Weighted Constraint Satisfaction Problems with Min-Max Quantifiers*

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*

Justin Yip
Brown University
Box 1910, Providence, RI 02912
USA
justin@cs.brown.edu

***Abstract*—Soft constraints are functions returning costs, and are essential in modeling over-constrained and optimization problems. We are interested in tackling soft constrained problems with adversarial conditions. Aiming at generalizing the weighted and quantified constraint satisfaction frameworks, a *Quantified Weighted Constraint Satisfaction Problem* (QWCSP) consists of a set of finite domain variables, a set of soft constraints, and a $\min$ or $\max$ quantifier associated with each of these variables. We formally define QWCSP, and propose a complete solver which is based on alpha-beta pruning. QWCSPs are useful special cases of QCOP/QCOP+, and can be solved as a QCOP/QCOP+. Restricting our attention to only QWCSPs, we show empirically that our proposed solving techniques can better exploit problem characteristics than those developed for QCOP/QCOP+. Experimental results confirm the feasibility and efficiency of our proposals.**

***Keywords*-Constraint Optimization, Soft Constraint Satisfaction, Quantified Constraint Satisfaction**

I. Introduction

The task at hand is that of an optimization problem with adversarial conditions. As an example, we begin with a generalized graph coloring problem in which numbers are used instead of colors. In addition, the graph is numbered by two players. The nodes are partitioned into two sets, A and B. Player 1 will number set A first, followed by player 2 numbering set B. The goal of player 1 is to maximize the total difference between numbers of adjacent nodes, while player 2 wishes to minimize the total difference. The aim is to help player 1 devising the best strategy.

The example is optimization in nature, and the adversaries originate from the numbers being placed on nodes by player 2. The question can be translated to maximizing total difference for all possible combinations of numbers player 2 can write. One way to solve this problem is by tackling many Constraint Optimization Problems [1] or Weighted CSPs [2], where each of them maximizes the total difference conditioned on a specific combination of numbers given by player 2. Solving these sub-problems independently, however, defies opportunities for exploiting global problem structure and characteristics, and reuse of computation. Another way is to model the problem as a QCSP [3] by finding whether there exists combinations of numbers for player 1 for all number placements by player 2 such that the total difference is less than a cost k. Trying different values of k progressively in separate QCSPs falls short in utilizing the objective function to guide search globally. We propose to combine the best of both worlds.

Weighted CSPs are minimization in nature. We introduce the $\max$ quantifier to further allow min-max operations on constraint costs in Weighted CSPs. A *Quantified Weighted Constraint Satisfaction Problem* (QWCSP) consists of an ordered sequence of finite domain variables, a set of soft constraints, and a $\min$ or $\max$ quantifier associated with each variable. Note that the existential ($\exists$) and universal ($\forall$) quantifiers in QCSPs can be expressed using $\min$ and $\max$ respectively. WCSPs and QCSPs are thus also special cases of QWCSPs. We define solutions of a QWCSP, and show how branch-and-bound search with alpha-beta pruning can be applied to solve a QWCSP. General pruning conditions of alpha-beta search are defined and discussed. The new framework aims to answer such interesting questions as "What is the best plan I can choose to minimize all the possible penalties for the worst case scenario?" QWCSPs can be modeled as QCOPs [4], which are a more general framework. We give a construction method followed by an example. We perform experimental evaluations, comparing our proposed solving techniques and those for QCOPs, on three benchmarks to show the efficiency and feasibility of our framework.

II. Background

We give the basic definitions of WCSPs and QCSPs. A *weighted constraint satisfaction problem* [2] (WCSP) is a tuple $(\mathcal{X}, \mathcal{D}, \mathcal{C}, k)$, where $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a finite set of *variables* and $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of *domains* of possible values. We denote $x_i = v_i$ be an *assignment* assigning value $v_i \in D_i$ to variable x_i, and the set of assignments $l = \{x_1 = v_1, x_2 = v_2, \ldots, x_n = v_n\}$ be a *complete assignment* on variables $\mathcal{X}$, where v_i is the value assigned to x_i. A *partial assignment* $l[S]$ is a projection of l

*We thank the anonymous referees for their constructive comments. The work described in this paper was substantially supported by grants (CUHK413710 and CUHK413808) from the Research Grants Council of Hong Kong SAR. We also thank Ian Gent for giving us QCSP-Solve, which serves as a basis for our solver.

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:

$$cost(l) = C_\varnothing \oplus \bigoplus_{C_s \in \mathcal{C}} C_s(l[S])$$

A complete assignment l on $\mathcal{X}$ is *feasible* if $cost(l) < k$, and is a *solution* of a WCSP if l has the minimum cost among all feasible tuples.

A *quantified constraint satisfaction problem* [3] (QCSP) $\mathcal{P}$ is a tuple $(\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{Q})$, where $\mathcal{X} = (x_1, x_2, \ldots, x_n)$ is an ordered sequence of variables, $\mathcal{D} = (D_1, D_2 \ldots, D_n)$ is an ordered sequence of finite domains, $\mathcal{C} = C_1 \wedge C_2 \wedge \ldots \wedge C_e$ is a conjunction of constraints, and $\mathcal{Q} = (Q_1, \ldots, Q_n)$ is a quantifier sequence in which each Q_i is either $\exists$ (existential, 'there exists') or $\forall$ (universal, 'for all') and associated with x_i. A *constraint* $C_k \in \mathcal{C}$ consists of a sequence $\mathcal{X}_k = (x_{k_1}, \ldots, x_{k_r})$ of $r > 0$ variables s.t. $\mathcal{X}_k$ is a subsequence of $\mathcal{X}$ and $\mathcal{D}_k$ is a subsequence of $\mathcal{D}$. C_k has an associated set $A[C_k] \subseteq D_{k_1} \times \ldots \times D_{k_r}$ of *tuples* which specify allowed combinations of values for the variables in $\mathcal{X}_k$. Let firstx($\mathcal{P}$) returns the first unassigned variable in the variable sequence. If there are no such variables, it returns $\perp$. The *semantics* of a QCSP $\mathcal{P}$ is defined recursively as follows: (1) In case $\mathrm{firstx}(\mathcal{P}) = \perp$, if all constraints $C_k \in \mathcal{C}$ are satisfiable, $\mathcal{P}$ is *satisfiable*; and if any constraint fails, $\mathcal{P}$ is *unsatisfiable*. (2) Otherwise, let $\mathrm{firstx}(\mathcal{P}) = x_i$. If $Q_i = \exists$ then $\mathcal{P}$ is *satisfiable* iff there exists a value $a \in D_i$ such that the simplified problem $\mathcal{P}$ with a assigned to x_i is satisfiable. If $Q_i = \forall$ then $\mathcal{P}$ is *satisfiable* iff for all values $a \in D_i$ the simplified problem $\mathcal{P}$ with a assigned to x_i is satisfiable.

