Appears in: *Proceedings of the IEEE 2003 Congress on Evolutionary Computation, pages 1267-1273.*

Evolving Binary Constraint Satisfaction Problem Instances that are Difficult to Solve

J.I. van Hemert
jvhemert@cwi.nl
National Research Institute for Mathematics and Computer Science
CWI, P.O. Box 94079, NL-1090 GB Amsterdam, The Netherlands

Abstract- We present a study on the difficulty of solving binary constraint satisfaction problems where an evolutionary algorithm is used to explore the space of problem instances. By directly altering the structure of problem instances and by evaluating the effort it takes to solve them using a complete algorithm we show that the evolutionary algorithm is able to detect problem instances that are harder to solve than those produced with conventional methods. Results from the search of the evolutionary algorithm confirm conjectures about where the most difficult to solve problem instances can be found with respect to the tightness.

1 Introduction

The evolutionary computation community, as many other communities, has a tradition of comparative studies that compete for the best algorithm on selected sets of problem instances of one or a couple of problem domains. This process is valuable in a sense that it aids in sharpening the tools that the community has to offer. One drawback of this approach is that it offers the temptation to restrict studies to just a couple of problem sets whereby running the risk of overfitting the tools to certain properties of the chosen problem sets. Two major reasons for this temptation are first that one likes to compare with previously published results and second that running many different experiments and describing them takes valuable time and space.

In this study we take a different approach as we try to search the space of problem instances of a specific problem domain for difficult to solve problem instances. Naturally, this search needs to be directed somehow. We use the search effort measured in the amount of conflict checks required by a complete constraint solving method. This process helps us to identify the weak spots of a constraint solving technique.

The study will be performed using binary constraint satisfaction problems. By using chronological backtracking these problem instances are then solved or determined unsolvable. An evolutionary algorithm is used to search in the space of binary constraint satisfaction problems where the fitness of each problem instance is defined as the amount of conflict checks a solving technique requires to solve the problem or prove the problem unsolvable.

To show the potential of evolving binary constraint satisfaction problem instances we look at the convergence of our evolutionary algorithm. Also, we will provide information on the success of the evolutionary algorithm in finding hard to solve problem instances with respect to theoretical predictions and to the conventional approach. Last, we shall present more evidence that binary constraint satisfaction contains a double phase transition.

In the next section binary constraint satisfaction problems are explained, then in Section 3 the difficulty of solving these is discussed. In Section 4 the evolutionary algorithm is introduced, which is used in the experiments in Section 5. Conclusions are drawn in Section 6 and future directions are given in Section 7.

2 Binary Constraint Satisfaction

Constraint satisfaction problems (CSPs) (Tsang, 1993) form a class of models representing problems that have as common properties, a set of variables and a set of constraints. The variables should be instantiated from a discrete domain while making sure the constraints that restrict certain combinations of variable instantiations to exist, are satisfied. Some well known CSP problems are graph k-colouring, n-queens and 3-SAT. It is well known that the class of CSP is a subset of the class of NP-complete problems (Garey and Johnson, 1979). The study of CSP has become focused on binary constraint satisfaction problems, which restrict the general model by only allowing constraints over at most two variables. Rossi *et al.* proof that for every CSP there exists an equivalent binary CSP (Rossi et al., 1990). However, from empirical evidence we know that the method used to convert one CSP into another may have a large impact on the performance of the algorithms that try to solve it (Bacchus and van Beek, 1998; Walsh, 2000). Here we will be concerned only with binary CSPs created directly, i.e., no conversion has been applied.

A *constraint satisfaction problem* is defined as a tuple $\langle X, D, C \rangle$ where,

- X is a finite set of variables,
- D is a finite set of domains, one domain for each variable
- and C is a finite set of constraints that restrict certain simultaneous value assignments.

The objective is to assign each variable $x \in X$ one value v from its domain, denoted as $\langle x, v \rangle$, such that none of the

constraints in C is violated. This set of assignments is called a solution to the CSP. It is possible to have so many constraints present in the CSP that it is impossible to find a valid value assignment for each variable. Such problem instances are called *unsolvable*.

