From Unsolvable to Solvable: An Exploration of Simple Changes

Susan L. Epstein[1, 2] and Xi Yun[1]

Department of Computer Science
[1]The Graduate Center and [2]Hunter College of The City University of New York
New York, NY 10065 USA
susan.epstein@hunter.cuny.edu, xyun@gc.cuny.edu,

Abstract

This paper investigates how readily an unsolvable constraint satisfaction problem can be reformulated so that it becomes solvable. We investigate small changes in the definitions of the problem's constraints, changes that alter neither the structure of its constraint graph nor the tightness of its constraints. Our results show that structured and unstructured problems respond differently to such changes, as do easy and difficult problems taken from the same problem class. Several plausible explanations for this behavior are discussed.

Introduction

Constraint satisfaction problems (*CSPs*) that model real-world challenges are often unsolvable. The thesis of this work is that it is possible to reformulate such a problem relatively easily so that it becomes solvable. This paper investigates the impact on a CSP's solvability of small changes in its constraints. Its principal result is that relatively few such changes can make a hitherto unsolvable problem solvable, and that structured problems and more difficult problems are actually more likely to become solvable under such changes.

A dynamic CSP (*DCSP*) subjects an original CSP P_0 to a sequence of perturbations, producing a sequence of new problems $P_1, P_2, \ldots, P_n$, where a *perturbation* is any change to P_0. The solutions to P_i may well be different from those to P_{i-1}. Prior work has suggested how search for solution to P_i may benefit from solutions to P_{i-1} (Verfaillie and Jussien, 2005) or how search performance changes when new constraints are introduced and old ones dropped (Wallace, Grimes and Freuder, 2009). The focus here is on a more fundamental change: whether or not, for some $i > 1$, P_i is solvable, although P_{i-1} was not. For real-world problems, which are often unsolvable in their original formulation, simple perturbations that make them solvable should be of considerable interest.

We focus here on perturbations that preserve both the domains and the structure of the problem, that is, ones for which the constraint graph remains unchanged. Our perturbations instead change the tuples that constraints declare acceptable. There are several ways to do this, two of which are explored carefully here. Because our concern is with realistic problems, we investigate such perturbations on both random (i.e., unstructured) problems and on small world problems, which have the kind of structure detected in many real-world domains (Walsh, 1999).

We show here that certain perturbations readily make some problems solvable. The next section provides definitions and describes the perturbations studied. Subsequent sections describe the experiments, report and discuss their results, and outline current work.

CSPs and Value Perturbation

A CSP $\langle X, D, C \rangle$ is defined by a set of variables X, a set of domains D, and a set of constraints C. Each variable $X_i \in X$ has a domain D_i of possible values. A *constraint* restricts how the variables in its *scope* $S \subseteq X$ may be assigned values together. Here, CSPs are *binary*, that is, all constraints have scope size at most two. Thus a constraint $C_{ij} \in C$ declares each tuple (a, b) where $a \in D_i$, $b \in D_j$ to be either acceptable (a *compatible*) or unacceptable (a *nogood*). A constraint can be expressed *extensionally*, as a set of nogoods or a set of compatibles, or *intensionally*, as a function that maps $D_x \times D_y$ into {*compatible, nogood*}. The *tightness* of a constraint is the fraction of tuples in $D_x \times D_y$ that it specifies as nogoods.

A *solution* to a CSP assigns to each variable a value from its domain so that all the constraints are satisfied. A CSP with at least one solution is *solvable*; otherwise it is *unsolvable*. To solve a CSP, *global search* repeatedly selects a variable and assigns it a value. At any point in search, the existing assignments form a *partial instantiation* of the problem, and a variable that does not yet have an assigned value is called a *future variable*. After each assignment, *propagation* identifies and temporarily removes values incompatible with that assignment from the do-

mains of the future variables. If any domain becomes empty (*wipeout*), search backtracks, removing assignments and restoring dynamic domains until there is an alternative to explore. If a problem has no solution, global search will demonstrate that by retracting every value in the domain of some variable at the beginning of the search.