III. Quantified Weighted CSPs

Standard WCSPs are minimization in nature, we aim at also optimizing problems with adversarial conditions, by modeling adversaries using $\max$ quantifiers. A *Quantified Weighted Constraint Satisfaction Problem* (QWCSP) $\mathcal{P}$ 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* as in WCSPs, $\mathcal{Q} = (Q_1, \ldots, Q_n)$ is a *quantifiers sequence* where Q_i is either $\max$ or $\min$ associated with x_i, and k is the global upper bound.

In a QWCSP, 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}. Suppose l is a complete assignment of $\mathcal{P}$. We reuse the definition of firstx($\mathcal{P}$) defined in QCSPs. The *A-cost($\mathcal{P}$)* of a QWCSP $\mathcal{P}$ is defined recursively as follows:

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

where $\mathbb{M}_i = \{\text{A-cost}(\mathcal{P}[x_i = v]) | v \in D_i\}$. A QWCSP $\mathcal{P}$ is *satisfiable* iff A-cost($\mathcal{P}$) $< k$. Similar to QCSPs, we define a *block* of variables in a QWCSP $\mathcal{P}$ to be a maximal subsequence of variables in $\mathcal{X}$ which has the same quantifiers. Changing the variable ordering within the same block of variables does not change the A-cost of a QWCSP.

Example 1: Given a QWCSP $\mathcal{P}$ with the ordered sequence of variables (x_1,x_2,x_3), domains $D_1 = \{a, b, c\}, D_2 = \{a, b\}$, and $D_3 = \{a, b, c\}$, the set of constraints represented in Figure 1, the quantifier sequence $(Q_1 = \max, Q_2 = \min, Q_3 = \max)$, and the global upper bound k. The problem is to find the A-cost of $\mathcal{P}$. Figure 1 indicates there are 3 unary constraints C_1, C_2, C_3 and 2 binary constraints $C_{1,2}, C_{2,3}$. For unary constraints, 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 show the computation for the A-cost of the QWCSP $\mathcal{P}$ as follows:

$$\text{A-cost}(\mathcal{P}) = \max_{v_1 \in D_1} \{ \min_{v_2 \in D_2} \{ \max_{v_3 \in D_3} \{\text{A-cost}(\mathcal{P}[x_1 = v_1][x_2 = v_2][x_3 = v_3])\}\}\}$$

$$\begin{aligned} = \quad & \max\{ \min\{\max\{cost(a,a,a), cost(a,a,b), cost(a,a,c)\}, \ldots\}, \\ & \min\{\max\{cost(b,a,a), cost(b,a,b), cost(b,a,c)\}, \ldots\}, \\ & \min\{\max\{cost(c,a,a), cost(c,a,b), cost(c,a,c)\}, \ldots\}\} \\ = \quad & \max\{ \min\{\max\{10,5,4\}, \max\{11,8,6\}\}, \\ & \min\{\max\{7,2,1\}, \max\{7,4,2\}\}, \\ & \min\{\max\{6,1,0\}, \max\{8,5,3\}\}\} \\ = \quad & \max\{ 10, 7, 6\} = 10 \end{aligned}$$

If $k > 10$ in Example 1, then the problem is satisfiable. Otherwise, Example 1 is unsatisfiable. Solution of a WCSP is a complete assignment with the minimum costs, while solution in QCSPs is winning strategy [5]. In this paper, we define a *solution* of a QWCSP $\mathcal{P}$ as a complete assignment $\{x_1 = v_1, \ldots, x_n = v_n\}$ s.t.: A-cost($\mathcal{P}$) = A-cost($\mathcal{P}[x_1 = v_1] \ldots [x_i = v_i]), \forall 1 \leq i \leq n$. Extracting solutions is easy after computing the A-cost of a QWCSP.

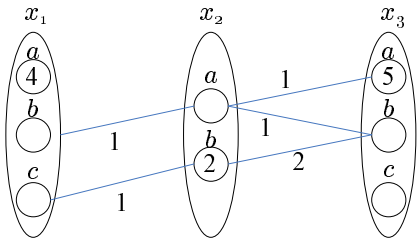

Figure 1. Constraints for Example 1

Figure 2 shows an adaptation of the standard labeling tree [1] for Example 1. The A-cost for each sub-problem is placed inside the corresponding node representing the sub-problem. Both WCSPs and QCSPs are special cases of QWCSPs. We state without proof the following theorems.

Theorem 1: A WCSP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C}, k)$ can be transformed by Karp reduction [6] to an equivalent QWCSP $\mathcal{P}' = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{Q}, k)$ with all $Q_i \in \mathcal{Q}$ equal to the $\min$ quantifier.

Given a hard constraint C. We can construct a soft constraint C' on the same set of variables. A soft constraint C' returns cost 0 if C is satisfiable on the same set of assignments; otherwise, C' returns cost k.

