Weighted Super Solutions for Constraint Programs*

Alan Holland and **Barry O'Sullivan**
Cork Constraint Computation Centre
University College Cork, Ireland
email: {a.holland,b.osullivan}@4c.ucc.ie

Abstract

Super solutions to constraint programs guarantee that if a limited number of variables lose their values, repair solutions can be found by modifying a bounded number of assignments. However, in many application domains the classical super solutions framework is not expressive enough since it only reasons about the number of breaks in a solution and the number of changes that are necessary to find a repair. For example, in combinatorial auctions we may wish to guarantee that we can always find a repair solution whose revenue exceeds some threshold while limiting the cost associated with forming such a repair. In this paper we present the *weighted super solution* framework that involves two important extensions. Firstly, the set of variables that may lose their values is determined using a probabilistic approach enabling us to find repair solutions for assignments that are most likely to fail. Secondly, we include a mechanism for reasoning about the cost of repair. The proposed framework has been successfully used to find robust solutions to combinatorial auctions.

Introduction

Solutions are *robust* if a repair solution is available should some assignments become invalid (break). A *super solution* to a constraint program guarantees that if the solution breaks, another solution can be found by changing a limited number of other assignments (Hebrard, Hnich, & Walsh 2004b). It is a generalization of both supermodels in propositional satisfiability (SAT) (Ginsberg, Parkes, & Roy 1998) and fault tolerance in constraint programming (CP) (Weigel & Bliek 1998). This differs from Branching Constraint Satisfaction (Fowler & Brown 2000) which focuses on finding robust partial solutions to a dynamically changing problem.

In the classical super solutions framework, an (a, b)-super solution guarantees that if at most a variables lose their assignments, a repair solution can be found by reassigning those a variables and at most b others. Therefore, a super solution is a preventative approach to dealing with uncertainty that ensures solution robustness. In many application domains, such as combinatorial auctions, we not only need to be able to guarantee solution robustness, but we need to be able to reason about the *cost of repair*, since transitions to some repairs may be more costly than others. We also propose a probabilistic approach to reasoning about *how failure is likely to occur* in a solution, thus facilitating contingency planning where it is needed. Therefore, the weighted super solution (WSS) framework, proposed here, extends the classical super solutions framework in two important ways: firstly, it can model the most likely causes of failure in a solution and secondly, can reason about the costs of forming alternative repairs.

The framework provides for two alternative approaches to describing the likelihood of failure: using *static* probabilities of failure or *dynamic* failure rates, i.e. time-dependent probability of failure. Using the *Weibull distribution* (Weibull 1951) we can represent many of the most common failure distributions such as normal, lognormal and exponential.

It is equally important that we can reason about the costs associated with alternative ways of repairing a solution. Changing the values of certain variables may incur a heavier cost than others, e.g. changing the production schedule on a large production line may be easier than on a smaller one.

The feasibility of finding robust solutions for combinatorial auctions was explored in (Holland & O'Sullivan 2004). This work motivated the necessary extensions to the super solutions framework proposed in this paper. An extensive study of robust solutions for combinatorial auctions using the WSS framework was conducted in (Holland & O'Sullivan 2005). The revenue of robust solutions for different bid distributions, non-incentive compatibility, positive discrimination in favor of trusted bidders and an auction mechanism for increased reparability were amongst the issues discussed in that work.

In this paper we present the weighted super solution framework. We present a MAC-based (Sabin & Freuder 1994) search algorithm for establishing a WSS, provide complexity results and demonstrate the framework on job-shop scheduling problems and combinatorial auctions.

The Weighted Super Solution Framework

The WSS framework uses probabilistic failures to determine the sets of variables that require repairs, as well as the costs of establishing such repairs. We define both *static* and *dynamic weighted super solutions* in terms of robustness and cost of repair.

*This work has received support from Science Foundation Ireland under grant number 00/PI.1/C075.