Let a *swap* be a perturbation with respect to a pair of tuples. An *internal swap* for a single constraint C_{ij} takes a compatible tuple (a_1, b_1) and a nogood tuple (a_2, b_2) and relabels them, so that (a_1, b_1) becomes a nogood and (a_2, b_2) becomes a compatible. An *external swap* with respect to a pair of constraints C_1 and C_2 exchanges a pair of nogoods, (a_1, b_1) from C_1 and (a_2, b_2) from C_2. After an external swap, C_1 considers (a_1, b_1) a compatible but (a_2, b_2) a nogood, and C_2 considers (a_2, b_2) a compatible but (a_1, b_1) a nogood. The DCSPs investigated here perform a sequence of internal or external swaps on P_{i-1} to produce P_i.

As defined here, both kinds of swap retain the domains, the scope, and the tightness of every constraint in P. In contrast, relaxation of a constraint can change its tightness. Swapping may also offer a realistic adaptation. In a scheduling problem, for example, an external swap of a nogood that forbids a pair of job-sharing workers from simultaneous assignments at night could shift that restriction to daytime hours. An internal swap on the same nogood could substitute a similar restriction on their spouses. Thus, although an external swap is syntactically equivalent to some pair of internal swaps, together the two kinds provide a more flexible perturbation semantics.

Experimental Design

The problems studied here are drawn from two classes: random and structured. The *random problems* are model B problems (MacIntyre et al., 1998). The *structured problems* are small world problems produced under the Watts-Strogatz model (Watts and Strogatz, 1998). All problems have 50 variables, each with the same domain of 10 values, density 0.204, and tightness 0.35. (Because we chose to make all problems the same density, our structured problems are somewhat denser than the typical small world problems inspired by social networks.) Their parameters place both problem classes at the *phase transition*, that is, they are particularly difficult for their size (50 variables, 10 values). 58.9% of the random problems and 96.0% of the structured problems are solvable.

We generated 1000 random problems and solved them with *MinDomDeg*, a heuristic that prefers variables with the smallest ratio of dynamic domain size to dynamic degree. (A variable's *degree* is the number of constraints in which it participates.) During search, propagation maintained arc consistency with MAC-3 (Sabin and Freuder, 1997); value selection and tie breaking were lexical. Although problems in the same class are putatively similar, classes at the phase transition exhibit a heavy tail (Pareto-Normal distribution) of difficulty (Gomes et al., 2000). We therefore ranked the 1000 problems by the number of nodes in their search trees to find a first solution (for solvable problems) or to prove that none existed (for unsolvable ones). Figure 1 bins the random problems by the number of nodes in their search trees, where each bar shows the proportion of solvable and unsolvable problems. The same process was repeated separately for 1000 structured problems; Figure 2 shows the result.

In each class we then identified *EU*, the 10 easiest unsolvable problems (that is, those with the smallest search trees, from the leftmost bins) and *HU,* the 10 hardest unsolvable problems (those with the largest search trees, from the rightmost bins). Out of 1000, the 10 easiest unsolvable random problems EU_R lay in the third through fifth bins from the left in Figure 1. They were ranked between 186 and 287 among all random problems, with search trees of sizes 1420 – 2454. The 10 hardest unsolvable random problems HU_R were ranked 900 to 999, with search trees of 25927 – 38012. The 10 easiest unsolvable structured problems EU_S were ranked between 572 and 915 with search trees of 1169 – 5003; the 10 hardest HU_S between 990 and 1000 with search trees of sizes 12913 – 24610.

To perturb problem P by an internal swap, we randomly selected a constraint, randomly selected one nogood and

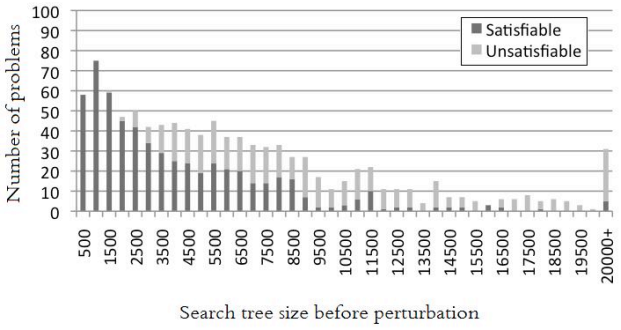

Figure 1: Random problems at the phase transition, binned by search tree size. Each bar represents the number of problems in a size range, where red denotes unsolvable and blue denotes solvable.

