction in the context of a decomposition heuristic for resource allocation viewed as a CSP (Choueiry & Faltings 1994). An earlier version of this paper appeared in (Freuder & Sabin 1995). Weigel and Faltings cluster variables to build abstraction hierarchies for configuration problems viewed as CSPs, and then use interchangeability to merge values on each level of the hierarchy (Weigel & Faltings 1996). Reformulation also has a long history in AI. Nadel discusses alternative CSP problem formulations (Nadel 1990). Geelen presents a rationale for switching between two views of a class of CSPs (Geelen 1992). Many CSP techniques could be regarded as re-

formulation. Classic arc consistency preprocessing, for example, could be regarded as reformulating problems into smaller ones, smaller in the sense that there are fewer potential solutions (alternative instantiations of all the variables).

Our contributions here are:

- ▷ Employing interchangeability to implement a novel form of abstraction.
- ▷ Generalizing and automating a classic instance of size-reducing reformulation.
- ▷ Applying our methods successfully to multi-dimensional problems.
- ▷ Experimentally observing the effect of problem parameters on performance.

Examples

To introduce our approach, we illustrate how interchangeability can be used to facilitate abstraction and reformulation in the context of the 4-queens problem, a version of the n-queens problem, which has often been used to investigate constraint satisfaction. It requires us to place 4 queens on a 4 by 4 corner of the chess board such that no two queens attack each other. Each queen is represented by a variable that has a domain consisting of the 16 squares it can occupy (assume the squares are numbered, with the first row numbered 1 through 4, the next 5 through 8, etc.). The constraints, one for each pair of variables, are all "not attack". Given the movements of the queen in chess, the "not attack" constraint is naturally presented as a multi-dimensional constraint: not attack vertically and not attack horizontally and not attack along "left" diagonals and not attack along "right" diagonals.

We abstract by simplifying the problem to only require that the queens do not attack horizontally, within a row. Weakening the constraints in this way introduces interchangeability. Interchangeability partitions the 16 squares into 4 equivalence classes, corresponding to the 4 rows. (This can be accomplished with a relatively efficient local computation.) We can replace each equivalence class with a single representative, without really giving up anything. For example, if there is a solution involving square 3 from the first row, we could replace square 3 with square 1 and still have a solution – in this simplified problem domain, where we only are concerned with horizontal attacks. Thus we have a much smaller search space (4^4 possibilities as opposed to 16^4). The solution 1, 5, 9, 13 is easily found. Now we replace each value in that solution with its interchangeability equivalence class (the row) and look for a solution satisfying the remaining constraints. This new problem is also relatively easy to solve. We have reduced solving the original problem with 16^4 possibilities to solving two problems each with 4^4 possibilities (and each with simpler constraints).

The really fascinating thing here is that the second problem we solve is the reformulation of the original problem that people who are experienced with the queens problem immediately employ. In testing a new algorithm, for example, one would not test it on the problem with 16 values per variable, but on the problem with 4 values per variable, where a variable essentially corresponds to a row, not the entire board. Simplification and interchangeability have automated the discovery of this problem reformulation.

If we abstracted on the column dimension, we would obtain an isomorphic result. If we abstracted on one of the diagonal dimensions, we would obtain a different reformulation. It would also be smaller than the original problem, but in this case the reformulation might not have a solution. This would require us to backtrack to the abstracted level to look for another solution there. For the row or column case we will never have to do this because any solution to the abstracted problem leads to a solution to the reformulated problem. The abstracted problem is essentially a coloring problem with 4 colors for a complete graph of 4 variables, meaning all solutions will be permutations of each other, and the reformulated problem is symmetric, so there is really only one case to consider.

The queens problem provides an example of a multi-dimensional constraint problem. To illustrate and motivate our approach for multi-dimensional domain problems, we briefly present a second example. For some real world problems the domains consist of elements that actually represent composite objects, rather than atomic values. Each particular object is then specified as a set of attribute values and the relation between two objects is naturally expressed as a conjunction of constraints between their attributes. It is possible that considering only the constraints related to one attribute, objects will cluster into equivalence classes of interchangeable values.

Let us consider a specific problem. The task is to configure a device consisting of three frames, $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$. Available are six different types of frames, $f_1, \ldots, f_6$. Each frame has a fixed number of slots and each slot can hold one module. There are two different types of modules, m_1 and m_2, but each frame with multiple slots can accommodate only modules of the same type.

Each frame type is described by a set of attributes, where the attributes of interest for us are the number of slots (SN) and the type of the modules (MT) that can be mounted on that frame. The concrete attribute values for each of the frame types are given in Table 1.

