Constraint Satisfaction as Global Optimization

Pedro Meseguer Javier Larrosa
Universitat Politecnica de Catalunya
Dep. Llenguatges i Sistemes Informatics
Pau Gargallo 5
08028 Barcelona, SPAIN

Abstract

We present a optimization formulation for discrete binary CSP, based on the construction of a continuous function *A(P)* whose *global maximum* represents the *best possible solution* for that problem. By the best possible solution we mean either (i) a solution of the problem, if it is solvable, or (ii) a partial solution violating a minimal number of constraints, if the problem is unsolvable. This approach is based on relaxation labeling techniques used to enforce consistency in image interpretation. We have used a projected gradient ascent algorithm to maximize *A(P)* on the n-queens problem obtaining good results but with a high computational cost. To elude this problem, we have developed a heuristic for variable and value selection inspired in the direction in which *A(P)* is maximized. We have tested this heuristic with forward checking on several classes of CSP.

1 Introduction

The purpose of this paper is to show how discrete constraint satisfaction problems (usually abbreviated as CSP) can be effectively analyzed and solved as a kind of optimization problems. This is not a strictly new approach; other authors have used it to solve specific problems [Sosic and Gu, 1991; Minton *et al.,* 1992; Selman *et a!.,* 1992; Morris, 1993]. The novel aspect we present here consists in the following: we provide a way to construct, for any instance of CSP with binary constraints, a continuous function whose *global maximum* represents the *best possible solution* for that problem. By the best possible solution we mean cither (i) a solution of the problem, if it is solvable, or (ii) a partial solution violating a minimal number of constraints, if the problem is unsolvable. Therefore, a CSP can be solved constructing such a function and using any kind of optimization techniques to compute its global maximum. This function, *A(P),* is the average local consistency function associated with the problem.

The present work is the result of applying relaxation labeling techniques, used to enforce consistency in image analysis, to CSP. Relaxation labeling considers labeling problems (LP), which can be seen as a generalization of CSP. LP and CSP have many points in common: both deal with a finite number of variables $\{X_{ij}\}_y$ which take values on discrete domains $\{D_i\}$ under a set of binary constraints $\{R_{ij}\}$. They differ in two main aspects: (i) in the problem formulation and (ii) in what they consider as a solution. With respect to the problem formulation, in CSP assignments and constraints are purely boolean (the assignment $(X_{it}\ v_j)$ is true or false, the constraint $R_{ij}\{v^*, v/)$ allows variables $X_{i;}$ and X_j to take the values vk and v/ or not), while in LP assignments can be weighted (assign several values with different positive weights to the same variable is a legal assignment, providing the sum of weights is equal to 1), and constraints can express a graded level of consistency (for a given pair of variables different pairs of values can be consistent, but some more consistent —and therefore preferred—than others). With respect to a solution, both approaches look for a *consistent* assignment but they differ in the level of consistency required. In CSP, a solution must be globally consistent, that is, it must satisfy all the constraints (or a maximal number of constraints in the case of maximal constraint satisfaction). In LP different criteria for a consistent solution have been proposed; in this paper we will follow the proposal of [Hummel and Zucker, 1983] which, for a LP with symmetric constraints, identifies consistent solutions with local maxima of *A(P),* the average local consistency function. Local maxima of *A(P)* are not guaranteed either to satisfy every constraint or to restrict the assignment weights to {0,1}, so they are not feasible solutions for CSP. After this description, we can see a CSP as a particular instance of LP, but with more demanding requirements for a solution. We will show that the necessary and sufficient condition for the best solution of a CSP is to be a global maximum of *A(P),* and in this case it is always possible to restrict the assignment weights to {0,1}.

This paper is organized as follows. In section 2, we revise some of the previous work on CSP and relaxation labeling. In section 3, we introduce the concepts needed to analyze a CSP as a LP. In section 4, we provide the results relating solutions with local and global maxima of *A(P).* In section 5, we discuss some approaches to compute a global maximum. In section 6, we discuss a heuristic approach for variable and value selection. Finally, in section 7 we summarize the main contributions of this work.