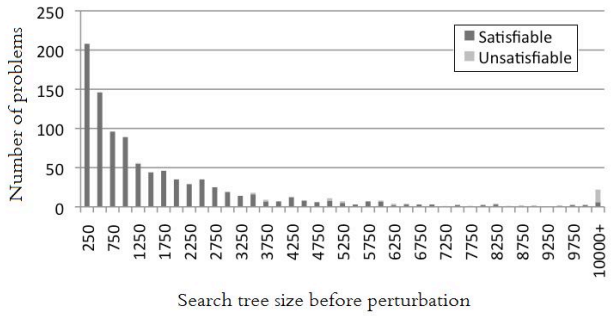

Figure 2: Structured problems at the phase transition, binned by search tree size. Each bar represents the number of problems in a size range, where red denotes unsolvable and blue denotes solvable. Note the change of scale on the vertical axis.

one compatible tuple from it, and interchanged their labels, so that the nogood became a compatible, and the compatible became a nogood. Perturbations were tested on u distinct constraints, each subjected to v internal swaps, where u = 1, 5, 25 internal swaps per tuple and v = 1, 5, 25 pairs of tuples.

To perturb problem P by an external swap, we randomly selected two distinct constraints, randomly selected one nogood from each of them (that was a compatible in the other constraint), and interchanged them, so that each constraint had a single nogood replaced by a nogood from the other constraint. Perturbations were tested for 1, 5, 10, 15, 20, and 25 external swaps. Note that both kinds of swaps retain P's structure and the tightness of all its constraints.

To gauge the impact of swaps on a given problem, we began with P and perturbed it with s swaps. Because swapping is non-deterministic, results were averaged over 100 trials, each time beginning with the same P. Results are reported here for EU_R, HU_R, EU_S, and HU_S under both internal and external swaps.

Results

Our intent is to observe when and how readily a problem *flips*, that is, changes from unsolvable to solvable under perturbations that change only nogoods and compatibles.

Internal swaps

Figure 3 reports the number of times, out of 100, that an unsolvable random CSP became solvable after u randomly-chosen constraints were each perturbed by v internal swaps. Each cell in the table represents 1000 trials (10 EU_R or HU_R problems each perturbed 100 times). Although (u,v) = (1,1), which is a single internal swap on a single constraint, rarely had any effect, the responsiveness of the hardest 10 problems to internal swaps is particularly noteworthy. Both (u,v) = (25, 5) and (u,v) = (5, 25) (25 internal swaps on 5 constraints, and 5 internal swaps on 25 constraints, respectively) flipped more than half of HU_R, that is, made more than half the hardest unsolvable random problems solvable.

Similarly, Figure 4 reports the number of times, out of 100, that an unsolvable structured CSP flipped after u constraints were each perturbed by v internal swaps. Again, each entry represents 1000 trials, and the hardest problems are easier to flip.

External swaps

As shown in Table 1, individual unsolvable random problems respond differently to swapping. In 100 attempts with 25 swaps in the perturbation, the first (very easiest) EU_R problem never became solvable. In contrast, on at least one occasion a single external swap made each of the hardest problems solvable. ("At least" because in the interest of time there was no intermediate examination of the problem after each swap. For perturbations involving more than one swap, first P was perturbed and then it was solved.) Table 2 reports how often in 100 trials the easiest and the hardest individual unsolvable structured problems became solvable under external swaps.

Because EU_R and EU_S were clearly more difficult to flip than the harder problems, we also tested them under 30, 35, 40, and 45 swaps in an attempt to increase the percentage of flips. The results, in Figure 5, indicate that easy unsolvable problems are remarkably reluctant to flip. Even at 150 swaps (not plotted), the easiest random problems flipped only 8.4 % of the time.

	Easiest: EU_R			Hardest: HU_R		
u	v =1	v = 5	v = 25	v = 1	v = 5	v = 25
1	0.1	0.5	4.1	0.9	6.2	26.2
5	0.8	3.3	13.0	6.7	23.0	57.8
25	5.3	10.9	24.9	24.5	51.9	68.2

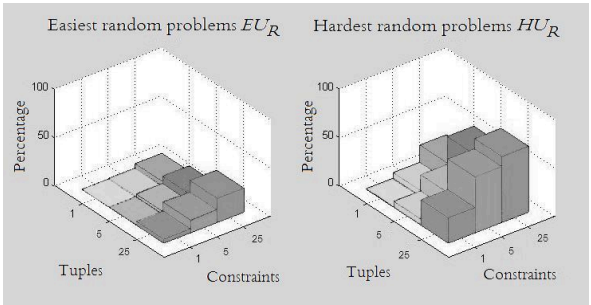

Figure 3: How often, in 100 trials, a DCSP among the 10 easiest and 10 hardest unsolvable random problems flipped due to internal swaps.

