Consistencies for Ultra-Weak Solutions in Minimax Weighted CSPs Using the Duality Principle

Arnaud Lallouet, Jimmy H.M. Lee, Terrence W. Mak

Consistencies for Ultra-Weak Solutions in Minimax Weighted CSPs Using the Duality Principle *

Arnaud Lallouet[1], Jimmy H.M. Lee[2], and Terrence W.K. Mak[2]

[1] Université de Caen, GREYC, Campus Côte de Nacre,
Boulevard du Maréchal Juin, BP 5186, 14032 Caen Cedex, France
Arnaud.Lallouet@unicaen.fr
[2] 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 QWCSPs) are a framework for modeling soft constrained problems with adversarial conditions. In this paper, we describe novel definitions and implementations of node, arc and full directional arc consistency notions to help reduce search space on top of the basic tree search with alpha-beta pruning for solving ultra-weak solutions. In particular, these consistencies approximate the lower and upper bounds of the cost of a problem by exploiting the semantics of the quantifiers and reusing techniques from both Weighted and Quantified CSPs. Lower bound computation employs standard estimation of costs in the sub-problems used in alpha-beta search. In estimating upper bounds, we propose two approaches based on the Duality Principle: duality of quantifiers and duality of constraints. The first duality amounts to changing quantifiers from $\min$ to $\max$, while the second duality re-uses the lower bound approximation functions on dual constraints to generate upper bounds. Experiments on three benchmarks comparing basic alpha-beta pruning and the six consistencies from the two dualities are performed to confirm the feasibility and efficiency of our proposal.

Keywords: constraint optimization, soft constraint satisfaction, minimax game search, consistency algorithms

1 Introduction

The task at hand is that of a constraint optimization problem with *adversaries* controlling parts of the variables. As an example, we begin with a generalized version of the Radio Link Frequency Assignment Problem (RLFAP) [7] consisting of assigning frequencies to a set of radio links located between pairs of sites, with the goal of preventing interferences. The problem has two types of constraints. One type prevents radio links that are close together from interfering with one another, by restricting the links not to take frequencies with absolute differences smaller than a threshold. In practice, the

* 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.

threshold is measured depending on the physical environment, and is often overestimated. The second type of constraints are technological constraints, where each constraint ensures the distance between frequencies of a radio link from site A to B and its reverse radio link from site B to A must be equal to a constant. If the problem is unsatisfiable, one approach is to find assignments violating the first type of constraints as little as possible. Suppose now a certain set of links are placed in unsecured areas, and *adversaries* (e.g. 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. One interesting question for this type of scenarios is to find frequency assignments such that we can minimize the degree of radio links affected for the worst possible case (i.e. finding the best-worst case). The prime goal is to understand how well we can defend against the worst adversaries for planning purposes.

The example is optimization in nature, and the adversaries originate from the uncontrollable frequencies being assigned on the links in unsecured areas. The question can be translated to minimizing the interferences for all possible combinations of frequency adjustments the adversaries can control. One way to solve this problem is by tackling many COPs [2]/Weighted CSPs [15], where each of them minimizes the interferences conditioned on a specific combination of frequency adjustments controlled by the adversaries. Another way is to model the problem as a Quantified CSP [15] by finding whether there exists combinations of frequency adjustments for us for all frequency placements by the adversaries such that the total interferences is less than a cost k. To avoid solving multiple sub-problems, Minimax Weighted Constraint Satisfaction Problems (MWCSPs) (previously called Quantified Weighted Constraint Satisfaction Problems) [16] are proposed to tackle such problems, combining quantifier structures from Quantified CSPs to model the adversaries and soft constraints from Weighted CSPs to model costs information. Previous work defines a solution as a complete assignment representing the best-worst case, gives an introduction on how to adopt alpha-beta prunings to tackle the problem in branch and bound, and suggests two sufficient pruning conditions to achieve prunings and backtrackings.

When tackling game problems, more specifically two-person zero-sum games with perfect information [22, 23], games can be solved at different levels. Allis [1, 13] proposes three solving levels for games: *ultra-weakly solved*, *weakly solved*, and *strongly solved*. Ultra-weakly solved means the game-theoretic value of the initial position has been determined, which means we can determine the outcome of the scenario when both players are playing perfectly (i.e. best-worst case). Weakly solved means a strategy, noted as winning strategy [4] in QCSPs, has been determined for the initial position to achieve the game-theoretic value against any opposition. Strongly solved is being used for a game for which such a strategy has been determined for all legal positions. 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 bi-level programs, there are cases in which we can assume there is a unique optimum for the follower or we are concerned with only the moves for the leader [11]. Finding ultra-weak solutions for these cases are sufficient, and the generalized RLFAP is an example. In adversarial game

playing, many game search algorithms, e.g. minimax and alpha-beta [24], computes strategies assuming optimal plays to reduce computation costs. In fact, even determining just the ultra-weak solution in an offline manner is also an important and interesting line of research, e.g. a recent breakthrough on checkers [25].

The main focus of this paper is to further introduce novel consistency notions for solving ultra-weak solutions, by approximating the lower and upper bounds of the cost of the problem. Lower bound computation employs standard estimation of costs in the sub-problems used in alpha-beta search. In estimating upper bounds, we adopt the Principle of Duality in (integer) linear programming, which suggest to convert an original (primal) problem to its dual form and tackle the problem using both forms. We consider two dualities: duality of quantifiers and duality of constraints. The first approach allows us to formulate upper bound approximation functions by changing quantifiers in the lower bound functions from $\min$ to $\max$, while the second approach re-uses the lower bound approximation functions on dual constraints to generate upper bounds. Discussions on whether our proposed techniques are applicable to the computation of the two stronger solutions will be given. Experimental evaluations on three benchmarks are performed to compare six consistencies defined using the two dualities to confirm the feasibility and efficiency of our proposal.

2 Background

In the first part, we give definitions and semantics of MWCSPs, followed by an example. In the second part, sufficient conditions allowing us to perform backtracking/prunings used in alpha-beta search are highlighted.

2.1 Definitions and Semantics

A *Minimax Weighted Constraint Satisfaction Problem* (MWCSP) [16] $\mathcal{P}$ 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*, 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:

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

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 $\text{A-cost}(\mathcal{P}) < k$.

