Heuristic Techniques for Variable and Value Ordering in CSPs

Bahareh Jafari
jafarijb@uregina.ca

Malek Mouhoub
mouhoubm@uregina.ca

Department of Computer Science
University of Regina
Regina, Saskatchewan, Canada

ABSTRACT

A Constraint Satisfaction Problem (CSP) is a powerful framework for representing and solving constraint problems. When solving a CSP using a backtrack search method, one important factor that reduces the size of the search space drastically is the order in which variables and values are examined. Many heuristics for static and dynamic variable ordering have been proposed and the most popular and powerful are those that gather information about the failures during the constraint propagation phase, in the form of constraint weights. These later heuristics are called conflict driven heuristics. In this paper, we propose two of these heuristics respectively based on Hill Climbing (HC) and Ant Colony Optimization (ACO) for weighing constraints. In addition, we propose two new value ordering techniques, respectively based on HC and ACO, that rank the values based on their ability to satisfy the constraints attached to their corresponding variables. Several experiments were conducted on various types of problems including random, quasi random and patterned problems. The results show that the proposed variable ordering heuristics, are successful especially in the case of hard random problems. Also, when using the proposed value and variable ordering together, we can improve the performance particularly in the case of random problems.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms

Keywords

Constraint Satisfaction Problems (CSPs), Ant Colony Optimization (ACO), Hill Climbing (HC), Variable Ordering.

GECCO'11, July 12–16, 2011, Dublin, Ireland.

1. INTRODUCTION

Constraint Satisfaction Problems (CSPs) [5] constitute a large number of combinatorial problems with many important "real-word" applications including scheduling, planning and timetabling. A CSP is essentially about selecting values for a set of variables so that all the relations between variables (constraints) hold. Consistent CSPs have one or more solutions that satisfy all the relations while inconsistent ones do not have any feasible solution. In the latter case, the solver should be able to prove the inconsistency of the problem.

There has been a lot of research on developing general CSP solvers and many solvers have been designed for specific problems. Generally, CSPs are either solved in a systematic way as in a backtracking-based method with constraint propagation or with non-systematic methods such as local search. Systematic methods are complete which means they are able to solve the consistent problems and prove the inconsistency in case the problem does not have any solution. Non-systematic methods on the other hand are incomplete and are not able to decide if the problem is consistent or not.

Neither of the complete or incomplete methods are sufficient for all kinds of problems. Backtracking can be inefficient in large size problems whereas local search is more scalable but does not guarantee finding a consistent solution. Many real world problems are structured. Backtracking enhanced with constraint propagation seems to be the better choice for these kinds of problems as local search usually cannot exploit these structures. Many attempts have been done to combine the advantages of these two methods for achieving a faster and more proficient method. Some methods generate partial solutions using local search then use backtracking to extend it to a complete one [23]. Another type of hybrid method uses constraint propagation within local search schema [11, 18]. Some approaches have used local search as a mean to guide the backtrack algorithm in terms of variable and value ordering. The following describes two of these approaches. The first approach uses a local search for finding the best branching variable for a complete backtracking algorithm [4]. This proposed method uses local search algorithm to derive weights for clauses in Boolean satisfiability problem (SAT), giving higher weights to the clauses that are harder to satisfy. The clause weights are then used to guide the variable ordering heuristic search for the backtracking algorithm. Each variable is given a weight which is the sum of the weights of the clauses it occurs in.

The backtracking algorithm is then guided by the shortest clause heuristic augmented by a "tie breaker" based on the weights of variables. The test results show an improvement in run time and the size of the tree search. The second technique that uses local search for deciding variable ordering is introduced in [17]. This approach uses the information gathered during local search probes to guide a refinement search. In this method there are two solvers interacting with each other, one is a refinement solver and the other is a local search solver.

Refinement search is the master process and it uses the local-search solver for heuristic guidance. Solvers have different internal representations of the job-shop scheduling problem and interact with each other regarding decision variables, which in this case are tasks starting times. Every time that the refinement solver makes a refinement decision, it informs the local search solver to add comparable constraints which allows both solvers to keep working on the same problem. Local search provides recommendations to the refinement search about the variables to choose for refinement. These variables will basically be those related to highest amounts of inconsistency. The results show that the use of domain-specific knowledge proves to be a much more effective basis for search control than information about constraint interactions that is gained by local search probes.

In this paper we introduce two hybrid methods that combine systematic and heuristic techniques. The first heuristic is an iterative algorithm based on Hill Climbing local search (HC). The second one is a constructive approach based on Ant Colony Optimization (ACO) [7]. The two methods that we propose tackle the problem first using HC or ACO to gather information about the search space during the search. Then, the information gained through the search is used to give advices to the systematic search. This information is passed as variable and value ordering. Variable ordering specifically has a tremendous effect on the size of a search tree. Therefore, it is very important for a backtrack based search. The idea is that a heuristic algorithm might be able to gather better information from larger parts of search space, which would enable it to identify harder constraints. In order to evaluate the performance of our methods, we conducted a comparative experimental study with the most powerfull techniques for variable ordering. The results of the experiments demonstrate that our proposed algorithms using HC and ACO for variable ordering are successful in the case of hard random problems. However, for problems with more structure such as graph coloring or quasi random problems, they are not comparable to the other conflict driven heuristics. Also, the use of the new proposed value orderings can boost the performance for some random and structured problems.

