Realtime Online Solving of Quantified CSPs*

David Stynes and Kenneth N. Brown

Cork Constraint Computation Centre,
Dept of Computer Science, University College Cork, Ireland
d.stynes@4c.ucc.ie, k.brown@cs.ucc.ie

Abstract. We define Realtime Online solving of Quantified Constraint Satisfaction Problems (QCSPs) as a model for realtime online CSP solving. We use a combination of propagation, lookahead and heuristics and show how all three improve performance. For adversarial opponents we show that we can achieve promising results through good lookahead and heuristics and that a version of alpha beta pruning performs strongly. For random opponents, we show that we can frequently achieve solutions even on problems which lack a winning strategy and that we can improve our success rate by using Existential Quantified Generalised Arc Consistency, a lower level of consistency than SQGAC, to maximise pruning without removing solutions. We also consider the power of the universal opponent and show that through good heuristic selection we can generate a significantly stronger player than a static heuristic provides.

1 Introduction

Many practical decision problems are not under the control of a single decision maker. For example, in planning under uncertainty, mixed initiative planning, interactive configuration or game playing, either the external environment or another actor refines the detail of the problem as decisions are being made. Such problems can be modeled as *online* constraint satisfaction, where the problem variables must be instantiated in a fixed sequence, but where some of those variables are set externally, and the aim is to achieve a complete satisfying assignment at the end of the process. *Quantified* constraint satisfaction (QCSP) is a generalization of CSP, which also has a fixed sequence of variables, but where some of the variables are universally quantified. The aim is to find a *winning strategy*, which guarantees a complete satisfying assignment for every possible combination of values for the universal variables. QCSP can be regarded as a model for online CSP: the existential variables represent the values under our control, while the universal variables represent the externally assigned variables, and if we can find a winning strategy for the QCSP, then we can guarantee to find a solution to the online CSP. But in many online problems, including for example delivery dispatch, game playing or reservation management, we have

* This work was supported in part by Microsoft Research through the European PhD Scholarship Programme, and by the Embark Initiative of the Irish Research Council for Science, Engineering and Technology (IRCSET).

I.P. Gent (Ed.): CP 2009, LNCS 5732, pp. 771–786, 2009.

limited time in which to make each decision, and so online CSP can be extended to *realtime* online CSP. Given the time limits, it may no longer be feasible to search for a winning strategy for the corresponding QCSP. If we are to continue using QCSP as a model, then we must develop methods for finding solutions to a QCSP interactively, or online, as the universal variables are assigned, and we must do this under time constraints.

Here, we investigate realtime online solving of QCSPs. We continue to use QCSP as an idealised model of online CSP, but at each time-limited step we search partial strategies to find the best decision, in the style of game tree search. We develop the use of constraint propagation, game-tree search and ordering heuristics for finding partial strategies. We consider two types of external actors: (i) adversarial opponents, who try to prevent us finding a solution, and (ii) random solvers, which simply select random values for the variables. We develop existential generalised arc consistency, which does not prune any solutions from a QCSP, and which is particularly effective against random solvers. We develop a version of alpha-beta pruning for adversarial opponents, and a method based on weighted estimates for random solvers. We evaluate our methods empirically. On random binary QCSPs, we show that alpha-beta is most effective on large adversarial problems, while against random solvers we show that weighted estimates frequently finds solutions where no winning strategy exists. We then consider Online Bin Packing problems, requiring non-binary constraints, where the external solver generates the items to be packed into the bins. For these problems we propose some online heuristics tailored to bin packing, and we show that these heuristics outperform static heuristics. Performance is obviously affected by the quality of the opponent, and we show the effect of different strategies and heuristics for the opposing solver.

2 Background

A *Quantified Constraint Satisfaction Problem*[1] is a tuple $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{Q})$ where $\mathcal{X} = \{X_1, \ldots, X_n\}$ is a set of variables, $\mathcal{D} = \{D_1, \ldots, D_n\}$ is their domains, $\mathcal{C} = \{C_1, \ldots, C_e\}$ is a finite set of constraints and $\mathcal{Q} = Q_1 X_1 \ldots Q_n X_n$ is a sequence of quantifiers for each variable, where each Q_i is either $\exists$ (existential) or $\forall$ (universal). We define a *Normal Form* QCSP as one with a strictly alternating sequence of quantifiers, starting with $\forall$ and ending with $\exists$. Each constraint C acts on its *scope*, an ordered subset of the variables $S_C = (X_i, \ldots, X_j)$, where $C \subseteq D_i \times \ldots \times D_j$. A *solution* is a tuple containing a value assignment for each variable in $\mathcal{X}$ which satisfies all of the constraints in $\mathcal{C}$. A *strategy* is a tree of value assignments which assigns a value for each existential variable for all possible sequences of assignments to the preceding universal variables. If every path in the strategy tree generates a solution tuple, it is a *winning strategy*. A QCSP is satisfiable if and only if a winning strategy exists, i.e. if we can guarantee to reach a solution no matter what values the universal variables take. Determining whether a winning strategy exists is PSPACE-complete[2]. Note that a QCSP may have solutions which are not part of any winning strategy. When generating binary QCSP problems, it is common[3] to generate only constraints where

the second variable is existentially quantified (i.e. $\exists\exists$ and $\forall\exists$ constraints), since other constraints can be removed during preprocessing.