A *binary constraint satisfaction problem* (BINCSP) is CSP where every constraint $c \in C$ restricts two variables. Often, network graphs are used to visualise CSP instances. In Figure 1 we provide an example of a hypergraph of a BINCSP. It consists of three variables $\{x_1, x_2, x_3\}$, two of which have a domain of $\{a, b\}$ and one that has a domain of one element $\{a\}$. In a hypergraph every vertice corresponds with one value of one variable in the BINCSP it represents. Every edge shows the value pairs which are forbidden by the set of constraints C. In the example, we show all the edges that correspond to the following set of forbidden value pairs $C = \{\ \{\langle x_1, a\rangle, \langle x_2, a\rangle\}, \{\langle x_1, b\rangle, \langle x_2, a\rangle\}, \{\langle x_1, b\rangle, \langle x_2, b\rangle\}, \{\langle x_1, b\rangle, \langle x_3, a\rangle\}, \{\langle x_2, a\rangle, \langle x_3, a\rangle\}\ \}$.

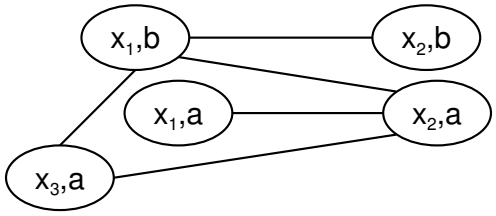

Figure 1: Example of a $|X|$-partite hypergraph of a BINCSP with one solution: $(\langle x_1, a\rangle, \langle x_2, b\rangle, \langle x_3, a\rangle)$

3 Problem difficulty

We can divide the class of all possible BINCSP into subclasses that share common properties. Quite commonly for BINCSP we choose to work on a number of these subclasses for which its members have the same number of variables, the same domain size for each variable and the same ratio of edges to possible edges in the corresponding hypergraph. These properties are called *order parameters*.

Cheeseman *et al.* show for a number of NP complete problems that these exhibit a transition from solvable to unsolvable problem instances when one looks at an order parameter that varies a certain aspect of the problem (Cheeseman et al., 1991). Such a transition in a complex system is referred to as a *phase transition*. BINCSP also exhibits such a transition (Prosser, 1994; Prosser, 1996) when one looks at the order parameters constraint density and constraint tightness. Here we will work with BINCSP in the form of hypergraphs, and the order parameter will be the ratio of forbidden value pairs to the number of possible value pairs. Basically, this is the overall tightness of the problem instance. We define this *tightness* of a BINCSP $\langle X, D, C\rangle$ as

$$\frac{|C|}{\binom{|X|}{2}|D|^2},$$

where C is the set of forbidden value pairs. The example in Figure 1 has a tightness of $5/8$.

An extensive study (Smith, 1994; Smith and Dyer, 1996; Prosser, 1996) on binary CSPs provides evidence that the phase transition coincides with a peak in the effort required to either find one solution for an instance or determine that it has no solution. The *search effort* is measured in the number of conflict checks needed by an algorithm to either find a solution or to determine that a problem instance has no solutions. A *conflict check* occurs when an algorithm tests whether the value assignment of two variables is valid, i.e., it is not forbidden by the set of constraints C.

To determine whether a problem instance is located in the phase transition an estimator is provided in (Williams and Hogg, 1994a), which uses the expected number of solutions and the conjecture that the most difficult problem instances have only one solution. In (Gent et al., 1996) a general order parameter is proposed, which also uses the number of solutions. By combining this with the estimator for the expected number of solutions we get,

$$\kappa = \frac{|X| - 1}{2} d \log_{|D|}\left(\frac{1}{1-t}\right), \qquad (1)$$

where t is the tightness of the problem instance and d is the ratio of constraints in the network graph, also called the density of the constraints. This latter notion is not evident in the representation of BINCSPs as hypergraphs, but the problem instances created in the experiments all have a density of one, which means that a constraint exists between every pair of variables. The phase transition from solvable to unsolvable occurs when $\kappa \approx 1$.

Apart from the needle in the haystack argument, i.e., one solution in a very large search space ($|D|^{|X|}$), this does not provide us with a reason why problem instances located at the phase transition are so difficult to solve. Another question related to this is why some problem instances that have the same tightness are more difficult to solve than others (Hogg, 1996). The most likely answer lies in the structure of the problem instances, i.e., the distribution of the conflicting value pairs. Some specific structural entities have been the object of study, among which are local graph topologies (Kwan et al., 1996; Dent and Mercer, 1996), parameters describing properties of the global graph (Williams and Hogg, 1994a) and the size of a substructure that has certain properties (Slaney and Walsh, 2001).