Definition 1 (Static WSS) *A solution to a CSP is a static weighted super solution, or (α,β)-static WSS, if any set of variables whose probability of losing their current assignments is greater than or equal to α, can be repaired by reassigning other values to these and other variables with a repair cost of at most β.*

Probabilistic failure rates can also be used to determine the sets of variables that require repairs. The only difference between static and dynamic weighted super solutions is in the selection of assignments for which repairs are necessary. We define *dynamic weighted super solutions* in terms of robustness and cost of repair as follows:

Definition 2 (Dynamic WSS) *A solution to a CSP is a dynamic weighted super solution, or (α,β,τ)-dynamic WSS, if any set of variables whose probability of losing their current assignments is greater than or equal to α before time τ, can be repaired by reassigning other values to these and other variables with a repair cost of at most β.*

Modeling Probabilistic Failure

As discussed above, breaks in solutions to constraint programs may or may not be time-dependant. For example, a solution to a combinatorial auction results in a break that is effectively immediate when a bidder refuses to pay. Therefore a constant probability of failure can be associated with each variable assignment. A factory scheduling problem however, may exhibit failure rates over a period of time.

Constant probabilities of failure (Static WSS). We propose that *assignments* have varying degrees of robustness to differentiate between those that are more or less likely to fail. We assume that assignments have independent probabilities of failure. Repair solutions can be determined for sets of assignments that are likely to fail, whilst more robust assignments may not require any repair solution if their probability of failure is below some threshold. Probabilistic robustness may be particularly useful in recurring scenarios where historical information pertaining to the reliability of assignments is available.

Probabilistic rates of failure (Dynamic WSS). The failure rates of mechanical components or electronic devices are typically defined in terms of probability distributions over time. In general, the type of failure distribution depends upon the component's inherent failure mechanisms.

The Weibull distribution can be used to model a variety of distributions including normal, lognormal, exponential and Rayleigh (Weibull 1951). It is the most widely used distribution in reliability engineering, thus ideal for simulating probabilistic failure rates in the WSS framework for the purposes of determining the sets of assignments that require repairs.

The probability density function of the 2-parameter Weibull distribution is defined as follows:

$$f(t) = \frac{\gamma}{\eta}(\frac{t}{\eta})^{(\gamma-1)} e^{(-(\frac{t}{\eta})^{\gamma})} \quad (1)$$

where γ and $\eta > 0$. γ is the shape parameter of the distribution. If γ is greater than 1, the failure rate is increasing; if γ is less than 1, the failure rate is decreasing; $\gamma = 1$ implies the failure rate is constant. η is the scale parameter and is also known as the characteristic life, where $\frac{1}{\eta}$ is defined as the time at which there is a 0.632 probability that failure will have occurred.

The cumulative distribution function (CDF) is the probability of failure before time t. This is relevant for the scenario in which we are interested, i.e. calculating which sets of assignments are likely to fail in a given time-frame. The CDF for the 2-parameter Weibull distribution is as follows:

$$F(t) = 1 - e^{-(\frac{t}{\eta})^{\gamma}}, t \geq 0; \gamma > 0; \eta > 0. \quad (2)$$

We use the Weibull CDF to describe the failure rates of assignments whose probability of failure is time-dependent. When this probability is at least some threshold, α, at a given time, τ, then the corresponding assignment is deemed brittle and requires a repair solution.

Modeling Repair Cost

The super solution framework guarantees the availability of repair solutions in cases where up to a variables may break and b other variables are allowed to change. The approach assumes that the cost of changing all variables is the same and that the cardinality of the repair set is a reasonable measure of the total cost of repair. However, in general, one needs to be able to differentiate between the costs of alternative repairs.

The cost of changing an assignment in a solution may depend on several parameters which, at the very least, include the original and final assignments, as well as the break variables themselves. This provides a more flexible and accurate description of the cost of making changes to a solution. For example, in a job-shop scheduling problem the values associated with variables may represent the various states of a machine and the cost of alternative state transitions may differ. Furthermore, this cost may be influenced by the cause of the break in the solution.