	Easiest: EU_S			Hardest: HU_S		
u	v =1	v = 5	v = 25	v = 1	v = 5	v = 25
1	1.2	5.4	23.9	3.8	19.3	57.8
5	5.1	22.3	57.8	18.7	54.5	86.9
25	23.7	58.8	83.1	53.2	86.7	92.5

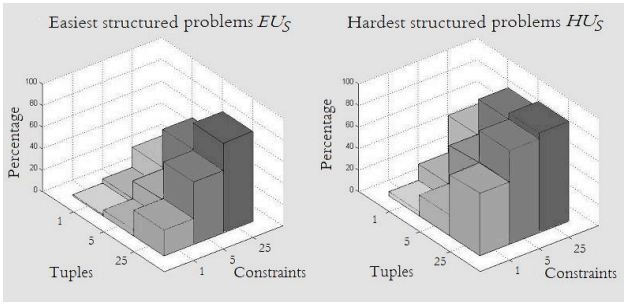

Figure 4: How often, in 100 trials, a DCSP among the 10 easiest and 10 hardest structured random problems flipped due to internal swaps.

Table 1: How often, in 100 trials, individual DCSPs based on unsolvable random problems became solvable. Perturbation was by 1, 5, or 25 external swaps.

	Easiest: EU_R			Hardest: HU_R		
Swaps / Problem	1	5	25	1	5	25
1	0	0	0	2	10	30
2	0	1	2	1	13	26
3	0	1	3	2	10	37
4	3	9	18	3	1	25
5	0	6	12	4	15	51
6	0	1	8	1	5	21
7	0	1	4	3	11	55
8	0	1	5	8	18	54
9	0	0	3	1	5	30
10	0	1	4	1	15	45
average	0.30	2.10	5.90	2.60	10.30	37.40
σ	0.95	2.96	5.40	2.17	5.31	12.87

Table 2: How often, in 100 trials, individual DCSPs based on unsolvable structured problems became solvable. Perturbation was by 1, 5, or 25 external swaps.

	Easiest: EU_R			Hardest: HU_R		
Swaps / Problem	1	5	25	1	5	25
1	2	7	34	3	19	55
2	2	6	21	7	18	49
3	0	5	31	11	30	84
4	2	8	33	8	29	79
5	2	5	30	2	25	62
6	1	14	47	4	19	79
7	0	3	18	10	56	89
8	6	27	64	7	32	68
9	1	12	36	19	56	89
10	3	20	51	14	40	81
average	1.90	10.70	36.50	8.50	32.40	73.50
σ	1.73	7.69	13.95	5.23	14.18	25.84

Discussion

As Figure 5 clearly indicates, unsolvable problems can become solvable with swapping. Harder unsolvable problems flip more readily than easier ones, and structured unsolvable problems flip more readily than random ones. How few changes were necessary to create a flip in these problems is noteworthy. Each of them has 250 constraints. Since all domains are of size 10 and all constraints have tightness 0.35, each constraint specifies its acceptance or rejection of 100 tuples with 35 nogoods. Thus a problem's full constraint description references 8750 nogoods, and 25 swaps represents only about 0.57% of them.

To confirm that lexical ordering did not bias the identification of EU_R, EU_S, HU_R, and HU_S, each of those 40 problems was solved 10 additional times with *MinDomDeg* and random tie breaking. Using the Mann-Whitney-Wilcoxon test (Hollander and Wolfe, 1999), the average search tree sizes for HU_R problems were compared to those for EU_R, and those for HU_S to EU_S. In both cases the search trees for *HU* were significantly larger than those for the corresponding *EU* ($p < 0.001$).