For QCSP, a constraint C is *Strongly Quantified Generalised Arc Consistent* (SQGAC)[4] iff for each variable $X_i \in S_C$ and value $a \in D_i$, a vertex labeled $X_i \leftarrow a$ is contained in M, where M is a multiple winning strategy tree representing the union of all winning strategies for the constraint C. Any values not in the tree are pruned from their domains, and following each domain reduction the tree is updated to be a valid representation of all remaining winning strategies. SQGAC applied to binary QCSPs reduces to arc consistency for QCSPs[5] which we shall refer to as *Quantified Arc Consistency*(QAC), and can be implemented without a complex tree structure.

Gent *et al.*[3] proposed QCSP-Solve as a solver for QCSP, and included a *Pure Value Rule* for binary constraints, later extended to cover non-binary constraints[4]. A value a for variable X_i is *pure* if and only if all possible tuples with $X_i \leftarrow a$ are actual solutions to $\mathcal{P}_i$, the sub-problem containing only all constraints over the variable X_i. As domains are reduced during search, values may dynamically become pure. Existential pure values can be instantly assigned, while universal pure values can be safely pruned from their domains (assuming one other value still remains) to reduce search. Value ordering heuristics have been developed for QCSP, including Dynamic Geelen's Promise[6,7].

A difficulty in modeling using QCSPs is that in many problems some values for the universal variables are only legal depending on preceding decisions, and so no winning strategy is possible. To handle such problem types, Strategic CSPs [8] extend QCSPs by allowing universal variables to adapt their domains to be compatible with previous choices. Alternatively, QCSP$^+$[9] introduces restricted quantification to state when values are legal in the universal domains. [10] proposes backpropagation methods for value ordering in QCSP$^+$. To use QCSP-Solve on such problems, *shadow variables*[4] and associated constraints can be introduced, which make universal values pure when they are no longer legal choices, and thus they are removed from the search.

Dynamic Constraint Satisfaction [11,12] considers problems that change over time, by the addition, retraction or modification of variables, domains and constraints. Formalisms that specifically focus on problems that progress by the assignment of values to variables include Mixed CSP [13] and Stochastic CSP [14], although in the latter case probability distributions are associated with the assignment of each uncontrolled variable. Sampling methods have been used[15,16] to solve Online Stochastic Optimization problems under time constraints, in which problems are gradually revealed, although again with an assumption that there is some model of the likely growth. Finally, Groetschel [17] considers the general problem of Realtime Online Combinatorial Optimization.

In AI game playing [18], the game is represented as a tree of game states. Players take turns making moves, which are a transition from one state in the tree to another. Players perform game-tree search to determine their best option, but it is typically infeasible for the player to search the entire tree. Players are forced to form estimates of which move will lead them to a win. In practice, a

subtree of limited depth is generated and the leaf states are evaluated according to a problem specific heuristic function. These values are then propagated back up the tree to the current root state, in order for the player to make a decision. For adversarial zero sum games, most algorithms for performing this propagation are based upon the *minimax* heuristic [18], which states that a player should make the choice which minimises the (estimated) maximum that the opponent can achieve. When propagating up the tree, a state in which it is the player's move will take the value of its child with the highest estimate, since that is the minimum for the opponent. Conversely, a state in which it is the opponent's move will take the value of its child with the lowest estimate. The best known algorithms use variants of *Alpha-Beta Pruning*[19], which uses minimax based reasoning to prune moves which cannot improve on already discovered scores. [20] applied game-tree search in what they call Adversarial CSP, in which solving agents take turns to choose instantiations for variables in a shared CSP.

In a *bin packing* problem [21], we are given a list of items of varying size, and required to place them into a minimal number of fixed capacity bins without causing any of the bins to overflow. In an *online bin packing* problem, we must permanently assign each incoming item to a bin with no knowledge of the future remaining items to come. Two of the simplest heuristics to achieve this are First Fit (FF) and Best Fit (BF). First fit tries to place a packet into the first bin it can fit into. Best Fit places the packet into whatever bin will have the least space remaining after inserting the packet, and is equivalent to ordering the bins in descending fullness and then applying FF. Best Fit is known[22] to have worst case performance of 1.7 times as many bins as an off-line optimal algorithm, where the theoretical best possible by any online algorithm is 1.54, and an average waste of $O(n^{1/2}log^{3/4}n)$ bins.

3 Realtime Online Solving of QCSP

When solving realtime online CSPs using QCSP as a model, we treat the QCSP as a two-player game, in which one player (the existential player) assigns values to the existentially quantified variables, and the other (the universal player) assigns values to the universally quantified variables. The variables are assigned in the order of the quantifier sequence, and a time limit is imposed on each decision. For the existential the objective is to reach a solution, while for an adversarial universal it is to cause a failure (i.e. prevent a solution being reached). Formally, we define Realtime Online solving of QCSP for the existential as:

Definition 1 (Existential RO-QCSP). *Given a normal form QCSP $\mathcal{P}$, an increasing sequence of time points $t_1, t_2, \ldots, t_n$, and a sequence of values $v_1, v_3, v_5, \ldots, v_{n-1}$ such that each value v_j is in D_j and is revealed at time t_j, generate at each time t_k for $k = 2, 4, 6, \ldots, n$ a value $v_k \in D_k$ such that the tuple $(v_1, v_2, \ldots, v_n)$ is a solution for $\mathcal{P}$.*

When choosing each value v_k, the existential player has a known time limit for making the decision. A competent player should reason about the best value

to select, taking into account the possible future actions of the other player. If the time limit is sufficiently large, the first player can search for a winning strategy, and if it finds one it can simply execute this for each successive decision. However, we assume that finding a winning strategy will be initially infeasible, and instead we will generate partial strategies. The player looks ahead at possible future moves of both itself and the opponent, performing a partial exploration of the search tree. Different lookahead methods determine which area of the tree is explored. While exploring the tree, constraint propagation prunes unwanted branches. The player heuristically evaluates nodes as they are generated and propagates the evaluations back up to the root node. Once the time limit is reached, it selects the root value with the highest evaluation.