For instance, a frame of type f_4 can hold up to 2 modules of type m_1. The problem is to equip the device with frames and the selected frames with modules, where the constraints restricting the possible configurations are 1) the number of slots on frame $\mathcal{F}_1$ must be less than the number of slots on frames $\mathcal{F}_2$ and $\mathcal{F}_3$ 2) the types of modules on all the three frames have to be the same.

		SN		
		1	2	3
MT	m_1	f_1, f_2	f_4	
	m_2	f_3	f_5	f_6

Table 1: Frame types and their attribute values

A possible CSP representation for this problem would have three variables, F_1, F_2 and F_3, each with domain $\{f_1, \ldots, f_6\}$. If we consider only the two constraints on the attribute number of slots, the domain of values for variable F_1 is divided into three equivalence classes of interchangeable values: $\{f_1, f_2, f_3\}$, $\{f_4, f_5\}$ and $\{f_6\}$.

Method

The general process we propose is as follows:

1. *Abstract.* We abstract by removing or loosening constraints.
2. *Reduce.* Remove redundant neighborhood interchangeable values. We first compute neighborhood interchangeability. Neighborhood interchangeability looks for values that are consistent with the same values at other variables. A set of mutually neighborhood interchangeable values forms an equivalence class. All but one representative of each equivalence class of mutually neighborhood interchangeable values can be eliminated as redundant at this point.
3. *Solve abstraction.* Find a(nother) solution to the reduced, abstracted problem. Any standard, complete CSP algorithm can be employed. If no solution can be found, the problem is unsolvable.
4. *Reformulate.* The value for each variable is obtained by replacing the value in the abstract-level solution by its interchangeability equivalence class. The constraints are the original constraints *minus* the abstract-level constraints. In other words any values in the equivalence classes already satisfy the abstracted constraints, now we seek values that also satisfy the rest of the constraints.
5. *Solve reformulation.* Again any standard, complete CSP algorithm can be employed.
6. *Solved?* If step 5 is successful, we are done; if not, return to step 3 to seek another solution.

The process can be summarized as moving from an initial problem P, to an abstracted problem, P_a, to a reduced, abstracted problem, P_{ra}, to a reformulated problem P_r. Note that we can also apply this process recursively, using it to solve P_r. Even if we do not apply the entire process to P_r, we can look to use interchangeability to remove redundant values.

This method can easily be shown to be complete in the sense that it will find a solution if one exists. We have replaced solving one problem by solving two: an abstracted problem and a reformulated problem, and we may have to solve (or attempt to solve) more than one incarnation of each. However, abstracted and reformulated problems are likely to be simpler than the original, and in some cases, such as the row abstraction of the queens problem, any solution of the abstracted problem will lead to a solution of the reformulated problem, so we are guaranteed that we only need to solve one incarnation of each.

Interchangeability can result in smaller problems (fewer values) at both the abstract and reformulated levels. Since the constraints at both the abstract and reformulated levels are fewer or looser than the original constraints there is more likelihood of interchangeability. Simpler or fewer constraints may also mean that the problems are easier to solve. This could result in an overabundance of solutions at the abstract level, but interchangeability can help us cope with this, by allowing an individual solution to represent the large class of solutions that would be obtained by substituting interchangeable values.

Multi-dimensional problems naturally lend themselves to this abstraction and reformulation process. For multi-dimensional constraints we can remove a subset of the dimensions, as we removed all but the row constraints in the queens example. For multi-dimensional domains we can remove constraints involving a subset of the dimensions, as we removed constraints not involving slots in the configuration example. It is quite reasonable to expect the constraints to be presented to us in multi-dimensional form (though future work might investigate automatically partitioning constraints into a multi-dimensional form). We can abstract on any of the dimensions (though future work might develop heuristics for choosing dimensions to abstract on, e.g. those that maximize interchangeability), or we can recursively abstract out one dimension at a time.

Experiments

We first tested the method on the n-queens problem, for varying values of n. Next we defined a class of multi-dimensional constraint problems, MC, and generated a variety of problems from that class. The third set of experiments was conducted on a class of problems with multi-dimensional domains, MD. For MC and MD we generated problems varying several key parameters and observed the effect of these parameters on the effectiveness of our method.