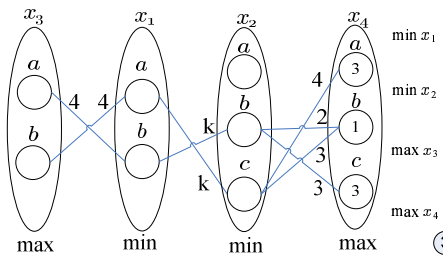

Fig. 1. Constraints for Example 1

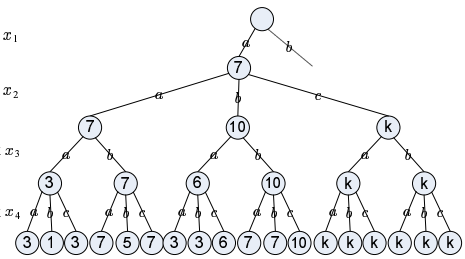

Fig. 2. Labeling Tree for Example 1

We now define three solution concepts for MWCSPs based on the definition of A-costs. An *ultra-weak solution* of an MWCSP $\mathcal{P}$ is a complete assignment $\{x_1 = v_1, \ldots, x_n = v_n\}$ s.t. $\text{A-cost}(\mathcal{P}) = \text{A-cost}(\mathcal{P}[x_1 = v_1]\ldots[x_i = v_i]), \forall 1 \leq i \leq n$. Solving an ultra-weak solution corresponds to finding the scenario when both players are playing perfectly. To capture weak (strong resp.) solutions, we re-use the concept of winning strategies [4]. Without loss of generality, we assume the $\max$ player is the adversary. A *weak solution* (*strong solution* 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

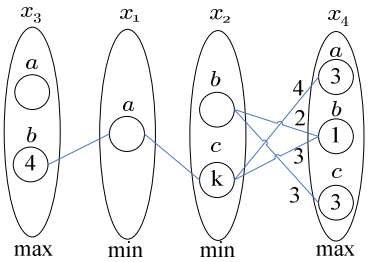

Fig. 3. Constraints for Example 2

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*. Note that computing ultra-weak solutions essentially computes the A-costs of an MWCSP, which are defined based on constraints and quantifiers, and in general, computing the A-costs of an MWCSP is PSPACE-hard [16]. A special case is that if all the quantifiers of an MWCSP are $\min$ quantifiers, finding an ultra-weak solution is equivalent to finding a complete assignment l with the minimum costs (i.e. $\text{argmin}_l\, cost(l)$). The problem reduces [16] to a Weighted CSP.

Example 1. We use the generalized Radio Link Frequency Assignment Problem introduced in the previous section as an example. The problem consists of four links $l_1, l_2, l_3,$ and l_4. Two of the links l_1 and l_2 connect sites A and B, and the other two links l_3 and l_4 connect sites B and C. Link l_2 (l_4 resp.) is the reverse link for l_1 (l_3 resp.). There is a variable x_i in the MWCSP $\mathcal{P}$ for each link l_i, which is used to represent the chosen frequency for link l_i. Site C is not secure and links l_3 and l_4 are subject to control. We need to pay costs if two links interfere with each other. Therefore, we want to find frequency assignments for l_1 and l_2 such that we can minimize the total costs for interference in the worst case. We set the quantifier sequence in $\mathcal{P}$ as $(Q_1 = \min, Q_2 = \min, Q_3 = \max, Q_4 = \max)$. For simplicity, we assume links l_1 and l_3 have two frequency choices, and the other two links have three. We measure the costs for interference only for links l_1 and l_3, and links l_2 and l_4. These costs will be modeled by constraints on variables x_1 and x_3, and also on variables x_2 and x_4. In addition, we maintain the technological constraint between links l_1 and l_2, which will be modeled by a binary constraint on variables x_1 and x_2. Figure 1 indicates there is one unary constraint C_4 and three binary constraints $C_{1,2}, C_{1,3}$, and $C_{2,4}$. For the unary constraint, non-zero unary costs are depicted inside a circle and domain values are placed above the circle. For binary constraints, non-zero binary costs are depicted as labels on edges connecting the corresponding pair of values. Only non-zero costs are shown. We set the global upper bound k to be 11. By following the partial labeling tree in Figure 2, we can easily infer the A-cost of the subproblem $\mathcal{P}' = \mathcal{P}[x_1 = a]$ is 7, and $\{x_1 = a, x_2 = a, x_3 = b, x_4 = a\}$ is one of the ultra-weak solutions for $\mathcal{P}'$.

2.2 Pruning Conditions in B & B

MWCSPs can be solved by applying alpha-beta pruning in branch and bound search [16] (Figure 4), 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 6 and 8). If $lb \geq ub$, then one of the previous branch must dominate over the current sub-tree, and we can perform backtrack (line 9).

Lee, Mak, and Yip [16] give pruning conditions that allow further derivation of consistency notions, and we introduce them as follows. Let $\mathcal{P}[x_{1..i-1} = v_{1..i-1}, x_i =$

```
function alpha_beta(P,lb,ub):
  if firstx(P) == ⊥: return cost(P)
  i = firstx(P)
  for v in Di:
    if Qi == min:
      ub = min(ub, alpha_beta(P[Xi=v],lb,ub))
    else:
      lb = max(lb, alpha_beta(P[Xi=v],lb,ub))
    if ub <= lb: break
  return (Qi == min)?ub:lb
```

Fig. 4. Alpha-beta for MWCSPs

$v]$ denote the subproblem $\mathcal{P}[x_1 = v_1][x_2 = v_2]...[x_{i-1} = v_{i-1}][x_i = v]$. Formally, we consider two conditions: $\exists v \in D_i \;\; s.t. \;\; \forall v_1 \in D_1, ..., v_{i-1} \in D_{i-1}$:

$$\text{A-cost}(\mathcal{P}[x_{1..i-1} = v_{1..i-1}, x_i = v]) \geq ub \tag{1}$$

$$\text{A-cost}(\mathcal{P}[x_{1..i-1} = v_{1..i-1}, x_i = v]) \leq lb \tag{2}$$

where ub and lb are the upper and lower bounds in alpha-beta prunings respectively. When either of the above conditions is satisfied, we can apply prunings according to Table 1.