2 Related Work

A significant amount of work has been made on CSP in the last twenty years (for an wide overview see [Tsang, 1993]). The most common approach to solve CSP has been a systematic search algorithm with backtracking. A consistent partial solution is formed by a subset of variables and it is extended by adding variables one by one until a complete solution is found. When no consistent value exists for the variable being added, backtracking occurs changing the value of a previously assigned variable. This approach is complete, it always finds a solution if it exists, but is has an important drawback: backtracking is extremely inefficient. To prevent this problem, several refinements and additions to this approach has been developed, such as local consistency pre-processing, backtrack-free problems, look-ahead and look-back algorithms, heuristics, and combinations of these strategies.

In the last years, a new iterative approach to solve CSP has been proposed. Starting from an inconsistent global assignment, each iteration modifies this assignment using local information in such a way that the number of violated constraints decreases (or in some cases, remains unchanged), until global consistency is achieved [Sosic and Gu, 1991; Minton *et al.*, 1992; Selman *et al.*, 1992]. This approach can be seen as a hill-climbing procedure, where the number of violated constraints is minimized. Given that it can be stuck in local minima (where some constraints are still violated), a way to escape from them is needed [Morris, 1993]. This approach is not complete. When a given limit of iterations is achieved without reaching a solution, the process is restarted from another initial assignment. In practice and for some kind of problems, a few restarts are enough to reach a solution, with less computational effort than systematic search with backtracking.

On the other hand, image interpretation considers the problem of assigning labels to image parts to produce a global consistent interpretation. Given that the presence of a particular object may impose constraints on other objects in its neighbourhood, a common approach uses local contextual information to obtain the most adequate label for each image part. This process is iterated to allow local information to propagate, until a stable state is reached [Davis and Rosenfeld, 1981]. An early example of this technique is the Waltz's work on interpretation of line segments [Waltz, 1975]. This work allowed unambiguous interpretations only. In general this is too restrictive, so ambiguous interpretations where several labels are assigned to the same image part with different weights are also allowed. In this context a number of techniques, called relaxation labeling, have been developed [Kittler and Illingworth, 1985; Torras, 1989]. A relaxation procedure is an iterative and parallel process which, starting from an initial weighted assignment (weights in [0,1]), performs a synchronous weight updating until it does not cause further changes in the current weighted assignment (it converges to a fixed point). Many updating formulas have been proposed [Rosenfeld *et al.*, 1976], some of which have been proved as approximations of a gradient ascent algorithm and their fixed points are local maxima of a continuous function.

3 CSP as Labeling Problems

In this section we introduce the basic concepts for CSP and LP, and we formulate a generic CSP as a LP. To keep the maximum of clarity in our exposition, we use different notations for CSP and LP. The real differences will become apparent at the end of the section.

A discrete binary CSP is defined by a finite set of variables $\{X_i\}$ taking values on discrete and finite domains $\{D_i\}$ under a set of binary constraints $\{R_{ij}\}$. A constraint R_{ij} is a logical expression which evaluates to true or false on each pair of potential values for X_i and X_j. An assignment of values to variables is *csp-consistent* if it satisfies every constraint. A solution for the CSP is a csp-consistent global assignment. The number of variables is n and, without loss of generality, we will assume a common domain D for all the variables, m being its cardinality.

A general LP is characterized by a finite set of units $\{U_i\}$, a set of labels for each unit $\{\Lambda_i\}$, a neighbour relation over the units, and a constraint relation over tuples of neighbouring units. In this paper we will assume that the sets of labels are discrete and finite, and that all the neighbour relations are binary, which imply binary constraints $\{r_{ij}\}$. The number of units is n and, without loss of generality, we will assume a common set of labels Λ for all the units, m being its cardinality. A constraint is a real-valued function, $r_{ij}: \Lambda \times \Lambda \rightarrow R$; a positive value of $r_{ij}(\lambda_k, \lambda_l)$ indicates that the assignment of λ_k to U_i is consistent with the assignment of λ_l to U_j, while a negative value indicates an inconsistent assignment. The magnitude of $r_{ij}(\lambda_k, \lambda_l)$ represents relative importance of r_{ij} in the set of constraints, as well as the relative preference for the pair of values (λ_k, λ_l) for this constraint. From now on, we will assume that constraints are symmetric, $r_{ij}(\lambda_k, \lambda_l) = r_{ji}(\lambda_l, \lambda_k)$, for $i, j = 1,\ldots,n$, and $l, k = 1,\ldots,m$.