For graph colouring the most difficult to solve instances lie just outside the previously described phase transition, giving rise to a double phase transition, one with on average the most difficult to solve instances and one where the most difficult problem instances are found together with easy to solve problem instances (Williams and Hogg, 1994b). Smith makes a conjecture in (Smith and Grant, 1994) that such a double phase transition might also exist for binary constraint satisfaction.

The search continues for order parameters that are more exact in pinpointing the hardest to solve problem instances, mostly by defining a new order parameter and then verifying how well it predicts where difficult problem instances are to be found. Here we take another approach by making no assumption about what structure or property makes a BINCSP difficult to solve. The evolutionary algorithm is able to

freely search the space of problem instances, guided only by the search effort requirements of the constraint solver.

4 Evolving binary CSPs

The evolutionary algorithm maintains a population of binary CSPs of which it changes the structure over time. Basically, its genetic operators alter the individual conflict pairs between two values of two variables, i.e., the edges of its corresponding hypergraph. The set of variables and each variable's domain values are left untouched. This means that the size of the problem will not change. Only the ratio of forbidden value pairs to the total amount of value pairs can vary.

The following components make up the evolutionary algorithm, they are summarised in Table 1. The population consists of 30 binary CSPs. The initial population is created using Model E (Achlioptas et al., 2001) from the program RandomCsp (van Hemert, 170) with 15 variables, every domain size set to 15 and the p parameter of Model E is set to 0.02. This last parameter determines how many forbidden value pairs will exist in on average in the randomly created problem instances. The value chosen here makes sure that the initial population will consist of problem instances with many solutions, which are easy to find. The precise definition of Model E follows.

> The graph C^{Π} is a random $|X|$-partite graph with $|D|$ nodes in each part that is constructed by uniformly, independently and with repetitions selecting $p\binom{|X|}{2}|D|^2$ edges out of the $\binom{|X|}{2}|D|^2$ possible ones.

The evolutionary algorithm does 200 generations before it terminates and uses a generational scheme with elitism. Hence, the total number of evaluations, i.e., the number of problem instances created, for one run of the evolutionary algorithm is 6,000. To create the new population it uses tournament selection of size two to select two parents, after which uniform crossover is performed to create one offspring. This offspring is then mutated using standard mutation where the mutation rate *pm* is varied over the generations using the scheme,

$$pm = pm_{end} + (pm_{start} - pm_{end}) \cdot 2^{\frac{-generation}{bias}},$$

from (Kratica et al., 2003) where the parameters are set as $bias = 3$, $pm_{start} = 5000/\text{chromsome-size} = 0.211$, $pm_{end} = 1/\text{chromsome-size} = 1/23625$, and *generation* is the current generation. This scheme makes it possible to take reasonably large steps in the search space at the start, while keeping changes very small at the end of the run.

The uniform crossover operator takes two individuals, i.e., two binary CSPs, as input and creates a binary CSP with the same number of variables and domain sizes as the parents. It then iterates through all possible value pairs of the offspring, i.e., possible edges in the corresponding hypergraph, and chooses every time with equal probability either the first or the second parent. It sets the value pair to be a forbidden value pair if and only if the chosen parent's value pair is forbidden. Another iteration of the value pairs is made for the mutation where, with a chance of *pm*, the state of a value pair will be flipped.

component	*value*
individual representation	BINCSP instance
initialisation	Model E: $\langle 15, 15, 0.02 \rangle$
population size	30
crossover	uniform
mutation	uniform with changed mutation rate
parent selection	2-tournament
evolutionary model	generational with elitism
termination condition	200 generations
fitness function	search effort of chronological backtracking
goal	maximisation

Table 1: A summary of the components of the evolutionary algorithm for evolving BINCSP instances

To calculate the fitness of newly created offspring we run a complete constraint solving algorithm and take as the fitness for that offspring the effort the algorithm needs to either find one solution or to determine that the problem instance is unsolvable. Here we use chronological backtracking (Golomb and Baumert, 1965) where we count the number of conflict checks it performs. The goal of the evolutionary algorithm is to maximise this fitness function, thus to search for the most difficult to solve problem instances.