The rest of the paper is organized as following: In section 2, we give an overview on basic concepts of CSPs and the related existing solving algorithms. In section 3, we review the existing variable ordering heuristics and propose two new variable ordering heuristics. Section 4 is dedicated to the value ordering heuristics that we have developed. In section 5, we report the experimental study we conducted to evaluate the proposed algorithms. Finally, section 6 lists concluding remarks and future research.

2. BACKGROUND

2.1 Constraint Satisfaction Problems

A constraint Satisfaction Problem consists of a set of variables of non-empty finite domains and a set of constraints or limitations [5]. Each constraint is valid for a subset of variables and specifies what combinations of values are allowed for that subset. Constraints can be defined by enumerating the set of allowed tuples or by using an algebraic or symbolic expression. An assignment of values to the variables involved in a CSP from their domains is a consistent assignment if they do not violate constraints. A complete assignment is obtained by assigning values to all the variables and it is a solution if it satisfies all the constraints of the problem. CSPs can be characterized by the following parameters. The arity of a constraint which is the number of variables in the scope of the constraint, the degree of a variable representing the number of constraints in which the variable participates and the domain size of a variable corresponding to the number of values in its domain. A constraint is called binary when there are just two variables involved in it and is unary if it puts limitation on just one variable. A CSP is called a binary CSP if all constraints in it are unary or binary. All CSPs can be transformed into binary CSPs. Therefore binary CSPs are of particular importance in constraint satisfaction studies.

2.2 Systematic Methods for Solving CSPs

In systematic search methods, the search starts with the set of all variables unassigned. At each step, one variable will be assigned a value from its domain and the partial solution gets checked for consistency. There is also a goal test to see if the assignment is a complete one (which, in this case, is a solution) or not. Backtracking is essentially a depth first search that assigns a value to one variable at a time and backtracks when there is no legal value left in the domain of the variable. Standard Backtracking is not efficient due to thrashing; that is the search repeatedly failing due to the same reason which could be identified earlier in the search. In order to overcome this difficulty in practice, local consistency techniques have been proposed [5]. More precisely, these techniques enforce the consistency on a subset of CSP variables before and during the backtrack search. One of the most known forms of local consistency is called Arc Consistency [15]. Arc Consistency (AC) applies the consistency on subsets of two variables. More formally, for each pair of variables (x_1, x_2) sharing a constraint, AC removes from the domain of x_1 any value that is inconsistent with all the values of x_2 domain. When used before the search, the goal of AC is to reduce the size of the search space before Backtracking takes place. When used during the search, AC helps to detect later failure earlier following a lookahead technique such as Maintaining Arc Consistency (MAC). Each time we assign a value to a variable, MAC enforces AC on this latter variable and all future active variables (variables not assigned yet). We use this method as our basic solving method in this paper.

2.3 Heuristic Methods for Solving CSPs

In most heuristic methods such as local search and evolutionary algortihms, the search starts with one or more complete assignments and then changes are made to make the assignments satisfy more and more constraints until reach-

ing the solution. The problem with these algorithms is that because of the randomness they use, they cannot be used to prove the inconsistency of a CSP since they do not enumerate all the solutions and cannot be traced back to previous states. In cases where the problem does not have a solution, these algorithms would run forever without success. In practice, a time limit should be set after which the algorithm returns no solution. To use these algorithms, we need to define a representation of the potential solution and a fitness function to measure the quality of a solution. In CSPs, the representation is an array of variables and the value in each index of the array represents the value of the corresponding variable. The fitness function measures the number of unsatisfied constraints or if constraints have weights, the sum of the weights of the unsatisfied constraints.