Queens Problems

We tested the abstraction process on the n-queens problem for increasing n. The results are shown in Table 2. The comparison was between a good CSP search algorithm running alone on the problem, and that same algorithm used in conjunction with abstraction and reformulation. The search algorithm employed was for-

Algorithm	Q	CChecks	Bktrks	Nodes Visited	Values Deleted		Time [sec]
					Ichg	Fwd	
FC-DMD	3	299	9	25	0	152	0.002
	4	231	7	18	0	136	0.001
	5	262	0	4	0	136	0.001
	6	19,669	730	1,438	0	7,882	0.089
	7	1,053	1	10	0	256	0.003
	8	351,221	12,583	25,942	0	136,923	1.851
	9	96,320	2,900	6,344	0	37,390	0.518
	10	1,757,641	61,661	125,070	0	673,968	10.416
	11	435,492	12,513	25,911	0	163,655	2.615
	12	30,038,489	996,164	2,023,367	0	11,276,472	194.198

Algorithm	Q	CChecks		Bktrks			Nodes Visited		Values Deleted				Time [sec]
									Ichg		Fwd		
		S†	R†	S†	I†	R†	S†	R†	S†	R†	S†	R†	
ABS&REF FC-DMD	3	30	108	10	6	18	15	30	18	0	17	74	0.003
	4	20	38	0	0	2	4	7	48	0	6	20	0.001
	5	40	34	0	0	0	5	4	100	0	10	14	0.001
	6	70	230	0	0	13	6	26	180	0	15	96	0.003
	7	112	92	0	0	0	7	6	294	0	21	30	0.005
	8	168	777	0	0	44	8	74	448	0	28	291	0.013
	9	240	362	0	0	10	9	24	648	0	36	115	0.019
	10	330	569	0	0	16	10	34	900	0	45	168	0.031
	11	440	987	0	0	31	11	59	1,210	0	55	295	0.056
	12	572	2,375	0	0	95	12	152	1,584	0	66	713	0.089

† S - simplified problem, I - between problems, R - reformulated problem

Table 2: Results for the n-queens problems

ward checking using a dynamic variable search order based on minimal domain size (FC-DMD). ABS&REF+FC-DMD basically uses the method presented in the previous section (there is some sophistication involved in moving back and forth between the abstract and reformulated levels).

We note values deleted by interchangeability (Ichg) and by forward checking (Fwd). (Here and in the next section, the efficiency of the search algorithm running alone could be improved by first intersecting the two constraint components to form a single constraint, but that should at best yield a factor of two speedup, and the advantages of abstraction and reformulation often far exceed that.)

When using abstraction and reformulation we distinguish effort made in solving simplified (abstracted) problems, in solving reformulated problems, and in moving between abstract and reformulated levels. In the case of the 3-queens problem, where there is no solution, the program backs up to the abstraction level repeatedly in an effort to find a solution at this level that will lead to a complete solution after reformulation. In the other cases, the first reformulation leads to a complete solution to the problem.

As n increases the advantage of the abstraction technique also increases. Constraint checks, backtracks and nodes visited in the search tree were counted, and CPU time measured (on a DEC Alpha 300XL). Constraint checks are a standard measure of CSP effort; a constraint check involves asking whether a value a for a variable X is consistent with a value b for a variable Y. Interchangeability was determined efficiently using a matrix representation of the constraints. Constraint check counts do not reflect the interchangeability computations, however those computations are, of course, included in the CPU time results.

Multi-Dimensional Constraints

Problems in MC have constraints that are a conjunction of two components. The first component is generated in a manner that permits us to specify the degree of interchangeability present. The second component is generated in a manner that permitted us to specify the tightness of the component (how constraining it is). A random element in the constraint generation process permits us to generate a variety of problems meeting a given set of specifications. By varying the interchangeability and tightness specifications, as well as the specification of the underlying density of the problem (roughly speaking how many constraints were present), we were able to generate a wide variety of problems, and observe the effects of the interchangeability, tightness and density specifications on the performance of our method.

The tests we conducted addressed the problem of finding a single solution to a CSP (or determining that no solution exists). We assume that constraints are expressed as a conjunction of two components and the abstraction process simplifies the problem by only considering the first component. Since we want to use in-

terchangeability in order to facilitate abstraction and reformulation, we wish to guarantee that the abstract problem has interchangeable values. We also wish to be able to vary the amount of interchangeability to study the effect on the effectiveness of our method. Therefore, we generate the random problems so that the first component of each constraint exhibits a specified degree of interchangeability. The degree of interchangeability is specified by the number of equivalence (interchangeability) classes in the partition induced by interchangeability on the domain of each variable. We generate problems so that the interchangeability classes will be of the same size to the extent possible.