5 Experiments and results

The experiment consists of 100 independent runs of the evolutionary algorithm presented in Section 4. During a run we save the following statistics about every generated problem instance, its tightness, the effort needed to find a solution or determine that it is unsolvable, and whether it is solvable or not. At the end of the run we save the problem instance that needed the most effort over the whole run, commonly referred to as the best individual of the run.

Over the whole experiment at least 96.2% of all the problem instances created by the evolutionary algorithm are different. Hence the quality of the search as performed by the evolutionary algorithm is quite high as it has a low resampling ratio (van Hemert and Bäck, 2002), i.e., it tends not re-visit points in the search space too often.

5.1 Convergence

Figure 2 shows how the fitness of the evolutionary algorithm is converging averaged over the 100 runs. It starts out with a very fast increase in the fitness, i.e., difficulty of the problem instances, in the first four generations. Also, at the same time the tightness of the problem is rapidly climbing to about 0.38, as shown in Figure 3. Problem instances that have such a large tightness have a high probability of not being solvable, which is confirmed by Figure 4. In the

latter figure we observe that the ratio of solvable instances drops from one to almost zero within the first five generations. Then this ratio increases to almost 0.2 at 28 generations, after which it slowly decreases again. This matches the tightness in Figure 3, which converges to 0.319.

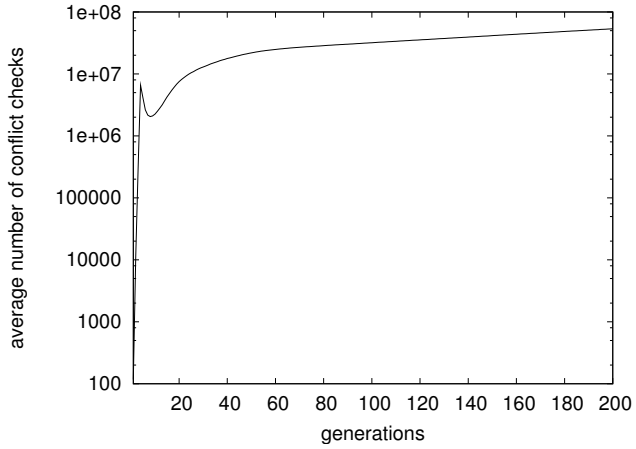

Figure 2: Average number of conflict checks required for solving the problem instances per generation, averaged over 100 runs

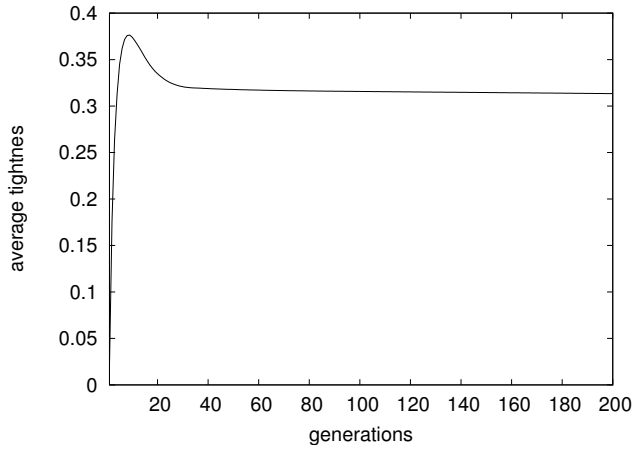

Figure 3: Average tightness of the problem instances per generation, averaged over 100 runs

The search of the evolutionary algorithm may be characterised as follows. The initial population of problem instances is very easy to solve (tightness of around 0.02). Then the search is directed towards unsolvable problem instances that lie quite far from the phase transition (tightness around 0.37). From there the focus of the search slowly moves towards the phase transition (tightness around 0.32) where it finds a small percentage of solvable problem instances. As the mutation rate approaches 1/mutation-rate the tightness keeps decreasing past the point where on average the hardest problem instances are found.

