

Structuring Techniques for Constraint Satisfaction Problems

Rainer Weigel and Boi V. Faltings

Artificial Intelligence Laboratory

Swiss Federal Institute of Technology in Lausanne (EPFL)

IN-Ecublens, CH-1015 Lausanne

Switzerland

Abstract

We present a structuring method for discrete constraint satisfaction problems (CSPs). It takes advantage of interchangeabilities to represent sets of equivalent values by meta-values and thus obtain more compact representations. Strongly related variables are clustered into meta-variables to create occurrences of interchangeabilities. By iterative application, a CSP can be transformed into an hierarchy of equivalent CSPs, where each problem is significantly simpler than the original one. This structure is particularly advantageous when a large set of possible solutions must be inspected.

1 Introduction

We consider binary CSPs defined by $P = (X, D, C, R)$, where $X = \{X_1, \dots, X_n\}$ is a set of variables, $D = \{D_1, \dots, D_n\}$ a set of finite domains associated with the variables, $C = \{C_1, \dots, C_m\}$ a set of constraints, and $R = \{R_{ij} \subseteq D_i \times D_j \text{ for } C_{ij} \text{ applicable to } X_i \text{ and } X_j\}$ a set of relations that define the value combinations allowed by the constraints. Solving such a CSP amounts to finding value assignments to variables subject to constraints. In this paper we address CSPs with many solutions which are to be inspected to select the most suitable one. *Structuring* this space of solutions means constructing a more efficient representation which simplifies such a selection. An example of such a structure is the crossproduct representation [Hubbe and Freuder, 1992]. Interchangeability [Freuder, 1991] is an important concept for structuring CSPs. It involves replacing equivalent values by single *meta values*, thus collapsing solutions with either value to a single one. Crossproduct representations can be constructed from interchangeabilities. The definition from Freuder for neighborhood interchangeability is recalled below. Neighborhood interchangeability captures the idea that values might satisfy exactly the same constraints and thus it is enough

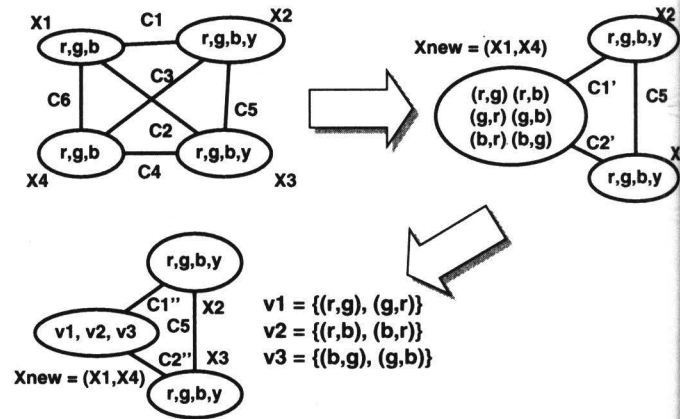


Figure 1: Coloring example

to consider only a single value which represents all the equivalent ones.

Definition 1 *Neighborhood interchangeability*: A value b for a CSP variable V is neighborhood interchangeable (NI) with a value c for V iff for every constraint C on V :

$$\{i \mid (b, i) \text{ satisfies } C\} = \{i \mid (c, i) \text{ satisfies } C\}$$

However, interchangeability is rarely valid for complete CSPs. It can often be found only for certain contexts, or for combinations of several variables. We call such combinations *meta variables*. The coloring example in Fig. 1 illustrates this fact. Assume that variables X_1 and X_4 are clustered into a new meta variable X_{new} . Then there are 6 values for X_{new} , namely the solutions of the subproblem induced by X_1 and X_4 . C_1, C_3 and C_2, C_4 are merged into C_1' and C_2' respectively. The constraint C_6 will no longer appear since it is already accounted in the value combinations allowed for X_{new} . The new structured value $(X_1 = r, X_4 = g)$ for X_{new} is then compatible with b and y for variable X_3 . It turns out that by calculating interchangeability for values of X_{new} , we can find three equivalence classes each of size two, namely $v_1 = \{(r, g), (g, r)\}$, $v_2 = \{(r, b), (b, r)\}$ and $v_3 = \{(b, g), (g, b)\}$. These are the meta values for X_{new} . One solution of the transformed CSP is $(X_{new} =$

