Adaptive Constraint Satisfaction: The Quickest First Principle

James E. Borrett, Edward P. K. Tsang and **Natasha R. Walsh**[1]

Abstract: The choice of a particular algorithm for solving a given class of constraint satisfaction problems is often confused by exceptional behaviour of algorithms. One method of reducing the impact of this exceptional behaviour is to adopt an adaptive philosophy to constraint satisfaction problem solving. In this paper we describe one such adaptive algorithm based on the principle of chaining and designed to avoid the phenomenon of exceptionally hard problems. Our algorithm shows how the speed of more naïve algorithms can be utilised safe in the knowledge that the exceptional behaviour is being bounded, clearly demonstrating the potential benefits of the adaptive approach.

1. MOTIVATION

The constraint satisfaction problem (CSP) can be defined in terms of the triple <Z, D, C>, where Z is a set of variables, D is a mapping of the variables in Z to domains and C is a set of constraints [1]. The task in solving a CSP is to assign a value to each of the variables in Z such that none of the constraints in C are violated. Given this definition there are many ways in which different types of CSPs can be classified, in terms of the elements of Z, D and C.[2] This classification can be used as a basis for the selection of a particular algorithm to solve that class of problems.

There is, however, a significant complication when defining classes of CSPs. Sometimes particular instances of problems in a class may exhibit exceptional qualities, in terms of the solving abilities of the chosen algorithm. One clear example of this is the phenomenon of exceptionally hard problems in easy CSPs [3,4,5], or EHPs as they shall be referred to in this paper.

The example of EHPs is illustrative of the dilemma posed to the problem solver. There is often a clear choice of either using a naïve algorithm which is likely to solve most instances very quickly, at the risk of catastrophic encounter with an EHP, or to choose a more complex algorithm, which has a far lower probability of encountering EHPs. The reduced susceptibility of more sophisticated algorithms to EHPs is discussed in [6,7]. However, their use often incurs a higher overhead.

One approach to overcoming this problem is to use parallel search agents [8]. This was found to be useful for CSPs in the hard region where the distribution of search costs is high.

In this paper we consider a more flexible approach which we describe as *adaptive constraint satisfaction*. It is designed to draw from the benefits of both simple and complex algorithms. The notion of adaptive constraint satisfaction can be encapsulated in the following description:

> *Adaptive Constraint Satisfaction is a general philosophy for solving constraint satisfaction problems. It aims to make use of the many algorithms and techniques available by relaxing the commitment to a single algorithm when solving a particular CSP, allowing for the active modification or switching of algorithms during the search process.*

We describe a particular instance of the adaptive approach where we make use of *algorithmic chaining*. The result is REBA (for Reduced Exceptional Behaviour Algorithm) which is designed to avoid the phenomenon of exceptionally hard problems in the so called easy region for solvable CSPs.

In the next section we discuss further our adaptive strategy. In section 3 we describe the REBA algorithm in detail. In section 4 the performance of the REBA algorithm is assessed and our conclusions are discussed in section 5.

2. THE ADAPTIVE STRATEGY

There are many possible strategies that might be used in the context of adaptive constraint satisfaction. We examine one particular adaptive strategy, designed to reduce the significance of EHPs by utilising algorithmic chaining. Algorithmic chaining uses a set of algorithms, arranged in a pre-determined order, combined with a switching mechanism. The switching mechanism monitors the search process of the current algorithm and, should certain conditions occur, stops the current algorithm, trying again with the next algorithm in the chain. In this section we discuss these two elements of the strategy.

2.1 Chain design

As noted in [6,7] the phenomenon of EHPs appears to affect different algorithms to different degrees. The trend tends to be for more naïve algorithms, such as simple chronological backtracking algorithms, to be more susceptible. This presents us with two potentially useful measures for ranking algorithms. The first is the cost to solve 'normal' occurrences of CSPs (e.g. the median cost), and the second is the algorithm's sensitivity to EHPs. An example of possible differences in ranking is given in Table 1.