An *unambiguous labeling* is a mapping from the units into their corresponding set of labels, associating each unit with exactly one label. For each unit U_i, this mapping can be represented by a vector of m bits, p_i, defined as follows,

$$p_i[\lambda_k] = \begin{cases} 1, & \text{if unit } U_i \text{ maps to label } \lambda_k \\ 0, & \text{if unit } U_i \text{ does not map to label } \lambda_k \end{cases}$$

Obviously, p_i has 1 in one position only and the remainder positions contain zeroes, so $\sum_{k=1}^{m} p_i[\lambda_k] = 1$. Concatenating the vectors $p_1, p_2, \ldots, p_n$, one can obtain an assignment vector $P \in R^{nm}$. The space of unambiguous labelings K^* is,

$$K^* = \{P \in R^{nm} \mid P = [p_1,\ldots,p_n];\ p_i = [p_i[\lambda_1],\ldots,p_i[\lambda_m]] \in R^m; \\ p_i[\lambda_k] = 0 \text{ or } 1;\ \sum_{k=1}^{m} p_i[\lambda_k] = 1, i=1,\ldots,n\}$$

A *weighted labeling* is obtained by replacing the condition $p_i[\lambda_k] = 0$ or 1 by the condition $0 \le p_i[\lambda_k] \le 1$ for all i, k. The space of weighted labelings K is defined as,

$$K = \{P \in R^{nm} \mid P = [p_1,\ldots,p_n];\ p_i = [p_i[\lambda_1],\ldots,p_i[\lambda_m]] \in R^m; \\ 0 \le p_i[\lambda_k] \le 1;\ \sum_{k=1}^{m} p_i[\lambda_k] = 1, i=1,\ldots,n\}$$

An *ambiguous labeling* is a weighted assignment which is not unambiguous. Given an assignment P, we define the *support* for label λ_k at unit U_i by the assignment P as,

$$s_i(\lambda_k, P) = \sum_{j=1}^{n} \sum_{l=1}^{m} r_{ij}(\lambda_k, \lambda_l)\, p_j[\lambda_l]$$

Given a labeling P, we define the *average local consistency* function as,

$$A(P) = \sum_{i=1}^{n} \sum_{k=1}^{m} s_i(\lambda_k, P)\, p_i[\lambda_k] \qquad (1)$$

If P is an unambiguous labeling which maps each unit U_i to the label λ^i, the expression of $A(P)$ is specialized as follows,

$$A(P) = \sum_{i=1}^{n} \sum_{j=1}^{n} r_{ij}(\lambda^i, \lambda^j) \qquad (2)$$

The unambiguous labeling P is *lp-consistent* provided,

$$s_i(\lambda^i, P) \geq s_i(\lambda_k, P), \qquad \text{for } i=1,\dots,n,\ k=1,\dots,m.$$

In other words, P is lp-consistent if the support that each label obtains from P is higher or equal than the support that any other label can obtain from P. Notice that the above expression involves the maximization of n quantities, the supports of the corresponding label on each unit. This definition can be extended to weighted labelings as follows: let P be a weighted labeling, P is *lp-consistent* provided,

$$\sum_{k=1}^{m} p_i[\lambda_k]\, s_i(\lambda_k, P) \geq \sum_{k=1}^{m} w_i[\lambda_k]\, s_i(\lambda_k, P) \quad i=1,\dots,n, \text{ all } W \in K$$

Given a CSP, we can formulate it as a LP in the following way: each variable X_i corresponds to a unit U_i, and the common domain of values D corresponds to the common set of labels Λ. The set of binary constraints $\{R_{ij}\}$ is transformed into the set of constraints $\{r_{ij}\}$ over binary neighbour relations in the following form:

```
for every pair of variables $(i, j)$, $i,j=1,\dots,n$,
    for every pair of values $(v_k, v_l)$, $k,l=1,\dots,m$
            if $R_{ij}(v_k, v_l)$ exists then
                    if $R_{ij}(v_k, v_l)$ then $r_{ij}(v_k, v_l) = 1$
                    else $r_{ij}(v_k, v_l) = -1$ endif
            else $r_{ij}(v_k, v_l) = 0$ endif
    endfor
endfor
```

A solution for this CSP will be an unambiguous labeling with the condition of being csp-consistent. Now, a natural question arises: *what is the relation between csp-consistency and lp-consistency*? A precise answer will be provided in section 4, but now we can show that they are not equivalent because there are lp-consistent labelings which are not csp-consistent. We can see this in a very simple example using the 3-queens problem, where we have to locate three queens on a 3 x 3 chessboard in such a way that they are not attacking each other. The problem is formulated as follows,

Units: $\{U_1, U_2, U_3\}$. U_i represents the queen at the row i.
Labels: $\Lambda=\{1, 2, 3\}$; label k represents a queen located at column k.
Constraints: $r_{ij}(k, k') = 1$ if queen i at column k does not attack queen j at column k', -1 otherwise.