To investigate the impact *MinDomDeg* had on the selection of the easiest and hardest problems, we re-ranked both original sets of 1000 problems with a different, and somewhat more powerful heuristic, *MinDomWdeg* (Boussemart et al., 2004). This heuristic learns weights on constraints as it searches. All weights begin at 1, and whenever propagation on a constraint produces a wipeout, the constraint's weight is incremented by 1. The *weighted degree* of a variable is then the sum of the weights of the constraints in whose scope it lies. *MinDomWdeg* prefers variables with a minimum ratio of dynamic domain size to weighted degree. The Pearson product moment coefficient (0.84) indicated little difference in the two rankings of the random problems produced this way, but somewhat more for the structured problems (0.61). When the experiment was repeated with *MinDomWdeg* instead of *MinDomDeg*, however, it produced results similar to those in Figure 5.

One must take care in this context to distinguish between individual problem difficulty and problem class difficulty. These experiments began with two classes of problems (one random, the other structured) that are already termed "difficult" because their value of κ (a predictor for phase transition) is very close to 1 (Gent et al., 1999). Among 1000 such problems of each type, however, there was a range of difficulty, as measured by nodes searched to solution in Figures 1 and 2. With EU_R and HU_R (and again with EU_S and HU_S) we extracted very small (size 10) subsets of those classes, and then generated 1000 more exam-

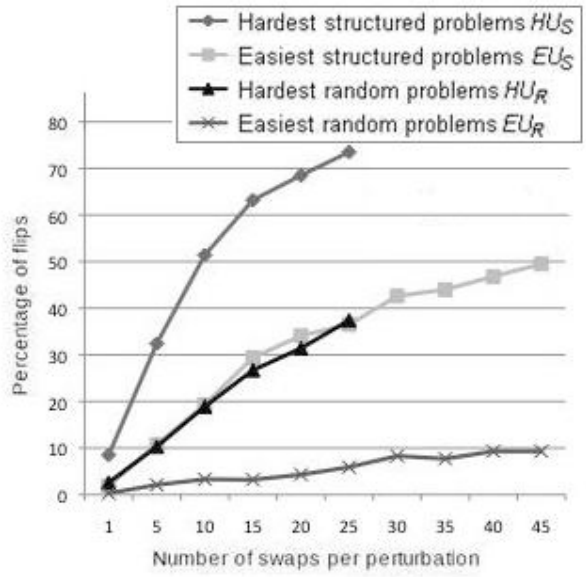

Figure 5: Flips under external swaps. Each data point represents 100 attempts to flip each of 10 unsolvable problems (1000 samples in all). EU_S and EU_R were tested on more swaps in an attempt to flip them more often.

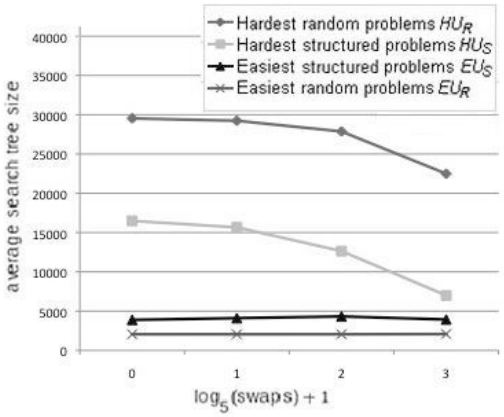

Figure 6: Search tree size for problems after perturbation by external swaps. Each data point represents 100 attempts to flip each of 10 problems (1000 samples in all), with 0 on the horizontal axis indicating no swaps. The easiest problems were tested on more swaps in an attempt to flip them more often.

ples by perturbation. Within each of these four larger sets, the structure of the problem and the tightness on its constraints were identical; only the labels on a (relatively) few tuples differed. Thus our results are about how readily swaps flip problems from a difficult (for their size) class.

The reader may wonder whether solvable problems *flop*, that is, become unsolvable under swapping. Because flopping seemed a less desirable property, we have thus far devoted less attention to it. Nonetheless, we can report that, in preliminary testing, harder solvable problems also flop more readily than easier ones under swapping. This suggests that harder problems are in general more susceptible to a change in solvability in either direction. Further computation is planned to confirm that.

Another question is whether the difficulty of the problems, as measured by tree size, changes under swapping. We emphasize again that the structure of these problems, along with their size and tightness, does not change under these perturbations. Nonetheless, the presence of a solution makes the search trees somewhat smaller, as shown in Figure 6, where the problems most likely to flip have the most noticeable reduction in search tree size.