Table 1. How the ranking of algorithms can differ when based on median cost of solving CSPs, and sensitivity to EHPs.

Rank	Algorithm Complexity	Median Cost	Sensitivity to EHPs
1	X	X	Z
2	Y	Y	Y
3	Z	Z	X

If we can determine rankings similar to those in Table 1, we would have enough information to design a useful chain for solving CSPs in the easy region whilst increasing the likelihood of avoiding the potentially catastrophic effects of encountering an EHP. The chain can simply be set to an ordering of algorithms based on the "Quickest First Principle" (QFP), where quickest indicates the algorithm with the best median performance.

We wanted to design an adaptive algorithm for solving easy solvable problems. Using QFP means that we always have the

[1] University of Essex, Wivenhoe Park, Colchester, Essex C04 3SQ, UK

[2] In [2] the issue of classifying different formulations of the same problem is considered.

ECAI 96, *12th European Conference on Artificial Intelligence*
Edited by W. Wahlster
Published in 1996 by John Wiley & Sons, Ltd

potential for solving the CSPs quickly. However, if we can detect that the current algorithm is not working well, we could switch to the next quickest algorithm, and so on[3]. As a result we can still benefit from the speed of the naïve algorithms, while at the same time, providing the capability to resort to more complex algorithms in the event that a switch scenario is detected.

While there is some overhead involved in this approach, the benefits can be considerable. For example, the ability to use a simple algorithm can result in an order of magnitude gain in performance over its more complex counterparts. Another advantage is that in the event of a bad initial choice of algorithm, we are not stuck with it. Mistakes of this nature will be rectified when we switch away from the bad choice.

2.2 Switching policy

One vital element to any adaptive algorithm is the ability to predict when it is profitable to abandon the current algorithm being used. At the same time, this prediction mechanism must add only minimal overheads. For REBA this means we need to predict the thrashing type behaviour associated with EHPs using a simple and efficient method.

At the heart of the switching mechanism of REBA is the MSL thrashing predictor which is described in detail in section 3.2. MSL attempts to predict when thrashing type behaviour is likely to occur such that only a small portion of any futile sub-search space is actually explored by the algorithm in question.

3. THE REDUCED EXCEPTIONAL BEHAVIOUR ALGORITHM (REBA)

Having outlined the basic strategy for our Reduced Exceptional Behaviour Algorithm, we give more details of its design and the switching mechanism it uses.

3.1 The REBA algorithm chain

The chain used by REBA consists of a selection of algorithms with good median performance on low density, easy soluble CSPs, and a selection of algorithms with good worst case performance. Having carried out some preliminary investigations, we chose to use the following algorithms;

BM+MWO - back-marking [10] with the minimum width ordering [11]
BMCBJ+MWO - back-marking with conflict-directed backjumping [12] and the minimum width ordering
BMCBJ+MDO - back-marking with conflict-directed backjumping and the maximum degree ordering[4]
FCCBJ+BZ - forward checking with conflict-directed backjumping [12] and the Brélaz ordering [13, 14]
MAC+MDO - maintain arc consistency [15] with the maximum degree ordering

We chained these algorithms in the following way;

BM+MWO → BMCBJ+MWO → BMCBJ+MDO → FCCBJ+BZ → MAC+MDO

The reasoning behind this chain is that BM+MWO is very fast for many easy soluble problems, but very susceptible to thrashing. However, it might succeed in finding a solution quickly, otherwise thrashing will be detected. In the event that BM+MWO fails, we try adding intelligent backjumping to it. If this fails, we try changing the ordering, since a bad ordering is often a contributing factor to EHPs [5]. If these simpler algorithms fall victim to an EHP, we attempt to use forward checking with conflict-directed backjumping and a dynamic variable ordering. Finally, if this fails, we resort to another algorithm which has relatively low susceptibility to EHPs, MAC+MDO.