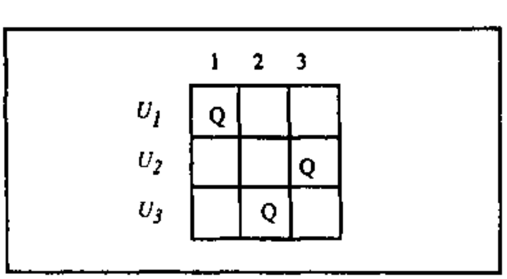

Figure 1. A labeling for the 3-queens problem which is lp-consistent but not csp-consistent.

It is well known that this problem has no solution (the general problem of n-queens has solutions for $n \geq 4$), so no unambiguous labeling can be csp-consistent. However, there are unambiguous labelings which are lp-consistent. For instance, the unambiguous labeling P = [[1,0,0] [0,0,1] [0,1,0]] (see Figure 1) is lp-consistent; computing the supports we have,

$s_1(1, P) = 2$	$s_1(2, P) = -2$	$s_1(3, P) = 0$
$s_2(1, P) = -2$	$s_2(2, P) = -2$	$s_2(3, P) = 0$
$s_3(1, P) = 0$	$s_3(2, P) = 0$	$s_3(3, P) = -2$

From these figures we can see that, $s_1(1, P) \geq s_1(i, P)$, i=2,3; $s_2(3, P) \geq s_2(j, P)$, j=1,2; $s_3(2, P) \geq s_3(k, P)$, k=1,3; so P is lp-consistent although obviously P is not csp-consistent. In consequence, these two notions of consistency are not equivalent. Lp-consistency is just a kind of partial consistency, as we will see in the next section.

4 Consistency as Optimization of $A(P)$

In this section we provide several results relating lp-consistency and csp-consistency with local and global maxima of the $A(P)$ function.

Theorem 1 [Hummel and Zucker, 1983; Banerjee, 1989]: Let us consider a labeling problem with symmetric binary constraints. A labeling P is lp-consistent if and only if it is a local maximum of $A(P)$ over K.

This first result establishes the equivalence between lp-consistent labelings and local maxima of $A(P)$. These labelings can be ambiguous, and therefore unfeasible when considering CSP (remember that only unambiguous labelings are meaningful as potential solutions for CSP). The following theorem relates ambiguous and unambiguous lp-consistent labelings.

Theorem 2 [Sastry and Thathachar, 1994]: Let us consider a labeling problem with symmetric binary constraints. If there exists an ambiguous lp-consistent labeling P_0, there exists an unambiguous lp-consistent labeling P_1 such that $A(P_0) = A(P_1)$. Further, P_0 is a convex combination of some unambiguous lp-consistent labelings, all sharing the value $A(P_0)$ for the average local consistency function.

This theorem guarantees the existence of an unambiguous lp-consistent labeling for each ambiguous lp-consistent labeling with the same value of $A(P)$. Therefore, this theorem allow us to ignore —at least in theory— the existence of ambiguous lp-consistent labelings and to consider only unambiguous ones.

Theorem 3: Let us consider a binary CSP formulated as a labeling problem, and let P_0 be an unambiguous labeling. P_0 violates a minimal number of constraints if and only if P_0 is a global maximum of $A(P)$.

Proof. Let us assume that P_0 violates a minimal number of constraints but it is not a global maximum of $A(P)$. There exist a $P_1 \in K$ such that $A(P_0) < A(P_1)$. By Theorem 2, there exists $P_2 \in K^*$ such that $A(P_1) = A(P_2)$. Using (2) $A(P_0)$ and $A(P_2)$ are,

$$A(P_0) = \sum_{i=1}^{n} \sum_{j=1}^{n} r_{ij}(\lambda_0{}^i, \lambda_0{}^j) \qquad A(P_2) = \sum_{i=1}^{n} \sum_{j=1}^{n} r_{ij}(\lambda_2{}^i, \lambda_2{}^j)$$

