The Goldilocks Problem*

Tudor Hulubei and Eugene C. Freuder
Department of Computer Science
University of New Hampshire
Durham, NH, 03824, USA
tudor,ecf@cs.unh.edu

Abstract

A lot of work in Constraint Satisfaction has been focused on finding solutions to difficult problems. Many real life problems however, while not extremely complicated, have a huge number of solutions, few of which are acceptable from a practical standpoint. In this paper we will present a heuristic that attempts to guide the search towards solutions that are acceptable given a set of metrics. In practice, this new heuristic can be used to suggest upgrades to existing configurations (upselling) and indeed that was the initial motivation of our work.

Introduction

"So away upstairs she went to the bedroom, and there she saw three beds. There was a very big bed for Father bear, but it was far too high. The middle-sized bed for Mother bear was better, but too soft. She went to the teeny, weeny bed of Baby bear and it was just right." – Goldilocks and the Three Bears.

Many practical problems have search spaces so huge that searching them exhaustively is impossible in practice. We have a lot more options to consider than Goldilocks did, and since we don't know if any of them will be exactly right, we are willing to be somewhat flexible about our interpretation of "just right", and stop our testing when we find one that is "close enough". When solutions can be ranked along a certain dimension, we may want to look for solutions at a desired point along that scale. There may not be a solution at that exact point, or it may be too costly to search until we find an exact match, so we are willing to specify a tolerance range around that point in which to search. We will present an algorithm that is good at finding a solution within such a range.

Most search algorithms use heuristics to guide the search towards those regions of the search space that are believed to contain solutions. The most common types of heuristics are variable and value ordering heuristics. When solving a particular class of problem, in addition to relying on general purpose heuristics (like `min-domain`), class-specific heuristics can be used in order to take advantage of the structure of the class at hand. In this paper we will describe a heuristic that can be used for a particular class of binary constraint satisfaction problems where each value in the domain of a variable and each allowed pair of values in a constraint has an associated weight(Bistarelli *et al.* 1996).

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

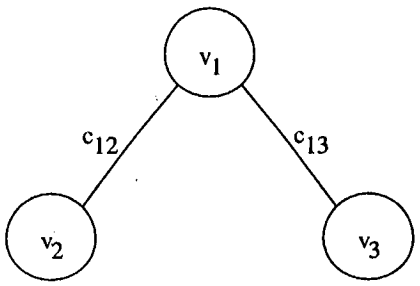

Figure 1: Small example (weights are given in *italics*):
v_1={0=*0.2*,1=*0.8*},
v_2={1=*0.1*,2=*0.7*},
v_3={-1=*0.8*,4=*0.9*},
c_{12}={(0,1)=*0.1*,(1,1)=*0.7*,(1,2)=*0.9*},
c_{13}={(0,-1)=*0.2*,(0,4)=*0.3*,(1,-1)=*0.5*}

Example

The following example will give a short preview of the things that are going to be discussed in the next sections. Consider a small problem consisting of three variables and two constraints (Fig.1). Both the values in the domain of a variable, and the pairs of values in a constraint have an associated weight, because the goodness of a solution is usually influenced by both the quality of the components and the way they are interconnected.

The weight of a solution to this problem is defined as the sum of the weights of all the values and pairs of values involved in the solution. We can easily see that v_1={1}, v_2={1} and v_3={-1} is a solution to our problem, and its weight is (0.8+0.1+0.8)+(0.7+0.5)=2.9.

We can compute lower and upper bounds for the so-

example, the lower and upper bounds are: MinSW=(0.2+0.1+0.8)+(0.1+0.2)=1.4, and MaxSW=(0.8+0.7+0.9)+(0.9+0.5)=3.8. That is, no solution weight can be smaller than MinSW or greater than MaxSW.

Definitions

Let $P = \{V, C\}$ be a constraint satisfaction problem with a set of variables $V = \{v_i | v_i$ a variable involved in $P\}$, and a set of constraints $C = \{c_{ij} | c_{ij}$ a constraint between v_i and $v_j\}$. For each $v_i \in V$ we define $domain(v_i) = \{x_{ik} | x_{ik}$ is a value that can be assigned to v_i, with $1 \leq k \leq |domain(v_i)|\}$. Each value x_{ik} has an associated weight $vw_{ik} \in [0, 1]$.