3.2 The Monitor Search Level (MSL) thrashing predictor

As a basis for the design of MSL we defined the following functional specification;

Given a CSP, an algorithm, and a variable ordering, the predictor should monitor the progress of the search and be able to predict if thrashing is likely to occur during the search.

One indication of thrashing is when the search from a particular level i never proceeds beyond a certain depth, d, and that a large proportion of the search space between level i and level i+d is explored (i.e. little pruning takes place between these two levels). Such a situation can occur when the culprits of the failure at level i+d precede the level i. MSL is a computationally inexpensive method which uses this observation to predict thrashing type behaviour.

Before discussing MSL in more detail, we must identify three distinct types of progress which occur during search. These are presented in figure 1.

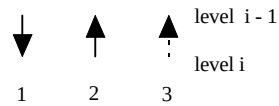


Figure 1. The types of progress during search

The types of progress are defined as;

1. A value is found for the current variable which is compatible with all past variables, or future variables in the case of lookahead algorithms.
2. Backtracking occurs after finding no values for the current variable which are compatible with past variables, or future variables in the case of lookahead algorithms. This will be known as a No Assigned Value (NAV) backtrack. The NAV backtrack occurs at the tail of the arrow, level i. At the head of the arrow, level i - 1 learns of an Unsuccessful Subspace Search (USS).
3. Backtracking occurs, but only after at least one value has been found for the current variable which is compatible with the assignments of previous variables, or future variables in the case of lookahead algorithms (meaning the search must have progressed at least one level further down than the current one). This will be known as a Successfully Assigned Values (SAV) backtrack. The SAV backtrack occurs at the tail of the arrow, level i. At the head of the arrow, level i - 1 learns of a USS.

[3] In [9] Frost and Dechter suggest a basic form of switching as a possible means of reducing the overhead incurred by their more complex Look-ahead Value Ordering based algorithms, when solving very easy CSPs

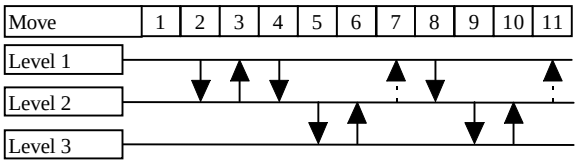
[4] A static ordering based on descending order of degree.

During the search MSL keeps track of the last level at which a NAV backtrack occurred. This is considered to be the deepest level of the current search sub-space. We will refer to this level as DEEPEST. In addition, for each level in the search, MSL keeps track of two values. Firstly a count indicating the number of USS's which returned to the level with the same value for DEEPEST. Secondly a record of the value of DEEPEST when this count is started. We will refer to these values as $count_i$ and DL_i respectively, where i is the level to which they refer.

In considering how the count is maintained, we must examine the seven possible cases. These depend on whether a USS, a NAV backtrack or a SAV backtrack is occurring, and what the value of DEEPEST is compared to the value of DL_i for the level. Table 2 illustrates the different actions taken at a given level, i, depending on these circumstances. DEEPEST and $count_i$ are initialised to 0 and DL_i is initialised to i.

Table 2. Possible actions of MSL on DEEPEST, $count_i$ and DL_i for level i

	(1) DEEPEST < DL_i	(2) DEEPEST = DL_i	(3) DEEPEST > DL_i
(a) USS	No action	Increase $count_i$ by 1; Check $count_i$ against threshold	Set $count_i$ to 1; Set DL_i to DEEPEST
(b) NAV	Set DEEPEST to i	Set DEEPEST to i	Not Possible
(c) SAV	Reset $count_i$ to 0; Set DL_i to DEEPEST	No action	Not Possible