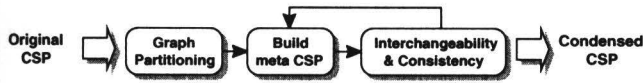


Figure 2: The Structuring Process

$v1, X2 = b, X3 = y$) and this solution represents the two detailed solutions ($X1 = r, X2 = b, X3 = y, X4 = g$) and ($X1 = g, X2 = b, X3 = y, X4 = r$). This small example shows that one can find interchangeability in problems after solving a subproblem, although no interchangeability could be found in the original problem formulation. In [Freuder, 1991], Freuder gives an algorithm for computing neighborhood interchangeability which runs in $O(n^2d^2)$ time, where n is the number of variables and d is the maximal domain size. This algorithm and its extension to k -interchangeability are the only polynomial time algorithms to find all interchangeable values in CSPs known so far. Finding all fully interchangeable values [Freuder, 1991] of a CSP, for example, requires finding all its solutions and is thus intractable in general. Thus we only consider neighborhood interchangeability. In order to compact the search space, we calculate neighborhood interchangeability in meta CSPs, and identify thereby *meta-interchangeability* as described in [Freuder, 1991]. We will first present the structuring algorithm. Then we describe an algorithm to calculate interchangeability for values of meta variables and illustrate in 4 the structuring algorithm with an example. In sec. 5 we describe a model for CSPs with clustered constraint graphs and conclude after an empirical evaluation of the algorithm.

2 The Structuring Algorithm

The various steps in the structuring algorithm are depicted in Fig. 2. A clustered structure of a constraint graph can be made explicit in the first step using graph partitioning algorithms. After having identified the clustered structure, it can be exploited for finding interchangeability and consistency in the meta CSPs that are generated iteratively. The iteration of calculating interchangeability and running consistency in meta CSPs continues until a single meta variable is created for each cluster identified by the graph partitioning step. Thus the amount of preprocessing that needs to be invested in the iteration depends on how many clusters the graph is decomposed.

Graph Partitioning: Given a CSP with clustered constraint graph $G(V, E)$, we apply a graph decomposition method called the *Recursive Bisection Partitioning* [Ciarlet *et al.*, 1994] that allows us to identify a set of edges E_p , which, if removed, decompose the constraint graph into several connected components. For clustered graphs, one can expect that the set E_p corresponds to

the intercomponent edges of the graph, and each connected component that remains after removing E_p , corresponds to a cluster with higher connectivity. The problem of finding a minimum cut bisection of a graph is a well known NP-complete problem [MacGregor, 1978], and among the heuristics for finding an approximation to the problem the Recursive Bisection Method [Ciarlet *et al.*, 1994] is a very successful one. It recursively applies the median cut partitioning method that could be described as follows:

1. Build the Laplacian matrix of the graph G , which is basically the adjacency matrix of the graph G and in the diagonal we put $-|adj(X_i)|$. Then calculate the second smallest eigenvalue of the matrix and the corresponding eigenvector f , which is called the fiedler vector [Fiedler, 1973].
2. Let x_l be the median value of the components of the fiedler vector f . Node X_i will be put in cluster $C1$ if its corresponding position f_i is smaller than x_l , otherwise node X_i belongs to cluster $C2$.