3.1 Constraint Propagation

The strongest level of consistency we consider is SQGAC. However, depending on the type of opponent, maintaining arc consistency can be detrimental, since our aim is simply to find any solution, and not necessarily a winning strategy. Consider a sequence of quantified variables $\exists X_1 \forall X_2 \exists X_3$, with domains $D_1 = D_3 = \{b, c\}$ and $D_2 = \{a, b\}$, and constraints $X_1 = X_3$ and $X_2 \neq X_3$. If we maintain arc consistency, then assigning $X_1 = b$ will remove c from D_3, causing b to be removed from D_2. At this point, we backtrack, since no winning strategy is now possible. For online problem solving with an adversarial opponent this is sensible, since when it reaches variable X_2 after $X_1 = b$, it is simple to detect that $X_2 = b$ removes all options for X_3 and thus the adversary would win. However, when a random opponent reaches X_2 after $X_1 = b$, it may still choose $X_2 = a$, and thus a solution is still possible. In this case, maintaining arc consistency may prevent us finding a solution. Therefore, to avoid losing solutions against random opponents but to keep some of the propagation power of arc consistency, we introduce *Existential Quantified Generalised Arc Consistency* (EQGAC):

Definition 2 (EQGAC). *A QCSP is* Existential Quantified Generalised Arc Consistent *(EQGAC) if for every X_i with $Q_i = \exists$, for all constraints C with $X_i \in S_C, \forall a \in D_i, \exists$ a tuple $t \in C$, s.t. each tuple element $t_j \in D_j$ and $t_i = a$.*

When all constraints are binary, EQGAC reduces to EQAC (*Existential Quantified Arc Consistency)*. Maintaining EQAC or EQGAC does not remove solutions from the problem. When maintaining EQAC, all $\exists\exists$ constraints are propagated like standard constraints in a CSP with MAC, while propagating $\forall\exists$ constraints never prunes values from the universal domain. Maintaining EQGAC uses the SQGAC algorithm[4], except we never place universal values on the remove list, and when removing a universal value from the tree, we do not remove sibling values, maintaining multiple *solution* trees, and not winning strategy trees. Maintaining SQGAC or EQGAC constructs a large tree in a preprocessing step, and as problem sizes increase, it becomes extremely expensive in time and space to construct and maintain these trees. To test the impact of this processing cost, we also implement solvers using forward checking. For non-binary QCSPs we extend

nFC0 [23] to QnFC0, which provides much weaker propagation, but requires no preprocessing and no significant extra data structures:

QnFC0: After assigning the current variable, achieve arc consistency on all constraints involving the current variable, past variables and exactly one future variable. If the future variable is existentially quantified, if the domain is not emptied, continue with a new variable, otherwise backtrack. If the future variable is universally quantified, if any value is removed from the domain, backtrack immediately, otherwise continue with a new variable.

3.2 Lookahead and Heuristics

The lookahead strategy and heuristic determine the sub-tree explored for each decision. Each strategy implements the general *lookahead* algorithm (Alg 1). *Nodes* is the main data structure, containing the unexpanded nodes in the search tree: each node represents a partial instantiation of the variables, in order, with the domains of uninstantiated variables reduced by propagation, and is $n_i(X_i, v_i, \sigma_i, p_i)$, recording the last instantiated variable, its value, the domains and propagation information, and its parent node. Initially *Nodes* contains a single node $n_0(\phi, \phi, \sigma_0, \phi)$, where σ_0 is the current state of the QCSP. We also maintain *Store*, a global store of evaluated nodes, initially empty. Different strategies implement *Nodes* differently, and are explained below.

Algorithm 1. lookahead(N,S) - the basic lookahead algorithm

```
   Input: Nodes,Store
   Data: τ, ϵ ;                                          // empty data structures
 1 n_i ← select-and-remove-node(Nodes) ;                 // select a node to expand
 2 if X_j ← next-variable(n_i) is not null then
 3     σ_i ← pure-value-rule(X_j, σ_i) ;                 // apply PV rule to X_j
 4     for each value w in X_j's domain do
 5         σ_w ← assign-and-propagate(X_j, w, σ_i) ;     // propagate the assignment
 6         τ ← τ + n_j(X_j, w, σ_w, n_i) ;               // record the new state
 7         e_w ← evaluate(σ_w) ;                         // evaluate by inspecting domains
 8         ϵ ← ϵ + (X_j, w, e_w) ;                       // record the evaluation
 9     Store ← Store + n_i(X_i, v_i, ϵ, p_i) ;           // store the old evaluated node
10     prop-eval(ϵ,n_i,Store) ;                          // propagate evaluations upwards
11     Nodes ← Nodes +τ ;                                // add new nodes to data structure
12 if Nodes is not empty and time remaining then
13     lookahead(Nodes, Store) ;                         // continue expanding nodes
```

Depth First(DF): Uses a stack; children are pushed onto the stack in order of generation; the next node to be expanded is popped off the top.