Table 1. When can we prune/backtrack

A-cost	$\geq ub$	$\leq lb$
$Q_i = \min$	prune v	backtrack
$Q_i = \max$	backtrack	prune v

Checking Condition (1)/(2) by finding the *exact* value of the A-cost for each sub-problem is computationally expensive. Alternatively, we allow approximating functions to perform bounds approximations. Function $\text{ubaf}(\mathcal{P}, x_i = v)$ ($\text{lbaf}(\mathcal{P}, x_i = v)$ resp.) is an upper bound (a lower bound resp.) approximation function if it approximates the A-cost for the set S of sub-problems, where:

$$\begin{aligned} S = \{&\mathcal{P}[x_{1..i-1} = v_{1..i-1}, x_i = v] | \forall v_1 \in D_1, \ldots, v_{i-1} \in D_{i-1}\} \\ \text{s.t.} \;\; &\forall \mathcal{P}' \in S, \text{A-cost}(\mathcal{P}') \leq \text{ubaf}(\mathcal{P}, x_i = v) \\ &\qquad\qquad (\geq \text{lbaf}(\mathcal{P}, x_i = v) \text{ resp. }) \end{aligned}$$

From the definition, we can easily obtain:

$$\text{lbaf}(\mathcal{P}, x_i = v) \geq ub \implies \text{Condition (1)}$$

$$\text{ubaf}(\mathcal{P}, x_i = v) \leq lb \implies \text{Condition (2)}$$

By implementing $\text{lbaf}()/\text{ubaf}()$ with good approximations, we can identify non-ultra-weak solution values from variable domains or perform backtracking earlier in search according to Table 1.

3 Consistency Techniques

In Weighted CSPs, consistency notions [15, 9] not only utilize constraint semantics, but also take the costs of constraints into account. This section discusses how we utilize costs information from unary constraints and binary constraints to formulate node and (full directional) arc consistencies. We start by giving an $lbaf()$ for node consistency

called $nc_{lb}()$, which formulates lower bounds by gathering unary costs. We then further describe a stronger $lbaf()$ for (full directional) arc consistency called $ac_{lb}()$. To approximate upper bounds, we propose two approaches by utilizing the Duality Principle: duality of quantifiers and duality of constraints. In the last part, we discuss how to strengthen our consistency notions, by incorporating techniques in Weighted CSPs. 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. To simplify our notations, we write the minimum costs $\min_{u \in D_j} C_j(u)$ and maximum costs $\max_{u \in D_j} C_j(u)$ of a unary constraint C_j as $\min C_j$ and $\max C_j$ respectively. We further write $Q_j C_j$ to mean $\min C_j$ if $Q_j = \min$, and $\max C_j$ if $Q_j = \max$.

3.1 Node Consistency: Lower Bound

We first give the definition for $nc_{lb}()$. We will then sketch the proof showing $nc_{lb}()$ is an $\mathrm{lbaf}()$ using a lemma. Without loss of generality, we now consider unary MWCSPs, which are MWCSPs with *unary constraints only*. We will show that computing $\mathrm{A\text{-}costs}$ for any sub-problems of unary MWCSPs are efficient (linear time), and therefore, computing the lower bound for these sub-problems are efficient. We then show using the same procedure on general MWCSPs, by viewing unary constraints only, the bound is still correct.

Definition 1. *The $nc_{lb}(\mathcal{P}, x_i = v)$ function approximates the $\mathrm{A\text{-}cost}$ for a set S of sub-problems $\{\mathcal{P}[x_{1..i-1} = v_{1..i-1}, x_i = v] | \forall v_1 \in D_1, \ldots, v_{i-1} \in D_{i-1}\}$. Define*

$$nc_{lb}(\mathcal{P}, x_i = v) \equiv C_{\varnothing} \oplus (\bigoplus_{j:j<i} \min C_j) \oplus (C_i(v)) \oplus (\bigoplus_{j:i<j} Q_j C_j)$$

where $Q_j \in \mathcal{Q}$ is the quantifier for variable x_j where $j > i$.

Lemma 1. *The $\mathrm{A\text{-}cost}$ of an MWCSP $\mathcal{P}$ with only unary constraints is equal to $\bigoplus_{i=1}^{n} Q_i C_i$.*

The proof of Lemma 1 follows directly from the definition of $\mathrm{A\text{-}costs}$ for MWCSPs.

Theorem 1. *The function $nc_{lb}(\mathcal{P}, x_i = v)$ is a lower bound approximating function $\mathrm{lbaf}(\mathcal{P}, x_i = v)$.*

Lemma 1 suggests the computation of $\mathrm{A\text{-}costs}$ for unary MWCSPs can be done in O(nd), where n is the number of variables and d is the maximum domain size. Therefore, computing the $\mathrm{A\text{-}costs}$ for any sub-problems is also efficient. The function $nc_{lb}()$ can be seen as a function extracting $\mathrm{A\text{-}costs}$ for the sub-problem in S with minimal $\mathrm{A\text{-}costs}$ following Lemma 1, by partitioning unary constraints into three groups: (a) $C_j, j < i$, (b) C_i, and (c) $C_j, j > i$. We skip the detailed reasoning on how to choose costs for these unary constraints. If $\mathcal{P}$ has only unary constraints, we can observe function $nc_{lb}()$ computes not only a correct lower bound for S, but also the exact $\mathrm{A\text{-}cost}$ for the sub-problem with minimum costs. Note that MWCSPs may have binary constraints and even high-arity constraints, but, these constraints must give positive costs to the problem. Therefore, by considering only unary constraints of general MWCSPs, $nc_{lb}()$ still returns a correct lower bound.

Example 2. We re-use Example 1. Suppose we are at sub-problem $\mathcal{P}' = \mathcal{P}[x_1 = a]$ and we have just visited the further sub-problem $\mathcal{P}'[x_2 = a]$ which have a new upper bound of 7. Before visiting $\mathcal{P}'[x_2 = b]$, we try to prune some values according to Table 1 using the new upper bound. Figure 3 shows the constraint graph for $\mathcal{P}'$. Suppose now $nc_{lb}()$ is applied and no unary costs for bounded variables, i.e. $C_\varnothing = 0$. We want to check if the value b can be pruned from D_2. In the sub-problem $\mathcal{P}'[x_2 = b]$, the quantifier Q_3 and Q_4 are both $\max$, and they will take at least the maximum unary cost $\max C_3$ and $\max C_4$. We have $C_\varnothing + C_2(b) + \max C_3 + \max C_4 = 0 + 0 + 4 + 3 = 7 \geq ub$. The cost of any assignment in the sub-problem $\mathcal{P}'[x_2 = b]$ is at least 7. The value b can therefore be removed from domain D_2. Notice that such a node cannot be pruned by basic alpha-beta pruning.