2.4 Variable and Value Ordering Techniques

The ordering of the variables in a backtrack search has a tremendous effect on the size of the search space. Typically, a heuristic is used to choose what variable to assign next at each step of the search. These heuristics should be designed based on some criterion. The well-known first-fail principle is a criterion that has been suggested for evaluating different heuristics. This principle supposes that the best search order is the one that minimizes the length or the depth of each branch. Variable ordering heuristics can be refined into two categories: Static Variable Ordering (SVO) and Dynamic Variable Ordering (DVO). SVO heuristics use the initial structure of the constraint network and maintain the same variable ordering during the search to decide the next variable to assign a value to. Smallest Domain First (SDF), is a SVO in which variables are sorted based on their domain size so that variables with smaller domain are checked first. The reasoning behind this is that given all the other factors are equal, the variable with the least number of values would have less sub trees rooted at those values. Another method for SVO is maximum degree(deg) [6] which chooses the variable that has the maximum degree in the constraint graph. DVO use the information about current state of the search at each point of the search to decide the next variable to assign value to. A very well known heuristic of this type, known as dom [10], selects the next variable that has the least remaining values in its domain and thus constrains the remainder of the search space the most. The dynamic version of deg, called ddeg, chooses the variables that are involved in the least amount of constraints with unassigned variables. dom/ddeg [20] is derived from combining these two heuristics and it selects a variable that has the minimum ratio of the current domain to the current dynamic degree. Impact based heuristics [19] focus on the search tree size as a criterion for variable ordering. They measure the importance of each variable in reducing the size of the search space by considering the changes its value assignments can make to the size of the search tree. Conflict driven variable ordering is a technique proposed by Boussemart et al. in [3]. This technique uses MAC as the basic solving method, which maintains the complete arc consistency throughout the search. In this technique, each constraint has a weight value that is incremented every time the constraint causes a domain wipe out during the constraint propagation phases. Each variable has a weighted degree that is the sum of the weights of the constraints that the variable is involved in. At each variable selection point, the variable that has the largest weighted degree is chosen. This technique is called wdeg. The combination of wdeg and dom is dom/wdeg, which prefers a variable that has the least ratio of current domain size to the current weighted degree. The obvious drawback to the wdeg technique is that for the first few choices which are also the most important ones, the search does not have enough information to choose the best variables, and that can tremendously affect the size of the search space. For solving this problem, Grimes et al proposed Weighted Information gathering (WNDI) and RANDom Information gathering (RNDI) in [9]. These two techniques use the dom/wdeg heuristic. However, they do a number of search restarts to gather information from different parts of the search space before starting the main search process. Having this information makes it possible to make better choices for first variables. In RNDI, a variable is selected randomly at each variable selection point during the search for the first R - 1 runs. Weights are incremented in the usual way (i.e. when a constraint causes a domain wipeout). On the final restart dom/wdeg is used in the normal way. However, the weights are not initialized to zero but they are set to the weights that the search learnt from previous random probes. This would enable the search to make better early decisions. In WNDI the search updates the weights consequently in all runs and uses the information gathered from each run at the start of the next run. Experiments in [2] show that RNDI can perform better in many test cases because it learns from more diverse parts of the search space and therefore can give a better approximation of the areas of global contention.

3. PROPOSED VARIABLE ORDERING METHODS

In this section, we propose two heuristics algorithms respectively based on HC and ACO that provide recommendations for backtracking based algorithms. All of these recommendations are based on assigning weights to constraints or domain values of variables; higher weights suggest more significance for corresponding variables or values in the variable or value selection process. In this manner, these methods are similar to the RNDI heuristic for variable ordering; the only difference is that RNDI gathers this information in a systematic way during the constraint propagation phase, whereas the methods we propose use non-systematic techniques for that purpose. RNDI also continues to weigh constraints during the last restart. Hence, it is considered a DVO method. However, in the case of the hybrid methods that we propose, the learning approach of the non-systematic methods cannot be continued during a systematic method. The non-systematic method is run for a specific amount of time or cycles, during which, the constraints gain weight. After the non-systematic phase, similar to the wdeg technique, each variable gets a weighted degree, which is the sum of the weights of the constraints that the variable is involved in. Variables are then sorted based on their weights and those with larger weight get more priority in the ordering. Since the ordering is decided before the start of the search, this approach is considered a SVO method.

3.1 Local Search for Weighing Constraints

Local search improves the quality of solution iteratively by changing one or more variable values at each step. Vari-

ables and values are either chosen randomly or based on some heuristic in a way that they increase the number of satisfied constraints. The problem with local search is that it might end up in a local minimum which means there would exist no move that can generate a better solution. Constraint weighting algorithms have been developed to help local search come out of local minima [16, 8]. One question that would be raised here is when and what to weigh. In the original work on constraint weighing in local search for escaping local minima by Morris in [16], weights are initialized to 1 and at each local minimum, the weights of the violated constraints are incremented by one. In the context of using constraint weights for variable ordering, weighing violated constraints just in local minima is not efficient for various reasons. For example, it might take a long time for the search to enter a local minimum, or the search might not detect sufficient local minima. Therefore, search would not be able to collect enough data about the search space to be able to give useful advice regarding variable ordering. For this reason, we used a cut off parameter that specifies the maximum number of iterations the local search should run for before it restarts again. Every time the search reaches the cut off point or enters a local minimum, it terminates. The weights of violated constraints are then incremented, and another search starts. This continues until a maximum number of cycles is reached. The reason for restarting the search is to be able to visit several parts of the search space and hence gaining a better understanding of the hot spots of the search spaces. For the experiments, we used a HC local search approach and we added the cut off parameter to it. Figure 1 shows the pseudo code of the modified version of the HC algorithm that we used.