Breadth First(BrF): Uses a queue; children are added to the end in order of generation; the next node to be expanded is taken from the front.

Best First: Uses an ordered list; children are evaluated and inserted into the appropriate position; the next node to be expanded is taken from the front (with highest evaluation).

Partial Best First(PBF): Best First can perform poorly against adversarial opponents, as the best move for a universal variable is the worst for an existential variable and vice versa. PBF is a modification which behaves as best first for existential nodes, but when expanding a universal node the best child for the universal (lowest scored evaluation) will be immediately explored and the remaining children have their estimation negated and then are added to the ordered list.

Alpha Beta Pruning(AB): As depth first but using alpha and beta bounds to prune parts of the search tree which would never be reached, identified when the node is popped off the stack.

Intelligent Depth First(IDF): As DF, but where children are ordered from best to worst before being pushed onto the stack.

Intelligent Alpha Beta(IAB): As AB, but with children ordered as in IDF.

AB and IAB do not search to the bottom of the tree since they are too time consuming and would not finish within the time limit. Instead, we perform an iteratively deepening form of AB which searches to a fixed depth limit and then increases that depth limit before performing AB lookahead again. We continue until we have searched to the final variable (maximum depth) or we have run out of time. In our tests we set the initial depth limit to 2 and at each iteration we increase the depth limit by 1.

For general heuristics to evaluate states, we use *Dynamic Geelen's Promise* (DGP)[7] which is the product of the future existential domain sizes and was shown to be a good value ordering heuristic for solving QCSPs. Since we are looking ahead we need to compare states at different depths of the search tree, where the heuristic evaluations are the product of different numbers of future existential domains, and thus the DGP evaluations are incomparable. We present two different modifications to DGP for achieving that. *Proportional Promise*(PP) is calculated as DGP divided by the product of the original sizes of those future domains. The *Geometric Mean*(GM) of a heuristic is calculated as the nth root of the evaluation given by the heuristic. For random QCSPs, n is the number of future existential domains. For the online bin packing problems, we introduce two heuristics based upon the principals of First Fit (FF) and Best Fit (BF). The *Ordered Fitting* (OF) heuristic is based on First Fit, and prefers states in which the first bin is the most filled, the second bin is the second most filled, etc.. For a problem with a set of k bins, $b = \{b_1, b_2, .., b_k\}$ of maximum capacity c, where f_i is how full the i^{th} bin is, we calculate OF as $\Sigma_{i=1}^{k} f_i * c^{k-i}$. The *Heavily Filled* (HF) heuristic is based on Best Fit, and prefers states in which bins are as highly filled as possible and the rest empty, to those in which many bins are only partially filled. We calculate HF as $\Sigma_{i=1}^{k} \sqrt{f_i/c}$. For applying GM in bin packing, n is the number of packets which have already arrived (since OF/HF evaluations increase with depth, unlike DGP evaluations which decrease).

To propagate heuristic evaluations back up the tree, we use either Minimax, or what we term *Weighted Estimates*(WE) reasoning. At each level, we switch the method of aggregating child scores depending on the quantifier of the parent variable. In a node where an existential variable's valuation is being decided,

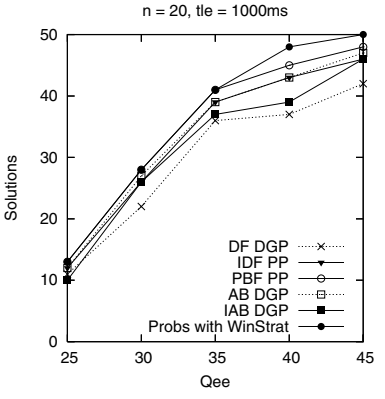

Fig. 1. Random RO-QCSP against universal using AB with no time limit

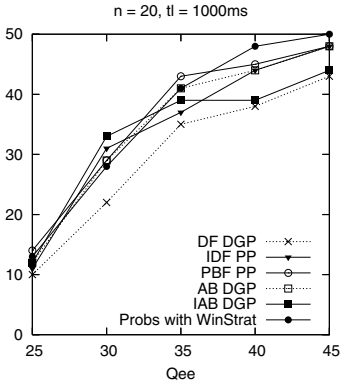

Fig. 2. Random RO-QCSP against universal using DF with time limit

both methods select the highest of the children's valuations. In a node where a universal variable's valuation is being decided, minimax selects the lowest of the children's valuations, while Weighted Estimates computes the average of the children's valuations. While minimax reports that certain values lead to failure, weighted estimates gives an indication of how likely it is an unintelligent opponent can pick a value which causes domain wipe outs.

4 Experiments: Random Binary QCSPs

We tested on randomly generated binary QCSPs with a strictly alternating sequence of $\exists$ and $\forall$ quantifiers, using the flawless generator described in Section 6 of [7]. In all our tests, the block size = 1, domain size = 8, constraint density = 0.20, $q_{\forall\exists} = 1/2$. $q_{\forall\exists}[3]$ is looseness for $\forall\exists$-constraints, $q_{\exists\exists}$ is the looseness of $\exists\exists$-constraints, and n is the number of variables. We denote as tl the time limit for both players, and tle the time limit for the existential. For each value of $Q_{\exists\exists}$ we generated 50 random problems. We measure the number of solutions reached. We omit many of the combinations of lookahead methods, propagation and heuristics from our graphs for clarity. Unless explicitly stated to be using Weighted Estimates(WE), all strategies are using Minimax. Both participants receive the same time limit, tl, per move unless explicitly stated otherwise. For the smaller problems with 20 variables we also show how many problems contained a winning strategy. For larger problems, $n = 51$, finding a winning strategy in reasonable time was not possible. Experiments ran on a 2.0GHz Pentium with 512MB RAM.