Theorem 2: A QCSP $\mathcal{P} = \langle \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{Q} \rangle$ can be transformed by Karp reduction [6] to an equivalent QWCSP $\mathcal{P}' = (\mathcal{X}, \mathcal{D}, \mathcal{C}', \mathcal{Q}', 1)$ where $\mathcal{C}'$ is the set of constraints constructed from $\mathcal{C}$. For each $Q'_i \in \mathcal{Q}'$, if $x_i \in \mathcal{X}$ is an existential variable in QCSP, then Q'_i is a $\min$ quantifier; otherwise, Q'_i is a $\max$ quantifier.

Corollary 3: QCSPs and WCSPs are special cases of QWCSPs, which are PSPACE-hard.

IV. Branch & Bound with Consistencies

This section outlines a complete solver for QWCSPs. The key idea of the solver is that, by applying alpha-beta pruning [7] in the Branch & Bound search and adapting consistency techniques used in WCSP [2], we can estimate the A-cost early in the search so as to reduce the search space.

We first discuss alpha-beta pruning. Then we discuss general conditions which can lead to prunings and backtrackings. Due to space limitations, we skip detailed node and arc consistency notions which were modified and integrated with alpha-beta pruning to solve QWCSPs more efficiently.

A. Alpha-Beta Pruning

Alpha-beta pruning attempts to reduce search nodes in a minimax algorithm by exploiting (a) semantics of the $\max$ and $\min$ quantifiers and (b) the upper and lower bounds of the costs of previously visited nodes. We can apply alpha-beta pruning in the Branch & Bound search directly in solving QWCSPs as only $\max$ and $\min$ quantifiers are

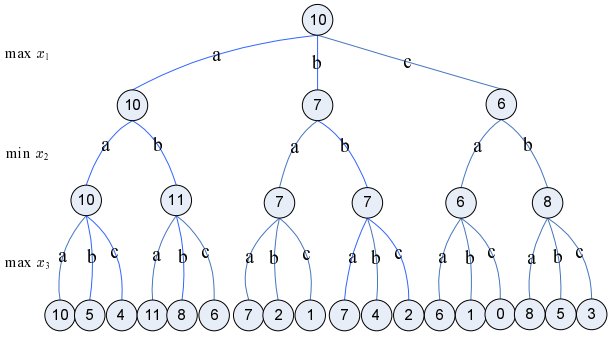

Figure 2. Labeling Tree for Example 1

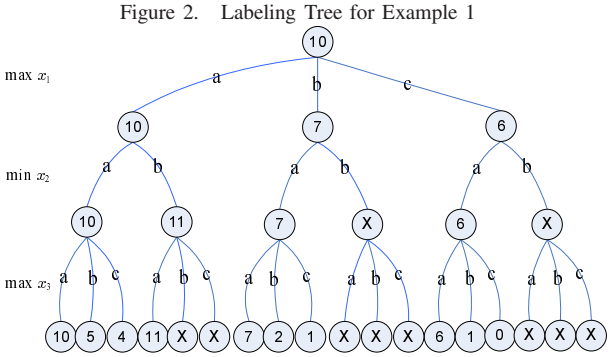

Figure 3. Labeling Tree for Example 1 after applying alpha-beta pruning

allowed. Alpha-beta pruning is also used for solving real-time online QCSPs [8].

Algorithm 1 is a high-level abstraction of a QWCSP solver. We first neglect the propagation routine in lines 7–31 and discuss the basic alpha-beta pruning algorithm. The search starts with `alpha_beta`$(\mathcal{P}, 0, k)$. The bounds lb and ub are the range of costs found by the alpha-beta pruning algorithm. QWCSP propagators further exploit these bounds to achieve stronger propagation. We use $\mathcal{P}[x_j \neq u]$ to denote a function pruning value u of variable x_j in $\mathcal{P}$, and $\mathcal{P}[x_i = v]$ to denote a function assigning value v to variable x_i in $\mathcal{P}$. Both functions return the modified problem. Line 3 is the base case in which all variables are bound. The routine `cost` returns the cost of the complete assignment. Lines 32–40 give the main routine of the traditional alpha-beta pruning algorithm. We only explain the cost for the $\min$ quantifier, since that of $\max$ is similar. The for loop in lines 6–42 evaluates all sub-problems $\mathcal{P}[x_i = v]$ by recursively invoking the alpha-beta algorithm. Since the goal is to find a minimum value, the upper bound is updated. When the upper bound is less than the lower bound (line 38), it triggers the short-cut to break out of the remaining search since every value returned by subsequent calls will be dominated by the current bounds. The function `alpha_beta` ends by returning the upper bound for the $\min$ quantifier (line 41). We illustrate the code with an example.

Example 2: Consider the tree in Figure 2, and assume values are labeled in the sequence of $[a, b, c]$. The search starts with `alpha_beta`$(\mathcal{P}, 0, k)$. Consider the node $\mathcal{P}' = \mathcal{P}[x_1 = a]$, which first visits its sub-problem $\mathcal{P}'[x_2 = a]$ by

Algorithm 1 A QWCSP Solver

```
1: function alpha_beta(P,lb,ub):
2: if firstx(P) == ⊥ then
3:     return cost(P)
4: end if
5: x_i = firstx(P)
6: for v ∈ D_i do
7:     {Pruning using QWCSP semantics}
8:     changed = true
9:     while changed do
10:        changed = false
11:        for j ∈ [i..n] do
12:            for u ∈ D_j do
13:                ap_lb = approx_lb(P, x_j = u)
14:                if ub <= ap_lb then
15:                    if Q_j == min then
16:                        P = P[x_j ≠ u], changed = true
17:                    else
18:                        return ub
19:                    end if
20:                end if
21:                ap_ub = approx_ub(P, x_j = u)
22:                if ap_ub <= lb then
23:                    if Q_j == min then
24:                        return lb
25:                    else
26:                        P = P[x_j ≠ u], changed = true
27:                    end if
28:                end if
29:            end for
30:        end for
31:    end while
32:    {Basic alpha-beta pruning}
33:    if Q_i == min then
34:        ub = min(ub, alpha_beta(P[x_i = v], lb, ub))
35:    else
36:        lb = max(lb, alpha_beta(P[x_i = v], lb, ub))
37:    end if
38:    if ub <= lb then
39:        break
40:    end if
41:    return (Q_i == min)?ub : lb
42: end for
```

