A Value Ordering Heuristic for Solving Ultra-Weak Solutions in Minimax Weighted CSPs*

Jimmy H.M. Lee & Terrence W.K. Mak
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
{jlee,wkmak}@cse.cuhk.edu.hk

***Abstract*—Minimax Weighted Constraint Satisfaction Problems (formerly called Quantified Weighted CSPs) are a framework for modeling soft constrained problems with adversarial conditions. In this paper, we study the effects of a value ordering heuristic in solving ultra-weak solutions on top of the alpha-beta tree search with constraint propagation. The value ordering heuristic is based on minimax heuristics from adversarial search, which selects values for variables according to the semantic of quantifiers by considering the problem as a two-player zero-sum game. In practice, implementing the heuristic requires costs approximations, and we devise three heuristic variants: HUnary, HBinary, and HFullBinary to approximate costs. In particular, we observe that combining these heuristic variants with consistency notions can achieve a better efficiency and a further reduction of search space. We perform experiments on three benchmarks to compare the effects on applying these heuristic variants, and confirm the feasibility and efficiency of our proposal.**

***Index Terms*—constraint optimization, soft constraint satisfaction, value ordering heuristics, minimax game search**

I. Introduction

Our work aims to tackle a class of optimization problems with adversarial conditions. As an example, we begin with a graph coloring game in which numbers are used instead of colors. The game is played by two players, and nodes are partitioned into two sets: a set S_1 for player 1 and a set S_2 for player 2. The game is played in a turn-based manner, and in each turn, a node corresponding to the turn will be chosen. If the chosen node belongs to set S_1 (S_2 resp.), player 1 (player 2 resp.) will write a number on the node. We assume both players know which node will be chosen at which turn before the game starts, and they can observe the numbers written at previous turns. After all the nodes are chosen, player 1 obtains a reward equal to the total difference between numbers of adjacent nodes. However player 2 is player 1's worst adversary, and the goal of player 2 is to minimize player 1's reward. The natural question for player 1 is to find the best strategy to maximize his/her reward against his/her worst adversary.

The graph coloring game described is a typical two-person zero-sum game with perfect information [1], [2], and we can solve it at different levels. Allis [3] proposes three solving levels for this type of games: *ultra-weakly solved*, *weakly solved*, and *strongly solved*. Ultra-weakly solved means the game-theoretic value of the initial position has been determined. This means we can determine the outcome of the scenario when both players are playing perfectly (i.e. best-worst case). Finding an ultra-weak solution for the graph coloring game indicates the guarantee reward for player 1, i.e. the reward player 1 can at least obtain regardless of player 2's moves. Weakly solved means a strategy, noted as winning strategy [4] in Quantified CSPs [5], for the initial position to achieve the game-theoretic value against any opposition is found. For the graph coloring game, solving the problem weakly allows player 1 to maximize his rewards against any moves (possibly not worst case move) played by player 2. Strongly solved is being used for a game for which such a strategy has been determined for all legal positions. A strongly solved solution for the graph coloring game allows player 1 to maximize his rewards not only against any moves played by player 2, but also against all moves played by himself/herself. Once a game is solved at a stronger level, the game is automatically solved at weaker ones. Finding solutions at stronger levels, however, implies substantially higher computation requirements. In particular in terms of space, ultra-weak solutions are linear in size, while the other two stronger ones are exponential. In this paper, we focus on ultra-weakly solved solutions, and the primal goal is to understand how well we can defend against the worst adversaries for planning purposes.

Minimax Weighted Constraint Satisfaction Problems (MWCSPs) (previously called Quantified Weighted Constraint Satisfaction Problems) [6], [7] are proposed to tackle optimization problems with adversarial conditions, combining quantifier structures from Quantified CSPs [5] to model the adversaries and soft constraints from Weighted CSPs [8] to model costs information. Previous work gives formal definitions for the framework, introduces how to adopt alpha-beta prunings to tackle the problem in branch and bound, suggests sufficient pruning conditions to achieve prunings and backtrackings, and introduces various consistency notions to increase the efficiency of the search.

In this paper, we study the effects of a value ordering heuristic in solving ultra-weak solutions of MWCSPs on top of the alpha-beta tree search. The value ordering heuristic is based on minimax heuristics from adversarial search [9], by viewing MWCSPs as a two-player zero-sum game. The two

*We are grateful to the anonymous referees for their constructive comments. The work described in this paper was generously supported by grants CUHK413808 and CUHK413710 from the Research Grants Council of Hong Kong SAR.

players, $\min$ player and $\max$ players, compete with each other. The $\min$ player controls minimization variables to minimize costs, while the $\max$ player controls maximization variables to maximize costs. Both of them can observe each other's moves, and play the game by assigning values to variables in a turn-based manner. By viewing the problem as a game, ultra-weak solutions for MWCSPs will then be scenarios when both players are playing perfectly, where the $\min$ player ($\max$ player resp.) chooses values leading to sub-problems with the smallest (largest resp.) costs. By following the game semantics, if a variable has $\min$ ($\max$ resp.) quantifier, the value ordering heuristic will then choose values leading to sub-problems with the smallest (largest resp.) costs. In practice, however, implementing such heuristic is infeasible and requires costs approximations. We propose three heuristic variants: HUnary, HBinary, and HFullBinary to perform costs approximation. In particular, we observe that combining these heuristic variants with consistency notions can achieve a better efficiency and a further reduction of search space. We perform experiments on three benchmarks to compare the effects on applying these heuristic variants, and confirm the feasibility and efficiency of our proposal.

II. Background

In the first part, we give definitions and semantics of MWCSPs, followed by an example. In the second part, we highlight the general branch and bound algorithm with alpha-beta prunings for solving MWCSPs.

A. Definitions and Semantics

A *Minimax Weighted Constraint Satisfaction Problem* (MWCSP) [6], [7] $\mathcal{P}$ is a tuple $(\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{Q}, k)$, where $\mathcal{X} = (x_1, \ldots, x_n)$ is defined as an ordered sequence of *variables*, $\mathcal{D} = (D_1, \ldots, D_n)$ is an ordered sequence of finite *domains*, $\mathcal{C}$ is a set of *soft constraints*, $\mathcal{Q} = (Q_1, \ldots, Q_n)$ is a *quantifiers sequence* where Q_i is either $\max$ or $\min$ associated with x_i, and k is the global upper bound. We denote $x_i = v_i$ an *assignment* assigning value $v_i \in D_i$ to variable x_i, and the set of assignments $l = \{x_1 = v_1, x_2 = v_2, \ldots, x_n = v_n\}$ a *complete assignment* on variables in $\mathcal{X}$, where v_i is the value assigned to x_i. A *partial assignment* $l[S]$ is a projection of l onto variables in $S \subseteq \mathcal{X}$. $\mathcal{C}$ is a set of *soft constraints* (sometimes called cost functions), each C_S of which represents a function mapping tuples corresponding to assignments on a subset of variables S, to a cost valuation structure $V(k) = ([0...k], \oplus, \leq)$. The structure $V(k)$ contains a set of integers $[0...k]$ with standard integer ordering $\leq$. Addition $\oplus$ is defined by $a \oplus b = \min(k, a + b)$. For any integer a and b where $a \geq b$, subtraction $\ominus$ is defined by $a \ominus b = a - b$ if $a \neq k$, and $a \ominus b = k$ if $a = k$. Without loss of generality, we assume the existence of $C_\varnothing$ denoting the lower bound of the minimum cost of the problem. If it is not defined, we assume $C_\varnothing = 0$. The *cost* of a complete assignment l in $\mathcal{X}$ is defined as: $cost(l) = C_\varnothing \oplus \bigoplus_{C_s \in \mathcal{C}} C_s(l[S])$.