Figure 1 shows the number of solutions against an opponent performing a complete lookahead with infinite time on problems with $n = 20$. This is the best a universal player is able to perform - if there is no winning strategy for

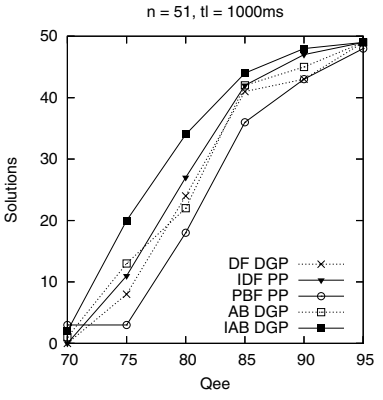

Fig. 3. Random RO-QCSP against universal using DF

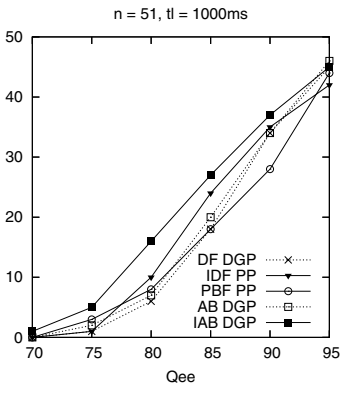

Fig. 4. Random RO-QCSP against universal using DGP AB

the QCSP or if the existential makes any choice which is not part of a winning strategy, the universal will win. We see that our time-limited existential solvers are still able to find most of the winning strategies, with PBF-PP finding them all for lower levels of Q_{EE}. Figure 2 shows performance against a time-limited universal opponent using depth-first lookahead, also with $n = 20$. Against this weaker universal, the existentials improve, and in a number of cases at lower Q_{EE} achieve solutions even when the QCSP has no winning strategy. In both figures, we see that PBF-PP outperforms AB and IAB; we believe that on these relatively small problems PBF-PP is able to search enough of the space to make intelligent decisions, while AB and IAB's higher overhead restricts their search. Figures 3 and 4 show performance on larger problems against weak and strong opponents respectively, and we see that IAB is now outperforming PBF-PP, finding solutions for up to five times as many problems at lower Q_{EE}. The value ordering used by IAB also allows it to prune the search space faster than AB, and thus enables it to search to a deeper level and make more informed decisions.

For problems against a random opponent, again we set $n = 20$, but reduce the time limit to 500ms, as it is significantly easier for the existential player to succeed. In Figure 5 the existential maintains QAC, while in Figure 6 it maintains EQAC. In both cases, IAB and methods using WE find many solutions even when the problem has no winning strategy. The WE approaches, which estimate the likelihood of an opponent picking a particular value, benefit significantly from the use of EQAC, and outperform IAB in Figure 6.

5 Modeling Online Bin Packing

We test on two types of Online Bin Packing problems in which the universal player selects packet sizes, while the existential player attempts to place them in

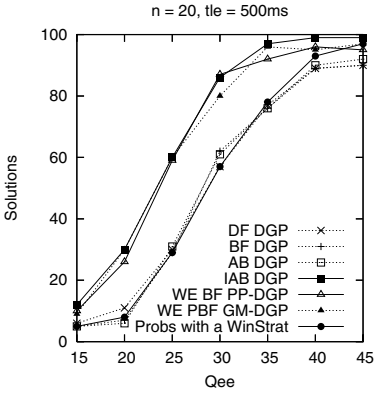

Fig. 5. Random RO-QCSP using QAC against random universal

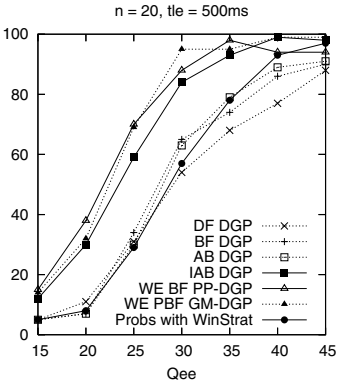

Fig. 6. Random RO-QCSP using EQAC against random universal

the bins. We present the models in an abstract form, in which we assume pruning the universals is unrestricted, to provide a clear and concise description. The footnotes describe how we transform this to a correct quantified form, through use of shadow variables and pure value pruning. The basic model is common to both problems and we describe it first. A known number of packets will be chosen from a limited set, and there is a fixed number of bins, each of the same capacity. We use state variables for each bin to record how much capacity it has left, we have decision variables for the universal which determine the size of each packet, and decision variables for the existential to state into which bin the current packet will be placed. Note that no lookahead is performed before assigning state variables, since they are uniquely determined by previous choices. As an example, the variables for the j^{th} packet choice are

$$\exists a_{(j-1)b_1} \exists a_{(j-1)b_2} \ldots \exists a_{(j-1)b_k} \forall p_j \exists l_j \exists a_{(j)b_1} \exists a_{(j)b_2} \ldots \exists a_{(j)b_k}$$

where $a_{(j-1)b_i}$ is the state of bin b_i before the j^{th} packet arrives, p_j is the size of the j^{th} packet, l_j is the bin the j^{th} packet is placed into, and $a_{(j)b_i}$ is the state of bin b_i after the j^{th} packet has been placed. The following constraints for each j and i ensure the state variables are consistent[1]:

$$(l_j = b_i) \Rightarrow a_{(j)b_i} = a_{(j-1)b_i} - p_j$$
$$(l_j \neq b_i) \Rightarrow a_{(j)b_i} = a_{(j-1)b_i}$$