Consider the example from Fig. 3. The median cut procedure partitions the graph into clusters $C1 = V1, \dots, V8$ and $C2 = V9, \dots, V16$ indicated by hatched edges E_p . A stopping criteria for the recursive part of the algorithm needs to be defined. The recursion stops when the number of edges that should be removed in each step is bigger or equal to the number of edges that rest in each component. In an interactive mode, the user might be prompted and asked for the decision.

For every cluster we can identify a subset of nodes V_E that are linked with nodes in other clusters, and a set of nodes V_I having only internal links. The nodes V_E can act like a set of articulation nodes that limit the interaction of assignments to the nodes V_I with the other clusters. At the same time, these nodes are intuitively the part of the cluster through which value interaction is transmitted among the different clusters. Thus by finding consistent assignments to the variable in V_E , we can localize the effects of assignments to the other variables of the CSP. The clustered structure is therefore a major source for interchangeability. In the example described in sec. 4, we cluster the variables $V13, V14, V15, V16$ into a new meta variable $AV4$. Any set of values for $AV4$ that have the **same** assignments for the variables in $V_E = V13, V14, V15$ are interchangeable wrt. the rest of the CSP.

Meta CSPs: Variables, inducing subproblems of the CSP, are clustered together and treated as a single meta-variable. Each solution to a subproblem is a composed label of the size of the subproblem and becomes a value of the new meta-variable. Adjusting the constraints must in turn be done for all the values of each new variable. Values of the new variables are compatible wrt. the new

constraint if all pairs of the compound labels are compatible wrt. the original constraints. The result is called a meta CSP in [Freuder, 1992]. There are many reasons why it can be interesting to cluster variables: the resulting meta CSP might have a nice structure so that it can be solved efficiently. This is the case in the tree clustering procedure [Dechter and Pearl, 1989] where variables are clustered together such that the resulting meta CSP is a tree. The ultimate goal, pursued here, is to find clusters of variables such that their values are meta-interchangeable.

There are many possible subproblems that could be clustered together. The clustering procedure presented here will satisfy the following two conditions: 1) the nodes should be tightly connected because the domain size for the new variable should be kept reasonably small¹, and 2) the nodes should belong to the same cluster identified by the graph partitioning method. Therefore we use a greedy maximal clique decomposition algorithm, described next, applied on the graph $G(V, E \setminus E_p)$. A *greedy clique decomposition* algorithm of a graph G partitions the nodes of G into cliques². To find such a decomposition, one starts with a single node and builds successively a bigger clique by putting nodes into the clique that are adjacent to all the nodes already in the clique until the clique cannot be extended further i.e. no other node is adjacent to all the nodes in the clique. Having found such a maximal clique one removes all the nodes in the clique from the graph. The algorithm finally stops when no nodes are left in G . There are many possible clique decompositions and a heuristic that always selects a node with the highest degree to be put into the current clique is used. An example of such a decomposition can be found in the example in Fig. 3, where the cliques are indicated with bold edges.

Using the clique decomposition as a heuristic to cluster variables can be justified by the fact that these problems are highly connected and thus might not allow many solutions that appear as new values for the meta variable in the next step. This heuristic corresponds partly to the maximum degree variable ordering heuristic [Dechter and Meiri, 1994] in backtrack search.

Interchangeability and Consistency: Calculating interchangeability for the values of the meta variables allows us to reduce the size of the CSP when sets of values can be treated as a single value. From the definition of neighborhood interchangeability one can see that the chance for finding these interchangeable values increases with a decreasing number of constraints that have to be checked. Thus the heuristic to choose cliques for cluster-

ing described above might result in more interchangeable sets of values compared to a random clustering.

When clustered variables have been replaced by meta variables, arc-consistency might find additional inconsistencies which were not apparent in the original problem. These pruned meta values are actually partial solutions of the original CSP that cannot be extended to a global solution.