where $\lambda_0{}^k$ and $\lambda_2{}^k$ are the labels assigned to unit k by the labelings P_0 and P_2 respectively. The summands $r_{ij}(\lambda, \lambda')$ can be 1, 0, or -1, and $A(P_0)$ and $A(P_2)$ have the same zero summands ($r_{ij}(\lambda, \lambda') = 0$ for every pair (X_i, X_j) of variables which do not constraint each other). Therefore, if $A(P_2) > A(P_0)$ it implies that $A(P_2)$ has more positive summands than $A(P_0)$. This means that P_2 satisfies more constraints than P_0, what contradicts the initial assumption on P_0. So P_0 is a global maximum.

Conversely, if P_0 is a global maximum of $A(P)$, $A(P_0) \geq A(P_1)$, for all $P_1 \in K$. In particular, $A(P_0) \geq A(P_2)$, for all $P_2 \in K^*$. Then, by construction of $A(P)$ it is easy to see that P_0 violates a minimal number of constraints. ♦

This theorem gives a necessary and sufficient condition for the best possible solution of a CSP: it must be a global maximum of $A(P)$. The theorem is restricted to unambiguous labelings, because ambiguous global maxima can exist. Theorem 2 assures the existence of unambiguous global maxima which are the vertices of a convex hull in which the ambiguous global maximum is located. It can be seen that every point in this convex hull has the same value for the function $A(P)$, that is, every point in this set is a global maximum including the edges forming the border. Then, if an ambiguous labeling is reached (a non-vertex point of the convex hull) one can find an unambiguous labeling (a vertex) after some exploration around the reached labeling.

Lemma 1: Let us consider a CSP formulated as a labeling problem. Let A_{max} be twice the number of constraints $\{R_{ij}\}$ with $i<j$. A_{max} is an upper bound of $A(P)$.

Proof. Let P_0 be an unambiguous labeling. Using (2), $A(P_0) = \sum_{i=1}^{n} \sum_{j=1}^{n} r_{ij}(\lambda_0{}^i, \lambda_0{}^j)$, where $\lambda_0{}^k$ is the label assigned to unit k by P_0. The summands $r_{ij}(\lambda_0{}^i, \lambda_0{}^j)$ different from zero are those corresponding to existing constrained pairs of variables. For a given R_{ij} the maximum value of $r_{ij}(\lambda_0{}^i, \lambda_0{}^j)$ is 1, so the maximum value for $A(P_0)$ is twice the number of existing constraints, which is the value of A_{max}. This proofs that A_{max} is an upper bound of $A(P)$ for unambiguous labelings. The extension to ambiguous labelings is straightforward, using Theorem 2. ♦

Corollary 1: Let us consider a binary CSP formulated as a labeling problem. An unambiguous labeling PQ is csp-consistent if and only if $A(Po)= A_{max}$.

Proof. This corollary is a trivial specialization of Theorem 3 using Lemma 1 when the global maximum satisfies all the constraints. ♦

5 Solving CSP by Gradient Ascent

Using the results of section 4, given a CSP we can compute a solution in the following steps: (i) construct the $A(P)$ function, (ii) compute a global maximum P_o, and (iii) if $A(P_0) = A_{max}$ then we can compute a solution from P_0, otherwise no solution exists. Steps (i) and (iii) are trivial but step (ii) is very difficult. Computing a global maximum of a continuous function which in general is not convex is a very difficult task [Horst and Tuy, 1993]. When possible, this issue is solved looking for a local maximum satisfying an additional condition which guarantees that it is a global one. In our case, we know that PQ is a global maximum when $A\{PQ) - A_{max}$, for solvable CSP. To look for a global maximum of $A(P)$ we take a very simple approach: starting from a random point, we look for a maximum using a continuous gradient ascent algorithm. If the value of $A(P)$ on this maximum is A_{max}, the point is a global maximum. Otherwise, we discard this local maximum and restart the process starting from another random point.