calling alpha_beta($\mathcal{P}'[x_2 = a], 0, k$) and a value of 10 is returned. Since $Q_2 = \min$, the upper bound is updated. The routine then invoke alpha_beta($\mathcal{P}'[x_2 = b], 0, 10$). After visiting $\mathcal{P}'[x_2 = b][x_3 = a]$, we get an A-cost of 11 for the sub-problem. The quantifier here is $\max$, the cost is greater than the upper bound and the condition in line 38 holds. No matter what costs the remaining sub-problems produce, they have no impact on the solution and a short-cut to break out of the remaining search is triggered. Hence, the sub-problems $\mathcal{P}'[x_2 = b][x_3 = b]$ and $\mathcal{P}'[x_2 = b][x_3 = c]$ are not explored.

Figure 3 illustrates the nodes pruned, denoted by the symbol X, by alpha-beta pruning.

B. Consistency Techniques

In traditional CSPs, we enforce different levels of consistency to prune infeasible domain values and hence reduce the search space. In WCSPs, the consistency algorithms take the cost of constraints into account. Various consistency notions (e.g. NC*, AC* [2], FDAC*, EDAC*, OSAC, and VAC [9]) have been proposed and proven to be useful in improving solver performance. Such techniques, however, cannot be directly applied to QWCSP since the quantifiers change the semantics of constraints. In particular, applying these consistency notions on constraints covering on $\max$ variables may result in unsound prunings. Consistency notions for QWCSPs need to take quantifiers into account.

To prune values of a QWCSP, the main idea is that if the A-cost of a sub-problem $\mathcal{P}' = \mathcal{P}[x_i = v]$ is greater than or equal to the upper bound ub (less than or equal to the lower bound lb resp.), the pruning techniques in alpha-beta can be applied. Let $\mathcal{P}[x_{1..i-1} = v_{1..i-1}, x_i = v]$ 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: $\exists v \in D_i \ \ s.t. \ \forall v_1 \in D_1, ..., v_{i-1} \in D_{i-1}$:

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

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

When any of the above conditions is satisfied, we can apply alpha-beta pruning according to Table I.

Table I
WHEN CAN WE PRUNE/BACKTRACK

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

Theorem 4: Given a QWCSP $\mathcal{P}$. If Condition (1)/(2) for $\mathcal{P}$ is satisfied, applying prunings and backtrackings according to Table I is sound.

Proof: (Sketch) Reasons to perform prunings and backtracking for $\min$ and $\max$ are symmetrical. We only describe the case where $Q_i = \min$. Suppose Condition (1) holds. We consider A-cost(s) for sub-problems $\mathcal{P}[x_{1..i-1} = v_{1..i-1}]$. Without loss of generality, we write $\mathcal{P}_{i-1}$ to be one of these sub-problems $\mathcal{P}[x_{1..i-1} = v_{1..i-1}]$ by fixing values $v_1 \in D_1$, $v_2 \in D_2$, ..., $v_{i-1} \in D_{i-1}$. We will see the proof using $\mathcal{P}_{i-1}$ applies for all sub-problems $\mathcal{P}[x_{1..i-1} = v_{1..i-1}]$, regardless on which values we fix. Given $Q_i = \min$, we obtain:

$$\text{A-cost}(\mathcal{P}_{i-1}) = \min_{a \in D_i} \mathcal{P}_{i-1}[x_i = a]$$

If A-cost$(\mathcal{P}_{i-1}) < ub$, the following must be true:

$$\exists v' \in D_i \text{ where } v' \neq v \ \ s.t. \ \ \text{A-cost}(\mathcal{P}_{i-1}[x_i = v']) < ub$$

Pruning value v does not change the A-cost of P_{i-1}. If A-cost$(\mathcal{P}_{i-1}) \geq ub$, i.e. $\mathcal{P}_{i-1}$ must not lead to solutions, the following must be true:

$$\forall v' \in D_i, \text{A-cost}(\mathcal{P}_{i-1}[x_i = v']) \geq ub$$

After pruning value v, either domain wipe out occurs or A-cost$(\mathcal{P}_{i-1}) \geq ub$. For both cases, the sub-problem $\mathcal{P}_{i-1}$ cannot lead to solutions. Combining the two cases, pruning value v does not change the problem from unsatisfiable to satisfiable(,and vice versa).

We now discuss Condition (2). Similar to the previous case, we consider the A-cost for these sub-problems

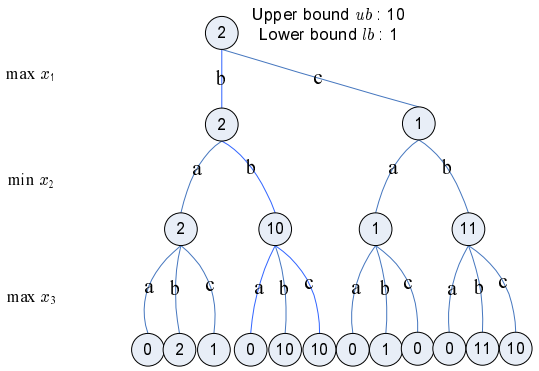

Figure 4. Labeling Tree for Example 3

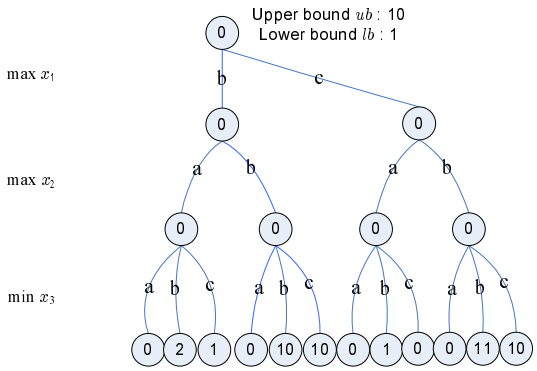

Figure 5. Labeling Tree for Example 4

$\mathcal{P}[x_{1..i-1} = v_{1..i-1}]$, and we fix $\mathcal{P}_{i-1}$ to be one of these sub-problems similarly. Given $Q_i = \min$, we obtain:

$$\text{A-cost}(\mathcal{P}_{i-1}) = \min_{a \in D_i} \mathcal{P}_{i-1}[x_i = a]$$