Action i		_	A	b1	A	A	b1	c2	A	A	b2	c2
Action i-1		_		a3			a3	a3			a2	a2
DL_i	L1	1	1	***2***	2	2	2	***3***	3	3	3	3
	L2	2	2	2	2	2	***3***	3	3	3	3	3
	L3	3	3	3	3	3	3	3	3	3	3	3
$count_i$	L1	0	0	***1***	1	1	1	1	1	1	1	***2***
	L2	0	0	0	0	0	***1***	1	1	1	***2***	2
	L3	0	0	0	0	0	0	0	0	0	0	0
DEEPEST		0	0	***2***	2	2	***3***	3	3	3	3	3

Figure 2. Example search

Figure 2 illustrates some of the possible situations encountered by MSL. Each column in Figure 2 represents either an assignment, a NAV backtrack, or a SAV backtrack together with a USS if applicable (except the first column). The numbers below the arrow indicate the values of DL_1,...,DL_3, $count_1$,...,$count_3$ and DEEPEST **after** the actions for that column have been carried out. The values of the actions indicate which entries in Table 2 applies to the above arrow[5]. This includes actions at both the tail and the head of the arrow. The first column simply shows the initial values before the search begins.

As an example consider columns 5 to 7. Column 5 shows a simple assignment to the variable at level 2, action A. No further actions take place.

Column 6 then shows a NAV backtrack from the variable at level 3. When the backtrack occurs, $DL_3 = 3$ and DEEPEST = 2, so DL_3 > DEEPEST and entry b1 in Table 2 applies to level 3. As a result DEEPEST is set to the value of i, i.e. DEEPEST = 3. At the head of the arrow USS entry a3 applies (because DEEPEST = 3 and $DL_2 = 2$) and $count_2$ is set to 1 with DL_2 being set to DEEPEST.

Column 7 shows a SAV backtrack from the variable at level 2. When the backtrack occurs DL_2 = DEEPEST and entry c2 in Table 2 applies and no action is taken at level 2. At the head of the arrow USS entry a3 applies and $count_1$ is set to 1 with DL_1 being set to DEEPEST.

3.2.1 Effectiveness of thrashing prediction mechanisms

Having defined the function of the prediction mechanism, we also define a set of criteria for evaluating its effectiveness. These criteria are;

1. It should predict as exceptionally hard those problems with high search cost for the current algorithm.
2. The computational cost of predicting a CSP to be exceptionally hard should be low and preferably not exceed the median cost. It should also be cheap in terms of space.
3. It should not be so sensitive that too many problems are predicted to be exceptionally hard. A high proportion of the problems with search costs of median or lower should not be predicted to be exceptionally hard for the current algorithm.

3.3 The REBA Switching Mechanism

The MSL predictor is used by REBA for its switching mechanism. This is done by REBA supplying the predictor with a formula for calculating the threshold. If $count_i$ exceeds the threshold at level i, then MSL suggests that a switch should take place. This causes REBA to switch to the next algorithm in the chain.

We have experimented with a threshold based on the domain size of the variables, and the number of levels separating the current level i and DL_i. The base threshold is a multiple of the domain size. The number of separating levels is taken as DL_i - i. The more separating levels, the lower the threshold has to be for switching to occur. The formula used is;

$$Threshold = base * \left(\frac{n - separation}{n} \right) \qquad (1)$$

Where *base* is the base threshold, which is a linear function of the domain size, *n* is the number of variables and *separation* is the number of separating levels (DL_i - i)

The threshold is adjusted according to separation to improve the sensitivity of detection when the subspace is only searched sparsely, as might be the case with intelligent backjumping algorithms. Note that in the subsequent experiments a suffix is given to the name of REBA. This suffix indicates the multiples of the domain size used for the base threshold.

4. EXPERIMENTS

In order to evaluate the overall performance of REBA and the effectiveness of its switching mechanism we carried out

[5] The entry A indicates a successful assignment, no action is taken.

experiments on sets of easy soluble CSPs (the class of problems REBA is designed to tackle).