5.2 Comparison with Model E

Using Model E we generate one million randomly created BINCSP instances with a tightness equally distributed over

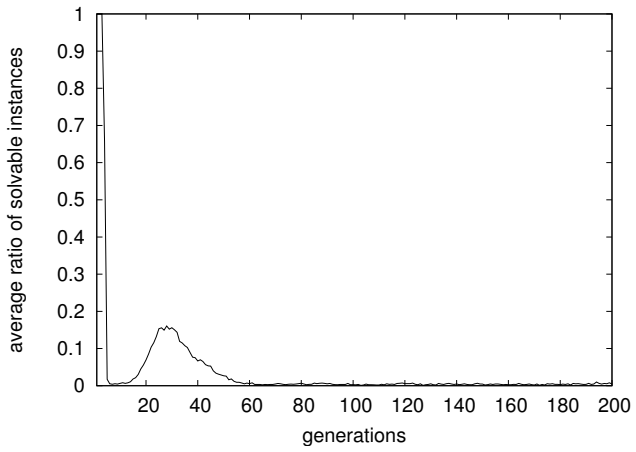

Figure 4: Average ratio of solvable instances per generation, averaged over 100 runs

the range $(0.29, 0.38)$. Of these problem instances we calculate the search effort in conflict checks using chronological backtracking. Then we separate them into two sets, those that are solvable and those that are not solvable. In Figure 5 and Figure 6 respectively the minimum, mean and maximum values are shown together with the most difficult problem instances found by the evolutionary algorithm for the tightness values in that range.

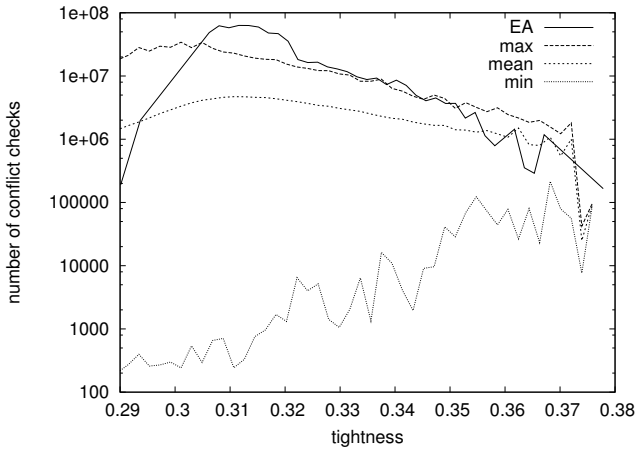

Figure 5: The number of conflict checks for *solvable* problem instances for the tightness in the range $(0.29, 0.38)$. Presented are the most difficult problem instances created by the EA over 100 runs and the maximum, mean and minimum of one million problem instances randomly created using Model E

On average the most difficult solvable problem instances are found at 0.312 (see Figure 5), however the most difficult to solve problem instances are found around 0.300. This provides evidence for the presence of a double phase transition. Nevertheless, when we observe the hardest problem instances found by the evolutionary algorithm we notice that these are found around the hardest average problem instances. Moreover, these problem instances are more difficult to solve than those randomly created using Model E.

Over the whole range we note that the easiest problem instances become more difficult to solve when approaching 0.38. At the same time the required search effort for the most difficult problem instances and problem instance of average difficulty is decreasing. As a result they approach each other, albeit with large variation in the required search effort between a tightness of 0.37 and 0.38. Thus, the variation in problem difficulty is highest around the phase transition.

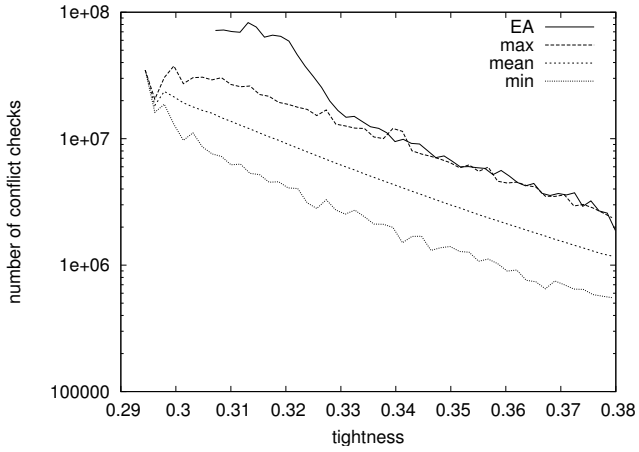

Figure 6: The number of conflict checks for *unsolvable* problem instances for the tightness in the range $(0.29, 0.38)$. Presented are the most difficult problem instances created by the EA over 100 runs and the maximum, mean and minimum of one million problem instances randomly created using Model E

The maximum, mean and minimum for unsolvable problem instances seems to contain less variation (see Figure 6). At the beginning of the range just below 0.30 we notice that the variation is very small. The smooth bump we get for the on average hardest solvable instances does not appear here. Instead no unsolvable problem instances are generated just below 0.295, where the three curves are converging and the hardest unsolvable problem instances are generated.