In an MWCSP, ordering of variables is important. Without loss of generality, we assume variables are ordered by their indices. We define a variable with $\min$ ($\max$ resp.) quantifier to be a minimization variable (maximization variable resp.). Let $\mathcal{P}[x_{i_1} = a_{i_1}][x_{i_2} = a_{i_2}] \ldots [x_{i_m} = a_{i_m}]$ be the *sub-problem* obtained from $\mathcal{P}$ by assigning value a_{i_1} to variable x_{i_i}, assigning value a_{i_2} to variable x_{i_2},..., assigning value a_{i_m} to variable x_{i_m}. Let firstx($\mathcal{P}$) returns the first unassigned variable in the variable sequence. If there are no such variables, it returns $\perp$. Suppose l is a complete assignment of $\mathcal{P}$. The *A-cost($\mathcal{P}$)* of an MWCSP $\mathcal{P}$ is defined recursively as follows:

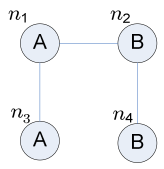

Fig. 2. Game graph G for Example 1

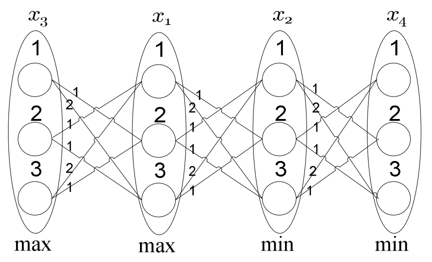

Fig. 3. Constraints for Example 1

$$\text{A-cost}(\mathcal{P}) = \begin{cases} cost(l), \text{ if } \text{firstx}(\mathcal{P}) = \perp \\ \max(\mathbb{M}_i), \text{ if } \text{firstx}(\mathcal{P}) = x_i \text{ and } Q_i = \max \\ \min(\mathbb{M}_i), \text{ if } \text{firstx}(\mathcal{P}) = x_i \text{ and } Q_i = \min \end{cases}$$

where l is the complete assignment of the completely assigned problem $\mathcal{P}$ (i.e. $\text{firstx}(\mathcal{P}) = \perp$), and $\mathbb{M}_i = \{\text{A-cost}(\mathcal{P}[x_i = v]) | v \in D_i\}$. An MWCSP $\mathcal{P}$ is *satisfiable* iff $\text{A-cost}(\mathcal{P}) < k$.

We now define three solution concepts corresponding to the three solved levels introduced in the introduction. An *ultra-weak solution* [7] of an MWCSP $\mathcal{P}$ is a complete assignment $\{x_1 = v_1, \ldots, x_n = v_n\}$ s.t. $\text{A-cost}(\mathcal{P}) = \text{A-cost}(\mathcal{P}[x_1 = v_1] \ldots [x_i = v_i]), \forall 1 \leq i \leq n$. Ultra-weak solution corresponds to the scenario where both players are playing perfectly. Without loss of generality, we assume the $\max$ player is the adversary. A *weak solution* [7] (*strong solution* [7] resp.) is a set of functions $\mathcal{F}$, where each function $f_i \in \mathcal{F}$ corresponds to a $\min$ variable x_i. Let G_i be the set of domains of max variables (all variables resp.) preceding x_i, i.e. $G_i = \{D_j \in \mathcal{D} | Q_j = \max \wedge j < i\}$ ($G_i = \{D_j \in \mathcal{D} | j < i\}$ resp.). We define $f_i : \times_{D_j \in G_i} D_j \mapsto D_i$. If G_i is an empty set, then f_i is a constant function returning values from D_i. Let $\mathcal{P}'$ be a sub-problem of an MWCSP $\mathcal{P}$, where the next unassigned variable x_i is a $\min$ variable, and l be the set of assigned values for $\max$ variables (all variables resp.) x_j where $j < i$. For weak solutions, we further require the assigned values of $\min$ variables x_j where $j < i$ in $\mathcal{P}'$ follow f_j. We require all f_i to satisfy: $\text{A-cost}(\mathcal{P}'[x_i = f_i(l)]) = \text{A-cost}(\mathcal{P}')$. In other words, we require $f_i(l)$ to return the best value for the $\min$ player, and the set of functions $\mathcal{F}$ will then be a best strategy for the $\min$ player. *This work focuses on ultra-weak solutions.*

Example 1: We use the graph coloring game described in the introduction as an example. Given a graph coloring game with a graph G of four nodes n_1, n_2, n_3, and n_4 and three edges $e_{1,2}, e_{1,3}$, and $e_{2,4}$ as shown in Figure 2. We have two players, player A and player B, playing in the game. Player

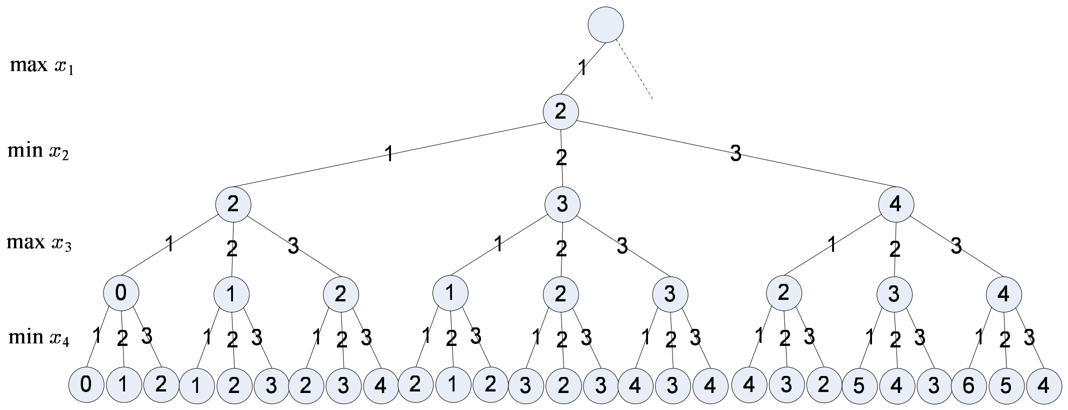

Fig. 1. The labeling tree for Example 1

A will write numbers on node n_1 and n_3 and player B will write numbers on node n_2 and n_4. To simplify the example, we assume both players can only write integers ranging from 1 to 3. At turn i, node n_i will be chosen. Both players are allowed to observe numbers written by players in previous turns. Suppose player B is player A's adversary. The problem is to help player A finding his/her guaranteed reward.