Definition 3 (Cost of Repair) *The cost of repair is represented as a non-negative real number that describes the cost associated with changing the value of variable x from v_1 to v_2 when A is the set of break variables, $C^{(x,A)}_{v_1 \to v_2} \in \mathbb{R}^+_0$.*

If at most k variables participate in any potential break, there are $\binom{n}{k}$ possible break sets, therefore the space complexity for storing the costs becomes $O(n^{k+1}d^2)$ when represented extensionally.

The total cost of repair is computed using a function, f, that combines the repair costs of the individual assignments that are modified to form the repair solution. Typically, a summation of the costs is used to determine the overall cost of repair.

The algorithm that we propose in this paper requires that f is monotone non-decreasing in the size of the repair set so that we can terminate the search for a WSS when an assignment is deemed irreparable. If R_1 and R_2 are two repair sets and $R_1 \subset R_2$, then $f(R_1) \leq f(R_2)$.This restriction is necessary because it allows us to cease searching for a repair solution in a branch once a threshold cost, β, has been

exceeded. Otherwise, the search for repair solutions would require the computation of a lower bound at each node of the search for a repair. This search would become prohibitively expensive without this restriction on the cost function. Concurrent search for repair solutions would become computationally infeasible.

Algorithm

The WSS framework requires several modifications to the algorithms used to find classical super solutions (Hebrard, Hnich, & Walsh 2004a; 2004b), to support repair solutions for *sets of break variables* of different size and repair sets of *arbitrary cardinality*. Algorithm 1 (`weighted-super-solve`) can be used to find both static and dynamic WSS. When τ is not defined, denoted $\emptyset$, the algorithm detects that a static WSS is required in the `failure` procedure and determines an assignment's probability of failure using its probabilities, $\alpha_{(x,v)}$, otherwise Weibull parameters $\gamma_{(x,v)}$ and $\eta_{(x,v)}$ are used to compute the probability of failure by time τ.

A repair solution, R_b, is provided for every possible set of break variables b. The `backtrack` procedure is called from `weighted-super-solve` and attempts to extend the current partial assignment by choosing a variable and assigning it a value. Backtracking may occur for one of two reasons: we cannot extend the current partial assignment to satisfy the given constraints, or it cannot be associated with a repair solution whose cost is less than or equal to β for a possible break. The procedure `reparable` searches for *partial* repair solutions using backtracking and attempts to extend the last repair found, just as in (1,b)-super solutions; the differences being that a repair is provided for a set of break variables rather than a single variable and the cost of repair is considered. The procedure `check-wss-repair` determines the cost of the repair solution and verifies consistency. A summation operator is used to determine the overall cost of repair in this case. This procedure can also include auxiliary break and repair restrictions described in (Hebrard, Hnich, & Walsh 2004a).

Algorithm 1: weighted-super-solve

input : α, β, τ, RepairCosts: I, CSP: P=$\{\mathcal{X}, \mathcal{D}, \mathcal{C}\}$ // Let $\tau = \emptyset$ for static-WSS
output: S: an (α,β,τ)-WSS: R: the set of repair solutions
begin
  $S \leftarrow \emptyset$ // Solution
  $R \leftarrow \emptyset$ // Set of repair solutions
  $Past \leftarrow \emptyset$ // Ordered set of assigned variables
  AC(P, S) // Perform arc-consistency
  backtrack(P,S,$Past$,R,0,α,β,τ,0)
end

Theorem 1 `weighted-super-solve` *terminates and is sound and complete.*

Proof. (Sketch)
Termination: The algorithm never revisits any partial assignment or repair for a given break set, of which there are