By Condition (2), $\mathcal{P}_{i-1}[x_i = v] \leq lb$ holds, and therefore:

$$\text{A-cost}(\mathcal{P}_{i-1}) \leq lb$$

Recall A-cost$(\mathcal{P}[x_{1..i-1} = v_{1..i-1}, x_i = v]) \leq lb$ applies regardless on which value $v_1, v_2, \ldots, v_{i-1}$ we fix. Therefore, we obtain:

$$\forall v_1 \in D_1, ..., v_{i-1} \in D_{i-1}, \text{A-cost}(\mathcal{P}[x_{1..i-1} = v_{1..i-1}]) \leq lb$$

We can easily obtain the following results using the definition of A-cost for a QWCSP: A-cost$(\mathcal{P}) \leq lb$. QWCSP $\mathcal{P}$ must be unsatisfiable, and the solver can backtrack. ■

Example 3: Given a QWCSP $\mathcal{P}$ with the ordered sequence of variables (x_1,x_2,x_3), domains $D_1 = \{a, b, c\}, D_2 = \{a, b\}$, and $D_3 = \{a, b, c\}$, the quantifier sequence $(Q_1 = \max, Q_2 = \min, Q_3 = \max)$, and the global upper bound 10. Suppose the A-cost for sub-problem $\mathcal{P}[x_1 = a]$ is 1. Figure 4 shows the upper bound ub, lower bound lb, and the A-costs for the remaining sub-problems. By inspecting the figure, we can easily observe Condition (2) holds: $\exists a \in D_3 \;\; s.t. \;\; \forall v_1 \in D_1, \forall v_2 \in D_2$,

$$\text{A-cost}(\mathcal{P}[x_1 = v_1][x_2 = v_2][x_3 = a]) \leq lb$$

By Table I, we can prune value a of x_3. We can easily observe the solution must not contain the assignment $x_3 = a$, and therefore, we can prune the value. After pruning value a of x_3, we can easily observe Condition (1) holds: $\exists b \in D_2 \;\; s.t. \;\; \forall v_1 \in D_1$,

$$\text{A-cost}(\mathcal{P}[x_1 = v_1][x_2 = b]) \geq ub$$

Similarly, we can prune value b of x_2.

Example 4: Suppose the quantifier sequence of Example 3 is replaced by $(Q_1 = \max, Q_2 = \max, Q_3 = \min)$, and the A-cost for sub-problem $\mathcal{P}[x_1 = a]$ remains unchanged (A-cost$(\mathcal{P}[x_1 = a]) = 1$). Figure 5 shows the upper bound ub, lower bound lb, and the A-costs for the remaining sub-problems. Costs for each complete assignment remain the same as in Example 3. The only difference is the modified A-costs for sub-problems, resulting from the change in quantifiers. By inspecting the figure, we can easily observe Condition (2) still holds: $\exists a \in D_3 \;\; s.t. \;\; \forall v_1 \in D_1, \forall v_2 \in D_2$,

$$\text{A-cost}(\mathcal{P}[x_1 = v_1][x_2 = v_2][x_3 = a]) \leq lb$$

As $Q_3 = \min$, we can easily observe all the A-costs for sub-problems $\mathcal{P}[x_1 = v_1][x_2 = v_2], \forall v_1 \in D_1, v_2 \in D_2$ must be less than or equal to the lb. By induction, we can conclude sub-problems $\mathcal{P}[x_1 = v_1], \forall v_1 \in D_1$ must be less than or equal to the lb, and finally obtain A-cost$(\mathcal{P}) \leq lb$. Therefore following Table I, the solver can backtrack.

One way to check Condition (1)/(2) is to find the exact value of the A-cost for each sub-problem, which is computationally expensive. The problem is essentially equivalent to determining if a variable assignment is a solution of a classical CSPs in general, which is NP-hard. A common technique in constraint programming is to formulate consistency notions and algorithms, which aim at extracting and making information in a problem explicit. Useful information includes pruning and cost information. Here we apply the same idea. We use efficient ways to extract a good upper bound and lower bound of A-cost, so as to backtrack or identify non-solution values from domains early in the search.

In the QWCSP solver (Algorithm 1), lines 7–31 prune or backtrack according to the conditions specified in Table I. Since finding A-cost is difficult, the algorithm finds the approximated bounds (`approx_lb` in line 13, and `approx_ub` in line 21). Functions `approx_lb`$(\mathcal{P}, x_i = v)$ and `approx_ub`$(\mathcal{P}, x_i = v)$ find the approximate A-cost for the set S of sub-problems, where:

$$S = \{\mathcal{P}[x_{1..i-1} = v_{1..i-1}, x_i = v] | \forall v_1 \in D_1, \ldots, v_{i-1} \in D_{i-1}\}$$

such that:

$$\forall P' \in S, \text{A-cost}(P') \leq \texttt{approx\_ub}(\mathcal{P}, x_i = v) \text{ ,and}$$
$$\forall P' \in S, \text{A-cost}(P') \geq \texttt{approx\_lb}(\mathcal{P}, x_i = v)$$

$\texttt{approx\_ub}(\mathcal{P}, x_i = v)$ is *tight* if:

$$\max_{P' \in S} \text{A-cost}(P') = \texttt{approx\_ub}(\mathcal{P}, x_i = v)$$

Similarly, $\texttt{approx\_lb}(\mathcal{P}, x_i = v)$ is *tight* if:

$$\min_{P' \in S} \text{A-cost}(P') = \texttt{approx\_lb}(\mathcal{P}, x_i = v)$$

Corollary 5: Given a QWCSP $\mathcal{P}$, If $\texttt{approx\_ub}(\mathcal{P}, x_i = v) \leq lb$, we can prune value v of variable x_i if $Q_i = \max$, and perform backtrack if $Q_i = \min$. If $\texttt{approx\_lb}(\mathcal{P}, x_i = v) \geq ub$, we can prune value v of variable x_i if $Q_i = \min$, and perform backtrack if $Q_i = \max$.