3 Calculating Interchangeability

In this section we present another algorithm to identify interchangeable values for meta variables. The basic idea is to take advantage of the set of articulation nodes V_E already presented in section 2. In order to identify interchangeability of two values v_1 and v_2 of a meta variable X with Freuder's algorithm, one needs to iterate over all constraints that link X with its neighbors and check if exactly the same tuples are valid for v_1 and v_2 . For two values having the same assignments for the variables in V_E , we do not even have to check the constraint, because the same assignments to V_E imply the same allowed tuples for the constraints. Thus these values are interchangeable. The following algorithm runs faster than Freuder's algorithm, but might not find all sets of interchangeable values.

Algorithm 1 *Interchangeability of Meta Values*

Let X be a meta variable created by clustering the set of variables V ($V = V_I \cup V_E$):

1. Identify $V_E \subseteq V$.
2. Unless $V_E = V$, iterate over the values of a meta variable X and use the projection of the compound labels of each value onto the set V_E to discriminate between the different equivalence classes of interchangeable values.

One can show that the interchangeable sets for values in a meta CSP found by algorithm 1 are always subsets of the interchangeable sets found by Freuder's algorithm. Thus, to find all interchangeable values, it is proposed to run algorithm 1 followed by Freuder's algorithm. Although this results in the same interchangeable sets as when running Freuder's algorithm only, it can be much faster. Algorithm 1 runs in $O(dn)$ time³. Thus the complexity of Freuder's algorithm can be calculated by $O(D^2n^2)$ where D is the number of equivalence classes returned by algorithm 1. Most experiments done so far indicate that the sets not found by our algorithm are rare. This is not very surprising knowing that interchangeability applied directly on CSPs is also rare.

¹The new domain size is exponential in the size of the subproblem.

²In the literature, however, clique decompositions often partition the edges and not the nodes into cliques.

³ d , n are the domain size and the number of variables of the meta CSP.