Procedure `backtrack`(P,S,$Past$,R,lvl,α,β,τ,m) :Boolean

begin
  if $\mathcal{X} = Past$ **then** return true
  choose $x \in \mathcal{X} \setminus Past$
  $b \leftarrow \emptyset$ // set of break variables
  $Past[lvl] \leftarrow x$
  foreach $v \in \mathcal{D}(x)$ **do**
    save $\mathcal{D}$, m and R
    $m \leftarrow \max(m, failure(\{x\}, S, \tau))$
    $k \leftarrow \lfloor \log_m \alpha \rfloor$ // Max size of any break
    $S \leftarrow S \cup \{(x, v)\}$
    if *AC(P,S)* **then**
      foreach $b \in \mathcal{P}(Past)$, $\mid b \mid \leq k$, *failure*(b, S, τ) $\geq \alpha$ **do**
        if ¬*reparable*(P,S,$Past$,R_b,0,β) **then** break
        if *backtrack*(P,S,$Past$,R,$lvl+1$,α,β,τ,m) **then**
        return true
    restore $\mathcal{D}$, m and R
    $S \leftarrow S \setminus (x, v)$
  $Past[lvl] \leftarrow \emptyset$
  return false
end

Procedure `reparable`(P,S,$Past$,R_b,lvl,β) :Boolean

begin
  if $lvl = |S|$ **then** return true
  $y \leftarrow Past[lvl]$
  for $v \leftarrow R_b[y]$ *to* $\max_{init} \mathcal{D}(y)$ *// Last value in lex order*
  do
    if $y \notin b$ *or* $S[y] \neq v$ **then** $R_b[y] \leftarrow v$
    if *check-wss-repair*(P,S,$Past$,R_b,lvl,β) **then**
      if *reparable*(P,S,$Past$,R_b,$lvl+1$,β) **then** return true
  $R_b[y] \leftarrow \min(\mathcal{D}(y))$
  return false
end

Procedure `check-wss-repair`(P,S,$Past$,R_b,lvl,β) : Boolean

begin
  $cost \leftarrow 0$
  for $i \leftarrow 0$ *to* lvl **do**
    $y \leftarrow Past[i]$
    if $y \notin b$ *and* $R_b[y] \neq S[y]$ **then**
      $cost \leftarrow cost + I(x, S[y], R_b[y])$
  if $cost > \beta$ **then** return false
  return consistency of the l first values in R_b
end

Procedure `failure`(b, S, τ) : Real

begin
  $fail \leftarrow 1.0$
  for $x \in b$ **do**
    $v \leftarrow S[x]$
    if $\tau = \emptyset$ **then** $fail \leftarrow fail \times \alpha_{(x,v)}$ // Static WSS
    else $fail \leftarrow fail \times$ CDF($\eta_{(x,v)}$,$\gamma_{(x,v)}$,τ) // Dynamic WSS
  return $fail$
end

finitely many.
Soundness: $\forall b \in \mathcal{P}(Past)$, R_b is the first repair solution of S, when taken in lexicographical order, for b in the problem restricted to $Past$.
Completeness: MAC (Sabin & Freuder 1994) is complete, therefore no partial assignment is omitted before checking for reparability. The check for reparability starts from the last repair found. The cost of repair function is monotone non-decreasing so no assignment before this last repair in the search tree can be extended to the current variable because each prior partial assignment had a minimum cost of repair that exceeded β. □

We also show that finding a WSS is $\mathcal{NP}$-complete in general for any fixed α (and τ for the case of dynamic WSSs). We define two decision problems: (α,β)-STATIC WEIGHTED SUPER SOLUBILITY and (α,β,τ)-DYNAMIC WEIGHTED SUPER SOLUBILITY as the problems of deciding whether there exists an (α,β)-static WSS or an (α,β,τ)-dynamic WSS to a problem, respectively.

Lemma 1 *Let $m = \max(\bigcup_{(x,v)\in s} \alpha_{(x,v)})$, where $\alpha_{(x,v)}$ is the constant probability of failure associated with the assignment of value v to variable x for static-WSS and the probability of failure at time τ in the dynamic case. If $\lfloor \log_m \alpha \rfloor$ is bounded by a constant k, the number of possible breaks requiring repair solutions is polynomial in k.*