The evolutionary algorithm shows a remarkable leap that starts at 0.33 climbing very fast to 0.32 and then showing a region between 0.307 and 0.320 where it finds unsolvable problem instances that are extremely difficult to solve. Remember that during the search of the evolutionary algorithm it first jumps to high tightness values (0.37) before decreasing towards this latter region, i.e., in time the problem instances in this graph are on average generated from right to left.

5.3 The hardest problem instances

We observe that the most difficult to solve problem instances are unsolvable. A similar observation can be made about a larger part of the space searched by the evolutionary algorithm. In Figure 7 the problem instances that require the most search effort are given for both solvable and unsolvable problem instances. The hardest unsolvable problem instances always require more search effort than the hardest solvable problem instances.

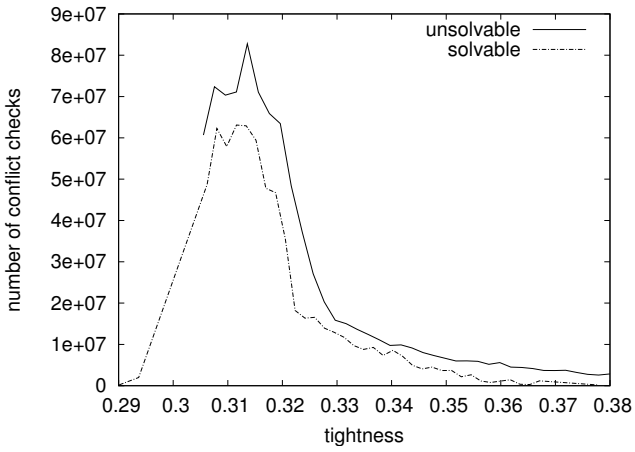

Figure 7: The number of conflict checks for the most difficult to solve instances found for the tightness in the range $(0.29, 0.38)$ over 100 runs, separated into solvable and unsolvable instances

This may explain the results in Figure 8, which shows that the evolutionary algorithm produces more unsolvable problem instances than solvable problem instances. Not only in total, but also for every tightness. This provides strong evidence that the property of not being solvable plays an important role in making problem instances hard to solve. Where solve means, to determine that a problem instance has no solutions.

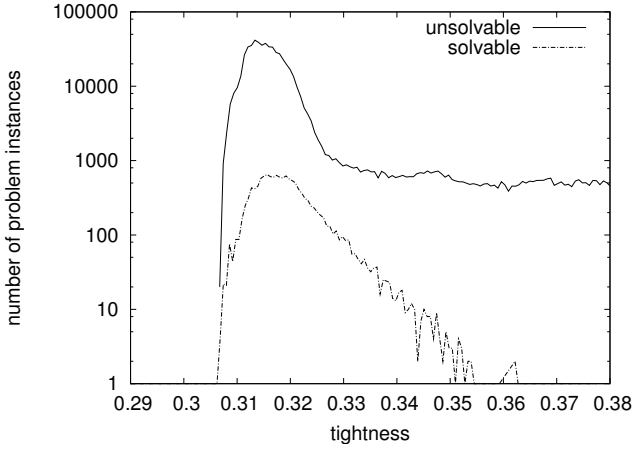

Figure 8: The number of problem instances created for the tightness in the range $(0.29, 0.38)$ over 100 runs, separated into solvable and unsolvable instances

By using Equation 1 we can determine κ for the most difficult to solve problem instances found in every run. On average this is 0.971, with a standard deviation of 0.011. This is very close to one, which shows that the evolutionary algorithm is well capable of locating the phase transition. Furthermore, it is also lower than one. This means the hardest to solve problem instances found by the evolutionary algorithm are in the region where we go from solvable to unsolvable problem instances.