3.2 ACO for Weighing Constraints

The second method uses ACO [21, 7] which is a stochastic probabilistic approach for solving computational problems that can be reduced to searching for a minimum cost path in a graph. Artificial ants walk on the edges of the graph looking for good paths. In real world, ants initially wander around randomly and while going back to their colony, they lay pheromone trails down on their path. If other ants find this trail, they may choose not to travel at random but to follow the trail and strengthen it on their way back if they eventually find food. The behavior of artificial ants is inspired from real ants. Artificial ants however, live in a discrete world, which is a graph and their movements should be on the edges of this graph. For CSP problems, this graph is called the construction graph. For CSPs, the construction graph associates a vertex to each variable-value pair $< V_i, d >$ such that $V_i \in V$ and $d \in D_i$ where V and D_i are respectively the set of variables and the domain of the variable V_i. There is a non oriented edge between any pair of vertices corresponding to two different variables. A completely constructed ant would be a potential solution for the problem. The algorithm that was proposed for solving CSPs with ACO is called ant solver. The details of the ant solver we used can be found in [21]. For being able to have a better exploration of the search space, we restart the ant solver a few times in every run. The number of restarts is a parameter for this algorithm. Figure 2 shows the pseudo code of the ACO algorithm that we propose.

The algorithm starts with initializing the pheromone trail amounts and setting the parameters. Then, at each itera-

Procedure HC
Input: A constraint network $R = (V, D, C)$
where $V = \{v_1, \ldots, v_n\}$ (set of variables)
$D = \{D_1, \ldots, D_n\}$ (variables domains)
$C = \{c_1, \ldots, c_m\}$ (set of constraints)
MAX_TRIES: maximum number of tries
CUT_OFF: cut off number
Output: Weights $(w_1, \ldots, w_m)$ for the constraints in C
Counter $\leftarrow 0$
LocalMinimum $\leftarrow$ false
solutionFound $\leftarrow$ false

While Counter < MAX_TRIES **and** solutionFound = false
Initialization: let $\bar{a} = (a_1, ., a_n)$ be a random initial assignment to all variables
Repeat
Let $Y = \{< v_i, a'_i >\}$ be the set of variable-value pairs where v_i is a variable in conflict and when v_i is assigned a'_i , it gives a maximum improvement to the cost of the assignment
If $Y \neq \emptyset$
Pick a pair $< v_i, a'_i > \in Y$
$\bar{a} \leftarrow (a_1, \ldots, a_{i-1}, a'_i, a_{i+1}, \ldots, a_n)$ (flip a_i to a'_i)
Else
LocalMinimum $\leftarrow$ true
If ($\bar{a}$ is consistent)
solutionFound $\leftarrow$ true
Counter $\leftarrow$ Counter + 1
Until Counter = MAX_TRIES **or** LocalMinimum=true
or Counter MOD CUT_OFF = 0
or solutionFound = true
If solutionFound = false
For each constraint $c_i \in C$
If c_i is violated in $\bar{a}$
$w_i \leftarrow w_i + 1$
End While
Return solutionFound

Figure 1: Pseudo code of the modified version of HC.

Procedure ACO
Input Restarts: the number of restarts for ACO
Max_Iters: the maximum iterations for the MC algorithm
Max_Cycles: the maximum number of cycles the ant solver can be run
N: the number of ants
Output The solution to the problem
or return "problem not consistent"
For R $\leftarrow$ 1 to Restarts **Do**
Initialize the pheromone trails and parameters
For C $\leftarrow$ 1 to Max_Cycles **Do**
For k $\leftarrow$ 1 to N **Do**
Construct a complete assignment A_k
using the pheromone trail
Apply HC to assignment A_k until Max_Iters is reached or search enters a local minimum
Update the weights of constraints with the improved version of A_k
End for
Update pheromone trails using the best ants in the cycle
End for
End for

Figure 2: Pseudo code of the used variant of ACO.

tion, a number of completely assigned ants are constructed and the pheromones are updated based on the cost values of the best ants in that iteration. Finally, ants are improved using the HC algorithm. A specific number of generations is considered as cut off. However, if the assignment enters a local minimum before the cutoff point, the local search will terminate. After applying HC, weights of the constraints that the ant was not able to solve get incremented by one. Ant algorithms have a lot of parameters.

Systematically optimizing these parameters before using them in algorithms is always preferable. However, it can take a very long time. For this reason, we used a set of parameters based on those that are found to be best in [21]. So parameters are set as following: $\rho = 0.01, \alpha = 2, \beta = 10, \tau_{min} = 0.01 and \tau_{max} = 4$. Number of ants was set to 8 in all of the problems. Other parameters for ACO are:

- Number of restarts: For being able to have a better exploration of the search space, I restart the ant solver a few times in every run. The number of restarts is a parameter for this algorithm.
- Number of ant solver cycles in each restart.
- Number of iterations for local search: as mentioned before, local search is applied to ants after they are constructed. Therefore, the number of iterations for local search is another parameter that should be set.