The maximization of $A(P)$ is subjected to a set of constraints, given that $0 \leq p_i[\lambda_k] \leq 1$ and $\sum_{k=1}^{m} p_i[\lambda_k] = 1$, for $i=1,\ldots,n$. That is, $A(P)$ should be maximized without leaving the set K. The pure continuous gradient ascent algorithm is not directly applicable, because the gradient may point out of K and in this case, this algorithm will compute a new point out of K. To prevent this, we have used the projected gradient ascent algorithm [Gill *et al.*, 1981], which uses the projection of the gradient on K as the direction of maximum increase of $A(P)$. The gradient of $A(P)$ at the point $P \in K$ is the vector $Q \in R^{nm}$, such that $Q = [q_1,\ldots,q_n]$, $q_i = [q_i[\lambda_1],\ldots,q_i[\lambda_m]]$. Each component $q_i[\lambda_k]$ has the following form,

$$q_i[\lambda_k] = 2\, s_i(\lambda_k, P) \tag{3}$$

The projection of Q on K is obtained using the operator $Proj$, which is built from the constraints defining K [Gill *et al.*, 1981]. The new point P' is computed as,

$$P' = P + \alpha\, Proj(Q) \tag{4}$$

where α is the length of the step taken in the direction $Proj(Q)$. Given that $A(P)$ is a quadratic form, the optimal value of α can be exactly computed.

We have implemented this algorithm to solve the n-queens problem. For $n \leq 50$ the algorithm achieves a maximum in approximately n iterations, and it needs between 2 to 4 restarts to achieve a global maximum (a solution). The number of restarts does not depend on n, the problem dimension. The local maxima reached present a low number of conflicts (1 or 2 pairs of attacking queens). The algorithm starts from a random unambiguous labeling. These results are reasonably good but the algorithm requires significant computational efforts when n increases, due to the dimension of vectors P and Q increases as n^2.

We have also implemented a discrete hill-climbing algorithm, which starts from a random configuration of queens (a queen for row), selects a row at random and performs a change in the position of its queen if the number of conflicts decreases. When the algorithm stops on a local minimum, the process restarts from another random configuration. For n<50, the algorithm reaches a minimum in approximately n iterations, and it needs between 10n to 100n restarts to achieve a solution. The number of conflicts in local minima commonly ranges from 2 to 4. Regarding computational cost, this approach is globally less expensive than the projected gradient algorithm described above. The reason is simple: an iteration of the projected continuous gradient is far more costly than an iteration of hill-climbing, and this cost is not compensated by the lineal number of restarts needed by hill-climbing with respect to the constant number of restarts of the projected gradient.

6 A Heuristic for Variable / Value Selection

While using the projected gradient algorithm to optimize $A(P)$ is not cost-effective with respect to previous approaches, intermediate results of this algorithm are still of interest for CSP solving. This algorithm computes the direction in which $A(P)$ increases more, and updates weights accordingly. At this point, we can use weights as a heuristic source for variable and value selection in a systematic search algorithm. In this way we obtain completeness (by systematic search), taking advantadge of the information coming from the optimization approach. With this idea, we have developed two optimization-inspired heuristics for variable and value selection inside forward checking.

In this new view, we have to relate systematic search with label updating. At a given time, the current state of past and future variables is reflected in a labeling P in the following form. If U_i is a past variable with value λ^j, $p_i[\lambda^j]$ = 1 and $p_i[\lambda] = 0$, $\lambda \in \Lambda$, $\lambda \neq \lambda^j$. If U_i is a future variable, $p_i[\lambda] = 1/m_i$, $\lambda \in feasible(\Lambda, U_i)$ and $p_i[\lambda] = 0$, $\lambda \in \Lambda - feasible(\Lambda, U_i)$, where the set $feasible(\Lambda, U_i)$ is the set of remaining values for variable U_i at this state of the search, and $m_i = card\ (feasible(\Lambda, U_i))$. In other words, the labeling of past variables is unambiguous and the labeling of future variables is ambiguous with an even distribution of weights among remaining values. To update weights, we substitute (4) by the following updating formula for relaxation labeling [Rosenfeld *et al.*, 1976],

$$p_i'[\lambda_k] = p_i[\lambda_k](2n + q_i[\lambda_k]) \Big/ \sum_{k=1}^{m} p_i[\lambda_k](2n + q_i[\lambda_k]) \qquad (5)$$

which is an approximation of the projected gradient [Hummel and Zucker, 1983], but less expensive to compute. Formula (5) does not cause any change for values with initial weights either 1 or 0. Therefore, (5) is only meaningful for remaining values of future variables, and in this case $p_i'[\lambda]$ can be simplified to,

$$p_i'[\lambda] = 2n + q_i[\lambda] \Big/ \sum_{\lambda \in feasible(\Lambda, U_i)} 2n + q_i[\lambda] \qquad (6)$$