Each constraint c_{ij} defines a set of pairs of values that are allowed by the constraint. That is, $c_{ij} = \{(x_{im}, x_{jn}) | x_{im} \in domain(v_i), x_{jn} \in domain(v_j)$, and (x_{im}, x_{jn}) is allowed by the constraint$\}$. We associate a weight $pw_{ij}(x_{im}, x_{jn}) \in [0, 1]$ with each pair of values $(x_{im}, x_{jn}) \in c_{ij}$[1]. If a pair $(x_{im}, x_{jn}) \notin c_{ij}$ then that pair is not allowed by the constraint. Finally, if $c_{pq} \notin C$, then all the possible pairs of values from v_p and v_q are implicitly allowed, but their weights are undefined.

A *solution* to the problem P is a set of values $S = \{x_{1a_1}, x_{2a_2}, ..., x_{|V|a_{|V|}}\}$ s.t. $\forall i, j \leq |V| \; \exists c_{ij} \in C \Rightarrow (x_{ia_i}, x_{ja_j}) \in c_{ij}$. A partial solution consists of a set of values $\{x_{a_1b_1}, x_{a_2b_2}, .., x_{a_kb_k}\}$ s.t. $k < |V|$, and $\forall m, n \leq k \; \exists c_{a_ma_n} \in C \Rightarrow (x_{a_mb_m}, x_{a_nb_n}) \in c_{a_mb_n}$. We define the *weight of a solution* as the sum[2] (over all the variables and constraints) of the weights of all the values and pairs of values involved[3]. Formally, $sw(S) = \sum_{v_i \in V} vw_{ik} + \sum_{c_{ij} \in C} pw_{ij}(x_{ia_i}, x_{ja_j})$ with $x_{ik}, x_{ia_i}, x_{ja_j} \in S$, and $(x_{ia_i}, x_{ja_j}) \in c_{ij}$. The weight of a partial solution is defined similarly, with the exception that only variables included in the partial solution and the constraints among them are taken into account.

For each variable v_i we can compute the *range* of possible weights by simply taking the minimum and maximum weights among all the values in the domain of that variable. For all k s.t. $1 \leq k \leq |domain(v_i)|$, and $x_{ik} \in domain(v_i)$:

$$MinVW(v_i) = min(vw_{ik})$$
$$MaxVW(v_i) = max(vw_{ik})$$

Similarly, for each constraint c_{ij} we can compute the *range* of possible weights by taking the minimum and maximum weights among all the pairs of values allowed by the constraint. For all a, b s.t. $1 \leq a \leq |domain(v_i)|, 1 \leq b \leq |domain(v_j)|$, and $(x_{ia}, x_{jb}) \in c_{ij}$:

[1] Exactly how the weights are selected is outside of the scope of this paper. See (von Winterfeldt & Edwards 1986) for details.

[2] See (Bistarelli *et al.* 1996) for a discussion on egalitarianism versus utilitarianism.

[3] Clearly, there will be only one such value per variable, and only one such pair of values per constraint.

Lower and upper bounds for the weight of a solution to P are computed as:

$$MinSW(P) = \sum_{v_i \in V} MinVW(v_i) + \sum_{c_{ij} \in C} MinCW(c_{ij})$$
$$MinSW(P) = \sum_{v_i \in V} MinVW(v_i) + \sum_{c_{ij} \in C} MinCW(c_{ij}).$$

Note that there might be no solutions with these extreme weights, all we are saying here is that $\forall S, MinSW(P) \leq sw(S) \leq MaxSW(P)$. In order to compute the exact minimum and maximum solution weights we would have to look at all the solutions.

The "acceptable-weight" Heuristic

One way of looking for solutions with weights in a given range is to simply use a general purpose search algorithm (like **MAC** (Sabin & Freuder 1994)) and every time a solution is found check whether or not its weight is within the acceptable range. However, unaware of the importance of the solution weight, such an algorithm will look for *any* solution, leaving to chance the discovery of an acceptable one.