4. PROPOSED VALUE ORDERING METHODS

In search for a solution to a CSP problem, value ordering heuristics provide an approximation of what values are more likely to be part of a solution. By assigning those values to the corresponding variables first, the search can be guided on a path that ends in a solution and decreases the whole computational cost. Good value ordering heuristics are generally highly problem specific and general heuristics can be costly. In this paper, we tried random value ordering for RNDI and dom/wdeg. We also propose two new value ordering heuristics that can be used with the proposed variable ordering heuristics. The first one uses HC/MAC and the second one uses ACO/MAC. For HC/MAC algorithm, we assume that the values that repeatedly participate in violated constraints are less likely to be part of a solution and values that satisfy their corresponding constraints frequently are more likely to be part of a solution. Thus, we assign weights to values, which are updated every time weights of constraints are updated. If a value is able to satisfy all the constraints attached to its corresponding variable, its weight is increased by one and if it causes conflicts, its weight is decreased by one. At the end it gives us a static value ordering to use in the backtrack-based search. With ACO/MAC algorithm, the values of a variable are ordered based on the sum of the pheromones that are laid on the edges between the corresponding variable-value node and other variable-value nodes in the ant colony graph.

Example

Figure 3 shows a graph coloring problem and its construction graph created by ACO. As it can be seen, each arc in the construction graph has a weight corresponding to the amount of pheromones laid on that arc at the end of the running of the ACO algorithm. Each node in the construction graph will have a weight that is equal to the sum of the weights of the arcs attached to it. The computation of weights for all the nodes is described below.

$W < V_1, r >= W(< V_1, r >, < V_2, r >) + W(< V_1, r >, < V_2, g >) + W(< V_1, r >, < V_3, r >) + W(< V_1, r >, < V_3, g >) = 0.5 + 0.5 + 0.5 + 0.5 = 2$
$W < V_1, g >= W(< V_1, g >, < V_2, r >) + W(< V_1, g >, < V_2, g >) + W(< V_1, g >, < V_3, r >) + W(< V_1, g >, < V_3, g >) = 4 + 0.5 + 0.5 + 4 = 9$
$W < V_2, r >= W(< V_2, r >, < V_1, r >) + W(< V_2, r >, < V_1, g >) + W(< V_2, r >, < V_3, r >) + W(< V_2, r >, < V_3, g >) = 0.5 + 4 + 4 + 0.5 = 9$
$W < V_2, g >= W(< V_2, g >, < V_1, r >) + W(< V_2, g >, < V_1, g >) + W(< V_2, g >, < V_3, r >) + W(< V_2, g >, < V_3, g >) = 0.5 + 4 + 4 + 0.5 = 4$
$W < V_3, r >= W(< V_3, r >, < V_2, r >) + W(< V_3, r >, < V_2, g >) + W(< V_3, r >, < V_1, r >) + W(< V_3, r >, < V_1, g >) = 0.5 + 0.5 + 0.5 + 0.5 = 5.5$
$W < V_3, g >= W(< V_3, g >, < V_1, r >) + W(< V_3, g >, < V_1, g >) + W(< V_3, g >, < V_2, r >) + W(< V_3, g >, < V_2, g >) = 0.5 + 4 + 4 + 0.5 = 9$

The values of each variable are then sorted, in decreasing order, based on the weight of their corresponding <variable, value> nodes: $D_{V_3} = \{g, r\}, D_{V_2} = \{r, g\}, D_{V_1} = \{g, r\}$. If the values of the variables are chosen based on these orders at each value selection point, no backtracking will be needed for this problem.

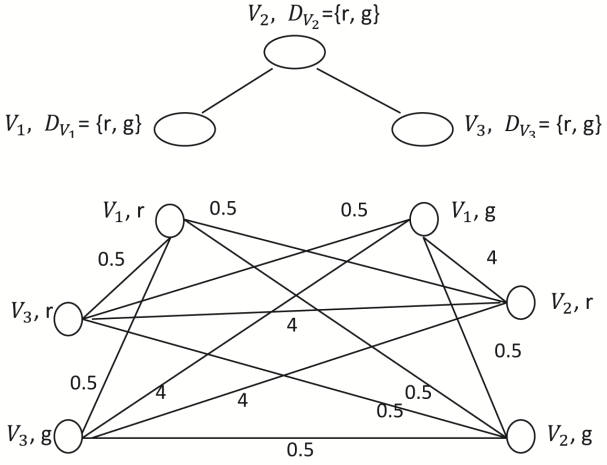

Figure 3: A graph coloring instance and its construction graph.