Proof. For each solution S there must be a repair solution for each subset $s \in \mathcal{P}(S)$, the power-set of S, whose probability of failure is greater than or equal to α, i.e. $\prod_{(x,v)\in s} \alpha_{(x,v)} \geq \alpha$. But $\alpha_{(x,v)} \leq m$, so we can say that:

$$\prod_{(x,v)\in s} \alpha_{(x,v)} \leq m^{|s|}.$$

If s requires a repair we have

$$\alpha \leq m^{|s|} \therefore \log \alpha \leq \mid s \mid \cdot \log m.$$

Since $\log \alpha$ and $\log m$ are both negative:

$$\frac{\log \alpha}{\log m} \geq \mid s \mid \therefore \log_m \alpha \geq \mid s \mid.$$

Since $\lfloor \log_m \alpha \rfloor \leq k$ and $\mid s \mid \in \mathbb{Z}$, $k \geq \mid s \mid$.

The size of the largest break-set needing consideration for repair is $\leq k$, therefore at most $\binom{n}{k}$ repair solutions are necessary, where n is the number of variables. □

Theorem 2 *(α,β)-STATIC WEIGHTED SUPER SOLUBILITY and (α,β,τ)-DYNAMIC WEIGHTED SUPER SOLUBILITY are $\mathcal{NP}$-complete when $\lfloor \log_{\max(\alpha_{(x,v)})} \alpha \rfloor$ is bounded by a constant k.*

Proof.
Hardness: To show they are in $\mathcal{NP}$ we need a polynomial witness. This is simply an assignment of the variables that satisfies the constraints in the problem and, for each of the $\mathcal{O}(n^k)$ possible breaks, the set of repair values. This is polynomial for fixed k from Lemma 1.
Completeness: We present a reduction from binary CSP. Duplicates of each value in the domain of all variables are created. Constraints are added to behave equivalently on the duplicate (primed) values. Additional constraints are added to enforce that a solution can either involve only primed or duplicate values. This problem is satisfiable *iff* the original problem is also satisfiable. If a solution does exist, a WSS does also because any set of k values may be primed to form a repair solution. □

Whilst an upper bound on α is fixed, there is no such constraint on β. If α is unbounded, then (α,β)-STATIC and (α,β,τ)-DYNAMIC WEIGHTED SUPER SOLUBILITY are in PSPACE.

Optimization Problems. It may not always be possible to find a robust solution for given values of α, β and τ. In such situations the problem constraints may be relaxed in several ways so that different trade-off scenarios are considered.

If α is minimized, we maximize the number of potential breaks that have repair solutions. Alternatively, when β is minimized we seek repair solutions for all potential breaks but seek to minimize the cost of repair for any such break. In the case of a dynamic WSS it is also possible to maximize τ so that the solution is reparable for as long as possible.

Applications

We applied the WSS framework to two separate applications, job shop scheduling problems (JSPs) and combinatorial auctions (CAs), to demonstrate its usefulness and versatility. Firstly, we present the effects of changing the input parameters to small JSPs as a pedagogical example. We then present results for finding robust solutions for combinatorial auctions with thousands of bids.

Job Shop Scheduling

Each problem consisted of 3 machines and 4 jobs, each comprising a sequence of 3 activities. Each activity required each machine for a duration chosen over a uniform random distribution [1,5]. The objective was to schedule all activities so that the precedence and resource constraints amongst activities were respected whilst minimizing the overall makespan.

In these experiments machines $M_1 - M_3$ exhibited different failure rates and we assumed Weibull distributions with γ as 1.0, 1.5 and 2.0, respectively, and a random repair period from [1,5] on each machine following a failure. We let the characteristic life $\eta = 100$ so that each machine had a probability of failure of 0.632 by this time. We modeled the activities as variables, whose domain values represented start-times. The duration of each activity allowed us to determine the end-time for each activity on each machine given the start-time. We could, therefore, assign a static probability of failure using the difference in the CDF on the relevant machine between these two times. The brittleness of a variable (activity on a particular machine), therefore, depended upon its assigned value (start-time). Our solver used

Table 1: Results (4x3 JSP).