The `acceptable-weight` heuristic is an attempt to do better than that by guiding the search towards areas of the search space that are likely to contain solutions with acceptable weights.

Consider a constraint satisfaction problem P and two positive real numbers, $MinASW$ and $MaxASW$, representing the minimum and maximum *acceptable* solution weights, with

$$MinSW(P) \leq MinASW \leq MaxASW \leq MaxSW(P).$$

A solution S is *acceptable* if:

$$MinASW \leq sw(S) \leq MaxASW.$$

Given the range of acceptable solution weights $[MinASW, MaxASW]$, we consider the *ideal* solution weight ($IdealSW$) as being at the center of that range[4].

During the search, a constraint that involves at least one variable that has not been instantiated (i.e. assigned a value) is considered *active*, while a constraint that involves only instantiated variables is considered *inactive*. This distinction is useful when computing the weight of a partial solution, since only *instantiated* variables and *inactive* constraints can be considered. For brevity, we define the *weight* of an instantiated variable as the weight of the value that has been assigned to it, and the *weight* of an inactive constraint as the weight of the pair of values assigned to the variables involved in that constraint.

The idea behind `acceptable-weight` is to keep track of the weight of the current partial solution S' and attempt to obtain a solution for the rest of the problem

[4] Actually, this can be viewed in two ways: either start with a $[MinASW, MaxASW]$ range and consider the midpoint as an useful point to head for, or start with a "target" weight and define a tolerance around it. Our algorithm will work both ways.

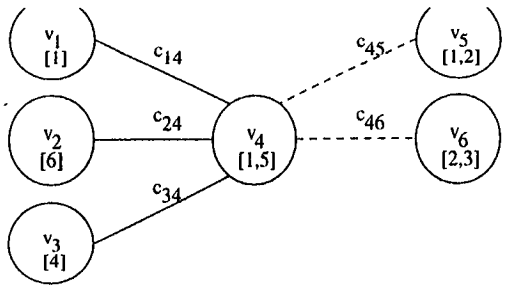

Figure 2: The instantiation of the current variable:
v_1={1=*0.1*},
v_2={6=*0.2*},
v_3={4=*0.4*},
v_4={1=*0.4*, 5=*0.8*},
c_{14}={(1,1)=*0.1*, (1,5)=*0.2*},
c_{24}={(6,1)=*0.8*, (6.5)=*0.9*},
c_{34}={(4,1)=*0.6*, (4,5)=*0.7*}

whose weight, combined with the first one, will bring the global solution weight as close to *IdealSW* as possible. Based on p, the number of currently *uninstantiated* variables, and q, the number of currently *active* constraints in the problem, `acceptable-weight` computes the average of the individual weights (AIW) that these variables and constraints would contribute to the global solution weight $sw(S)$, should it equal *IdealSW*:

$$AIW = \frac{\text{ISW} - \text{sw(S}')}{p+q}.$$

When the heuristic attempts to suggest a value for a variable, it considers the subproblem $P'' = \{V'', C''\}$ with $V'' \subset V$ containing the current variable and the past variables with which it is connected by constraints, and $C'' \subset C$ containing all the constraints between the current variable and past variables. The ideal weight of a solution S'' to P'' would be:

$IdealSW'' = \sum_{v_i \in V''} vw_{ik} + AIW + |C''| \cdot AIW$,

with $x_{ik} \in domain(v_i)$ and $x_{ik} \in S'$.

The `acceptable-weight` heuristic will select the value that will minimize the absolute difference between $sw(S'')$ and $IdealSW''$. The following example will illustrate a typical situation.

The values in the domain of each variable and the pairs of allowed values in each constraint are given between parenthesis. The number printed in *italics* represents the corresponding weight. Future variables are depicted, but not considered - this might be the subject of further improvements.

In this example the search is at the point of instantiating v_4. The possible values are 1 (whose weight is 0.4) and 5 (whose weight is 0.7), and the `acceptable-weight` heuristic is supposed to suggest one of them. Let us take a look at the implications of selecting each value, assuming that $AIW = 0.4$ at this point in the search.