5. EXPERIMENTATION

In order to evaluate the performance of our proposed methods, we conducted an experimental study on a variety of satisfiable and unsatisfiable problems. All the methods and techniques are coded in a C++.Net and each of the results is an average of 10 runs (note that, in most of the cases the results for each of the 10 runs are almost the same). The

Problem		wdeg	RNDI	HC/MAC	ACO/MAC
Composed_75_1_80_0(sat)	t	0.13	0.5	0.14	0.4
	n	865	9	10	10
Composed_75_1_80_9(sat)	t	0.15	0.18	0.2	0.4
	n	716	10	10	10
geo50-20-d4-75-60_ext	t	105	6	7	18
	n	127537	2024	5052	21042
geo50-20-d4-75-65_ext	t	2	1	3	7
	n	4227	1848	200	4614
ehi-85-297-17_ext(unsat)	t	2	3	-	-
	n	234	200	-	-

Table 1: Test results for quasi random problems.

tests were conducted on a computer running windows XP, with Intel(R) Core 2Duo CPU P8400 @ 2.26GHz processor and 2 GB RAM of memory. The problem sets have been taken from Lecoutre's benchmarks at [13].

Our methods are compared to our implementation of RNDI and wdeg. The first reason for comparing with RNDI and wdeg is that both of these methods also weigh the constraints to achieve a variable ordering. The second reason is that, according to Balafoutis and Stergiou in [2] who have done an experimental evaluation on modern variable ordering heuristics, conflict driven heuristics are almost always faster than the other existing variable ordering heuristics including impact based heuristics. Hence, they are considered to be some of the most powerful proposed variable ordering techniques.

The purpose of this experimentation is to compare the algorithms only based on how they weigh the constraints. For this reason, wdeg heuristic was used instead of dom/wdeg in the RNDI algorithm. For the HC problem, the total number of iterations was set based on the difficulty of the problem and ranges from 200 to 4000. Easy problems need less and harder problems need more iterations to explore the search space. The range of the cutoff parameter was between 1 and 50 for all of the instances.

For the ACO algorithm, the number of restarts ranges from 1 to 5. The number of cycles for each restart ranges from 10 to 50, depending on the hardness of the problem. Local search did not seem to be very effective for many of the test instances and hence, the number of local search iterations was set to a number between 0 and 20. RNDI has two parameters. The first one is the number of restarts and the second one is the maximum number of nodes that each of these restarts can visit.

These parameters were also set based on the difficulty of each instance. Harder instances need a more through exploration of the search tree to be able to provide a good variable ordering. The largest numbers assigned to the parameters restarts and maximum nodes were 20 and 4000 and they were used for the harder instances of pure random problems.

The algorithms are compared based on the average CPU time for reaching the solution (t) and the average number of visited nodes by MAC algorithm (n). The CPU time includes the preprocessing phase time (the phase that provides the weights). For RNDI, only the number of visited nodes in the last restart has been taken. For all the problems, the timeout was set to 700 seconds. For each instance, the parameters were tuned manually to get the best overall time performance. That means that the goal was to minimize the sum of the CPU time for preprocessing step and the main MAC search.

The first set of problems (satisfiable `Composed, geo` and unsatisfiable `ehi` instances listed in Table 1) are quasi random instances generated as described in [14]. `Composed` instances are made of a main part which is under-constrained and some auxiliary parts. Each of the auxiliary parts is attached to the main by using some binary constraints.

`ehi85` is a 3 SAT unsatisfiable instance that have been transformed into a binary CSP instance using the dual method described in [1]. The geometric instances (`geo50`) are random instances generated using a distance parameter. More precisely, for each variable, a point with two coordinates in a unit square is chosen randomly. Two variables will share a constraint if their corresponding points have a distance less or equal to the distance parameter.

As it can be seen in Table 1, dom/wdeg has the best performance amongst all the other algorithms. In the case of `ehi` test case, only RNDI and dom/wdeg are able to solve them.

The second set of problems (`frb` instances listed in Tables 2 and 3) are random binary satisfiable CSP instances created by the RB model [22]. This model is capable of generating problems close to phase transition as well as forced satisfiable instances that have the same hardness as unsatisfiable instances.

Table 2 reports the results for these problems. As we can see, the wdeg heuristic fails when the size and the hardness of the problem increase. HC/MAC has the best performance when it comes to the harder instances of frb series. ACO/MAC is also comparable to RNDI and HC/MAC for these hard instances. For easier instances, RNDI seems to have the best performance. For large instances, RNDI seems to have the best performance. Our experiments with different values of cut off parameter show that the cut off parameter for the HC algorithm should not be set to a large number, so that enough number of individuals can go under the constraint weighing process. For smaller numbers (0 to 100), the performance of the HC algorithm does not seem to be very dependent on this parameter.

Table 3 shows the results for cases when the proposed value and variable ordering methods were used together. As it can be seen, the results are significantly better than when variable ordering was used alone especially in the case of HC/MAC algorithm. It is worth mentioning that in a 2005 competition for solving these instances only about half of the solvers were able to solve the frb-40-19 series in under ten minutes.