We model the problem by a Minimax Weighted CSP $\mathcal{P}$ with four variables x_1,x_2,x_3 and x_4, and each variable is associated with a domain of $[1..3]$. Variable x_i is used to represent the number being written on node n_i. We associate $\max$ quantifiers to variable x_1 and x_3 and $\min$ quantifiers to variable x_2 and x_4. We use three binary constraints $C_{1,2}, C_{1,3}$ and $C_{2,4}$ to model costs given by edges $e_{1,2}$, $e_{1,3}$ and $e_{2,4}$ in the graph G. Figure 3 shows the constraint graph of the problem $\mathcal{P}$. For unary constraints, unary costs are depicted inside a circle and domain values are placed above the circle. For binary constraints, binary costs are depicted as labels on edges connecting the corresponding pair of values. To simplify the drawing, we skip all zero unary costs and remove edges with zero binary costs. By observing the constraint graph, we can easily infer the maximum costs player A can get is less than 7. Therefore, we set the global upper bound k to 7.

We show a labeling tree in Figure 1 for one sub-problem $\mathcal{P}' = \mathcal{P}[x_1 = 1]$. By following the tree, we can easily infer the A-cost for the sub-problem $\mathcal{P}'$ is 2, and $\{x_1 = 1, x_2 = 1, x_3 = 3, x_4 = 1\}$ is one of the ultra-weak solutions for $\mathcal{P}'$.

B. Alpha-Beta Prunings in B & B

MWCSPs can be solved by applying alpha-beta pruning in branch and bound search [6] as shown in Algorithm 1, by treating $\max$ and $\min$ variables as $\max$ and $\min$ players respectively. Alpha-beta pruning utilizes two bounds, α and β, for storing the current best costs for $\max$ and $\min$ players. We rename α and β as lower lb and upper ub bounds to fit with the common notations for bounds in constraint and integer programming. We initialize lb (ub resp.) to the lowest (largest resp.) possible costs, i.e. 0 (k resp.), and maintain the two bounds during assignments by the branch and bound. When a smaller costs (larger costs resp.) for $\min$ ($\max$ resp.) variable is found after exploring sub-trees, ub (lb resp.) will be updated (line 8 and 10). If $lb \geq ub$ (line 12), then one of the previous branch must dominate over the current sub-tree, and we can perform backtrack.

Algorithm 1 Alpha-beta for MWCSPs

```
1:  function alpha_beta($\mathcal{P}$,$lb$,$ub$):
2:  if firstx($\mathcal{P}$) == $\perp$ then
3:      return $cost(\mathcal{P})$
4:  end if
5:  $x_i$ = firstx($\mathcal{P}$)
6:  for $v \in D_i$ do
7:      if $Q_i == \min$ then
8:          $ub$ = $\min(ub$, alpha_beta$(\mathcal{P}[x_i = v], lb, ub))$
9:      else
10:         $lb$ = $\max(lb$, alpha_beta$(\mathcal{P}[x_i = v], lb, ub))$
11:     end if
12:     if $ub <= lb$ then
13:         break
14:     end if
15: end for
16: return $(Q_i == \min)?ub : lb$
```

III. A Value Ordering Heuristic for Minimax Weighted CSPs

Value ordering heuristic is an important topic in constraint solving community, and has been successfully applied in solving many constraint problems including: classical constraint optimization problems (COPs), Weighted CSPs, and Quantified CSPs. This section is splitted into two parts. In the first part, we describe the principle of our value ordering heuristic, followed by an example. In the second part, three variants approximating the heuristic are given.

A. Principle

In Weighted CSPs, our goal is to find complete assignments with minimum costs. We often apply a value ordering heuristic which orders values with increasing order of unary costs [10], [11]. The idea behind is that choosing a value with smaller unary costs will have a higher chance leading to optimal solutions. Even if the search cannot obtain optimal solutions, the heuristic will have a high chance leading to solutions close to the optimal and improve the upper bound quickly. In Quantified CSPs, Stynes and Brown devise a type of heuristics called solution-focused adversarial heuristics [12].

The heuristics view Quantified CSPs as two-person zero-sum games and successfully applied the minimax heuristics from adversarial search. MWCSPs are generalizations of Weighted CSPs and Quantified CSPs [6], and can be also viewed as two-person zero-sum games. We show that concepts and ideas for both frameworks can be adapted and re-used, and our value ordering heuristic can be seen as a further extension of Stynes and Brown's work [12] to MWCSPs, which also considers costs information from soft constraints.

We now describe the idea of our value ordering heuristic. There are two players: $\min$ player and $\max$ player competing with each other. The $\min$ player controls minimization variables to minimize costs, while the $\max$ player controls maximization variables to maximize costs. They can observe the other player's moves and are played in a turn-based manner, by assigning values to variables. By viewing the problem as a game, ultra-weak solutions for MWCSPs will then be scenarios where both players are playing perfectly. If both players are playing perfectly, then the $\min$ player will choose values which can lead to sub-problems with the smallest A-costs. On the other hand, the $\max$ player will choose values which can lead to sub-problems with the largest A-costs. We now define the desired heuristic formally.

Definition 1: Given an MWCSP $\mathcal{P}$ with the next unassigned variable x_i (i.e. $\text{firstx}(\mathcal{P}) = x_i$). The *minimax adversarial heuristic* is a function selecting $v \in D_i$ for x_i where $\forall u \in D_i$:

$$\text{A-cost}(\mathcal{P}[x_i = v]) \leq \text{A-cost}(\mathcal{P}[x_i = u]), \text{ if } Q_i = \min$$
$$\text{A-cost}(\mathcal{P}[x_i = v]) \geq \text{A-cost}(\mathcal{P}[x_i = u]), \text{ if } Q_i = \max$$

The above definition always assumes assigned values during backtrack (in branch and bound) will be removed from the domain, and these values will not be considered in the heuristic.

Lemma 1: Given an MWCSP $\mathcal{P}$. By applying the minimax adversarial heuristic on branch and bound search (in Algorithm 1), the first encountered complete assignment must be an ultra-weak solution.

The proof follows from the heuristic and problem definitions.

However, computing the heuristic *exactly* in branch and bound for an MWCSP in general is too computational expensive. It is essentially equivalent to solving the whole problem. One way to relax the requirement is to restrict the heuristic to examine only parts of the problem, hence, reducing the computational requirements. Even the heuristic cannot lead the search to the desired solution, if we can obtain (complete) assignments which are close enough, we can tighten the bound earlier and achieve better prunings. We give an example in Example 2 to illustrate the idea.