3.2 Arc Consistency: Lower Bound

To obtain stronger lower bound, we further define function $ac_{lb}()$ based on $nc_{lb}()$. Without loss of generality, we restrict our attention to MWCSPs which have *only unary constraints and one binary constraint*. We will show that computing any sub-problems for these MWCSPs are efficient (polynomial time), and therefore, computing the lower bound for these sub-problems are again efficient. By similar argument, viewing unary constraints plus one binary constraint on general MWCSPs, the bound is still correct.

Definition 2. *The $ac_{lb}[C_{i,j}](\mathcal{P}, x_i = v)$ function approximates the* A-cost *for the set S of sub-problems $\{\mathcal{P}[x_{1..i-1} = v_{1..i-1}, x_i = v] | \forall v_1 \in D_1, \ldots, v_{i-1} \in D_{i-1}\}$. Define*

$$ac_{lb}[C_{i,j}](\mathcal{P}, x_i = v) \equiv C_\varnothing \oplus (\bigoplus_{k:k<i} \min C_k) \oplus (C_i(v)) \\ \oplus (\bigoplus_{k:i<k \wedge j \neq k} Q_k C_k) \oplus (\underset{u \in D_j}{Q_j} \{C_j(u) \oplus C_{i,j}(v,u)\})$$

where $Q_j \in \mathcal{Q}$ is the quantifier for variable x_j, and $Q_k \in Q$ is the quantifier for variable x_k where $k > i$ and $k \neq j$.

The first three terms are the same as in $nc_{lb}()$. The fourth term is equivalent to the last term in $nc_{lb}()$, except we do not consider costs for constraint C_j, which will be considered in the fifth term.

Lemma 2. *The* A-cost *of an MWCSP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{Q}, k)$ with only unary constraints and **one** binary constraint $C_{i,j}$ is equal to*

$$\bigoplus_{k \in [1 \ldots n] \setminus \{i,j\}} \underset{u \in D_k}{Q_k} C_k(u) \oplus \underset{u \in D_i}{Q_i} [\underset{v \in D_j}{Q_j} [C_i(u) \oplus C_j(v) \oplus C_{i,j}(u,v)]]$$

where $Q_i, Q_j, Q_k \in \mathcal{Q}$.

The proof of Lemma 2 follows from the definition of A-costs. Theorem 2 follows.

Theorem 2. *The function $ac_{lb}[C_{i,j}](\mathcal{P}, x_i = v)$ for binary constraint $C_{i,j}$ is a lower bound approximating function* $\text{lbaf}(\mathcal{P}, x_i = v)$.

Note that Definition 2 is only one possible approach to define a lower bound approximation function for AC, following Lemma 2. It is designed in such a way that only **one** binary constraint is used in bounds calculation for costs estimation, and our approach is similar to AC in QCSPs [20, 12]. It is natural for us to further ask for stronger/tighter functions which consider more than one binary constraint. Note that in classical local consistency enforcement such as: AC in CSPs [2]; AC* in WCSPs [15]; and (Q)AC [20] in QCSPs, we usually handle one (binary) constraint at a time. Consistency enforcement will be performed many times at each node of the search tree, and considering multiple constraints at a time may cause a huge increase in time complexity. We have to maintain a balance between amount of reasoning at each search node and amount of pruning achieved. There are stronger consistency notions with efficient algorithms which consider more than one binary constraint, e.g. Max Restriced Path Consistency [10] in CSPs and OSAC [8] in WCSPs/VCSPs. Investigations on stronger notions for MWCSPs is an interesting future work. One possibility to enhance ac_{lb} is to consider a subset of constraints that form a tree and employ a dynamic programming approach to compute such notions.

3.3 NC & AC Upper Bounds by the Duality Principle

In linear programming, duality [21, 27] provides a standard way to obtain lower bounds (for minimization problems). In fact, the Principle/Theory of Duality [21] suggests that we can convert the original (primal) problem to its dual form, and tackle the problem by using both forms. In QCSPs, dual consistency [5] was defined by creating the dual QCSP problem, involving negation of the original constraints. We will now show how to implement upper bound approximation functions $nc_{ub}()$ and $ac_{ub}()$ by using the duality principle in MWCSPs.

Duality of Constraints One approach to create $nc_{ub}()/ac_{ub}()$ is to utilize the constraint duality property, which is similar to dual consistency [5] in QCSPs. We first define the dual problem of an MWCSP.

Definition 3. *Given an MWCSP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{Q}, k)$. The dual problem of $\mathcal{P}$ is an MWCSP $\mathcal{P}^\dagger = (\mathcal{X}, \mathcal{D}, \mathcal{C}^\dagger, \mathcal{Q}^\dagger, k)$ s.t. for a complete assignment l,*

$$C_\varnothing \oplus \bigoplus_{C_S \in \mathcal{C}} C_S(l[S]) = -1 \times (C^\dagger_\varnothing \oplus \bigoplus_{C^\dagger_S \in \mathcal{C}^\dagger} C^\dagger_S(l[S]))$$

where the valuation structure of $\mathcal{P}^\dagger$ is $([-k...k], \oplus, \leq)$, $Q^\dagger_i = \min$ if $Q_i = \max$, and $Q^\dagger_i = \max$ if $Q_i = \min$.

We can observe that $\text{A-cost}(\mathcal{P}) = -1 \times \text{A-cost}(\mathcal{P}^\dagger)$, and a straightforward method to construct the **dual constraints** in the dual problem is to multiply costs for all constraints in the original problem by -1. We then show how we utilize the dual problem to check $\text{ubaf}(\mathcal{P}, x_i = v) \leq lb$ (Condition 2) for an MWCSP $\mathcal{P}$.

Theorem 3. *Given an MWCSP $\mathcal{P}$ and its dual problem $\mathcal{P}^\dagger$. Suppose there is a lower bound approximation function* $\text{lbaf}()$.

$$\text{lbaf}(\mathcal{P}^\dagger, x_i = v) \geq -1 \times lb \implies \textit{Condition (2).}$$

The proof of Theorem 3 can be shown by observing the labeling tree of the dual problem, and inferring $\mathrm{lbaf}(\mathcal{P}^\dagger, x_i = v) \times -1$ is an upper bound approximation function for the original problem. In fact, the upper bound $ub^\dagger$ (lower bound $lb^\dagger$ resp.) of $\mathcal{P}^\dagger$ is equal to -1 times the lower bound lb (upper bound ub resp.) of $\mathcal{P}$. Therefore, we further define $ub^\dagger = -1 \times lb$, and $lb^\dagger = -1 \times ub$. We then implement $nc_{ub}()$ and $ac_{ub}()$, via checking the $nc_{lb}()$ and $ac_{lb}()$ for the dual problem.