4.1 Experimental design

The main aim of our experiment was to compare the performance of REBA with two types of algorithms - those exhibiting good median performance in the easy soluble region, and those that have a good worst case performance on easy soluble region. The actual CSPs we used were based on randomly generated binary CSPs classified by the tuple <n, m, p1, p2>, where the elements of the tuple are defined as;

- n number of variables
- m uniform domain size
- p1 density of constraints in the constraint graph
- p2 tightness of individual constraints[6] i.e. the percentage of incompatible assignments between the two variables involved in the constraint

Specifically, we wanted to conduct our experiments on problems in the easy soluble region where exceptionally hard problems were likely to occur. As a result, we chose the class <50, 10 , 0.1, 0.35 - 0.5 >. This range of p2 gives us a spread of problems in the region of interest and it also includes some of the sets of problems used in [6,7] where EHPs were investigated.

The algorithms we chose for comparison, based on initial tests of problems in the class described above, were BMCBJ+MWO, giving a low median performance but a sensitive worst case performance in the region of interest, FCCBJ+BZ, giving a relatively high median performance but a good worst case performance in the region of interest and finally MAC+MDO giving a relatively high median performance but a good worst case performance in the region of interest.

The CSPs for our experiments were generated at intervals of p2 of 0.01 and the sample size for each data point was 1000. In order to limit the impact of EHPs on our experimentation time, we limited the actual process CPU time for any given run to 30 minutes. Where this time is exceeded, the compatibility check count up to that time was recorded[7]. The effect of using such a limit is that for a few data points, for the BMCBJ+MWO combination, the limit was reached. This does not detract from the essence of our results, however, since the effect of any EHP is still clearly visible. The truncated values are many orders of magnitude above the median search cost. We also only present CPU time results for MAC since our implementation is the same as that of [15] where the compatibility check count is not a true reflection of the work done by MAC.

4.2 The effectiveness of REBA

The results of our experiment in measuring the effectiveness of REBA are presented in figures 3-6. They clearly show that the use of algorithmic chaining in REBA has produced a good worst case performance where the impact of EHPs has been significantly reduced. This is evident in the worst case plots of figures 4 and 6. REBA even outperforms FCCBJ+BZ in many cases. At the same time, the median performance of REBA is much better than that of the more complex algorithms, in most cases. This is particularly apparent when the CPU time is considered as in figures 5 and 6.

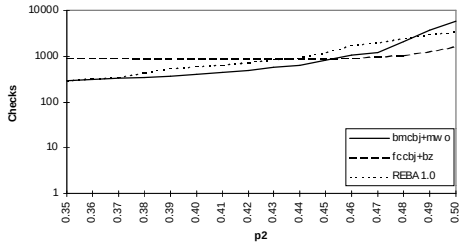


Figure 3. median performance on 50 variable problems in terms of compatibility checks

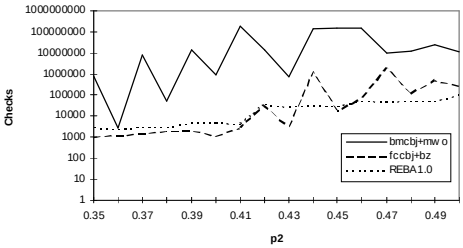


Figure 4. worst case performance on 50 variable problems in terms of compatibility checks