Example 2: We re-use the graph coloring game in Example 1. Suppose now the search finished searching the sub-problem $\mathcal{P}' = \mathcal{P}[x_1 = 1][x_2 = 1]$, and we obtain an A-cost of 2 for $\mathcal{P}'$. Since the quantifier of x_2 is $\min$, we set the upper bound ub in alpha-beta pruning (Algorithm 1) to 2. By observing the labeling tree in Figure 1, we can see the search now aims to find whether there exists better, i.e. smaller, A-costs for the problem. Suppose the search explores sub-problem $\mathcal{P}'' = \mathcal{P}[x_1 = 1][x_2 = 2]$. Figure 4 shows the alpha-beta search, by ordering values (from left to right) in lexicographic ordering. Nodes marked with 'X' are pruned by alpha-beta search. We can see the right-most sub-tree is pruned. Suppose now we order values according to minimax adversarial heuristics. Figure 5 shows we can obtain a smaller search tree by pruning more nodes in the alpha-beta search. The reason is that variable x_3 will first explore sub-problem $\mathcal{P}''[x_3 = 3]$ instead of sub-problem $\mathcal{P}''[x_3 = 1]$. Traversing sub-problem $\mathcal{P}''[x_3 = 3]$ earlier allows $\mathcal{P}''$ to obtain a stronger lower bound lb of 3 earlier. This causes an earlier backtrack.

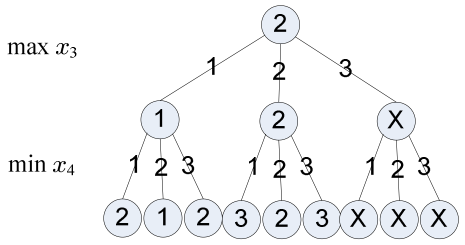

Fig. 4. Alpha-beta search with lexicographic ordering in Example 2

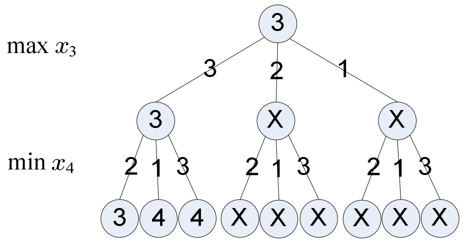

Fig. 5. Alpha-beta search with minimax adversarial heuristics in Example 2

B. Implementing the heuristic via approximations

We propose three approximations to implement the minimax adversarial heuristic. All consider only parts of the problem by examining variables and constraints related to the variable being assigned. We start by showing the basic heuristic HUnary, which covers only one unary constraint, followed by the other two: HBinary and HFullBinary. We write C_i for the unary constraint on variable x_i, $C_{i,j}$ for the binary constraint on variables x_i and x_j where $i < j$, $C_i(u)$ for the cost returned by the unary constraint when u is assigned to x_i, and $C_{i,j}(u, v)$ for the cost returned by the binary constraint when u and v are assigned to x_i and x_j respectively.

Definition 2: Given an MWCSP $\mathcal{P}$ and the next unassigned variable x_i. *HUnary*$(\mathcal{P})$ is a function returning value $v \in D_i$ for x_i where $\forall u \in D_i$, $C_i(v) \leq C_i(u)$ if $Q_i = \min$, and $C_i(v) \geq C_i(u)$ if $Q_i = \max$.

Definition 3: Given an MWCSP $\mathcal{P}$ and the next unassigned variable x_i. We denote set B_i to be the set of binary constraints constraining variable x_i and all future variables x_j, i.e. $B_i = \{C_{i,j} \in \mathcal{C} | j > i\}$. *HBinary*$(\mathcal{P})$ is a function returning value $v \in D_i$ for x_i s.t. $\forall u \in D_i$:

$$\text{If } Q_i = \min : C_i(v) \oplus \bigoplus_{C_{i,j} \in B_i} \Big(\underset{w \in D_j}{Q_j} \{C_{i,j}(v, w)\} \Big)$$
$$\leq C_i(u) \oplus \bigoplus_{C_{i,j} \in B_i} \Big(\underset{w \in D_j}{Q_j} \{C_{i,j}(u, w)\} \Big)$$

$$\text{If } Q_i = \max : C_i(v) \oplus \bigoplus_{C_{i,j} \in B_i} \big(\underset{w \in D_j}{Q_j} \{C_{i,j}(v,w)\}\big)$$
$$\geq C_i(u) \oplus \bigoplus_{C_{i,j} \in B_i} \big(\underset{w \in D_j}{Q_j} \{C_{i,j}(u,w)\}\big)$$

where ${Q_j}_{w\in D_j}\{C_{i,j}(v,w)\}$ is equal to $\min_{w\in D_j} C_{i,j}(v,w)$ if $Q_j = \min$, and equal to $\max_{w\in D_j} C_{i,j}(v,w)$ if $Q_j = \max$. Similarly, *HFullBinary*$(\mathcal{P})$ is a function returning value $v \in D_i$ for x_i s.t. $\forall u \in D_i$:

$$\text{If } Q_i = \min : C_i(v) \oplus \bigoplus_{C_{i,j} \in B_i} \big(\underset{w \in D_j}{Q_j} \{C_{i,j}(v,w) \oplus C_j(w)\}\big)$$
$$\leq C_i(u) \oplus \bigoplus_{C_{i,j} \in B_i} \big(\underset{w \in D_j}{Q_j} \{C_{i,j}(u,w) \oplus C_j(w)\}\big)$$
$$\text{If } Q_i = \max : C_i(v) \oplus \bigoplus_{C_{i,j} \in B_i} \big(\underset{w \in D_j}{Q_j} \{C_{i,j}(v,w) \oplus C_j(w)\}\big)$$
$$\geq C_i(u) \oplus \bigoplus_{C_{i,j} \in B_i} \big(\underset{w \in D_j}{Q_j} \{C_{i,j}(u,w) \oplus C_j(w)\}\big)$$

where ${Q_j}_{w\in D_j}\{C_{i,j}(v,w) \oplus C_j(w)\}$ is equal to $\min_{w\in D_j}\{C_{i,j}(v,w) \oplus C_j(w)\}$ if $Q_j = \min$, and equal to $\max_{w\in D_j}\{C_{i,j}(v,w) \oplus C_j(w)\}$ if $Q_j = \max$.

Theorem 1: Given an MWCSP $\mathcal{P}$ and $\text{firstx}(\mathcal{P}) = x_i$. If C_i is the *only* constraint constraining x_i, then HUnary is equivalent to minimax adversarial heuristic.

Theorem 2: Given an MWCSP $\mathcal{P}$ and $\text{firstx}(\mathcal{P}) = x_i$. We denote S_i to be the set of variables constraining with x_i and $U[S_i]$ to be the set of unary constraints on these variables, i.e. $U[S_i] = \{C_j | x_j \in S_i\}$. If the set of constraints $B_i \cup \{C_i\}$ ($B_i \cup \{C_i\} \cup U[S_i]$ resp.) is the *only* set of constraints constraining x_i and all variables in S_i, HBinary (HFullBinary resp.) is equivalent to minimax adversarial heuristic.

To prove Theorem 1 and Theorem 2, we begin by introducing Lemma 2.