Definition 4. *An MWCSP $\mathcal{P}$ is dual constraint node consistent (DC-NC) iff $\forall x_i \in \mathcal{X}, \forall v \in D_i : nc_{lb}(\mathcal{P}, x_i = v) < ub \wedge nc_{lb}(\mathcal{P}^\dagger, x_i = v) < ub^\dagger$.*

Definition 5. *An MWCSP $\mathcal{P}$ is dual constraint arc consistent (DC-AC) iff $\mathcal{P}$ is DC-NC, $\forall C_{i,j} \in \mathcal{C}, \forall v \in D_i : ac_{lb}[C_{i,j}](\mathcal{P}, x_i = v) < ub$, and $\forall C^\dagger_{ij} \in \mathcal{C}^\dagger, \forall v \in D_i : ac_{lb}[C^\dagger_{ij}](\mathcal{P}^\dagger, x_i = v) < ub^\dagger$.*

Theorem 4. *DC-AC is strictly stronger than DC-NC.*

The proof follows from the definitions.

Duality of Quantifiers Another way to check condition (2) for an MWCSP $\mathcal{P}$ is to scrutinize functions implementing $\mathrm{ubaf}(\mathcal{P}, x_i = v)$, by repeating similar reasonings for $nc_{lb}()$ on unary MWCSPs (plus a binary constraint). The idea is to use the duality of quantifiers, by replacing $\min$ quantifiers to $\max$ in the reasoning process. Recall we have three groups of unary constraints to consider. One direct way is to consider the maximum costs, instead of minimum costs from constraints in the first group (group (a)), hence changing quantifiers from $\min$ to $\max$. However, using the resulting upper bound approximation functions, by reasoning on unary MWCSPs is incorrect for general MWCSPs. We cannot neglect costs given by high arity constraints. One way to make the bound correct is to add the maximum costs for constraints which will not be covered in the function, and we pre-compute these costs before search. Function $nc_{ub}(\mathcal{P}, x_i = v)$ and $ac_{ub}(\mathcal{P}, x_i = v)$ are given as follows, and we write $\max C^\star$ to mean the maximum costs for constraints which are not considered in the function.

Definition 6. *The $nc_{ub}(\mathcal{P}, x_i = v)$ function approximates the* A-cost *for a set S of sub-problems $\{P[x_{1..i-1} = v_{1..i-1}, x_i = v] | \forall v_1 \in D_1, v_2 \in D_2, \ldots, v_{i-1} \in D_{i-1}\}$. Define:*

$$nc_{ub}(\mathcal{P}, x_i = v) \equiv C_\varnothing \oplus (\bigoplus_{j:j<i} \max C_j) \oplus (C_i(v)) \oplus (\bigoplus_{j:i<j} Q_j C_j) \oplus (\max C^\star)$$

where $Q_j \in \mathcal{Q}$ is the quantifier for x_j, $j > i$.

We can easily observe $\max C^\star$ is equal to $\bigoplus_{j,k:j\neq k} \max C_{jk}$ if there are only unary and binary constraints.

Definition 7. *The function $ac_{ub}[C_{i,j}](\mathcal{P}, x_i = v)$ approximates the* A-cost *for the set S of sub-problems: $\{P[x_{1..i-1} = v_{1..i-1}, x_i = v] | \forall v_1 \in D_1, v_2 \in D_2, \ldots, v_{i-1} \in D_{i-1}\}$. Define:*

$$ac_{ub}[C_{i,j}](\mathcal{P}, x_i = v) \equiv C_\varnothing \oplus (\bigoplus_{j:j<i} \max C_j) \oplus (C_i(v)) \oplus (\bigoplus_{k:i<k\wedge j\neq k} Q_k C_k)$$
$$\oplus Q_{j\,u\in D_j} \{C_j(u) \oplus C_{i,j}(v,u)\} \oplus (\max C^\star)$$

where Q_k is the quantifier for variable x_k where $k > i$ and $k \neq j$, and Q_j is the quantifier for variable x_j.

If there are only unary and binary constraints, $\max C^\star$ is equal to $\bigoplus_{C_{k,l} \in B} \max C_{k,l}$, where $B = \{C_{k,l} \in \mathcal{C} | k \neq l\} - \{C_{i,j}\}$. We now define the node and arc consistencies by utilizing the constructed functions.

Definition 8. *An MWCSP $\mathcal{P}$ is dual quantifier node consistent (DQ-NC) iff $\forall x_i \in \mathcal{X}, \forall v \in D_i : nc_{lb}(\mathcal{P}, x_i = v) < ub \wedge nc_{ub}(\mathcal{P}, x_i = v) > lb$.*

Definition 9. *An MWCSP $\mathcal{P}$ is dual quantifier arc consistent (DQ-AC) iff $\mathcal{P}$ is DQ-NC, and $\forall C_{i,j} \in \mathcal{C}, \forall v \in D_i : ac_{lb}[C_{i,j}](\mathcal{P}, x_i = v) < ub \wedge ac_{ub}[C_{i,j}](\mathcal{P}, x_i = v) > lb$.*

Theorem 5. *DQ-AC is strictly stronger than DQ-NC.*

The proof follows from the definitions.

3.4 Consistency Enforcement

To enforce DC-NC and DC-AC, one major step is to compute $nc_{lb}()$ and $ac_{lb}()$, by computing costs from unary and binary constraints in both the original and dual MWCSPs. For DQ-NC and DQ-AC, we compute $nc_{ub}()$ and $ac_{ub}()$ instead of the dual. To achieve these consistencies, we perform prunings/backtrackings according to Table 1. Similar to cascade propagation [2] in CSPs, a value of a variable being pruned may trigger prunings of other values in other variables and re-computation of the $\mathrm{lbaf}()$ and $\mathrm{ubaf}()$ functions. In addition, prunings caused by lower bound approximations may tighten upper bound approximations (and vice versa), and triggers extra prunings. Our propagation routine repeats until no values can be further pruned, or backtracks occur.