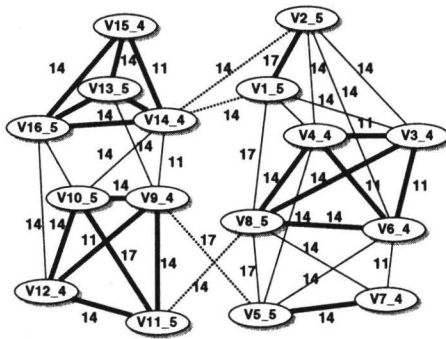


Figure 3: 16 variables example with clique decomposition and partitioning

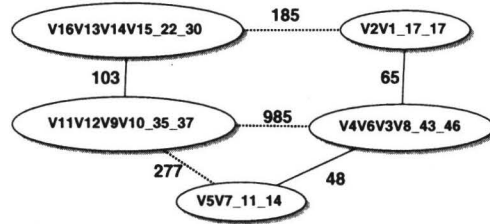


Figure 4: Meta-CSP and Interchangeability

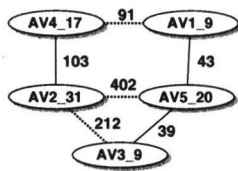


Figure 5: Renaming and Arc-Consistency in the Meta-CSP

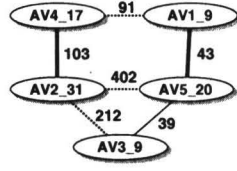


Figure 6: Clique Decomposition

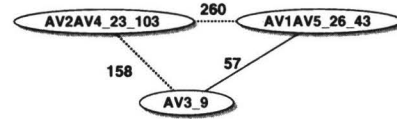


Figure 7: Again Meta-CSP and Interchangeability

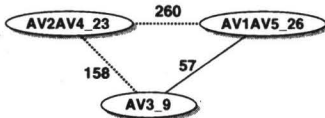


Figure 8: Unsuccessful Arc-Consistency

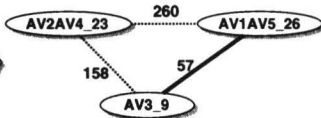


Figure 9: Clique decomposition

Figure 10: Resulting CSP with 2 variables, one for each cluster

4 Example

Fig. 3 shows the constraint graph of a CSP with 16 variables. Every variable name is followed by its domain size and every constraint is labeled with the number of tuples allowed by the constraint. The median cut partitioning finds 4 edges between the nodes $(V14, V2), (V14, V1), (V9, V5)$ and $(V11, V8)$ that partition the graph into cluster $V1, V2, \dots, V8$ and a second cluster $V9, V10, \dots, V16$. The partitioning algorithm is then stopped and the iteration on building the meta CSP calculating interchangeability and running consistency is repeated until a single meta variable is created for each cluster. The greedy clique decomposition algorithm determines in the first run 5 subproblems with complete graphs indicated by the bold edges in Fig. 3. Every subproblem is solved individually and appears in Fig. 4 as a new meta variable. The variables $V13, V14, V15$ and $V16$ for example are clustered together into the meta variable, named "V16V13V14V15". The number of solutions of this subproblem is 30, i.e. the last number of the node label. Interchangeability identifies 8 equivalence classes of val-

ues each of size 2. Thus the domain size can be reduced to 22. As an example of a set of interchangeable values consider $S1 = (V16 = 0, V13 = 4, V14 = 2, V15 = 3)$ and $S2 = (V16 = 0, V13 = 4, V14 = 2, V15 = 2)$. One can see how the nodes $V16, V13$ and $V14$ act as the articulation nodes V_E . Algorithm 1 finds the set V_E , the projection of $S1$ and $S2$ onto V_E is $(V16 = 0, V13 = 4, V14 = 2)$ for both values, and thus $S1$ and $S2$ are interchangeable. Renaming $V16V13V14V15$ to $AV4$ (AV stands for Abstract Variable) and applying arc-consistency to the CSP is shown in Fig. 5. With arc-consistency one can remove 5 values from the domain of $AV4$. The four edges found by the graph partitioning in the first step corresponds now to the three edges $(AV1, AV4), (AV2, AV5), (AV2, AV3)$ that are not allowed to appear in a clique in the next step. Consequently two new clusters $AV1AV5$ and $AV2AV4$ are created and $AV3$ is not changed (Fig. 8). In a third step "AV1AV5" and $AV3$ are clustered together resulting finally in a CSP with two nodes $AV2AV4$ with domain size 23 and $AV1AV5AV3$ with domain size 26. These two nodes corresponds to the two clusters $V1, V2, \dots, V8$

and $V_9, V_{10}, \dots, V_{16}$ determined in the first step. 210 tuples are allowed by the constraint between the two nodes and these tuples correspond to the solutions for the CSP on this level (Fig. 10). Every meta-value in a solution of the resulting CSP can be replaced by the set of values it represents. In this way one obtains the cross product representation [Hubbe and Freuder, 1992] of solutions for the next lower level. Replacing meta values for each solution in the crossproduct continues until the lowest level is reached. The 210 solutions can be refined to 438 on the next lower level then to 2910 and finally to the 5366 solutions of the original CSP. It is worth to mention that the size of the equivalence classes of values plays a multiplicative role when determining the number of solutions on the next lower level. If a solution S contains t meta values that represent classes of size s_i with $i = 0, \dots, t-1$, then the number of solutions on the next level that are represented by S equals $\prod_{i=0}^{t-1} s_i$.