Proof: (Sketch) We can easily observe the following:

$$lb \geq \texttt{approx\_ub}(\mathcal{P}, x_i = v) \geq \mathcal{P}[x_{1..i-1} = v_{1..i-1}, x_i = v]$$
$$ub \leq \texttt{approx\_lb}(\mathcal{P}, x_i = v) \leq \mathcal{P}[x_{1..i-1} = v_{1..i-1}, x_i = v]$$

$\forall v_1 \in D_1, v_2 \in D_2, \ldots, v_{i-1} \in D_{i-1}$. Condition $\texttt{approx\_ub}(\mathcal{P}, x_i = v) \leq lb$ implies Condition (2), and condition $\texttt{approx\_lb}(\mathcal{P}, x_i = v) \geq ub$ implies Condition (1). We then apply Table I according to these conditions. ■

We have designed two kinds of consistency notions: node consistency and arc consistency, which extract the approximated upper and lower bounds (hence implement `approx_ub` and `approx_lb`) for QWCSPs. Unfortunately, we have to skip the technical details due to space limitations.

V. QCOP VERSUS QWCSP

QWCSPs are special cases of QCOP [4]. Given a QWCSP, this section shows how to construct a QCOP instance by the "Soft As Hard" approach [10], which was used to construct classical COP instances from WCSP instances. In the next section, we provide empirical evidence to demonstrate that our proposed solving techniques are more efficient than those for QCOPs for tackling QWCSPs.

A. Definitions of QCOP

We first give the definition of QCOP [4]. A *restricted quantified set of variables* (rqset) is a tuple (q, W, C) where $q \in \{\exists, \forall\}$, W is a subset of variables $W \subseteq V$, and C is a CSP. A *prefix* P of rqsets is a sequence of rqsets $((q_1, W_1, C_1), \ldots, (q_n, W_n, C_n))$ such that $W_i \cap W_j = \varnothing, \forall i \neq j$. We define *range(P)* for a prefix P with n rqsets to be $[1..n]$, *before$_i$(P)* to be $\bigcup_{j \leq i} W_j$, and $nu_i(P) = \min_{j>i}\{j | q_j = \forall\}$ to be the index of the next universal block of variables located after an index i. If no such index exists, we denote $nu_i(P)$ by $n+1$. Let $\mathcal{A}$ be a set of aggregate names and $\mathcal{F}$ a set of aggregate functions. An *aggregate* is an atom of the form $a : f(X)$ where $a \in \mathcal{A}$, $f \in \mathcal{F}$, and $X \in V \cup \mathcal{A}$. An *optimization condition* is an atom of the form $\min(X)$, $\max(X)$ where $X \in V \cup \mathcal{A}$ or the atom any. An atom $\min(X)$ ($\max(X)$ resp.) means the user is interested in strategies that minimize (maximize resp.) this value, while any indicates the user does not care about the returned strategy. A $\exists$-*orqset* is a tuple $(\exists, W, C, o)$ where $(\exists, W, C)$ is a rqset and o is an optimization condition. A $\forall$-*orqset* is a tuple $(\forall, W, C, A)$ where $(\forall, W, C)$ is a rqset and A is a set of aggregates. We denote *names* of the set of aggregates A in a $\forall$-orqset by $names(A)$. An *orqset* is either a $\exists$-orqset or a $\forall$-orqset. A *QCOP+* is a pair (P, G) where G is a CSP and $P = (orq_1, \ldots, orq_n)$ is a prefix of orqsets such that $\forall i \in range(P)$, with $k = nu_i(P)$: 1) if $orq_i = (\exists, W, C, o)$ with $o = \min(X)$ or $o = \max(X)$, then we must have $X \in$ *before*$_{k-1}(P) \cup (k < n+1 ? names(A_k) : \varnothing)$, and 2) if $orq_i = (\forall, W, C, A)$, then for all $a : f(X)$ in A, we must have $X \in$ *before*$_{k-1}(P) \cup (k < n+1 ? names(A_k) : \varnothing)$. A QCOP is a QCOP+ in which no orqset has restrictions, *i.e.* no constraints in the CSP of all orqsets.

B. Transforming QWCSPs into QCOPs

We can transform any QWCSP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{Q}, k)$ into a QCOP $\mathcal{P}' = (P', G')$ based on the modified "Soft As Hard" approach as follows. For each variable x_i in $\mathcal{P}$, there is an orqset $orq_i = (\exists, \{x'_i\}, C'_i, o'_i)$ in P', where C'_i has no constraints. For every soft constraint C in $\mathcal{C}$, there is a corresponding cost variable x'_c in the CSP G' with domain being equal to all possible costs given by C. We construct an auxiliary cost variable s in CSP G' which is equal to the sum of all cost variables x'_c. A constraint $s < k$ is added to restrict the total cost to be less than the global upper bound. If x_i is a minimization variable, then we add $o'_i = \min(s)$; otherwise, $o'_i = \max(s)$. Suppose C on a set S of variables giving cost m when a tuple of assignment l is assigned. There is a *reified constraint* in the CSP G' restricting x'_c to take value m if variables in S are assigned with tuple l.