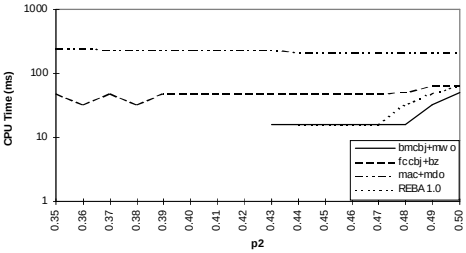


Figure 5. median cpu time[8] performance on 50 variable problems

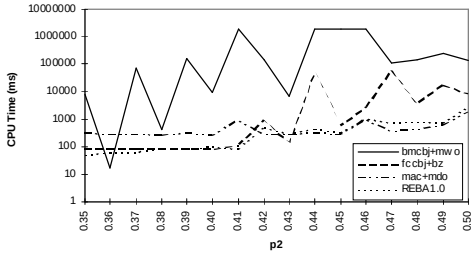


Figure 6. worst case cpu time performance on 50 variable problems

It should be noted that we have tested REBA on easy problems. This is because we advocate that different types of problem would be tackled by different algorithms as discussed in [20]. REBA, by design, appears to be useful in tackling problems in the easy region on the soluble side of the phase transition. It is the subject of further work to investigate the applicability of the strategies

[6] The original definition, which we have used in our previous work, was given in [16,17] as being the constraint looseness, or percentage of compatible labels in the binary constraint relation matrix. The more modern interpretation such as used in [18,19] has p2 as being the percentage of incompatible labels in the binary constraint relation matrix.

[7] The algorithms were implemented in C++ and run on Sun SPARCstation 5 workstations running at 85 MHz with SunOS 4.1.3

[8] Where the plot for REBA and BMCBJ+MWO does not exist this means the median time was less that one clock cycle and hence does not show in the logarithmic scale

used in REBA to tackling other problem types such as those in the phase transition.

4.3 Evaluation of the MSL predictor

To see how effective the MSL predictor was we carried out further experiments. We ran a version of BM+MWO, which included the MSL predictor, and monitored where any switch was predicted. If a switch was predicted, the number of compatibility checks was recorded and the algorithm was allowed to continue running to completion to see what the actual outcome would have been. A problem set of 1000 CSPs was generated with the specification <50, 10, 0.1, 0.4>, and a base threshold equal to the domain size was used.

From the sample there were 589 cases where a switch was predicted and many of these had high cost searches if allowed to complete. We found that the median cost for predicting a switch in BM+MWO was always less than the median search cost when all CSPs are considered. We also found that for this particular problem set, the 411 CSPs where no switch was predicted were solved within the median cost.

More comprehensive analysis of the performance of MSL is given in [21].

5. DISCUSSION

In this paper we have demonstrated the potential of adaptive constraint satisfaction. We have outlined a particular application of the adaptive approach using the technique known as algorithmic chaining. This technique was incorporated in an algorithm that we have named REBA, and has been shown to be effective in reducing susceptibility to exceptionally hard problems.

The REBA algorithm makes use of a mechanism for predicting when thrashing type behaviour is likely to occur. This notion of prediction is one of the keys to the adaptive approach since it is prediction that allows algorithms to avoid problem search spaces before they can impact significantly on the overall search. The MSL mechanism used here is very cheap to implement and it has been shown to be reasonably accurate.

Experiments with the REBA algorithm, which is specifically designed to reduce the impact of exceptionally hard problems, show that it is possible to take advantage of the speed of basic constraint satisfaction algorithms when solving easy, soluble CSPs, while at the same time allowing us to bound the exceptional behaviour of these algorithms when exceptional problem instances are encountered. The principle of using the quickest algorithm first means that the best case performance of the naïve algorithms always has a chance of being achieved. It also gives the opportunity for fast solutions to be provided in the event that "exceptionally easy" problems are encountered - this could be significant if a similar method were to be used on, for example, hard classes of CSPs.

We believe our work has opened many new areas of future work. We intend to further investigate the use of chains and similar methods of choosing appropriate algorithms to switch to in types of problems other than soluble easy CSPs. We also intend to look at other methods for detecting when it would be useful to switch between algorithms. This would involve identifying useful information that can be gathered during search. The actual process of switching could also be a source of improvement in efficiency, with the possibility of transferring information gathered to successive algorithms.

ACKNOWLEDGEMENTS

This work was supported by EPSRC research grant ref. GR/J/42878. The authors would like to thank Alvin Kwan for his useful comments on the contents of this paper. We should also like to thank the ECAI 96 reviewers for their useful comments.