$sw(S'') = (0.1+0.2+0.4)+0.4+(0.1+0.8+0.6) = 2.6$. If we choose 5, then $sw(S'') = (0.1+0.2+0.4)+0.8+(0.2+0.9+0.7) = 3.3$. After computing $IdealSW'' = (0.1+0.2+0.4)+AIW+3 \cdot AIW = 2.3$, we see that selecting the value 1 will yield a better weight for S'' (closer to $IdealSW''$). After the assignment of v_4, `acceptable-weight` recomputes the AIW, to compensate for the amount we were off.

The strategy behind the heuristic described is twofold. Locally, we try to make sure that each small subproblem centered around the current variable has a weight that is in line with the global *IdealSW*. Globally, by constantly adjusting the AIW we try to control the overall deviation of the solution weight. This strategy appears to work well in practice, as we will see in the next section.

Experimental Results

We have performed some experimental tests on problems that have been randomly generated (weights included) using a random parameter value model. Our tests compare the performance of MAC[5] with MAC+`acceptable-weight`. The algorithms were used to find acceptable solutions to problems with 100 variables, each variable having 5 values in its domain. The constraint density and tightness[6] were used to vary the difficulty of the generated problems and study the changes in the behavior of `acceptable-weight`. In the following tests we will compare the time required by the two algorithms to find an acceptable solution. Both algorithms will look for solutions in ranges of a given size (0.05 and 0.1 in the examples below) centered around the solution weight on the X axis, which has been translated and scaled from $[MinSW(P), MaxSW(P)]$ to $[0, 1]$. We arbitrarily set a timeout limit at 5 seconds.

Before going any further, we need to briefly explain the behavior of MAC. Without going into details, we will just say that, probabilistically speaking, there are many solutions with weights around 0.5 and very few solutions with extreme weights (close to 0 or 1). MAC doesn't take weights into account, and thus from its point of view the solutions are uniformly distributed throughout the search space. However, since most solutions have weights around 0.5, MAC will tend to find those pretty quickly.

The first test we did was with density = 0 and tightness = 0. Fig.3 shows the performance difference between the two algorithms. While MAC can only find solutions with weights in the 0.455-0.54 range,

[5]To be exact, in all the tests we have used MAC-3+`min-domain`.

[6]Constraint density is defined as the fraction of the possible constraints beyond the minimum n-1, that the problem has. Constraint tightness is defined as the fraction of all possible pairs of values from the domains of two variables that are not allowed by the constraint.