Example 5: Given a QWCSP $\mathcal{P}$ with an ordered sequence of variables (x_1, x_2), domains $D_1 = D_2 = \{a, b\}$, a set of constraints $\{C_1, C_2\}$, a quantifier sequence $\{Q_1 = \min, Q_2 = \max)$, and a global upper bound $k = 7$. $C_1(a) = 0, C_1(b) = 5, C_2(a) = 1, C_2(b) = 3$. The QCOP $\mathcal{P}'$ can be expressed using the "Soft As Hard" approach as shown in Figure 6.

$$\begin{array}{l}
\exists x'_1 \in D_1 \\
\qquad \exists x'_2 \in D_2 \\
\qquad\qquad x'_{C_1} \in \{0,5\}, x'_{C_2} \in \{1,3\}, s \in \{1,3,6\} \\
\qquad\qquad s = x'_{C_1} \oplus x'_{C_2} \wedge s < 7 \\
\qquad\qquad [x'_1 = a] \Rightarrow [x'_{C_1} = 0] \\
\qquad\qquad [x'_1 = b] \Rightarrow [x'_{C_1} = 5] \\
\qquad\qquad [x'_2 = a] \Rightarrow [x'_{C_2} = 1] \\
\qquad\qquad [x'_2 = b] \Rightarrow [x'_{C_2} = 3] \\
\qquad \max s \\
\min s
\end{array}$$

Figure 6. Transformed QWCSP in Example 5

Theorem 6: A QWCSP $\mathcal{P}$ can be transformed into a QCOP $\mathcal{P}'$. The A-cost of P can be found by solving the optimal strategy [4] of $\mathcal{P}'$.

The proof follows directly from the "Soft As Hard" construction. In particular, A-cost of $\mathcal{P}$ is equal to the value assigned to s in the optimal strategy [4] of $\mathcal{P}'$.

The central theme of this paper is to show QWCSP, a more restricted but useful subclass of QCOP+, can be solved more efficiently. Based on WCSPs, QWCSP's consistency techniques redistribute constraint costs by projections and extensions [2]. In turn, we conjecture that this allows extra prunings over the classical consistencies used in QCOP+. The situation is similar to how Lee and Leung [11] show WCSP (soft approach) consistencies to be stronger than classical optimization used in "Soft As Hard".

VI. Performance Evaluation

In this section, we compare the QCOP+ solver QeCode against our solver in three progressive modes: Alpha-beta pruning, Node Consistency (NC), and Arc Consistency (AC). 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 number of solved instances, average time used, and average number of tree nodes encountered. We take average for solved instances *only*. Winning entries for average time used and average number of tree nodes encountered are highlighted in bold. A symbol '-' represents all instances fail to run within the time limit of 900 seconds. The experiment is conducted on a Pentium 4 3.2GHz with 3GB memory. We compare our solver against QeCode, which uses minimax. All QWCSP instances are transformed to QCOP using the "Soft As Hard" approach outlined. We note alpha-beta prunings can be employed for QCOP+, but we believe there will be less prunings by enforcing classical consistencies.

A. Random Generated Problems

We generate random binary QWCSPs with parameters (n, s, d), where n is the number of variables, s is the domain size for each variable, and d is the probability for a binary constraint to occur between two variables. We purposefully do not generate unary constraints to make the problem harder to solve. The costs for each binary constraint are generated uniformly in [0...30]. Quantifiers are generated randomly with half probability for min (max resp.), and number of quantifier levels vary from instances to instances. Table II shows the results. For all instances, even just alpha-beta pruning is two orders of magnitude faster than QeCode, which cannot handle even moderately sized instances. NC and AC both run faster than alpha-beta pruning, with the search space dramatically decreased and runtime significantly faster. AC, utilizing information from both unary and binary constraints, performs best among all solver modes.

B. Graph Coloring Game

We have generated instances (v, c, d) for a graph coloring game similar to the one in the introduction, 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, players play in turn, and in each turn a node is numbered. The node corresponding to each turn is generated randomly. Table III shows the results. Similar to Random Problems, alpha-beta pruning runs faster than QeCode in all instances two orders of magnitude faster. NC and AC both run faster than alpha-beta pruning. Comparing AC and alpha-beta pruning, we can gain up to six times speedup for AC, which prunes up to two-third of the search space of NC. Again, AC betters in almost all instances in runtime.

C. Min-Max Resource Allocation Problem

Suppose N units of resources are allocated to t activities. We let x_i be the amount of resources allocated to activity i, and a function $c_i(x_i)$ returns the cost incurred from activity i by allocating x_i units of resources the activity. The resource allocation problem [12] is to find an optimal resource allocation so as to minimize the total cost. Suppose now there are s functions c_i^s. The min-max resource allocation problem [13] is to find a resource allocation to minimize the maximum cost functions. Table IV shows the results. Alpha-beta pruning runs an order of magnitude faster than QeCode. Again, NC and AC both run faster than alpha-beta pruning. AC can prune up to 90% of the search space of NC and runs the fastest in all instances.

VII. Concluding Remarks

We define the QWCSP framework for modeling optimization problems with adversaries. Our work allows us to model and solve interesting problems, such as graph coloring game and min-max resource allocation problem, efficiently. We propose and implement a complete solver incorporating alpha-beta pruning into branch-and-bound. We evaluate our proposal using three benchmarks and compare with QeCode. Experimental results show consistency enforcement is worthwhile in general and our solver is orders of magnitude faster than QeCode in tackling QWCSPs, although QeCode is a more general solver for solving QCOP/QCOP+. When restricting attention to QWCSPs, we are able to devise specific inference to prune and guide search. Our results suggest alpha-beta pruning techniques can be a future possible direction for enhancing QeCode.