Microstructure

One way to think about flipping is with the notion of microstructure (Jégou, 2003). The *microstructure* $m(P)$ of a CSP is a graph in which each possible variable-value assignment is a vertex, that is, for $P = \langle X, D, C \rangle$,

$m(P) = \langle V, E \rangle$ where $V = \{(X_i, d_i) \mid X_i \in X, d_i \in D_i\}$

In addition, an edge in the microstructure between $X_i = a$ and $X_j = b$ means that the pair (a, b) is compatible with the constraint between X_i and X_j. If there is no constraint between variables X_i and X_j in P, then in the microstructure every vertex that represents an assignment to X_i has an edge to every vertex that represents an assignment to X_j. A solution to P is represented in $m(P)$ by a complete graph (a *clique*) on n vertices, where the values in those vertices constitute a solution to P.

When a problem of size n is unsolvable, there is no clique of size n in the microstructure. (Of course, no clique larger than n can be present in the microstructure, by definition.) If a single swap makes an unsolvable problem solvable, then there must have been a subgraph on n vertices in the microstructure that was one edge shy of a clique on n vertices. In Figure 7, for example, there is no 3-clique, so the problem is unsolvable. An internal swap of the compatible ($X_1 = 2$, $X_2 = 3$) with the nogood ($X_1 = 2$, $X_2 = 1$) would produce a 3-clique, and therefore a solution. The external swap of the nogoods ($X_1 = 1$, $X_2 = 1$) with ($X_2 = 1$, $X_3 = 3$) will also produce a clique and a solution. This led us to believe that good swaps might be predictable from inspection of the maximum clique in a graph.

We searched some microstructures for a maximum clique, using Variable Neighborhood Search (*VNS*), a state-of-the-art local search algorithm that finds maximum cliques in a graph (Hansen, Mladenovic and Urosevic, 2004). In the microstructure of the five hardest solvable random problems HS_R, on 100 attempts per problem VNS found maximum cliques only of size 45 – 47. We also compared all solutions to each problem to the largest clique found in its microstructure. There was little similarity. Let the similarity between a clique in $m(P)$ and a solution to P be the number of assignments they share. The closest a largest identified clique ever came to any solution to its problem was 25, that is, half its assignments matched some solution and the other half did not. Indeed, the typical clique was similar in only 10-20 of its assignments.

By design, VNS is heavily attracted to vertices of high degree. Degree in the microstructure, however, indicates compatibility with many other values. Recall that if a constraint is not present between two variables in P, all possible edges between the vertices associated with them appear in $m(P)$. Thus variables with low degree in the graph of P are likely to be associated with vertices of very high degree in $m(P)$, and therefore attract VNS indiscriminately. The nature of the solution space provides better information.

The shape of the solution space

Let the *distance* between two solutions be the number of value assignments by which they differ, and let two solu-

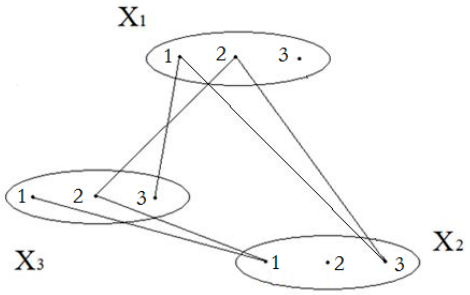

Figure 7: The microstructure of a simple unsolvable problem on 3 variables with 3 values, where a swap would produce a flip.

tions be *adjacent* if the distance between them is 1. In easy solvable SAT problems, most solutions are adjacent to one another, forming a dominant cluster (Krzakala et al., 2007). As the problems become more constrained, however, the solution space separates into exponentially many clusters that are smaller and more distant from one another (Achlioptas and Ricci-Tersenghi, 2006). Moreover, inside each cluster most variables assume only a single value. Eventually, as the problem moves through the range of the phase transition (Gent et al., 1999), the number of clusters decreases until there are no solutions at all.

The effectiveness of local search is heavily dependent on the shape of the solution space, and it is possible that a local search algorithm is effective because it exploits idiosyncrasies in that shape. (See, for example, (Feldman, Provan and van Gemund, 2008).) If a local search metaheuristic like VNS fails on the microstructure of a solvable problem drawn from classes at the phase transition, it is probably because the solution clusters are isolated and distant from one another. The relationship among sets of partial instantiations and the solution space also influences how readily a problem will flip. For instance, the hardest unsolvable problems are likely laden with isolated, nearly-complete instantiations; a few fortuitous swaps could turn one into a solvable problem, and our results suggest that that is relatively easy to do.