[1] Note that this describes the abstract model. The limited set of possible packets constrains the universal choices, and so in the implementation we use shadow variables to render illegal universal values pure, and modify the constraints accordingly. Each universal variable p_j has an existential shadow variable sp_j placed immediately after it in the variable sequence (and they will be linked in later constraints). Thus the variables for the j^{th} packet choice become $\ldots \exists a_{(j-1)b_k} \forall p_j \exists sp_j \exists l_j \ldots$, and we replace the first constraint by $(l_j = b_i) \Rightarrow a_{(j)b_i} = a_{(j-1)b_i} - sp_j$.

5.1 Type 1 Problems

In type 1 problems, the universal player has a fixed set of m packets, for which the sum of their sizes is the value B, and must decide on which order to provide them to the existential player. By testing with both a random and an adversarial universal we can evaluate the existential's performance against both average case and worst-case order scenarios for randomly generated sets of packets. The existential player does not know the sizes of the packets before they arrive, but does know the upper bound on the sum of their sizes. Thus the existential is less informed than the universal and the two actually assign values in slightly different synchronised problem models.

We represent the fixed set of packets with a single global cardinality constraint over the p_j variables[2]:

$$gcc(p_1, p_2, \ldots, p_m, c_{s_1}, c_{s_2}, \ldots, c_{s_t}).$$

The domain for each p_j is a set $\{s_1, s_2, ..., s_t\}$ of possible sizes, and the c_{s_i} state exactly how many of the p_j must take each value s_i. Note that the c_{s_i} in this case are constants, rather than constrained variables. The existential player does not see the gcc constraint; instead it sees a less restrictive global sum constraint[3]:

$$\Sigma_{j=1}^{m} p_j \leq B$$

In the Type-1 problems, our aim is to show how the existential player can improve over strategies like First Fit or Best Fit, even when its perception of the problem is more restricted than that of the opponent.

5.2 Type 2 Problems

In the second type of problems, the universal and existential players both share the same problem. This time the list of packets for the universal is *larger* than

[2] The universal value must respect the gcc constraint. Instead of posting the gcc constraint, for each p_i we post an extensional constraint with scope $(sp_1, sp_2, ..., sp_{i-1}, p_i, sp_i)$, such that each possible tuple satisfies the following rule: if the values for the sp_j should disallow a value v for p_i ($p_i \leftarrow v$ would cause a violation of the gcc constraint), then $p_i \leftarrow v$ is compatible with all values of sp_i, and otherwise, $p_i = sp_i$. Thus as soon as choices for some p_j disallow a value v for p_i, v becomes pure; immediately before p_i is to be assigned, the pure value rule removes v from its domain.

[3] As before, we replace this with a shadow variable form for each p_i with scope $(sp_1, sp_2, ..., sp_{i-1}, p_i, sp_i)$, as an extensional constraint which implements the rule: if the values of the sp_j plus the number of remaining packets would disallow v for p_i ($p_i \leftarrow v$ would cause the sum to exceed B), then $p_i \leftarrow v$ is compatible with all values of sp_i, and otherwise $p_i = sp_i$. Note that this never makes a value pure if it has not also been made pure in the universal's problem model.

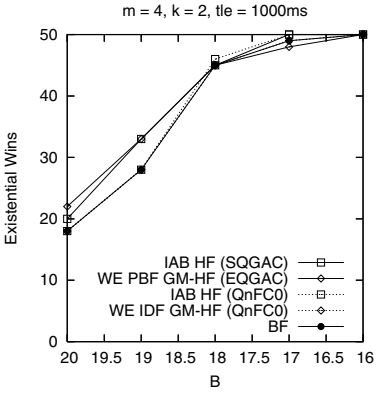

Fig. 7. Bin packing type I against random universal

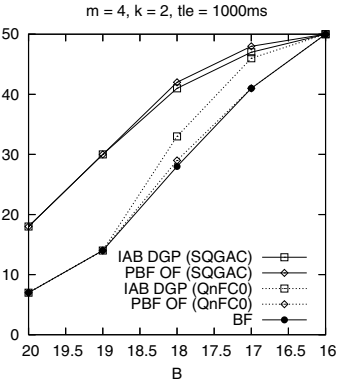

Fig. 8. Bin packing type I against universal using IAB OF

the number of packets it must pick. However, the subset of packets it can pick is restricted by an upper bound, B, on their combined size[4]:

$$gcc(p_1, p_2, \ldots, p_m, v_{s_1}, v_{s_2}, \ldots, v_{s_t})$$
$$\Sigma_{j=1}^{m} p_j \leq \mathrm{B}$$

where the v_{s_i} are now variables, each of which has its own upper bound and a lower bound of 0. In these type-2 problems, the universal has more freedom as to what values to pick and in what order, and our aim is to show that by exploiting constraint propagation, lookahead and heuristics the universal can have a significant effect on the success rate of the existential.