Lemma 2: Given an MWCSP $\mathcal{P}$. Suppose there exists a set S of variables in $\mathcal{P}$, such that each of these variables $x_j \in S$ is being constrained by the respective unary constraint C_j *only*. We let $U[S]$ be the set of unary constraints covering variables in S, and further let $\mathcal{P}'$ be a modified sub-problem of $\mathcal{P}$ by removing all variables in the set S, including the associated domains, the associated quantifiers, and the set of associated constraints $U[S]$. Then,

$$\text{A-cost}(\mathcal{P}) = \bigoplus_{C_j \in U[S]} Q_j C_j \oplus \text{A-cost}(\mathcal{P}')$$

where $Q_j C_j$ is equal to $\min_{v\in D_j} C_j(v)$ if $Q_j = \min$, and equal to $\max_{v\in D_j} C_j(v)$ if $Q_j = \max$.
The proof follows from the definition of A-costs.

One direct consequence of Lemma 2 is that if there exists a set of variables (after a series of assignments in branch and bound) which are constrained by unary constraints only, we can divide the A-cost of the problem $\mathcal{P}$ into two parts/sub-problems: $\bigoplus_{C_j \in U[S]} Q_j C_j$ and $\text{A-cost}(\mathcal{P}')$. It is easy to observe that computing the first part is easy and essentially solves the ultra-weak solution for variables in S.

Proof: (Theorem 1) Let $\mathcal{P}'$ be a modified sub-problem of $\mathcal{P}$ by removing variable x_i, the associated domain D_i, the associated quantifier Q_i, and all constraints associated with x_i. Suppose variable x_i is covered by only constraint C_i. By Lemma 2,

$$\text{A-cost}(\mathcal{P}) = Q_i C_i \oplus \text{A-cost}(\mathcal{P}')$$

$\mathcal{P}'$ does not involve variable x_i. Therefore,

$$\forall u \in D_i, \text{A-cost}(\mathcal{P}[x_i = u]) = C_i(u) \oplus \text{A-cost}(\mathcal{P}')$$

By taking minimum/maximum on values in D_i,

$$\min_{v \in D_i} \text{A-cost}(\mathcal{P}[x_i = v]) = \min_{v \in D_i}\{C_i(v)\} \oplus \text{A-cost}(\mathcal{P}'), \text{ and}$$
$$\max_{v \in D_i} \text{A-cost}(\mathcal{P}[x_i = v]) = \max_{v \in D_i}\{C_i(v)\} \oplus \text{A-cost}(\mathcal{P}')$$

The above two equations show if we want to find a value v minimizing/maximizing $\text{A-cost}(\mathcal{P}[x_i = v])$, it is equivalent to finding the value v giving the minimum/maximum costs of C_i, hence implementing HUnary. This shows we can implement minimax adversarial heuristic by implementing HUnary. ■

Proof: (Theorem 2) The proof is similar to the proof for Theorem 1, by reusing Lemma 2. We skip the proof for HBinary, and show the proof for HFullBinary which is more general. Let $\mathcal{P}'$ be a modified sub-problem of $\mathcal{P}$ by removing all variables in the set $S_i \cup \{x_i\}$. Similar to previous proof, we also remove all the associated domains, all the associated quantifiers, and all the associated constraints related to the set $S_i \cup \{x_i\}$ of variables in $\mathcal{P}'$. To simplify our notations, we denote the expression $C_i(u) \oplus \bigoplus_{C_{i,j}\in B_i}({Q_j}_{w\in D_j}\{C_{i,j}(u,w) \oplus C_j(w)\})$ as $E[i,u]$.

Suppose the set $S_i \cup \{x_i\}$ of variables covers by only the set of constraints $B_i \cup \{C_i\} \cup U[S_i]$. If we assign a value u of x_i to the problem $\mathcal{P}$, the set of binary constraints B_i will become unary. We can merge B_i and the set of unary constraints $U[S_i]$ to give a new set of unary constraints $U'[S_i]$. The set of unary constraints $U'[S_i]$ is constructed as follows. For all unary constraints $C_j \in U[S_i]$, if there exists a binary constraint $C_{i,j} \in B_i$, then for all values w in D_j, the merged unary constraint $C'_j \in U'[S_i]$ will return $C_j(w) \oplus C_{i,j}(u,w)$. We can see after a value assignment $x_i = u$ on x_i, there are only unary constraints covering on the set of variables $S_i \cup \{x_i\}$. By using Lemma 2, $\forall u \in D_i$,

$$\text{A-cost}(\mathcal{P}[x_i = u]) = C_i(u) \oplus \{\bigoplus_{C'_j \in U'[S_i]} Q_j C'_j\} \oplus \text{A-cost}(\mathcal{P}')$$

We then replace the set of merged unary constraint $U'[S_i]$ into the set of unary constraints $U[S_i]$ and the set of binary constraints B_i in the equation. This gives,

$$\forall u \in D_i, \text{A-cost}(\mathcal{P}[x_i = u]) = E[i,u] \oplus \text{A-cost}(\mathcal{P}')$$

Recall sub-problem $\mathcal{P}'$ does not involve the set of variables $S_i \cup \{x_i\}$. By taking minimum/maximum on values D_i,

$$\min_{v \in D_i} \text{A-cost}(\mathcal{P}[x_i = v]) = \min_{v \in D_i}\{E[i,v]\} \oplus \text{A-cost}(\mathcal{P}'), \text{ and}$$
$$\max_{v \in D_i} \text{A-cost}(\mathcal{P}[x_i = v]) = \max_{v \in D_i}\{E[i,v]\} \oplus \text{A-cost}(\mathcal{P}')$$

The above two equations show if we want to find a value v minimizing/maximizing $\text{A-cost}(\mathcal{P}[x_i = v])$, it is equivalent to finding the value v giving the minimum/maximum costs for $E[i, v]$, hence implementing HFullBinary. By similar arguments, we can prove the case of HBinary. This completes the proof. ∎

By observing the two theorems, it is natural for us to ask whether applying the three heuristic variants on general MWCSPs can achieve better performance and smaller search space. We will test and show these heuristics are worthwhile in the experimental section. At current stage, all three heuristic variants consider *unary* and *binary* constraints only. Generalizing these heuristics to high arity/global constraints is an interesting future work. Note that our approach only consider value ordering heuristic for a variable at a time. By considering multiple variables at a time, there may exists cases where many variables are constrained by unary constraints only. By following Lemma 2, this could allows us to derive stronger and more effective heuristics in the future.

IV. Performance Evaluation

In this section, we compare the effects of the three value ordering heuristics: HUnary, HBinary, and HFullBinary. To further evaluate the effects on using the three heuristics, we also perform experiments on three heuristics: HUnary Rev, HBinary Rev, HFullBinary Rev, which are the reverse of HUnary, HBinary, and HFullBinary respectively. All six heuristics will be compared against the static lexicographic ordering (Lex). Apart from running these heuristics using only alpha-beta prunings in the branch and bound search, we also test our heuristics on two consistencies: DC-NC[proj-NC*] and DC-AC[proj-AC*] [7], which are used to increase prunings and backtrackings in the search, by filtering infeasible values and inferring backtrackings on top of alpha-beta prunings. We note there are also other consistency notions introduced in previous work [7]. However, as the results are similar to DC-NC[proj-NC*] and DC-AC[proj-AC*], we skip reporting the details.