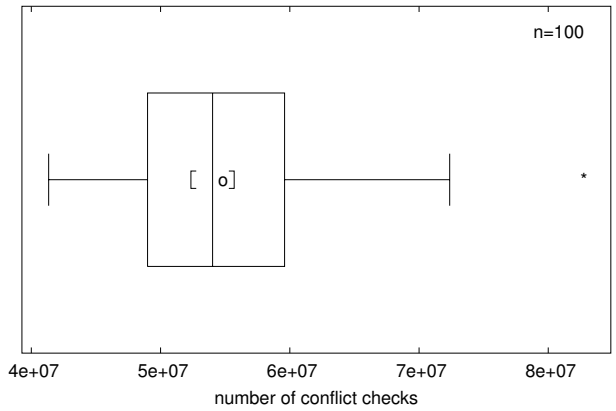

Figure 9: A box-plot for the conflict checks required to solve the hardest problem instances from each of the 100 runs

In Figure 7 three peaks may be identified for the unsolvable problem instances, at 0.308, at 0.314 and at 0.320. The latter one, which is more a bump than a peak, corresponds with the predicted phase transition, where problem instances are found that are harder than any of the solvable problem instances, but significantly easier to solve than in the first region. The other two peaks show that for lower than the predicted value and even lower than the peak in average problem difficulty we find hard to solve problem instances. Looking at the hardest problem instances created in each of the 100 runs we note than on average it requires chronological backtracking 5.48×10^7 conflict checks, with a standard deviation of 7.90×10^6, to determine that these instances have no solution. The most difficult to solve problem instance requires chronological backtracking 8.28×10^7 conflict checks to solve. However, statistically over the 100 runs this may be considered an outlier as we see in the box-plot in Figure 9.

6 Conclusions

Although on average problem instances become more difficult to solve near the phase transition, the variation in the search effort required to solve problem instances in a small range of the tightness may vary immensely around the phase transition. By using an evolutionary algorithm to evolve BINCSPs we are able to find the hard to solve problem instances without needing to know the exact structural properties that make these problem instance so difficult to solve. This enables us to create valuable sets of problem instances in an easy manner.

The hardest problem instances produced by the evolutionary algorithm surpass the difficulty of the conventional method where a model is used that produces problem instances with given properties by randomly distributing constraints and conflicts in every problem instance it creates. Also, the evolutionary algorithm shows a jump in search effort for unsolvable problem instances, which is lacking when observing the conventional method.

Two major differences may be identified between solvable and unsolvable problem instances. First, the hardest unsolvable problem instances found by the evolutionary algorithm are more difficult to solve than the hardest solvable problem instances for any tightness. Second, the solvable problem instances clearly show a smooth phase transition in search effort when observing one million randomly generated problem instances. Such a phase transition is lacking in the unsolvable problem instances, instead a peak is observed at a tightness of 0.295 below which no unsolvable problem instances are generated.

The exceptionally difficult problem instances, as they are called in (Smith and Grant, 1997), form a very small subset of the set of problem instances with the same order parameters. In (Smith and Grant, 1997) 50,000 problem instances are created for each setting of the tightness in order to locate them. Our evolutionary algorithm detects such problem instances and even harder ones by generating only 6,000 of them. Potentially, this makes it a valuable tool to locate such problem instances for analysis purposes.

The search of the evolutionary algorithm shows a clear picture of a double phase transition. One peak in search effort coincides with the predicted phase transition, where on average we find difficult problem instances. But for lower tightness values the evolutionary algorithm is able to find considerable harder problem instances.

7 Future directions

By changing the fitness function of the evolutionary algorithm we want to focus the search of the evolutionary algorithm towards solvable problem instances. This may provide a useful feature for experimental research as this sometimes requires a test set of solvable instances. For instance when dealing with incomplete methods where we want to measure how accurate a technique is in finding the solution.

To verify the success of using evolutionary computation to evolve hard problem instances we need to extend our study in two ways. First, we need to test other complete and incomplete methods as a basis for our fitness function. Preliminary experiments show that our method is rather robust when applying other complete methods for the same experimental setup described here. Second, we need to test if this method of acquiring difficult problem instances also applies to other problem domains. In the near future we are planning to study graph colouring problems and routing problems. The first is very similar to binary constraint satisfaction problems, albeit with other structural difficulties in the problem. The latter is more complex in the description of problem instances as these type of problems often contain many different components such as, vehicle, networks, customers and depots.

Another research direction is to have an evolutionary algorithm that evolves a constraint solving algorithm, where it undergoes co-evolution with the evolutionary algorithm presented here such that the best constraint solvers will have to compete with the hardest to solve problem instances.