To facilitate controlling the degree of interchangeability we wish the interchangeability classes to be the same at each variable. To accomplish this we restrict the problems in several ways: We use the same domains for each variable. We use the same first component for each constraint. The first components are symmetric, in the sense that if a for X and b for Y satisfies the component constraint, then b for X and a for Y will also. The method we used to generate problems with the desired characteristics has the side effect of producing a first component constraint that is antireflexive (if we consider the matrix representation of the component, where 0's indicate inconsistency and 1's consistency, the diagonal elements are all 0) and a reduced abstracted problem where all the constraints are the inequality constraint. To make the problems more general, there is no specific requirement on the second component of each constraint. It is just a random constraint with the specified tightness.

We ran tests on 4 sets of problems, using 4 different values for the number of interchangeability classes (2 – 5) induced by the first component of the constraints. The density of the constraint graph in each set varies between 0.1 and 0.9, with a step of 0.05, while the tightness for the second component of the constraints varies between 0.1 and 0.9 with a step of 0.1. For each pair of values (density, tightness), we generated 5 instances of random problems, which gives us roughly a total of 800 problems per set. Since some of the problems were extremely hard, we limited the running time for both algorithms to one hour per problem. If the algorithm did not end during this period of time, it was aborted and the results at that moment were reported. The results are discussed below and shown in Figure 1 (which omits the case of 2 interchangeability classes for space reasons). The figure is divided into three sections, (a) – (c). Each section presents the results of the tests on one set of problems, in the form of two plots. The first plot shows the difference between the running (CPU) times for FC-DMD and ABS&REF+FC-DMD, while the second one shows the minimum of the two. Note that the time scales are very different on these two plots. Each point of the plot was computed as the average over the 5 instances of random problems generated for each pair of density and tightness values.

For 2 and 3 interchangeability classes, utilizing abstraction and reformulation (ABS&REF+FC-DMD) won all the time. Thus the "min" plot in Figure 1(a) shows in fact the CPU time for ABS&REF+FC-DMD, while the "difference" plot shows basically the CPU time for FC-DMD. For 4 and 5 interchangeability

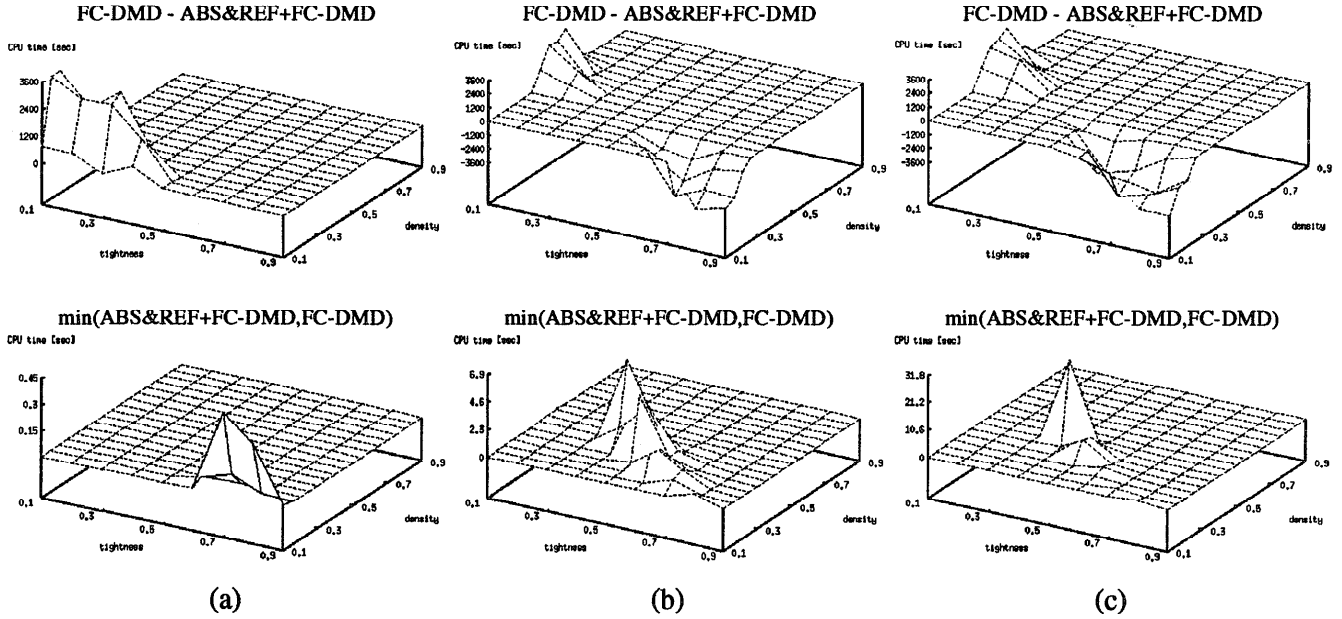


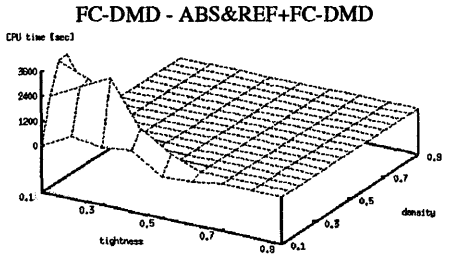
Figure 1: CPU time results for problems with 20 variables, 30 values in each domain, 3 (a), 4 (b) and 5 (c) interchangeability classes (note the different time scales)