3.5 Strengthening Consistencies by Projection/Extension

Consistency algorithms for WCSPs use an equivalence preserving transformation called *projection* [9] to move costs from higher arity constraints to lower arity ones to extract and store bound information. Some further utilizes *extension* [9], which is the inverse of projection, to increase the consistency strength. We propose to re-use WCSP consistencies, especially the parts related to projections and extensions, to strengthen the approximating functions for MWCSPs.

WCSPs consistencies consist of two kinds of conditions: one for pruning and one for projection/extension. Since their pruning conditions are unsound w.r.t. MWCSPs, we adopt only their projection/extension conditions so as to strengthen DC-NC, DC-AC, DQ-NC, and DQ-AC. The projection/extension conditions for NC*, AC*, and FDAC* [15, 14] are as follows:

$$\begin{aligned}
\text{proj-NC*} :& \forall C_i, \exists v \in D_i : C_i(v) = 0 \\
\text{proj-AC*} :& \text{ proj-NC* } \wedge \forall C_{i,j}, \forall v_i \in D_i, \exists v_j \in D_j : C_{i,j}(v_i, v_j) = 0 \wedge \\
& \forall C_{i,j}, \forall v_j \in D_j, \exists v_i \in D_i : C_{i,j}(v_i, v_j) = 0 \\
\text{proj-FDAC*} :& \text{ proj-AC* } \wedge \forall C_{i,j} : i < j, \forall v_i \in D_i, \exists v_j \in D_j : C_{i,j}(v_i, v_j) \oplus C_j(v_j) = 0
\end{aligned}$$

Note that the enforcing algorithm for proj-FDAC* may decreases unary costs for $\max$ variables and increases unary costs for $\min$ variables; hence weakening the approximating functions. We tackle this issue by re-ordering the variables when enforcing proj-FDAC*, with $\max$ variables first. To further enforce these projecting conditions on the dual problem in DC-NC/DC-AC, we need to perform normalization, by transferring costs from $C_\varnothing$ to constraints with negative costs until all constraints except $C_\varnothing$ return non-negative costs. We now re-define DC-NC, DC-AC, DQ-NC, and DQ-AC, to allow users plugging in general projection/extension conditions τ.

Definition 10. *An MWCSP $\mathcal{P}$ is DC-NC[τ] (DC-AC[τ] resp.) iff $\mathcal{P}$ is DC-NC (DC-AC resp.), and all projection/extension conditions τ for both $\mathcal{P}$ and the dual problem $\mathcal{P}^\dagger$ are satisfied. An MWCSP $\mathcal{P}$ is DQ-NC[τ] (DQ-AC[τ] resp.) iff $\mathcal{P}$ is DQ-NC (DQ-AC resp.), and all the projection/extension conditions τ for $\mathcal{P}$ are satisfied.*

Previous work [16] shows experimental results on an implementation of DQ-NC[proj-NC*] and DQ-AC[proj-AC*], where DQ-NC[proj-NC*] and DQ-AC[proj-AC*] are named as node and arc consistency respectively.

3.6 Tackling Stronger Solution Definitions

This section discusses the scopes and limitations of our techniques on solving MWCSPs for the other two stronger solved levels: weakly solved and strongly solved.

In terms of space, the solution sizes for solving MWCSPs ultra-weakly, weakly, and strongly vary from $O(n)$, $O((n-m)d^m)$, to $O(d^n)$ respectively, where n is the total number of variables, $m \leq n$ is the number of variables owned by adversaries, and d is the maximum domain size of the MWCSP. A direct consequence is that we need exponential space to store weak/strong solutions during search, and most often, compact representations to represent weak/strong solutions are more desirable.

In terms of prunings in branch and bound tree search, a sound pruning condition when solving a weaker solution concept may not hold in stronger ones. This is caused by the removal of the assumption of optimal/perfect plays when dealing with stronger solution concepts. For example in alpha-beta prunings, when the $\min$ player obtains an A-costs which is lower than the lb (i.e. $\max$ player's last found best), we cannot immediately backtrack if we want to tackle weakly solved solutions, where we assume the $\max$ player is the adversary. The reason behind is that we cannot assume the $\max$ player must play a perfect move. We have to consider all moves for the $\max$ player. The situation is similar if we assume the $\min$ player is the adversary. By similar reasonings and inductions, we cannot perform prunings/backtrackings for the $\leq lb$ column ($\geq ub$ column resp.) in Table 1 if we want to tackle weakly solved solutions, assuming the $\max$ player ($\min$ player resp.) is the adversary. For solving strong solutions, the situation is even worse. We cannot assume optimal plays for both players. Therefore, we have to find A-costs for all sub-problems, and all prunings/backtrackings conditions in Table 1 cannot be used. In general, the fewer sound pruning/backtracking conditions available, the larger search space we have to search. By using tree search, we can observe finding stronger solutions is much harder than weaker ones.

When tackling real-life problems, one can ask for solutions which solve the problem in an intermediate level. For example, if the adversaries have multiple optimal strategies, we can require solutions containing responses to every different optimal choice

the adversaries may choose. In this case, the solved level lies between ultra-weak and weak. One way to handle is to relax the bound updating procedure for the lower bound (upper bound resp.) in alpha-beta pruning (Line 6 and 8 in Figure 4), where we assume the $\max$ ($\min$ resp.) player is the adversary. When a larger lower bound lb (smaller upper bound ub resp.) is found, we update the lower bound to $lb - 1$ (upper bound to $ub + 1$ resp.). The major focus of this work is to give consistency notions to improve the search in finding the best-worst case, i.e. ultra-weak solutions, of a game.

4 Performance Evaluation

In this section, we compare our solver in seven modes: Alpha-beta pruning, DC-NC[proj-NC*], DQ-NC[proj-NC*], DC-AC[proj-AC*], DQ-AC[proj-AC*], DC-AC[proj-FDAC*], and DQ-AC[proj-FDAC*]. Values are labeled in static lexicographic order. 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 instances fail to run within the time limit. The experiment is conducted on a Core2 Duo 2.8GHz with 3.2GB memory. We have also performed experiments on QeCode, a solver for QCOPs [3], by transforming the instances to QCOPs according to the transformation in previous work [16].