5 Clustered Constraint Graphs

Constraint graphs of problems that arise for example from physical interactions often resemble clustered graphs. We shortly describe a simplified model for generating clustered graphs, however details can be found in [Hogg, 1996]. The nodes of the graph to be generated will appear as the leaves of a balanced binary tree. Thus to generate graphs with 2^D number of nodes, we build a balanced tree with depth $d = D - 1$. The tree induces a ultrametric distance between pairs of nodes, defined as the number of levels up in the tree from the nodes to reach a common ancestor. For a given p with $0 < p < 1$, edges in the graph between nodes at distance t ($0 < t \leq d$) can then be selected with a relative probability p^t . As an example consider the clustered constraint graph in Fig. 3. The nodes $V_1, \dots, V_{16}$ are the leaves of a balanced tree with depth 3 arranged from left to right in increasing order. Given $p = 0.5$ the probability of having an edge in between two nodes from the set $V_1, \dots, V_8$ and from the set $V_9, \dots, V_{16}$ is $0.5^3 = 0.125$. An edge in between two nodes in $V_1, \dots, V_4$ and $V_5, \dots, V_8$ appears with probability 0.25 etc.. Having build the clustered graph one can easily construct a CSP with clustered constraint graph by randomly selecting domain sizes and allowed tuples for the constraints.

6 Evaluation

Any evaluation of a technique depends on the objectives it is used for. Our objective is to structure a CSP so that its solution space can be easily inspected. Thus, a reasonable measure would be the average number of choices a user has to make to select one particular solution among all possibilities. The results for a small number of test cases generated with $n = 32$, $p = 0.4$, $domainsize = 3$,

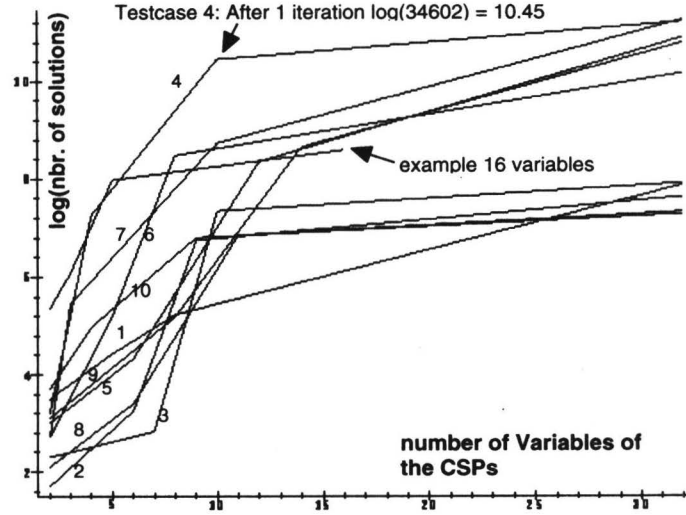


Figure 11: Number of solutions on the different levels for the Testcases on a logarithmic scale

number of constraints = 110, 70% from all possible tuples for each constraint are allowed, can be found in Table: 1 and in Fig. 11. The problems are transformed until CSP with two variables are returned. Over-constrained problems and problems with less than 100 solutions are omitted. We define a condensing factor of the solution space $\rho_i = \log|solutions\ level_i| / \log|all\ solutions|$ with $1/\log|all\ solutions| \leq \rho_i \leq 1$ and for different abstraction levels $i = 0, \dots, k$. Using this measure, we can discriminate between CSPs where the algorithm returns good and bad results⁴. A factor close to 0 implies a very condensed representation, while a factor of 1 could not condense the solution space much. A series of runs confirm the intuition that for clustered CSPs with “many” solutions, the number of solutions of the resulting meta-CSPs is condensed considerably, often by a large factor (see. Fig. 11). Testcase 2 for example could be condensed from 2066 solutions to 6 solutions after 3 iterations. Resulting in a $\rho_3 = \log 6 / \log 2066 = 1.7917595 / 7.6333694 = 0.234$. Thus, the structured problem will considerably reduce the effort required to search through the solution space of such a CSP.

The evaluation method needs to enumerate all solutions for the condensed CSP. This, however, is only possible for small CSPs. For large CSPs (i.e. when the resulting CSP has still > 100 variables), we will use a probabilistic approach that determines the expected number of solutions of the CSPs (based on [Knuth, 1975]) on the different levels to determine ρ_i .