Conclusions and Current Work

This paper is the beginning of an extensive study on value perturbation. It is being extended to larger and smaller problems, and to problems with larger and smaller domains. To address additional facets of real-world problems, we also expect to explore structures other than small world problems, to work with problems that do not have uniform tightness, and to consider ways to extend the study to intensional constraints and to one-of-a-kind real-world problems as well.

There are, of course, many other ways to perturb values with or without structural changes to *P*. This work will move gradually from the low-level disruption of value swaps described here toward more extensive perturbation, eventually reaching the kinds of structural disruption studied in (Wallace, Grimes and Freuder, 2009).

A class of constraint satisfaction problems is characterized by some set of shared features. The two classes explored here have problems that are the same size (number of variables and domain size), and have the same number of edges and the same tightness on those edges. These experiments make clear that characterization along such dimensions does not address the inherent nature of a CSP — as researchers have long suspected, structure makes a difference. But, beyond structure, some problems differ fundamentally in the geometry of their solution space, which dictates how partial instantiations are permitted to become solutions, and how solutions arise together.

Acknowledgements

This work was supported in part by NSF IIS-0811437. We thank Gene Freuder and Rick Wallace for their thoughtful comments.

References

Achlioptas, D. and F. Ricci-Tersenghi 2006. On the solution-space geometry of random constraint satisfaction problems. In *Proceedings of STOC-06: 38th Annual ACM Symposium on Theory of Computing*, 130–139.

Boussemart, F., F. Hemery, C. Lecoutre and L. Sais 2004. Boosting systematic search by weighting constraints. In *Proceedings of ECAI-2004*, 146-149. IOS Press.

Feldman, A., G. Provan and A. J. C. van Gemund 2008. Computing minimal diagnoses by greedy stochastic search. In *Proceedings of AAAI-08*, 911-918.

Gent, I. E., E. MacIntyre, P. Prosser and T. Walsh 1999. The Constrainedness of Search. In *Proceedings of Thirteenth National Conference on Artificial Intelligence*, 246-252.

Gomes, C. P., B. Selman, N. Crato and H. Kautz 2000. Heavy-tailed Phenomena in Satisfiability and Constraint Satisfaction Problems. *Journal of Automated Reasoning* 24: 67-100.

Hansen, P., N. Mladenovic and D. Urosevic 2004. Variable neighborhood search for the maximum clique. *Discrete Applied Mathematics* 145: 117-125.

Hollander, M. and D. A. Wolfe 1999. *Nonparametric Statistical Methods*, 2nd edition. New York, John Wiley.

Jégou, P. 2003. Decomposition of domains based on the micro-structure of Finite Constraint-Satisfaction Problems. In *Proceedings of AAAI-93*, 731-736.

Krzakala, F., A. Montanari, F. Ricci-Tersenghi, G. Semerjian and L. Zdeborová 2007. Gibbs states and the set of solutions of random constraint satisfaction problems, *Proc Natl Acad Sci* 104(25): 10318-23.

MacIntyre, E., P. Prosser, B. Smith and T. Walsh 1998. Random Constraint Satisfaction: theory meets practice. In *Proceedings of CP-98*, 325-339. Springer Verlag.

Sabin, D. and E. C. Freuder 1997. Understanding and Improving the MAC Algorithm. *Proceedings of CP-97,* Berlin, Springer Verlag**:** 167-181.

Verfaillie, G. and N. Jussien 2005. Constraint solving in uncertain and dynamic environments: A survey. *Constraints* 10(3): 253-281.

Wallace, R. J., D. Grimes and E. C. Freuder 2009. Solving Dynamic Constraint Satisfaction Problems by Identifying Stable Features In *Proceedings of IJCAI-09*.

Walsh, T. 1999. Search in a Small World. In *Proceedings of IJCAI-99*, 1172-1177. Stockholm, Morgan Kaufmann.

Watts, D. and S. Strogatz 1998. Collective dynamics of small-world networks. *Nature* 393: 440.