4.1 Randomly Generated Problems and Graph Coloring Games

We re-use benchmark MWCSP instances and graph coloring game instances by Lee, Mak, and Yip [16]. 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. For the graph coloring game instances, numbers are used instead of colors, and the graph is numbered by two players. We partition the nodes into two

Table 2. Randomly Generated Problem

	Alpha-beta		DC-NC[proj-NC*]		DC-AC[proj-AC*]		DC-AC[proj-FDAC*]	
(n, d, p)	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes
(12, 5, 0.4)	68.20	5,967,461	5.89	131,468	2.54	30,165	**2.13**	**20,397**
(12, 5, 0.6)	52.05	4,782,541	4.63	101,690	2.61	26,093	**2.24**	**16,178**
(14, 5, 0.4)	$263.04^{(18)}$	19,770,953	52.72	948,783	19.33	198,476	**14.82**	**117,155**
(14, 5, 0.6)	$271.72^{(17)}$	17,249,858	70.12	1,185,087	29.97	246,459	**23.11**	**143,197**
(16, 5, 0.4)	$517.24^{(2)}$	26,269,025	$332.65^{(19)}$	4,617,612	121.78	1,047,900	**102.82**	**706,913**
(16, 5, 0.6)	$693.31^{(2)}$	36,315,673	$461.68^{(16)}$	6,157,070	259.51	1,816,642	**208.52**	**1,054,326**
	QeCode		DQ-NC[proj-NC*]		DQ-AC[proj-AC*]		DQ-AC[proj-FDAC*]	
(n, d, p)	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes
(12, 5, 0.4)	-	–	3.68	158,179	3.23	53,845	4.27	58,619
(12, 5, 0.6)	-	-	2.85	118,401	3.24	41,596	4.17	45,698
(14, 5, 0.4)	-	-	33.39	1,135,378	26.20	369,185	41.74	482,053
(14, 5, 0.6)	-	-	46.81	1,510,946	45.85	450,407	68.63	522,715
(16, 5, 0.4)	-	-	217.13	5,780,075	141.07	1,654,538	173.96	1,745,527
(16, 5, 0.6)	-	-	$364.51^{(19)}$	9,401,844	341.71	3,071,036	$362.12^{(17)}$	2,659,294

Table 3. Graph Coloring Game

	Alpha-beta		DC-NC[proj-NC*]		DC-AC[proj-AC*]		DC-AC[proj-FDAC*]	
(v, c, d)	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes
(14, 4, 0.4)	19.88	1,572,978	6.71	122,266	3.20	37,252	**1.90**	**16,732**
(14, 4, 0.6)	24.12	1,730,473	10.38	185,111	5.88	59,359	**3.48**	**23,515**
(16, 4, 0.4)	167.75	10,050,800	48.37	688,200	22.67	221,484	**12.09**	**92,875**
(16, 4, 0.6)	166.83	9,213,029	45.71	625,944	27.03	212,934	**15.64**	**85,920**
(18, 4, 0.4)	$784.47^{(3)}$	33,914,968	288.90	2,839,962	114.63	792,220	**65.58**	**357,457**
(18, 4, 0.6)	-	-	350.29	3,400,265	163.70	993,099	**80.06**	**343,146**
	QeCode		DQ-NC[proj-NC*]		DQ-AC[proj-AC*]		DQ-AC[proj-FDAC*]	
(v, c, d)	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes
(14, 4, 0.4)	-	-	4.52	170,843	3.36	63,298	3.74	53,722
(14, 4, 0.6)	-	-	7.29	269,179	6.36	99,972	6.88	74,187
(16, 4, 0.4)	-	-	34.43	1,002,145	23.21	363,539	24.36	281,229
(16, 4, 0.6)	-	-	33.82	949,861	29.19	352,694	31.99	280,426
(18, 4, 0.4)	-	-	204.86	4,095,993	118.65	1,315,346	140.95	1,207,566
(18, 4, 0.6)	-	-	267.23	5,295,433	180.38	1,711,948	182.66	1,270,797

Table 4. Generalized Radio Link Frequency Assignment Problem

	Alpha-beta		DC-NC[proj-NC*]		DC-AC[proj-AC*]		DC-AC[proj-FDAC*]	
(i, n, d, r)	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes
(1, 24, 4, 0.2)	-	-	86.38	442,362	50.54	74,182	53.85	**55,988**
(0, 24, 4, 0.4)	-	-	148.87	828,286	105.95	295,743	128.01	**286,122**
(1, 22, 6, 0.2)	-	-	618.93	3,580,885	**307.58**	352,439	309.63	**299,361**
(0, 24, 6, 0.2)	-	-	$1230.33^{(19)}$	6,822,412	500.18	738,245	479.50	**651,762**
	QeCode		DQ-NC[proj-NC*]		DQ-AC[proj-AC*]		DQ-AC[proj-FDAC*]	
(i, n, d, r)	Time	#nodes	Time	#nodes	Time	#nodes	Time	#nodes
(1, 24, 4, 0.2)	-	-	**45.62**	449,164	50.75	77,286	47.08	62,734
(0, 24, 4, 0.4)	-	-	**96.55**	1,046,150	101.49	451,090	208.79	692,470
(1, 22, 6, 0.2)	-	-	338.42	3,719,348	374.34	374,385	309.96	368,643
(0, 24, 6, 0.2)	-	-	$682.60^{(19)}$	7,224,677	539.69	803,087	**434.99**	812,048

sets A and B. Player 1 (Player 2 resp.) will number set A (B resp.). The goal of player 1 is to maximize the total difference between numbers of adjacent nodes, while player 2 wishes to minimize. The aim is to help player 1 extracting the best-worst case. We generate instances 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 for both benchmarks are 900 seconds. Table 2 and 3 show the results.

4.2 Generalized Radio Link Frequency Assignment Problem (GRLFAP)

We generate the GRLFAP according to two small but hard CELAR sub-instances [7], which are extracted from CELAR6. All GRLFAP instances are generated with parameters (i, n, d, r), where i is the index of the CELAR sub-instances (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 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 7200 seconds. Table 4 shows the results.

4.3 Results & Discussions

For all benchmarks, all six consistencies are significantly faster and stronger than alpha-beta pruning.

Comparing the two duality approaches, we observe duality of constraints (DC) is stronger than duality of quantifiers (DQ), and we conjecture for any projection/extension conditions τ, DC-NC[τ] (DC-AC[τ] resp.) is stronger than DQ-NC[τ] (DQ-AC[τ] resp.). Note that enforcing projection/extension conditions on DQ-NC/DQ-AC may strengthen one approximation function, and weaken the other at the same time. DC-NC/DC-AC extracts costs from different copies of constraints and resolve this issue.