classes, ABS&REF+FC-DMD won in one region of the space and lost in another one. Given the small "min" values, the "difference" plot basically shows the running time for FC-DMD as positive values and the running time for ABS&REF+FC-DMD as negative values. Also, the "min" plot makes it clear that running both algorithms in parallel would give an overall winner, and that the magnitude of the win can be quite large. The complementary nature of the performance of the two approaches makes parallelism especially appropriate.

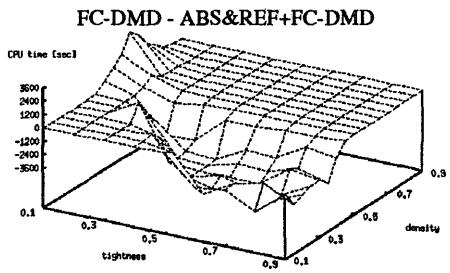
The reduced, abstracted problem will have relatively few values in each domain; the number of values, d, is the number of interchangeability classes. Since the constraints are the inequality constraint, a subset of more than d variables that are all pairwise constrained will be impossible to satisfy. Thus much of the success of our method on these problems may be in quickly identifying unsolvable problems. However, the queens problems, which fit the MC profile, demonstrate that the method can be effective when there are solutions.

Multi-Dimensional Domains

The values for the variables in this class have two attributes and the constraints are a conjunction of two components, each relating values of the same attribute: the first component is a constraint between values for the first attribute, while the second component restricts the combinations of values for the second attribute. The components are generated in a manner similar to the one used for class MC. We again tested



(a)



(b)

Figure 2: CPU time results for problems with 20 variables, 5 value domain for the first attribute, 6 value domain for the second attribute, 4 (a) and 5 (b) interchangeability classes

for 2-5 interchangeability classes. The difference plots for 4 and 5 interchangeability classes are shown in Figure 2. Again we observe the complementary behavior.

Conclusion

Higher level reasoning techniques like abstraction and reformulation can be successfully used in the constraint satisfaction domain. The key is to find concrete techniques that will repay their overhead, at least for suitable classes of problems. In this paper we present a specific approach to abstraction and reformulation that is supported by interchangeability processing. We demonstrate the success of this approach on a classic problem and on two classes of multi-dimensional problems.

Acknowledgments

This material is based on work supported by the National Science Foundation under Grant No. IRI-9207633 and Grant No. IRI-9504316.

References

Choueiry, B., and Faltings, B. 1994. A decomposition heuristic for resource allocation. *ECAI-94*, 585–589.

Darr, T., and Birmingham, W. 1996. An agent architecture and algorithm for solving distributed configuration-design problems. *Configuration, AAAI Technical Report FS-96-03.* AAAI Press.

Ellman, T. 1993. Abstraction via approximate symmetry. *IJCAI-93*, 916–921.

Freuder, E., and Sabin, D. 1995. Interchangeability supports abstraction and reformulation for constraint sa-tisfaction. *SARA-95, Symposium on Abstraction, Reformulation and Approximation.*

Freuder, E. 1991. Eliminating interchangeable values in constraint satisfaction problems. *AAAI-91*, 227–233.

Geelen, P. 1992. Dual viewpoint heuristics for binary constraint satisfaction problems. *ECAI-92*, 31–35.

Nadel, B. 1990. Representation selection for constraint satisfaction: A case study using n-queens. *IEEE Expert* 5(3):16–23.

Schrag, R., and Miranker, D. 1996. Abstraction and the CSP phase transition boundary. *Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics (AI/Math-96)*, 138–141.

Stefik, M. 1981. Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence* 16(2):111–140.

Weigel, R., and Faltings, B. 1996. Abstraction techniques for configuration systems. *Configuration, AAAI Technical Report FS-96-03.* AAAI Press.

Yoshikawa, M., and Wada, S. 1992. Constraint satisfaction with multi-dimensional domain. *AIPS-92: The First International Conference on AI Planning Systems.*