We generate 20 instances for each benchmark's particular parameter setting. Results for each benchmark are tabulated with average time used (in sec.) and average number of tree nodes encountered. We take average for solved instances *only*. If there are any unsolved instances, we give the number of solved instances beside the average time (superscript in brackets). Winning entries are highlighted in bold. A symbol '-' represents all 20 instances fail to run within a time limit set differently for each benchmark problem. The experiment is conducted on a Core2 Duo 2.8GHz with 3.2GB memory.

A. Randomly Generated Problems

We perform experiments on randomly generated MWCSPs. These benchmarks are previously used by Lee, Mak, and Yip [6]. The random MWCSP instances are generated with parameters (n, d, p), where n is the number of variables, d is the domain size for each variable, and p is the probability for a binary constraint to occur between two variables. There are no unary constraints which makes the instances harder, and the costs for each binary constraint are generated uniformly in [0..30]. Quantifiers are generated randomly with half probability for $\min$ ($\max$ resp.), and the number of quantifier levels vary from instances to instances. Time limit is set to 900 seconds, and Table I shows the result.

B. Graph Coloring Games

We perform experiments on the graph coloring game in the introduction section. The benchmark instances are collected from Lee, Mak, and Yip [6]. The instances are generated with parameters (v, c, d), where v is an even number of nodes in the graph, c is the range of numbers allowed to place, and d is the probability of an edge between two vertices. Player 1 (Player 2 resp.) is assigned to play the odd (even resp.) numbered turns, and the node corresponding to each turn is generated randomly. Time limit is set to 1800 seconds, and Table II shows the results.

C. Generalized Radio Link Frequency Assignment Problem

We re-use the benchmark Generalized Radio Link Frequency Assignment Problem (GRLFAP) instances by Lallouet, Lee, and Mak [7]. The problem consists of assigning frequencies to a set of radio links located between pairs of sites, with the goal of preventing interferences. However, a certain set of links are placed in unsecured areas, and terrorists/spies may hijack/control these links. We are not able to re-adjust the frequencies for the other links immediately to minimize the interferences on the functioning ones. We want to find frequency assignments such that we can minimize the degree of radio links affected for the worst possible case. The instances are generated according to two small but hard CELAR sub-instances [13], which are extracted from a large instance CELAR6. All GRLFAP instances are generated with parameters (i, n, d, r), where i is the index of the CELAR sub-instances used ($\texttt{CELAR6-SUB}_i$), n is an even number of links, d is an even number of allowed frequencies, and r is the ratio of links placed in unsecured areas, $0 \leq r \leq 1$. For each instance, we randomly extract a sequence of n links from $\texttt{CELAR6-SUB}_i$ and fix a domain of d frequencies. We randomly choose $\lfloor (r \times n + 1)/2 \rfloor$ pairs of links to be unsecured. If two links are restricted not to take frequencies f_i and f_j with distance less than t, we measure the costs of interference by using a binary constraint with violation measure $\max(0, t - |f_i - f_j|)$. We set the time limit to 3600 seconds. Table III shows the results.

D. Results & Discussions

In all the three benchmarks, applying value ordering heuristics have runtime improvement over the static lexicographic ordering. When the solver is maintaining consistencies, e.g. DC-NC[proj-NC*]/DC-AC[proj-AC*], even the simplest heuristic HUnary runs faster than applying lexicographic ordering. We first discuss results when the solver is running alpha-beta prunings *only* (i.e. no consistency enforcement), followed by results when the solver enforces consistencies.

TABLE I

RANDOMLY GENERATED PROBLEM

	Alpha-beta Lex		HUnary		HUnary Rev		HBinary		HBinary Rev		HFullBinary		HFullBinary Rev	
(n, d, p)	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes
(12, 5, 0.4)	66.84	5,967,461	75.39	5,897,438	75.40	5,897,438	**39.06**	**2,830,764**	141.76	10,339,204	39.18	**2,830,764**	143.00	10,339,204
(12, 5, 0.6)	53.32	4,782,541	90.46	6,796,546	90.43	6,796,546	**41.82**	**2,947,470**	109.73^{19}	7,515,682	42.02	**2,947,470**	110.61^{19}	7,515,682
(12, 5, 0.8)	37.65^{19}	2,795,500	61.65^{19}	3,691,135	61.76^{19}	3,691,135	**30.95**^{19}	**1,666,716**	145.51^{19}	8,166,657	31.39^{19}	**1,666,716**	146.66^{19}	8,166,657
(16, 5, 0.4)	510.46^{2}	26,269,025	-	-	-	-	**461.87**^{10}	**20,304,664**	-	-	463.83^{10}	**20,304,664**	-	-
(16, 5, 0.6)	679.83^{2}	36,315,673	-	-	-	-	**620.32**^{11}	**22,689,337**	-	-	627.52^{11}	**22,689,337**	-	-
(16, 5, 0.8)	738.34^{3}	33,096,934	-	-	-	-	**641.04**^{9}	**21,783,228**	-	-	646.40^{9}	**21,783,228**	-	-
(20, 5, 0.4)	-	-	-	-	-	-	-	-	-	-	-	-	-	-
(20, 5, 0.6)	-	-	-	-	-	-	-	-	-	-	-	-	-	-
(20, 5, 0.8)	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	DC-NC[proj-NC*] Lex		HUnary		HUnary Rev		HBinary		HBinary Rev		HFullBinary		HFullBinary Rev	
(n, d, p)	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes
(12, 5, 0.4)	5.73	131,468	1.30	26,185	32.74	861,341	0.82	15,318	41.62	1,040,848	**0.77**	**14,159**	54.13	1,393,713
(12, 5, 0.6)	4.52	101,690	1.23	24,307	45.10	1,178,818	0.85	15,340	65.18	1,607,882	**0.71**	**12,402**	83.64	2,096,126
(12, 5, 0.8)	6.61	147,525	1.82	34,649	47.96	1,163,180	1.43	24,362	65.49	1,495,977	**1.28**	**21,263**	81.18	1,875,538
(16, 5, 0.4)	325.82^{19}	4,617,612	30.94	380,305	252.52^{2}	3,242,046	25.59	290,328	583.41^{2}	7,311,226	**20.99**	**232,784**	346.20^{1}	3,860,844
(16, 5, 0.6)	454.36^{16}	6,157,070	34.94	426,025	889.40^{1}	11,811,844	26.71	296,878	-	-	**20.71**	**226,329**	-	-
(16, 5, 0.8)	428.38^{18}	5,681,283	39.19	470,231	-	-	31.24	339,084	-	-	**26.13**	**277,702**	-	-
(20, 5, 0.4)	-	-	464.59^{14}	3,631,220	-	-	310.82^{16}	2,339,663	-	-	**240.10**^{16}	**1,752,855**	-	-
(20, 5, 0.6)	-	-	663.10^{11}	5,081,140	-	-	363.75^{13}	2,599,439	-	-	**339.72**^{14}	**2,379,827**	-	-
(20, 5, 0.8)	-	-	544.28^{9}	4,113,674	-	-	416.80^{16}	2,903,194	-	-	**344.94**^{16}	**2,361,805**	-	-
	DC-AC[proj-AC*] Lex		HUnary		HUnary Rev		HBinary		HBinary Rev		HFullBinary		HFullBinary Rev	
(n, d, p)	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes
(12, 5, 0.4)	3.03	30,165	0.46	3,075	17.55	244,219	0.38	2,431	18.79	254,453	**0.35**	**2,072**	27.76	391,213
(12, 5, 0.6)	2.96	26,093	0.68	4,047	30.00	413,976	0.66	3,691	33.76	434,281	**0.46**	**2,345**	51.13	688,959
(12, 5, 0.8)	4.43	37,663	0.99	4,832	34.58	407,730	0.98	4,400	39.96	453,413	**0.83**	**3,541**	51.98	612,295
(16, 5, 0.4)	161.42	1,047,900	11.02	45,766	434.01^{4}	3,828,107	10.03	39,879	96.41^{2}	548,633	**7.11**	**26,312**	106.15^{2}	654,593
(16, 5, 0.6)	319.80	1,816,642	17.42	63,918	670.46^{2}	4,138,239	17.55	61,632	560.04^{1}	3,124,293	**14.55**	**48,842**	639.42^{1}	4,254,183
(16, 5, 0.8)	281.85	1,454,321	20.47	68,347	-	-	21.04	67,158	-	-	**15.79**	**47,765**	-	-
(20, 5, 0.4)	-	-	234.46^{18}	665,502	-	-	201.52^{18}	572,188	-	-	**125.62**^{18}	**338,949**	-	-
(20, 5, 0.6)	-	-	296.41^{16}	720,776	-	-	306.22	704,623	-	-	**227.17**	**495,816**	-	-
(20, 5, 0.8)	-	-	426.33^{18}	965,016	-	-	430.55^{18}	918,563	-	-	**326.23**	**661,739**	-	-