For all benchmarks, DQ-NC[proj-NC*] runs faster than DC-NC[proj-NC*]. In randomly generated problems and the graph coloring game, DC-AC[proj-(FD)AC*] runs faster than DQ-AC[proj-(FD)AC*], with DC-AC[proj-FDAC] the fastest. In GRLFAP, DQ-NC[proj-NC*] runs faster than the others for smaller instances and stronger consistencies are faster for larger ones. Enforcing proj-FDAC* is more computational expensive than proj-AC* and proj-NC*, and implementing duality of constraints requires implementing two copies of constraints. Therefore, stronger consistencies are worthwhile for larger instances, but not for smaller ones due to the large computational over-head.

It is worth noting DQ[proj-FDAC*] prunes less than DQ[proj-AC*], suggested by the fact that adding stronger projection/extension conditions from Weighted CSPs naively may not always strengthen our approximation functions. We have to further consider quantifier information.

All QCOP instances for even the smallest parameter settings for all benchmarks fail to run within the time limit. QCOPs are, in fact, more general [16] than MWCSPs. By viewing a more specific problem, it is natural for us to devise consistency techniques outperforming QeCode.

5 Concluding Remarks

We define and implement node and (full directional) arc consistency notions to reduce the search space of an alpha-beta search for MWCSPs, by approximating lower and upper bounds of the cost of the problem. Lower bound computation employs standard estimation of costs in the sub-problems and we propose two approaches: duality of quantifiers and duality of constraints, based on the Duality Principle in estimating upper bounds. Details on strengthening the approximation functions by re-using Weighted CSPs consistencies are given. We also discuss capabilities and limitations of our approach on other stronger solution concepts. Experiments on comparing basic alpha-beta pruning and the six consistencies from the two dualities are performed.

There are two closely related frameworks, where both tackle constraint problems with adversaries. Brown et al. propose adversarial CSPs [6], which focuses on the case where two opponents take turns to assign variables, each trying to direct the solution towards their own objectives. Another related work is Stochastic CSPs [26], which can represent adversaries by known probability distributions. We seek actions to minimize/-maximize the expected cost for all the possible scenarios. Our work is similar in the sense that we are minimizing the cost for the worst case scenario.

Possible future work includes: consistency algorithms for high arity (soft) constraints similar to those for WCSPs [18, 19, 17], value/variable ordering heuristics, theoretical comparisons on different consistency notions, tackling stronger solutions, and online algorithms.

References

1. Allis, L.V.: Searching for solutions in games and artificial intelligence. Ph.D. thesis, University of Limburg (1994)
2. Apt, K.: Principles of Constraint Programming. Cambridge University Press, New York, USA (2003)
3. Benedetti, M., Lallouet, A., Vautard, J.: Quantified constraint optimization. In: CP'08. pp. 463–477 (2008)
4. Bordeaux, L., Cadoli, M., Mancini, T.: CSP properties for quantified constraints: Definitions and complexity. In: AAAI'05. pp. 360–365 (2005)
5. Bordeaux, L., Monfroy, E.: Beyond NP: Arc-consistency for quantified constraints. In: CP'02. pp. 371–386 (2002)
6. Brown, K.N., Little, J., Creed, P.J., Freuder, E.C.: Adversarial constraint satisfaction by game-tree search. In: ECAI'04. pp. 151–155 (2004)
7. Cabon, B., de Givry, S., Lobjois, L., Schiex, T., Warners, J.: Radio link frequency assignment. CONSTRAINTS 4, 79–89 (1999)
8. Cooper, M.C., De Givry, S., Schiex, T.: Optimal soft arc consistency. In: IJCAI'07. pp. 68–73 (2007)
9. Cooper, M., de Givry, S., Sanchez, M., Schiex, T., Zytnicki, M., Werner, T.: Soft arc consistency revisited. Artificial Intelligence 174(7-8), 449–478 (2010)
10. Debruyne, R., Bessiere, C.: From restricted path consistency to max-restricted path consistency. In: CP'97. pp. 312–326 (1997)
11. Dempe, S.: Foundations of Bilevel Programming. Kluwer Academic Publishers (2002)
12. Gent, I.P., Nightingale, P., Stergiou, K.: QCSP-Solve: A solver for quantified constraint satisfaction problems. In: IJCAI'05. pp. 138–143 (2005)
13. van den Herik, H.J., Uiterwijk, J.W.H.M., van Rijswijck, J.: Games solved: Now and in the future. Artif. Intell. 134(1-2), 277–311 (2002)
14. Larrosa, J., Schiex, T.: In the quest of the best form of local consistency for weighted CSP. In: IJCAI'03. pp. 239–244 (2003)
15. Larrosa, J., Schiex, T.: Solving weighted CSP by maintaining arc consistency. Artificial Intelligence 159(1-2), 1–26 (2004)
16. Lee, J.H.M., Mak, T.W.K., Yip, J.: Weighted constraint satisfaction problems with min-max quantifiers. In: ICTAI'11. pp. 769–776 (2011)
17. Lee, J.H.M., Shum, Y.W.: Modeling soft global constraints as linear programs in weighted constraint satisfaction. In: ICTAI'11. pp. 305–312 (2011)
18. Lee, J.H.M., Leung, K.L.: Consistency techniques for flow-based projection-safe global cost functions in weighted constraint satisfaction. JAIR 43, 257–292 (2012)
19. Lee, J.H.M., Leung, K.L., Wu, Y.: Polynomially decomposable global cost functions in weighted constraint satisfaction (to appear). In: AAAI'12 (2012)
20. Mamoulis, N., Stergiou, K.: Algorithms for quantified constraint satisfaction problems. In: CP'04. pp. 752–756 (2004)
21. Murty, K.G.: Linear and Combinatorial Programming. R. E. Krieger (1985)
22. Neumann, J.V., Morgenstern, O.: Theory of Games and Economic Behavior. Princeton University Press (1944)
23. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V.: Algorithmic Game Theory. Cambridge University Press (2007)
24. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Pearson Education (2003)
25. Schaeffer, J., Burch, N., Bjrnsson, Y., Kishimoto, A., Mller, M., Lake, R., Lu, P., Sutphen, S.: Checkers is solved. Science 317(5844), 1518–1522 (2007)
26. Walsh, T.: Stochastic constraint programming. In: ECAI '02. pp. 111–115 (2002)
27. Wolsey, L.A.: Integer Programming. Wiley (1998)