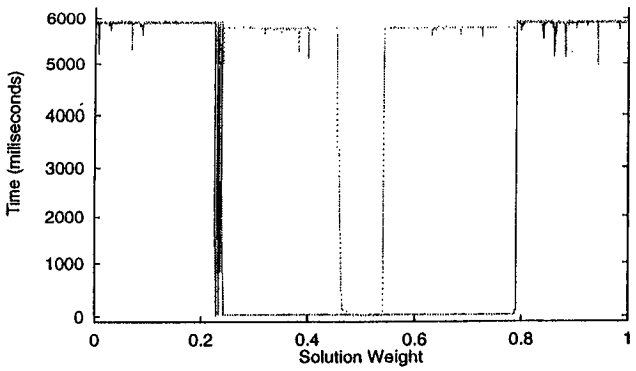

Figure 3: Range=0.05, Density=0, Tightness=0

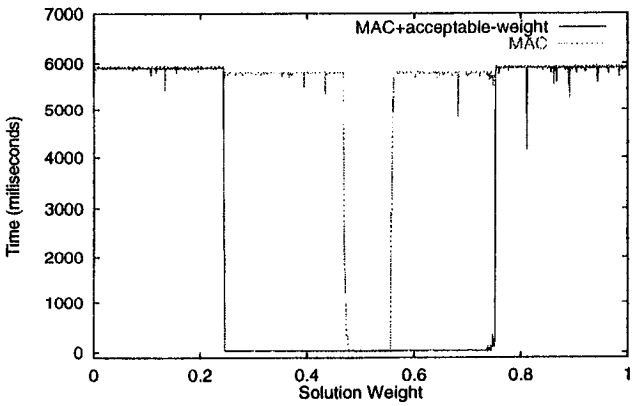

Figure 4: Range=0.05, Density=0, Tightness=0.25

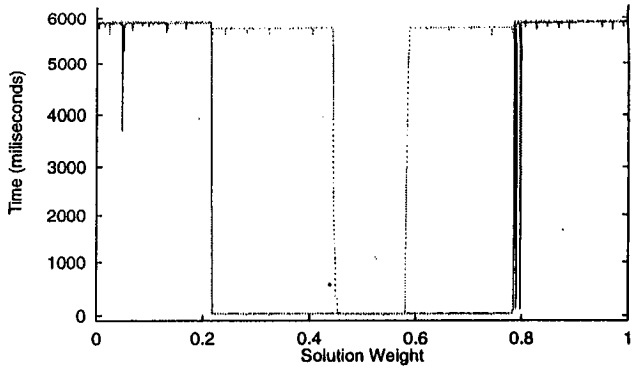

Figure 5: Range=0.1, Density=0, Tightness=0.25

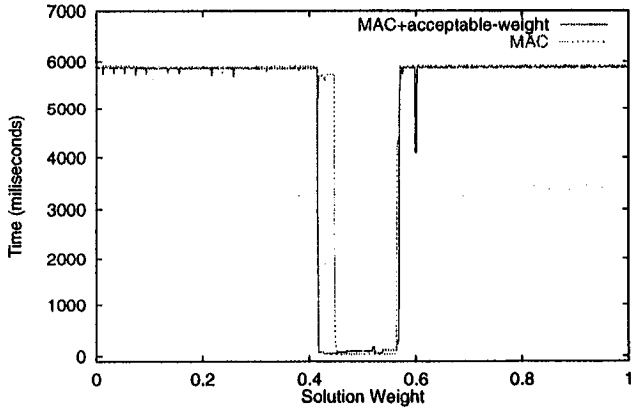

Figure 6: Range=0.1, Density=0.055, Tightness=0.25

MAC+`acceptable-weight` is capable of finding solutions with weights in the 0.226-0.79 range. Moreover, in most of the cases the very first solution that MAC+`acceptable-weight` finds is within the acceptable range.

The interval in which MAC combined with the `acceptable-weight` heuristic quickly finds acceptable solutions narrows as the density and/or tightness increase.

The test in Fig.4 was similar to the one in Fig.3, except that the constraint tightness has been increased to 0.25. As a result, the range in which MAC+`acceptable-weight` can find acceptable solutions decreases to 0.244-0.752. The range in which MAC finds acceptable solutions also changes a little: 0.469-0.560.

In terms of weights, the density of the solutions is given by a normal distribution. The reason for the performance degradation is that as density and/or tightness increase, the number of solutions decreases, and the areas affected the most are those at the sides of the normal distribution curve, which contain a very small number of solutions to begin with. Gradually, the range covered by MAC+`acceptable-weight` will shrink to a range comparable to that covered by MAC alone (Fig.6), because there will be virtually no solution outside a small range around 0.5 and MAC will be able to find those just as fast.

The first two results presented were performed with ranges of size 0.05. That is, any solution with a weight that differs in absolute value by at most 0.025 from the weight on the **X** axis was considered acceptable. As we widen the acceptable range, the interval in which both algorithms quickly find solutions widens. In the test pictured in Fig.5, MAC was capable of finding acceptable solutions with weights in the 0.444-0.584 range. MAC+`acceptable-weight` performed much better, covering the 0.216-0.798 range (also note that this compares better with the 0.244-0.752 range in Fig.4).

Finally, in our fourth test (Fig.6), MAC alone covered the 0.447-0.569 range, while MAC+`acceptable-weight` covered the 0.415-0.568 range (the spike around 0.601 marks a small region where MAC+`acceptable-weight` found acceptable solutions - with a longer time limit it would have covered the 0.568-0.601 range as well).