Initially, we used updated weights $p_i'[\lambda]$ as heuristic source, selecting as the next variable that with the highest weight corresponding with a value of its domain. However, (6) is a ratio between supports (from (3)), where 2n is added to assure a positive fraction. This ratio is sensitive to small variations of support and it exhibits some unstable behaviour. Looking for robustness in variable selection, we moved to another criterion: select the variable with the lowest sum of supports for its remaining values, and order its values by decreasing support. We will refer to this heuristic as the *lowest-support* heuristic. It is obviously related to the optimization approach: it means to select a variable with a low denominator in (6) which implies that this variable will have high values of Pi[A] although not necessarily the highest.

The lowest support heuristic is computed each time a new variable has to be selected. Assignment is performed as in standard forward checking. When a new variable is assigned, future domains are filtered, a new labeling is constructed and the heuristic is computed again to select the next variable. When backtracking occurs (a future domain becomes empty), the last assigned variable is reassigned using the value ordering set when it was selected.

This heuristic is expensive to compute. To simplify computation, we have considered two approximations. First, restrict heuristic computation to the set of variables with minimal domains, because there is a high chance that the selected variable will belong to this set. Second, after assigning a variable we still use the computed supports for further assignments if the next lowest sum of supports is close enough to the initial one. We have accepted sum of old supports when they have become smaller (due to domain pruning) than the initial one.

We will refer to forward checking with lowest-support without approximations as fc-ls, and with approximations as fc-ls-app. We will refer to the standard forward checking with first-fail (selecting variable with the smallest domain, breaking ties randomly and selecting values randomly) as fc-ff. We have tested these algorithms on two problems: random solvable graph colouring, and random problems. Empirical results are given in the following.

Graph colouring. Following [Minton et al. 92], we tested our heuristic on graph 3-colourability problems. Specifically, we have considered solvable random graphs with n nodes and m arcs, forming two classes: sparsely-connected graphs *(m = 2n)* and densely-connected graphs *(m = n (n - 1) / 4)*. We generated graphs on the range from n = 10 to n = 180, incrementing n by 1. For densely-connected graphs, fc-ls does not bring any real improvement to fc-ff. Almost all problem instances were solved without backtracking for both algorithms, and regarding CPU time, fc-ls took a bit longer than fc-ff, because the time required by support computing. For sparsely-connected graphs, both fc-ls and fc-ls-app outperformed clearly to fc-ff in backtrackings and CPU time. Results are given in Table 1.

#inst.	algorithm	#backs	CPU time
170	fc-ff	18,719,163	12,693
	fc-ls	4,503	564
	fc-ls-app	13,988	64

Table 1. Sparsely-connected graph 3-colourability problems.

n m	#inst.	algorithm	#backs	CPU time
10, 10	300	fc-ff	9,466	7.0
		fc-ls	4,446	19.3
		fc-ls-app	5,124	13.0
20, 10	200	fc-ff	156,540	127.7
		fc-ls	53,258	435.3
		fc-ls-app	64,000	91.3
30, 10	50	fc-ff	686,247	752.3
		fc-ls	285,965	3,204.5
		fc-ls-app	320,752	497.9

Table 2. Sparsely-connected random problems.

Random problems. We tested our heuristic on random problems following [Prosser, 1994]. We worked on problem instances on the peak by choosing the appropriate tightness (P2) for a given connectivity (p1) and problem size (n, m). We considered two types: sparsely-connected instances, (with p1; = 0.4) and densely-connected instances (with p1 = 1). To select problems on the peak, we used the Prosser's formula to compute the appropriate tightness. Around this value, we varied p2 by steps of 0.001, in the interval where a fifty percent of problems happened to be solvable. Ten instances of each p2 setting were used for testing. The results are given in Tables 2 and 3. For sparsely-connected problems, fc-ls saves around half number of backtrackings required by fc-ff, although it needs more CPU time. Including approximations, fc-ls-app is slightly worse than fc-ls in backtrackings, but it requires less time than fc-ff (except in the 10, 10 case). For densely-connected problems, although the heuristic decreases the number of backtrackings performed by fc-ff, it is not cost-effective regarding time.