References

Achlioptas, D., Kirousis, L., Kranakis, E., Krizanc, D., Molloy, M., and Stamatiou, Y. (2001). Random constraint satisfaction: A more accurate picture. *Constraints*, 4(6):329–344.

Bacchus, F. and van Beek, P. (1998). On the conversion between non-binary and binary constraint satisfaction problems. In *Proceedings of the 15th International Conference on Artificial Intelligence*. Morgan Kaufmann.

Cheeseman, P., Kenefsky, B., and Taylor, W. M. (1991). Where the really hard problems are. In *Proceedings of the* IJCAI *'91*, pages 331–337.

Cohn, A. G., editor (1994). *Proceedings of the European Conference on Artificial Intelligence*. Wiley.

Dent, M. J. and Mercer, R. E. (1996). A new model of hard binary constraint satisfaction problems. In *Proceedings of the Canadian Conference on AI 1996*, pages 14–25.

Garey, M. and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of* NP*-Completeness*. W.H. Freedman and Co.

Gent, I. P., MacIntyre, E., Prosser, P., and Walsh, T. (1996). The constrainedness of search. In *Proceedings of the* AAAI*-96*, pages 246–252.

Golomb, S. and Baumert, L. (1965). Backtrack programming. *A.C.M.*, 12(4):516–524.

van Hemert, J. (Version 1.7.0). Randomcsp: binary constraint satisfaction problem generator. Freely available under the GNU Public License at `http://freshmeat.net/projects/randomcsp`.

van Hemert, J. and Bäck, T. (2002). Measuring the searched space to guide efficiency: The principle and evidence on constraint satisfaction. In Merelo, J., Panagiotis, A., Beyer, H.-G., Fernández-Villacañas, J.-L., and Schwefel, H.-P., editors, *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, number 2439 in LNCS, pages 23–32, Berlin. Springer.

Hogg, T. (1996). Refining the phase transition in combinatorial search. *Artificial Intelligence*, 81:127–154.

Kratica, J., Ljubić, I., and Tošic, D. (2003). A genetic algorithm for the index selection problem. In et al., G. R., editor, *Applications of Evolutionary Computation*, volume 2611 of *LNCS*, pages 281–291. Springer-Verlag.

Kwan, A., Tsang, E., and Borrett, J. (1996). Predicting phase transitions of binary CSPs with local graph topology. In Wahlster, W., editor, *12th European Conference on Artificial Intelligence*, pages 185–189.

Prosser, P. (1994). Binary constraint satisfaction problems: Some are harder than others. In (Cohn, 1994), pages 95–99.

Prosser, P. (1996). An empirical study of phase transitions in binary constraint satisfaction problems. *Artificial Intelligence*, 81:81–109.

Rossi, F., Petrie, C., and Dhar, V. (1990). On the equivalence of constraint satisfaction problems. In Aiello, L. C., editor, *ECAI'90: Proceedings of the 9th European Conference on Artificial Intelligence*, pages 550–556, Stockholm. Pitman.

Slaney, J. and Walsh, T. (2001). Backbones in optimization and approximation. In Nebel, B., editor, *Proceedings of the seventeenth International Conference on Artificial Intelligence (*IJCAI*-01)*, pages 254–259, San Francisco, CA. Morgan Kaufmann Publishers, Inc.

Smith, B. (1994). Phase transition and the mushy region in constraint satisfaction problems. In (Cohn, 1994), pages 100–104.

Smith, B. and Grant, S. (1994). Sparse constraint graphs and exceptionally hard problems. Technical Report RR 94.36, University of Leeds.

Smith, B. M. and Dyer, M. E. (1996). Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1-2):155–181.

Smith, B. M. and Grant, S. A. (1997). Modelling exceptionally hard constraint satisfaction problems. In Smolka, G., editor, *Principles and Practice of Constraint Programming —* CP*97*, pages 182–195. Springer-Verlag.

Tsang, E. (1993). *Foundations of Constraint Satisfaction*. Academic Press.

Walsh, T. (2000). SAT *v* CSP. In *Proceedings of the 6th Conference on Principles and Practice of Constraint Programming (CP 2000)*, pages 441–456.

Williams, C. and Hogg, T. (1994a). Exploiting the deep structure of constraint problems. *Artificial Intelligence*, 70:73–117.

Williams, C. and Hogg, T. (1994b). The hardest constraint problems: A double phase transition. *Artificial Intelligence*, 69:359–377.