For the problem tested here (100 variables, domain size 5), the difficulty peak for tightness=0.25 is around density=0.0651 (Fig.7). As the problem gets harder

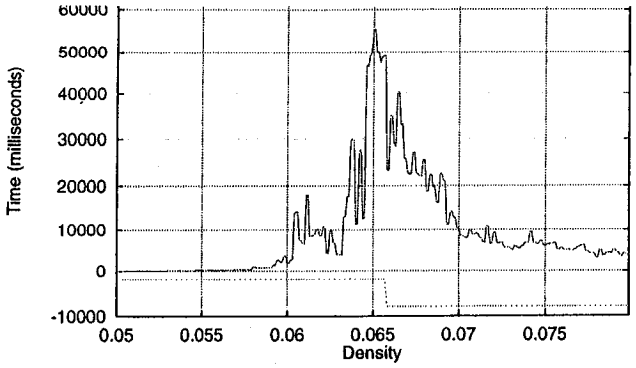

Figure 7: The difficulty peak for tightness=0.25

and harder, the impact of `acceptable-weight` becomes less noticeable. Intuitively, as the problem gets harder, the number of solutions gets smaller, and if an acceptable one exists, chances are that MAC alone will find it pretty quickly by just looking for *any* solution, not necessarily an acceptable one.

It should be noted however that we are not targeting hard problems with this heuristic. Configuration problems for instance are relatively easy - there are lots of solutions, the real challenge is to find one that is acceptable. For problems around the difficulty peak, a standard algorithm would have similar performance characteristics.

Future Work

One way to improve the performance of the heuristic might be by trying to detect situations where the search gets stuck in a given range. There are situations where MAC+`acceptable-weight` obtains a partial solution, but there is no way of extending it to a complete solution with an acceptable weight. It is clear that in those situations the heuristic has made a few mistakes, and it might be interesting to see if the overall performance can be improved by refusing to listen to the heuristic's suggestions until the range of solution weights obtained changes. Somewhat related ideas can be found in (Harvey & Ginsberg 1995).

Another idea would be to avoid searching in subtrees where there is no way of extending the current partial solution to a complete solution with an acceptable weight. Rough estimates of the potential contribution of the unsolved part of the problem can be obtained by computing its minimum and maximum solution weights ($MinSW$ and $MaxSW$).

Finally we plan to look at ways of obtaining better weight evaluations for the small subproblems centered around the current variable. Taking future variables into account is on top of our list of potential improvements.

Applications for this algorithm can range from suggesting initial solution by a matchmaker (Freuder & Wallace 1997) or deep-interview strategy, then the vendor suggests successive upgrades. The upgrades can be computed by looking for solutions with weights within a range that is close to (but not centered in[7]) the weight of the original solution.

Conclusions

Goldilocks learned a very important lesson. There are things in life that are not of the "right size" for everyone. It is often the case that there is a wide range of choices to pick from, and we have to determine which one is "too big", "too small", or "just right". Worst yet, sometimes we have to relax our notion of "just right", to get something that is "good enough".

The `acceptable-weight` heuristic presented here is designed to guide the search towards solutions with acceptable weights, when solutions can be ranked along a given dimension. Although we believe there are avenues for improvement, experiments with random problems showed that this heuristic, combined with MAC, can find acceptable solutions very quickly.

References

Bistarelli, S.; Fargier, H.; Montanari, U.; Rossi, F.; Schiex, T.; and Verfaillie, G. 1996. Semiring-based csps and valued csps: Basic properties and comparison. In Jampel, M.; Freuder, E., and Maher, M., eds., *Over-Constrained Systems (LNCS 1106, Selected Papers from the Workshop on Over-Constrained Systems at CP95.).*

Freuder, E., and Wallace, R. 1997. Suggestion strategies for constraint-based matchmaker agents. *AAAI Workshop on Constraints and Agents.*

Harvey, W., and Ginsberg, M. 1995. Limited discrepany search. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence.*

Sabin, D., and Freuder, E. 1994. Contradicting conventional wisdom in constraint satisfation. In *11th European Conference on Artificial Intelligence.*

von Winterfeldt, D., and Edwards, W. 1986. *Decision Analysis and Behavioral Research.* Cambridge: Cambridge University Press. chapter 8, 259–313.

[7] As pointed out before, `acceptable-weight` attempts to guide the search towards the center of the range.