Our results on graph colouring and random problems show that the proposed heuristic provides good advice (fc-ls always performs less backtrackings than fc-ff in every problem type), although it may not be cost-effective for some problem classes. It seems to be appropriate for sparsely-connected problems with a high number of variables relative to the number of values. In these problems, first fail variable selection heuristic is not enough to reduce drastically the number of backtrackings. Relation between the usefulness of the heuristic and the shape of A(P) requires further investigatioa

7 Conclusions

This paper offers two main contributions. On the theoretical side we have shown that any binary discrete CSP can be formulated as an optimization problem of a continuous function A(P), which is constructed from the set of initial constraints. A global maximum of A(P) corresponds to the best possible solution of the CSP, that is, an assignment violating a minimal number of constraints. On the practical side we have applied this result to solve some CSP problems. Computing a global maximum of A(P) is quite costly, so we moved to an heuristic approach: use the direction of change that increases A(P) to generate a heuristic for variable and value selection. The results on two kind of CSP show that it causes a low number of backtrackings, being cost-effective for sparsely-connected problems.

n m	#inst.	algorithm	#backs	CPU time
10, 10	170	fc-ff	19,500	11.8
		fc-ls	13,926	30.5
		fc-ls-app	14,573	17.4
20, 10	100	fc-ff	377,867	280.6
		fc-ls	265,393	1,487.4
		fc-ls-app	289,905	327.2

Table 3. Densely-connected random problems.

Acknowledgements

First author has been supported by the Spanish CICYT under the project #TAP93-0451. We thank Carme Torras and David Cohen for their useful comments.

References

[Banerjee, 1989] Banerjee S. Stochastic relaxation paradigms for low level vision. PhD thesis, Dept. of Elect. Eng., Indian Inst., of Science, India, 1989.

[Davis and Rosenfeld, 1981] Davis L. A. and Rosenfeld A. Cooperating Processes for Low-level Vision: A Survey, Artificial Intelligence, vol. 17, 245-263, 1981.

[Gill et al., 1981] Gill P., Murray W. and Wright M. Practical Optimization, Academic Press, 1981.

[Horst and Tuy, 1993] Horst R. and Tuy H. Global Optimization, 2nd edition, Springer-Verlag, 1993.

[Hummel and Zucker, 1983] Hummel R. A. and Zucker S. W. On the Foundations of Relaxation Labeling Processes, IEEE Trans. Pattern Analysis Machine Intelligence, vol. 5, no. 3, 267-287, 1983.

[Kittler and Illingworth, 1985] Kittler J. and Iliingworth J. Relaxation labelling algorithms - a review, Image and Vision Computing, vol. 3, no. 4, 206-216, 1985.

[Minton et al, 1992] Minton S., Johnston M. D., Philips A. b. and Laird P. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, Artificial Intelligence, 58, 161-205, 1992.

[Morris 1993] Morris P. The breakout method to escape from local minima, Proceedings ofAAAI-93, 40-45, 1993.

[Prosser, 1994] Prosser P. Binary constraint satisfaction problems: Some are harder than others, Proceedings of ECAI-94, 95-99, 1994.

[Rosenfeld et al., 1976] Rosenfeld A, Hummel R. and Zucker S. Scene Labeling by Relaxation Operators, IEEE Trans. Systems, Man, Cybernetics, vol. 6, no.6,420-433, 1976.

[Sastry and Thathachar, 1994] Sastry P. S. and Thathachar M. A. L. Analysis of Stochastic Automata Algorithm for Relaxation Labeling, IEEE Trans. Pattern Analysis Machine Intelligence, vol. 16, no. 5,538-543, 1994.

[Selman et al., 1992] Selman B., Levesque H. and Mitchell D. A new method for solving hard satisfiability problems, Proceedings ofAAAI-92, 440-446, 1992.

[Sosic and Gu, 1991] Sosic R. and Gu J. Fast Search Algorithms for the N-Queens Problem, IEEE Trans. Systems, Man, Cybernetics, vol. 21, no.6, 1572-1576, 1991.

[Torras, 1989] Torras C. Relaxation and neural learning: points of convergence and divergence, Journal of Parallel and Distributed Computing, vol. 6, 217-244, 1989.

[Tsang, 1993] Tsang E. Foundations of Constraint Satisfaction, Academic Press, 1993.

[Waltz, 1975] Waltz L. D. Understanding line drawings of scenes with shadows, in The psychology of computer vision, Winston P. H. editor, McGraw-Hill, 1975.