6 Experiments: Online Bin Packing

In our Online Bin Packing problems, packets range in size from 1 to 10 and each bin's capacity is 10. We test 50 problems, each with a different randomly generated list of packets, at each of the different upper bounds on the sum of the sizes of all packets. The size of the list of packets in Type 2 problems is twice the number of incoming packets. The number of incoming packets is m, the number of bins is k, and the time limit per decision for the existential is tle. In all problems the universal has 1000ms per decision. Again, many of the lookaheads and heuristics are omitted from these graphs for clarity. In general, we plot the best AB-based heuristic+lookahead combination, and the best non-AB heuristic+lookahead combination, with other relevant combinations shown

[4] The shadow variable form is an extensional constraint for each p_i with scope $(sp_1, sp_2, ..., sp_{i-1}, p_i, sp_i)$ such that if the sp_j values disallow v for p_i (due to the gcc constraint or the upper bound constraint), then $p_i \leftarrow v$ is compatible with all values of sp_i, and otherwise $p_i = sp_i$.

when appropriate. As a baseline for our tests, we compare our results against an existential using Best Fit, as it is always at least as good as First Fit.

Figures 7 and 8 show the results for small Type 1 problems against Random and IAB OF universals respectively. Against a random opponent we see that the simple BF strategy does well, and our combination of propagation, heuristics and lookahead only achieves a small performance gain over it. Against an adversarial opponent however we see that we can perform significantly better than BF. We also note that the existential using QnFC0 propagation instead of SQGAC or EQGAC performs worse due to the reduced amount of propagation. In all these tests the universal is using SQGAC.

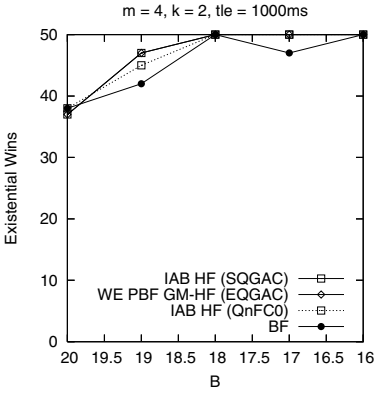

Fig. 9. Bin packing type II against random universal

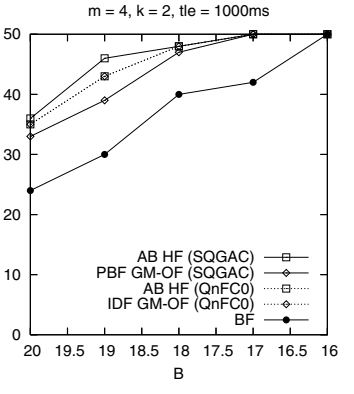

Fig. 10. Bin packing type II against universal using IAB OF

Figures 9 and 10 show the results for small Type 2 problems. The relative performance is similar to Type 1. In Figure 11 we increase the problem size, and use QnFC0 propagation for every player, as SQGAC is too slow on large problems. These larger problems reveal an flaw with IAB OF for the universal, and IAB DGP for the existential player exploits it, obtaining exceptionally good results. A universal using IAB OF essentially assumes the existential wants to maximise the content of the first bin, and so initially picks a small packet to be placed into it. However, IAB DGP's implementation places the first packet into the final bin, so the universal ends up continually feeding small packets expecting them to be placed into the first bin, making it easy for the existential to win.

To overcome this flaw, we develop a new heuristic intended for the universal called *MinSpace*(MS). MS tries to leave a minimal non-zero empty space in each bin. By leaving these small gaps, it makes it hard for the existential to succeed at high upper bounds. We calculate the MS measure using $\Sigma_{i=1}^{k} g(i)$, where $g(i)=0$ when $c - f_i = 0$, and $g(i)=\sqrt{(c-f_i)/c}$ otherwise. Figure 12 shows a universal using MS on the same problems as in Fig. 11. As can be seen the performance

of the universal is drastically improved. When both the existential and universal have 1000ms time limits, the existential struggles to do well against MS and only achieves close to the performance of BF, as shown by AB OF (1000ms). The remainder of the plots in Figure 12 show how well we can do when the existential's time limit is raised to 5000ms. With this additional time on these larger problems, we can achieve many more solutions than BF.

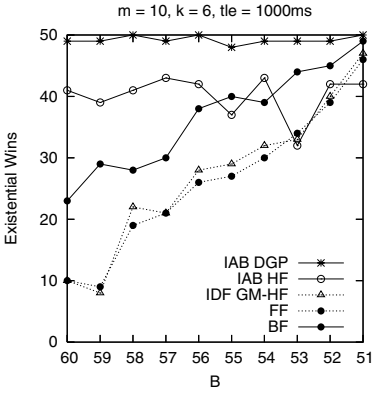

Fig. 11. Bin packing type II against universal using IAB OF

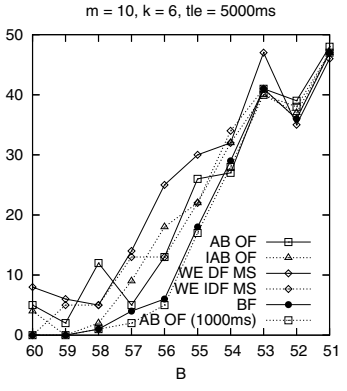

Fig. 12. Bin packing type II against universal using IAB MS

We also compared against a universal using a policy of picking the largest packet it can at each turn. Due to the nature of this universal policy, our choice of existential strategy has no effect until far into the packet stream and BF and most of the heuristics perform almost identically against it. A universal using IAB MS performs consistently significantly better than this Largest First approach at all upper bounds.