Brown et. al propose a similar framework, Adversarial CSPs [14], which focuses on the case where two opponents take turns to assign variables, each trying to direct the solution towards their own objectives. They describe the process as a game-tree search, and the solving algorithm is based on the minimax algorithm. Our work pushes the idea further by: 1) applying alpha-beta pruning, and 2) exploiting costs information using soft constraint techniques. Another related work is Stochastic CSPs [15], which can represent

Table II
RANDOM GENERATED PROBLEM

	QeCode			Alpha-beta			Node Consistency			Arc Consistency		
(n, s, d)	#solve	Time	#nodes	#solve	Time	#nodes	#solve	Time	#nodes	#solve	Time	#nodes
(9, 5, 0.4)	20	401.62	4,394,531	20	0.84	110,738	20	0.19	11,115	20	**0.17**	**4,221**
(9, 5, 0.6)	20	550.32	4,394,531	20	1.95	228,017	20	**0.23**	11,817	20	0.28	**4,024**
(12, 5, 0.4)	0	-	-	20	91.23	5,967,461	20	5.04	158,179	20	**4.25**	**53,866**
(12, 5, 0.6)	0	-	-	20	66.54	4,782,541	20	**3.87**	118,40	20	4.27	**41,710**
(16, 5, 0.4)	0	-	-	2	706.51	26,269,025	20	299.15	5,780,075	20	**193.63**	**1,657,203**
(16, 5, 0.6)	0	-	-	1	830.29	32,859,735	17	438.42	8,085,200	19	**428.35**	**2,909,388**
(18, 5, 0.4)	0	-	-	0	-	-	1	815.69	11,210,135	3	**595.33**	**3,514,426**
(18, 5, 0.6)	0	-	-	0	-	-	1	718.15	9,171,827	2	**470.27**	**2,613,430**

Table III
GRAPH COLORING GAME

	QeCode			Alpha-beta			Node Consistency			Arc Consistency		
(v, c, d)	#solve	Time	#nodes	#solve	Time	#nodes	#solve	Time	#nodes	#solve	Time	#nodes
(10, 4, 0.4)	20	141.16	2,446,677	20	0.41	43,509	20	0.13	6,158	20	**0.11**	**2,580**
(10, 4, 0.6)	20	179.58	2,446,677	20	0.49	49,029	20	**0.17**	8,124	20	**0.17**	**3,617**
(12, 4, 0.4)	0	-	-	20	3.46	266,589	20	0.95	33,739	20	**0.70**	**13,124**
(12, 4, 0.6)	0	-	-	20	4.22	302,255	20	1.34	47,010	20	**1.19**	**18,088**
(16, 4, 0.4)	0	-	-	20	214.93	10,050,800	20	44.73	1,002,145	20	**30.20**	**363,523**
(16, 4, 0.6)	0	-	-	20	210.21	9,213,029	20	43.85	949,861	20	**37.93**	**352,691**
(18, 4, 0.4)	0	-	-	0	-	-	20	278.00	4,095,993	20	**158.71**	**1,315,212**
(18, 4, 0.6)	0	-	-	0	-	-	20	362.04	5,295,433	20	**238.51**	**1,711,880**

Table IV
MIN-MAX RESOURCE ALLOCATION

	QeCode			Alpha-beta			Node Consistency			Arc Consistency		
(t, N, s)	#solve	Time	#nodes	#solve	Time	#nodes	#solve	Time	#nodes	#solve	Time	#nodes
(10, 8, 7)	20	49.03	656,221	20	2.17	146,778	20	1.13	39,605	20	**0.48**	**4,317**
(12, 8, 7)	20	166.42	1,889,371	20	9.26	444,340	20	4.95	132,257	20	**1.91**	**15,096**
(12, 9, 7)	20	399.39	4,408,771	20	20.63	981,967	20	8.61	226,456	20	**3.05**	**21,563**
(13, 9, 7)	20	716.22	7,461,106	20	41.82	1,732,234	20	17.17	398,072	20	**5.87**	**39,831**
(13, 9, 8)	20	852.29	8,455,920	20	42.07	1,700,088	20	17.25	399,221	20	**6.34**	**39,526**

adversaries by known probability distributions. Probabilities for every action decided by adversaries are known beforehand. We then seek actions to minimize/maximize the expected cost for all possible scenarios. Our work is similar in the sense that we are minimizing the cost for the worst case scenario. It will be interesting to study how to model non-zero sum optimization problems, allow restricted quantiers [16] for soft constraints, devise arc consistency algorithms for high arity soft constraints, and investigate on variable / value ordering heuristics.

REFERENCES

[1] K. Apt, *Principles of Constraint Programming*. New York, USA: Cambridge University Press, 2003.

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

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

[4] M. Benedetti, A. Lallouet, and J. Vautard, "Quantified constraint optimization," in *CP'08*, 2008, pp. 463–477.

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

[6] S. Arora and B. Barak, *Computational Complexity: A Modern Approach*, 1st ed. Cambridge University Press, 2009.

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

[8] D. Stynes and K. N. Brown, "Realtime online solving of quantified CSPs," in *CP'09*, 2009, pp. 771–786.

[9] M. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki, and T. Werner, "Soft arc consistency revisited," *Artificial Intelligence*, vol. 174, no. 7-8, pp. 449–478, 2010.

[10] T. Petit, J.-C. Régin, and C. Bessière, "Specific filtering algorithms for over-constrained problems," in *CP'01*, 2001, pp. 451–463.

[11] J. H. M. Lee and K. L. Leung, "Towards efficient consistency enforcement for global constraints in weighted constraint satisfaction," in *IJCAI'09*, 2009, pp. 559–565.

[12] K. M. Mjelde, *Methods of Allocation of Limited Resources*. John Wiley, 1983.

[13] G. Yu, "Min-max optimization of several classical discrete optimization problems," *Journal of Optimization Theory and Applications*, vol. 98, pp. 221–242, 1998.

[14] K. N. Brown, J. Little, P. J. Creed, and E. C. Freuder, "Adversarial constraint satisfaction by game-tree search," in *ECAI'04*, 2004, pp. 151–155.

[15] T. Walsh, "Stochastic constraint programming," in *ECAI '02*, 2002, pp. 111–115.

[16] M. Benedetti, A. Lallouet, and J. Vautard, "QCSP made practical by virtue of restricted quantification," in *IJCAI'07*, 2007, pp. 38–43.