Problem		wdeg	RNDI	HC/MAC	ACO/MAC
frb-35-17-1(sat)	t	14	8	10	14
	n	27497	15178	18898	22063
frb-40-19-2(sat)	t	429	86	85	99
	n	572319	96710	97871	83110
frb-35-19-3(sat)	t	414	387	368	355
	n	633499	519902	478107	518501
frb-35-17-4(sat)	t	469	387	233	359
	n	633499	345338	322696	446268
frb-35-17-5(sat)	t	460	390	304	431
	n	726460	544810	452435	439321

Table 2: Test results for random binary constraint problems generated by the RB model.

Problem		HC/MAC	ACO/MAC
frb-40-19-2(sat)	t	60	124
	n	36830	114332
frb-40-19-3(sat)	t	70	128
	n	63269	89072
frb-40-19-4(sat)	t	232	202
	n	308880	226713
frb-40-19-5(sat)	t	207	151
	n	254584	155381

Table 3: Test results when value and variable ordering methods are used together.

Figure 4 depicts the weights obtained by different algorithms for ehi test case. As it can be seen, RNDI was able to make a pretty good distinction between variables, since in the RNDI graph, only a few variables have noticeably high weights and the other variables have very low weights. Thus, RNDI can provide the MAC search with a more precise variable ordering.

Problem		HC/MAC	ACO/MAC
anna-8(unsat)	t	60	124
	n	36830	114332
anna-9(unsat)	t	70	128
	n	63269	89072
jean-7(unsat)	t	232	202
	n	308880	226713
david-9(unsat)	t	207	151
	n	254584	155381

Table 4: Test results for the graph coloring problem.

The graph coloring problem is a special case of graph labeling in graph theory. The problem is to assign colors to vertices of a graph such that no two adjacent vertices have the same color. The sets of unsatisfiable instances used for

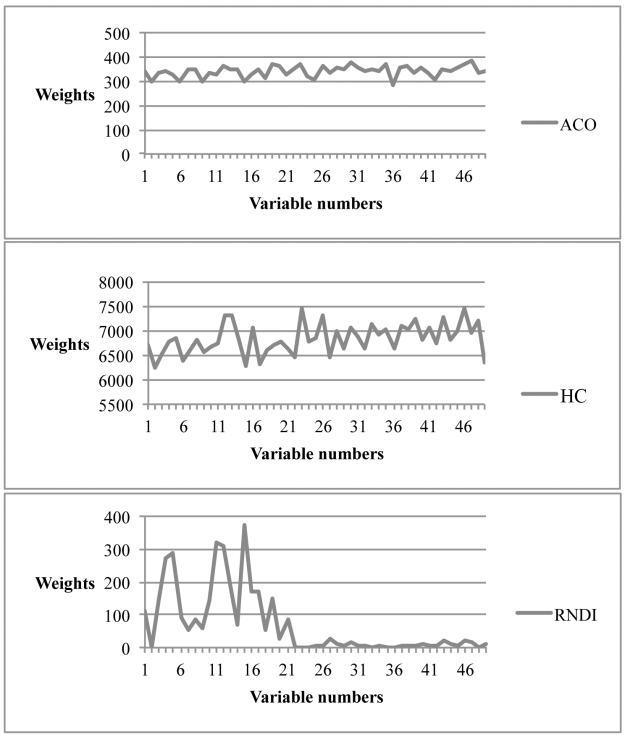

Figure 4: Weights obtained for ehi test case.

Problem		wdeg	RNDI	HC/MAC	ACO/MAC
anna-8(unsat)	t	175	8	10	10
	n	1402564	69828	129760	100192
anna-9(unsat)	t	-	60	96	111
	n	-	627739	1784745	1485369
jean-7(unsat)	t	1	0.4	1	1
	n	35543	8659	35543	16219
david-9(unsat)	t	-	40	23	26
	n	-	624251	623529	623529

Table 5: Test results when using both variable and value ordering methods.

our tests are from [12]. Given a work of literature, a graph is created where each node represents a character. Two nodes are connected by an edge if the corresponding characters encounter each other in the book. The three test sets `anna`, `david`, and `jean` correspond respectively to the following novels: *Anna Karenina, David Copperfield*, and *Les Misérables*.

As it can be seen in Table 4, RNDI has the best performance among other algorithms in terms of CPU time and the number of visited nodes. Only in the `david-9` test instances, HC/MAC and ACO/MAC result in a better time performance.

Table 5 shows the results of running HC/MAC and ACO/-MAC where suggested variable and value ordering were used together. The results show that the ACO value ordering makes the running time slightly better for the `anna-9` instances.