TABLE II

GRAPH COLORING GAME

	Alpha-beta Lex		HUnary		HUnary Rev		HBinary		HBinary Rev		HFullBinary		HFullBinary Rev	
(v, c, d)	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes
(14, 4, 0.4)	19.50	1,572,978	22.72	1,572,978	22.82	1,572,978	**8.08**	**500,744**	108.26	6,910,142	8.11	**500,744**	109.15	6,910,142
(14, 4, 0.6)	23.81	1,730,473	29.25	1,730,473	29.32	1,730,473	**8.21**	**428,177**	299.31	16,328,103	8.28	**428,177**	301.15	16,328,103
(24, 4, 0.4)	-	-	-	-	-	-	-	-	-	-	-	-	-	-
(24, 5, 0.6)	-	-	-	-	-	-	-	-	-	-	-	-	-	-
(22, 6, 0.4)	-	-	-	-	-	-	-	-	-	-	-	-	-	-
(32, 4, 0.4)	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	DC-NC[proj-NC*] Lex		HUnary		HUnary Rev		HBinary		HBinary Rev		HFullBinary		HFullBinary Rev	
(v, c, d)	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes
(14, 4, 0.4)	6.53	122,266	0.65	11,053	57.50	1,145,837	0.31	4,922	117.72	2,227,353	**0.21**	**3,346**	129.41	2,436,701
(14, 4, 0.6)	10.08	185,111	0.85	13,957	104.00	2,025,421	0.33	4,991	261.54	4,825,831	**0.29**	**4,348**	243.36	4,516,025
(24, 4, 0.4)	-	-	696.27	3,934,005	-	-	118.24	643,280	-	-	**77.97**	**421,340**	-	-
(24, 5, 0.6)	-	-	1,322.95^{1}	7,225,728	-	-	209.88	1,063,488	-	-	**206.64**	**1,045,374**	-	-
(22, 6, 0.4)	-	-	408.73^{1}	2,527,346	-	-	694.77^{17}	4,164,208	-	-	**655.62**^{19}	**3,930,443**	-	-
(32, 4, 0.4)	-	-	-	-	-	-	1,229.56^{2}	3,807,544	-	-	**1,103.53**^{4}	**3,410,632**	-	-
	DC-AC[proj-AC*] Lex		HUnary		HUnary Rev		HBinary		HBinary Rev		HFullBinary		HFullBinary Rev	
(v, c, d)	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes
(14, 4, 0.4)	4.30	37,252	0.62	4,230	31.04	335,624	0.35	2,290	50.19	520,860	**0.23**	**1,500**	64.37	651,782
(14, 4, 0.6)	7.49	59,359	0.94	5,602	59.23	582,871	0.42	2,350	108.84	1,012,054	**0.35**	**1,944**	129.50	1,191,190
(24, 4, 0.4)	-	-	626.15	1,485,890	-	-	128.95	290,397	-	-	**84.66**	**189,681**	-	-
(24, 5, 0.6)	-	-	1,509.69^{2}	2,979,983	-	-	263.27	469,303	-	-	**245.79**	**437,889**	-	-
(22, 6, 0.4)	-	-	338.84^{1}	756,394	-	-	762.24^{17}	1,722,831	-	-	**716.31**^{19}	**1,642,269**	-	-
(32, 4, 0.4)	-	-	-	-	-	-	1,301.11^{2}	1,666,841	-	-	**1,199.46**^{4}	**1,506,866**	-	-

TABLE III

GENERALIZED RADIO LINK FREQUENCY ASSIGNMENT PROBLEM

	Alpha-beta Lex		HUnary		HUnary Rev		HBinary		HBinary Rev		HFullBinary		HFullBinary Rev	
(i, n, d, r)	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes
(1, 24, 4, 0.2)	-	-	-	-	-	-	-	-	-	-	-	-	-	-
(0, 24, 4, 0.4)	-	-	-	-	-	-	-	-	-	-	-	-	-	-
(1, 22, 6, 0.2)	-	-	-	-	-	-	-	-	-	-	-	-	-	-
(0, 22, 6, 0.4)	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	DC-NC[proj-NC*] Lex		HUnary		HUnary Rev		HBinary		HBinary Rev		HFullBinary		HFullBinary Rev	
(i, n, d, r)	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes
(1, 24, 4, 0.2)	79.42	442,362	**77.92**	421,818	105.63	664,543	81.55	424,536	108.03	641,524	80.22	**416,453**	120.81	738,909
(0, 24, 4, 0.4)	136.06	828,286	72.50	426,532	704.26	4,887,260	73.35	420,142	731.51	4,968,383	**32.17**	**180,044**	821.89	5,551,107
(1, 22, 6, 0.2)	577.32	3,580,885	**546.69**	**3,292,629**	731.60	4,971,870	585.65	3,303,538	787.67	5,081,591	594.15	3,319,014	841.33	5,452,801
(0, 22, 6, 0.4)	1,101.79^{15}	7,862,914	571.24^{19}	3,757,999	1,058.49^{4}	9,531,187	580.28^{19}	3,641,583	1,002.93^{4}	8,855,429	**260.53**	**1,583,673**	248.97^{3}	1,995,544
	DC-AC[proj-AC*] Lex		HUnary		HUnary Rev		HBinary		HBinary Rev		HFullBinary		HFullBinary Rev	
(i, n, d, r)	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes
(1, 24, 4, 0.2)	49.87	74,182	48.96	68,381	67.64	133,192	49.34	67,766	70.64	138,577	**47.38**	**64,413**	75.67	156,642
(0, 24, 4, 0.4)	99.92	295,743	54.49	137,407	673.12	2,605,763	61.63	159,094	651.44	2,516,190	**38.61**	**91,439**	836.59	3,165,730
(1, 22, 6, 0.2)	309.21	352,439	**304.87**	316,659	394.91	582,929	305.54	**311,648**	405.28	594,559	306.15	311,994	424.82	680,745
(0, 22, 6, 0.4)	1,281.30	4,448,644	576.36	1,506,982	975.49^{5}	4,463,350	742.64	2,058,202	1,211.09^{6}	5,492,821	**410.46**	**972,607**	1,121.03^{5}	5,116,597