7 Conclusions and Future Work

Quantified CSPs can be used as a model for solving online CSPs, generating winning strategies in advance. But when decisions in the online problem have to be made in realtime, complete solving of a QCSP is infeasible. We have developed techniques for realtime online solving of QCSP using a combination of propagation, lookahead and heuristics, for online CSPs involving both adversarial opponents and random external solvers. We have proposed existential quantified generalised arc consistency for handling random solver opponents, which allows us to achieve solutions even when the underlying QCSP has no winning strategy. We have demonstrated that a version of alpha-beta pruning with a constraint-based value-ordering heuristic outperforms other heuristics on large binary QCSPs against adversarial opponents. We have developed a non-binary constraint model of Online Bin Packing, and we have shown that with

good heuristic selection, a significantly stronger universal player can be generated using our reasoning, but that against even a strong opponent the existential reasoning can help us reach more solutions.

In future work we will consider weaker consistency levels than SQGAC or EQGAC to avoid the large overhead, we will investigate the use of other forms of quantified constraint problems for realtime online problem solving, and we will consider solution methods based on sampling.

References

1. Bordeaux, L., Cadoli, M., Mancini, T.: CSP Properties for Quantified Constraints: Definitions and Complexity. In: Proceedings of AAAI, pp. 360–365 (2005)
2. Börner, F., Bulatov, A., Jeavons, P., Krokhin, A.: Quantified constraints: Algorithms and complexity. In: Baaz, M., Makowsky, J.A. (eds.) CSL 2003. LNCS, vol. 2803, pp. 58–70. Springer, Heidelberg (2003)
3. Gent, I.P., Nightingale, P., Stergiou, K.: QCSP-Solve: A solver for quantified constraint satisfaction problems. In: Proceedings of IJCAI, pp. 138–143 (2005)
4. Nightingale, P.: Consistency and the Quantified Constraint Satisfaction Problem. PhD thesis, University of St Andrews (2007)
5. Bordeaux, L., Monfroy, E.: Beyond NP: Arc-consistency for quantified constraints. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 371–386. Springer, Heidelberg (2002)
6. Stynes, D., Brown, K.N.: Value Ordering for Quantified CSPs. In: Proceedings of CP2007 Doctoral Programme, pp. 157–162 (2007)
7. Stynes, D., Brown, K.N.: Value Ordering for Quantified CSPs. Constraints 14(1), 16–37 (2009)
8. Bessiere, C., Verger, G.: Strategic constraint satisfaction problems. In: Proceedings of CP Workshop on Modelling and Reformulation, pp. 17–29 (2006)
9. Benedetti, M., Lallouet, A., Vautard, J.: QCSP made Practical by Virtue of Restricted Quantification. In: Proceedings of IJCAI, pp. 38–43 (2007)
10. Verger, G., Bessiere, C.: Guiding Search in QCSP$^+$ with Back-Propagation. In: Stuckey, P.J. (ed.) CP 2008. LNCS, vol. 5202, pp. 175–189. Springer, Heidelberg (2008)
11. Dechter, R., Dechter, A.: Belief maintenance in dynamic constraint networks. In: Proceedings of AAAI, pp. 37–42 (1988)
12. Brown, K.N., Miguel, I.: Uncertainty and Change. In: Handbook of Constraint Programming, ch. 21, pp. 731–760 (2006)
13. Fargier, H., Lang, J., Schiex, T.: Mixed constraint satisfaction: a frameworkfor decision problems under incomplete knowledge. In: Proceedings of AAAI, pp. 175–180 (1996)
14. Walsh, T.: Stochastic constraint programming. In: Proceedings of ECAI, pp. 111–115 (2002)
15. Bent, R., van Hentenryck, P.: Regrets only! online stochastic optimization under-time constraints. In: Proceedings of AAAI, pp. 501–506 (2004)
16. Hentenryck, P.V., Bent, R.: Online Stochastic Combinatorial Optimization. The MIT Press, Cambridge (2006)
17. Grötschel, M., Krumke, S.O., Rambau, J., Winter, T., Zimmermann, U.T.: Combinatorial Online Optimization in Real Time. Online Optimization of Large Scale Systems, 679–704 (2001)

18. Shannon, C.E.: Programming a computer for playing chess. Philosophical Magazine (Series 7), 256–275 (1950)
19. Knuth, D.E., Moore, R.W.: An Analysis of Alpha-Beta Pruning. Artificial Intelligence 6(4), 293–326 (1975)
20. Brown, K.N., Little, J., Creed, P.J., Freuder, E.C.: Adversarial constraint satisfaction by game-tree search. In: Proceedings of ECAI, pp. 151–155 (2004)
21. Johnson, D.S.: Fast Algorithms for Bin Packing. Journal of Computing and System Sciences 8(3), 272–314 (1974)
22. Version, P., Kenyon, C., Rabani, Y., Sinclair, A.: Biased Random Walks, Lyapunov Functions, and Stochastic Analysis of Best Fit Bin Packing. J. Algorithms, 351–358 (1998)
23. Bessière, C., Meseguer, P., Freuder, E.C., Larrosa, J.: On forward checking for non-binary constraint satisfaction. In: Jaffar, J. (ed.) CP 1999. LNCS, vol. 1713, pp. 88–102. Springer, Heidelberg (1999)