6. CONCLUSION

In this paper, we introduced two new methods for deciding variable ordering and two heuristics for deciding value ordering in CSPs. These techniques use heuristic algorithms to identify the hard constraints, while comparable approaches try to learn the hard constraints during constraint propagation in a systematic search. RNDI, HC/MAC and ACO/MAC all do a search prior to the main search process to identify the global sources of difficulty. This way, the last search can start with more informed decisions.

These hybrid approaches make it possible to use incomplete algorithms to gather information even when it comes to inconsistent problems. The two value ordering heuristics we propose rank the values based on their ability to satisfy the constraints attached to their corresponding variables.

The experiments we conducted on random, quasi random, and structured graph coloring problems demonstrate that HC/MAC and ACO/MAC have better performance in the case of hard random problems and some structured problems. However, in some types of structured problems such as ehi problem, RNDI seems to be able to find the hardest spots of the search space much easier and boost the performance by focusing on those spots, whereas using a non-systematic algorithm lowers the chances of finding those areas.

In the near future, we are looking to doing more tests on different kinds of problems including real world problems. We will also study different forms of local search and evolutionary algorithms to learn their ability in recognizing the hard parts of the search space.

7. REFERENCES

[1] F. Bacchus. Extending forward checking. In R. Dechter, editor, *CP*, volume 1894 of *Lecture Notes in Computer Science*, pages 35–51. Springer, 2000.

[2] T. Balafoutis and K. Stergiou. Experimental Evaluation Of Modern Variable Selection Strategies In Constraint Satisfaction Problems. In *Proceedings of the 15th RCRA workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion, online at CEUR Workshop Proceedings (ISSN 1613-0073)*, volume 451, 2008.

[3] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting Systematic Search By Weighting Constraints. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004), Valencia, Spain*, pages 146–150. IOS Press, 2004.

[4] J. Crawford. Solving Satisfiability Problems Using A Combination Of Systematic And Local Search. In *Second DIMACS Challenge: cliques, coloring, and satisfiability*, 1993.

[5] R. Dechter. *Constraint processing*. Morgan Kaufmann, 2003.

[6] R. Dechter and I. Meiri. Experimental Evaluation Of Preprocessing Techniques In Constraint Satisfaction Problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, volume 1, pages 271–277. Citeseer, 1989.

[7] M. Dorigo and T. Stutzle. *Ant Colony Optimization*. The MIT Press, 2004.

[8] J. Frank. Weighting for Godot: Learning heuristics for GSAT. In *Proceedings Of The National Conference On Artificial Intelligence*, pages 338–343, 1996.

[9] D. Grimes and R. Wallace. Learning To Identify Global Bottlenecks In Constraint Satisfaction Search. In *20th International FLAIRS conference*, 2007.

[10] R. Haralick and G. Elliott. Increasing Tree Search Efficiency For Constraint Satisfaction Problems. *Artificial intelligence*, 14(3):263–313, 1980.

[11] N. Jussien and O. Lhomme. The Path repair algorithm. *Electronic Notes in Discrete Mathematics*, 4:2–16, 2000.

[12] D. Knuth. *The Stanford Graphbase: A Platform For Combinatorial Computing*. Addison-Wesley Reading, MA, 1994.

[13] C. Lecoutre. http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html , October 2010.

[14] C. Lecoutre, F. Boussemart, and F. Hemery. Backjump-based techniques versus conflict-directed heuristics. In *ICTAI*, pages 549–557. IEEE Computer Society, 2004.

[15] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[16] P. Morris. The Breakout Method For Escaping From Local Minima. In *Proceedings Of The National Conference On Artificial Intelligence (AAAI-93)*, pages 40–45. John Wiley & Sons Ltd, 1993.

[17] A. Nareyek, S. Smith, and C. Ohler. Integrating Local-Search Advice Into Refinement Search (Or Not). In *Proceedings of the CP 2003 Third International Workshop on Cooperative Solvers in Constraint Programming*, pages 29–43, 2003.

[18] S. Prestwich. Maintaining Arc-Consistency In Stochastic Local Search. In *Workshop on Techniques for Implementing Constraint Programming Systems*, 2002.

[19] P. Refalo. Impact-Based Search Strategies For Constraint Programming. *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP 2004)*, pages 557–571, 2004.

[20] B. Smith and S. Grant. Trying Harder To Fail First. *Research Report Series-University Of Leeds School Of Computer Studies LU SCS RR*, 1997.

[21] C. Solnon. Ants Can Solve Constraint Satisfaction Problems. *IEEE Transactions on Evolutionary Computation*, 6(4):347–357, 2002.

[22] K. Xu, F. Boussemart, F. Hemery, and C. Lecoutre. Random Constraint Satisfaction: Easy Generation Of Hard (Satisfiable) Instances. *Artificial Intelligence*, 171(8-9):514–534, 2007.

[23] J. Zhang and H. Zhang. Combining local search and backtracking techniques for constraint satisfaction. In *Proceedings Of The National Conference On Artificial Intelligence*, pages 369–374, 1996.