REFERENCES

[1] Tsang, E. P. K., 1993, Foundations of Constraint Satisfaction, Academic Press, London

[2] Borrett, J. E. & Tsang, E. P. K., 1995, On the Selection of Constraint Satisfaction Problem Formulations, Technical Report CSM254, Dept. of Computer Science, University of Essex, Colchester

[3] Gent, I. P. & Walsh, T., 1994, Easy Problems are Sometimes Hard, Artificial Intelligence **70**, 335-345

[4] Hogg, T. & Williams, C. P., 1994, The Hardest Constraint Satisfaction Problems, Artificial Intelligence **69**, 359-377

[5] Smith, B., 1994, In search of Exceptionally Difficult Constraint Satisfaction Problems, Proceedings of the Workshop on Constraint Processing, 11th European Conference on Artificial Intelligence, 79-86

[6] Smith, B. & Grant, A., 1995, Sparse Constraint Graphs and Exceptionally Hard Problems, Proceedings 14th International Joint Conference on Artificial Intelligence, 646-651

[7] Smith, B. & Grant, A., 1995, Where the *Exceptionally* Hard Problems are, Workshop on Studying and Solving Really Hard Problems, Principles and Practice of Constraint Programming - CP95, 172-182

[8] Hogg, T. & Williams, C. P., 1994, Expected Gains form Parallelizing Constraint Solving for Hard Problems, Proceedings 12th National Conference on Artificial Intelligence, 331-336

[9] Frost, D. & Dechter, R., 1995, Look-ahead value ordering for constraint satisfaction problems, Proceedings 14th International Joint Conference on Artificial Intelligence, 572-578

[10] Gaschnig, J., 1977, A General Backtrack Algorithm That Eliminates Most Redundant Tests, Proceedings 5th International Joint Conference on Artificial Intelligence, 457

[11] Freuder, E. C., 1982, A sufficient Condition for Backtrack-Free Search, Journal of ACM, vol. 29, 24-32

[12] Prosser, P., 1993, Hybrid Algorithms for the Constraint Satisfaction Problem, Computational Intelligence, Vol. 9, 268-299.

[13] Brélaz, D., 1979, New methods to color the vertices of graphs, Communications of the ACM, 22(4), 251-256

[14] Turner, J. S., 1988, Almost all k-Colorable Graphs are Easy to Color, Journal of Algorithms, vol 9, 63-82

[15] Sabin, D. & Freuder, E. C., 1994, Contradicting Conventional Wisdom in Constraint Satisfaction, Proceedings 11th European Conference on Artificial Intelligence, 24-129

[16] Haralick, R. M. & Elliott, G. L., 1980, Increasing Tree Search Efficiency for Constraint Satisfaction Problems, Artificial Intelligence **14**, 263-31

[17] Nudel, B., 1983, Consistent-Labelling Problems and their Algorithms: Expected-Complexities and Theory-Based Heuristics, Artificial Intelligence **21**, 135-178

[18] Prosser, P., 1994, Binary Constraint Satisfaction Problems: Some are Harder than Others, Proceedings 11th European Conference on Artificial Intelligence, 95-99

[19] Smith, B., 1994, Phase Transition and the Mushy Region in Constraint Satisfaction Problems, Proceedings 11th European Conference on Artificial Intelligence, 100-104

[20] Tsang, E. P. K., Borrett, J. E. & Kwan A. C. M., 1995, An Attempt to Map a Range of Constraint Satisfaction Algorithms and Heuristics, in Proceedings AISB 95, 203-216.

[21] Borrett, J. E., Tsang, E. P. K. & Walsh, N. R, 1995, Adaptive Constraint Satisfaction: The Quickest First Principle, Technical Report CSM256, Dept. of Computer Science, University of Essex, Colchester