Alpha-beta prunings only For randomly generated problems and the graph coloring games, applying HUnary and its reverse on alpha-beta prunings gives the same number of backtracks and similar runtimes. In addition, applying HBinary (Rev) and HFullBinary (Rev) on alpha-beta prunings has the same number of backtracks. However, HBinary (Rev) runs slightly faster than HFullBinary (Rev). The reason is that there are no unary constraints on these two problems. For HUnary and HUnary Rev, both of them cannot distinguish which value of a variable is better. In this case, we degenerate them into the same ordering (in an arbitrary manner). For HFullBinary (Rev), it is degenerated into HBinary (Rev) according to definitions. However, HFullBinary (Rev) requires computing costs from more constraints than HBinary (Rev). Therefore, it runs slightly slower than HBinary (Rev). Overall, HBinary on alpha-beta prunings runs the fastest and maintains the smallest number of backtracks on these two benchmarks.

In fact, we can also observe applying the reverse of HBinary and HFullBinary, which can be viewed as approximating the reverse of the minimax adversarial heuristic, both have the worst number of backtracks. This suggest devising heuristics to approximate minimax adversarial heuristic is worthwhile. One point to note is that when maintaining the two consistency notions, we are required to use *projection* operations [7], which is an equivalence-preserving transformation by transferring costs from binary constraints to unary constraints. Even when unary constraints do not exist in the problem, they may be created when enforcing consistencies. The degeneration of HUnary/HFullBinary (Rev) is unlikely to appear once we enforce consistencies.

With Consistencies For all of the three benchmarks, even applying the simplest heuristic HUnary gives runtime and backtrack improvements.

For randomly generated problems and the graph coloring games, applying HFullBinary (HBinary resp.) has a smaller number of backtracks than applying HBinary (HUnary resp.). For the runtime, applying HFullBinary (HBinary resp.) on both DC-NC[proj-NC*] and DC-AC[proj-AC*] runs faster than applying HBinary (HUnary resp.). The only exception is the two parameters (16, 5, 0.6) and (16, 5, 0.8) in Table I, where HBinary on DC-AC[proj-AC*] runs slower than HUnary. In addition, we can observe applying the reverse heuristics is always worse than their counterparts.

One possible explanation is that by using a heuristic considering more constraints (e.g. HFullBinary), the heuristic is more accurate towards the minimax adversarial heuristic. This will allow bounds to tighten quicker and gives more prunings. However, computing these heuristics requires higher computational resources, and therefore, the runtime improvement for these heuristics may not always be worthwhile. On the other hand, the three reverse heuristics can be viewed as functions approximating the reverse of the minimax adversarial heuristic. Applying the reverse of the minimax adversarial heuristic should tighten bounds slower and gives less prunings. Therefore, all reverse heuristics runs slower (with a larger search space) comparing to their counterparts.

For GRLFAP, the number of backtracks is the smallest when applying HFullBinary (exception for (1, 22, 6, 0.2)) and HFullBinary has shown best runtime in many cases. However, the results for other heuristics, especially for HBinary, are different from the other two benchmarks. Applying HBinary on DC-NC[proj-NC*]/DC-AC[proj-AC*] does not always achieve a smaller number of backtracks and run faster than HUnary. We also observe for benchmarks when $r = 0.2$, applying the three heuristics: HUnary, HBinary, and HFullBinary can only slightly improve the runtime and backtracks.

This suggests when two heuristics are both approximating the costs for minimax adversarial heuristic, a heuristic considering more constraints does not necessarily work better than the other one. It is problem/benchmark dependent.

V. Concluding Remarks

In this paper, we study the effects of minimax adversarial heuristic, which orders values of variables according to the semantic of quantifiers, in solving ultra-weak solutions for MWCSPs. The heuristic selects values by viewing MWCSPs as a two-player zero-sum game based on minimax heuristics from adversarial search. To implement the heuristic for efficient solving in practice, we propose and define three heuristic variants: HUnary, HBinary, and HFullBinary. We further give sufficient conditions to show when these heuristics will be equivalent to minimax adversarial heuristic. Experiments on three benchmarks comparing the effects of the three heuristic variants with different consistencies are shown. Other future work on MWCSPs includes: consistencies for (high arity) soft global constraints [14], algorithms for tackling stronger solution concepts, and distributed versions of MWCSPs.

References

[1] J. V. Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[2] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic Game Theory*. Cambridge University Press, 2007.

[3] L. V. Allis, "Searching for solutions in games and artificial intelligence," Ph.D. dissertation, University of Limburg, 1994.

[4] L. Bordeaux, M. Cadoli, and T. Mancini, "CSP properties for quantified constraints: Definitions and complexity," in *AAAI'05*, 2005, pp. 360–365.

[5] L. Bordeaux and E. Monfroy, "Beyond NP: Arc-consistency for quantified constraints," in *CP'02*, 2002, pp. 371–386.

[6] J. H. M. Lee, T. W. K. Mak, and J. Yip, "Weighted constraint satisfaction problems with min-max quantifiers," in *ICTAI '11*, 2011, pp. 769–776.

[7] A. Lallouet, J. H. M. Lee, and T. W. K. Mak, "Consistencies for ultra-weak solutions in minimax weighted csps using the duality principle," in *CP'12 (to appear)*, 2012.

[8] J. Larrosa and T. Schiex, "Solving weighted CSP by maintaining arc consistency," *Artificial Intelligence*, vol. 159, no. 1-2, pp. 1–26, 2004.

[9] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[10] J. Larrosa and T. Schiex, "In the quest of the best form of local consistency for weighted CSP," in *IJCAI'03*, 2003, pp. 239–244.

[11] S. de Givry, M. Zytnicki, F. Heras, and J. Larrosa, "Existential arc consistency: Getting closer to full arc consistency in weighted CSPs," in *IJCAI'05*, 2005, pp. 84–89.

[12] D. Stynes and K. N. Brown, "Value ordering for quantified CSPs," *Constraints*, vol. 14, no. 1, pp. 16–37, 2009.

[13] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. Warners, "Radio link frequency assignment," *CONSTRAINTS*, vol. 4, pp. 79–89, 1999.

[14] J. H. M. Lee and K. L. Leung, "Consistency techniques for flow-based projection-safe global cost functions in weighted constraint satisfaction," *JAIR*, vol. 43, pp. 257–292, 2012.