⁴ ρ is not defined for CSPs with only a single solution

Solutions on level:	Example and Testcases 1 - 10										
	Ex.(16 vars)	1	2	3	4	5	6	7	8	9	10
3	210	36	6	11	26	22	15	15	9	24	43
2	438	84	26	17	1428	76	184	235	30	180	146
1	2910	188	862	1550	34602	4260	4832	6220	951	5712	904
original CSP	5366	2632	2066	2750	74718	54280	25872	77828	1512	48972	1460
ρ_3	0.62	0.44	0.23	0.29	0.28	0.27	0.26	0.23	0.28	0.28	0.50

Table 1: Condensing the solution space of some examples. The size of the CSPs on the various levels is omitted.

7 Future Work and Conclusion

In this paper we have presented a structuring method for CSPs. Applying the method to a CSP results in a more condensed representation that allows the enumeration of sets of similar solutions and thus facilitates the selection of single solutions. One should contrast this with search algorithms for solutions in the original CSP: They find one solution after the other without showing the similarity in between them. Furthermore, in many applications it is not enough to present a human decision maker with a single solution, because this single solution might violates other criteria which the user has not made explicit to the problem. Thus one should give the user a chance to integrate hidden knowledge and unquantifiable preferences in the decision process by proposing sets of similar solutions from which he can make the selection.

The evaluation of our algorithm imposes many difficulties and a lot of work needs still to be done. It is for example not enough to know how many interchangeable sets we get and how big they are; it is very important to know in how many solutions a meta value will finally appear. This always requires us to look at all solutions of a CSP, a set which is impossible to generate for large CSP. In the future work we will also look at hard random problems. Basically one would like to compact the search space such that certain instantiations might have to be made only once for each equivalence class of values, thus reducing the search effort. Similarly these hard problems often exhibit a high degree of consistency and thus low consistency algorithms will not find inconsistencies. Running arc-consistency on a meta CSP, which is related to restricted form of *k-consistency* with *k* equal to the sum of size of two adjacent clusters, might therefore be very interesting for hard CSPs.

References

[Ciarlet *et al.*, 1994] P. Ciarlet, F. Lamur, and B.F. Smith. On the influence of the partitioning schemes on the efficiency of overlapping domain decomposition methods. Technical Report 94-23806, UCLA Computational and Applied Mathematics, Los Angeles, CA 90024-1555, 1994.

- [Dechter and Meiri, 1994] R. Dechter and I. Meiri. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. *Artificial Intelligence*, 68:211-241, 1994.
- [Dechter and Pearl, 1989] Rina Dechter and Judea Pearl. Tree Clustering for Constraint Networks. *Artificial Intelligence*, 38:353-366, 1989.
- [Fiedler, 1973] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23 (98):298-305, 1973.
- [Freuder, 1991] Eugene C. Freuder. Eliminating Interchangeable Values in Constraint Satisfaction Problems. In *Proc. of AAAI-91*, pages 227-233, Anaheim, CA, 1991.
- [Freuder, 1992] Eugene C. Freuder. Constraint solving techniques. In E. Tyngu B. Mayoh and J. Penjaen, editors, *Constraint Programming of series F: Computer and Systems Sciences*, pages 51-74. NATO ASI Series, 1992.
- [Hogg, 1996] T. Hogg. Refining the phase transition in combinatorial search. *Artificial Intelligence*, 81:127-154, 1996.
- [Hubbe and Freuder, 1992] P.D. Hubbe and E.C. Freuder. An efficient Cross Product Representation of the Constraint Satisfaction Search Space. In *Proc. of AAAI-92*, pages 421-427, San Jose, CA, 1992.
- [Knuth, 1975] D. E. Knuth. Estimating the efficiency of backtrack programs. *Mathematics of Computations*, 29:121-136, 1975.
- [MacGregor, 1978] R. M. MacGregor. *On Partitioning a Graph*. PhD thesis, University of California, Berkeley, 1978.