(a) $\alpha = 0.01$

β	mksp	nodes $\times 1000$	breaks
0	18.82	67,624	303,848
50	18.92	638,739	743,005
100	19.26	533,368	298,325
150	18.88	500,723	234,598
200	18.32	383,725	136,273
250	18.42	577,310	159,820
300	18.20	449,798	227,233
350	18.48	821,485	174,816
400	17.84	525,212	209,574

(b) $\alpha = 0.02$

β	mksp	nodes $\times 1000$	breaks
0	18.28	15,878	53,369
50	18.32	354,186	128,888
100	18.32	290,072	78,951
150	18.28	412,006	105,511
200	18.12	415,698	74,169
250	17.32	243,148	45,748
300	17.80	360,334	53,963
350	17.50	281,017	60,642
400	16.98	421,379	63,510

(c) $\alpha = 0.03$

β	mksp	nodes $\times 1000$	breaks
0	18.40	18,915	41,493
50	17.68	33,864	10,959
100	17.80	175,650	40,622
150	18.24	182,240	38,803
200	17.74	193,117	24,728
250	18.18	257,233	73,544
300	17.68	290,801	52,235
350	17.70	370,700	44,384
400	18.16	318,035	59,601

(d) $\alpha = 0.04$

β	mksp	nodes $\times 1000$	breaks
0	16.96	3,748	5,329
50	16.98	6,005	1,159
100	17.06	473	163
150	17.14	1,517	362
200	17.08	10,853	1,126
250	16.84	3,445	639
300	16.20	66,100	6,260
350	15.96	249	33
400	15.82	8,158	1,123

a dynamic minimum degree heuristic over the variables and values were chosen in lexicographical order.

We assumed that the cost of repairing a solution was machine dependant. Given a break in a solution, the costs of changing an activity on $M_1 - M_3$, were 25, 50 and 75, respectively. Rescheduling activities on different machines may vary because of calibration/setup/labor costs, etc. We examined the trade-off between robustness and makespan for 50 randomly generated instances. Problems had 12 variables, with domain sizes containing approximately 15-20 values. The last value in the domain was an upper bound on the latest start-time required for that activity. A solution was found by bounding the makespan to an initial lower bound. If a solution was not found, this makespan was incremented by one and the problem was resolved. This process was repeated until a valid solution is found.

Tables 1(a)-1(d) show how the minimal makespan, number of nodes visited in the search tree and breaks checked for reparability vary with α and β. It is noticeable that the optimal makespan decreases as β increases and is more pronounced when $\beta > 200$. When β is low reparability is inhibited because fewer repair solutions can be considered, so the makespan increases. It is also clear that as α decreases there is a larger number of activities that require repair solutions so the makespan increases as some repair solutions require later start-times.

Finding a WSS can become computationally intensive when α is low, increasing the number of combinations of breaks needing consideration. For example, from Tables 1(a)-1(d) we can see how the number of nodes in the search tree can grow exponentially. As the number of possible breaks decreases, the number of concurrent searches for separate repair solutions also decreases. The search-effort is greatly reduced when $\alpha = 0.04$ (Table 1(d)) because fewer assignments require repair solutions.

The number of nodes visited and breaks checked are not tightly correlated with β for the following reason. When β is low, searches for repair solutions may fail quickly, whereas when it is high, the repair searches visit more nodes but the success rate increases thereby leading to initial solutions more quickly. These two effects counteract one another as β increases.

Tables 1(a)-1(d) also show the number of breaks checked for reparability before we extend the search tree (column breaks). This corresponds to the number of calls made to the `reparable` procedure. From these results it is clearly important that repair solutions are not sought for assignments that are robust in order to minimize the computational burden. This is why probabilistic failures in the WSS framework are critical in subtly differentiating between assignments that are brittle and those that are robust.

Combinatorial Auctions

A combinatorial auction (CA) involves the sale of multiple distinguishable items in which bids can be submitted on any combination of items that interest a bidder. The winner determination problem (WDP) for CAs is $\mathcal{NP}$-complete (Rothkopf, Pekeč, & Harstad 1998) and inapproximable (Sandholm 2002), and is equivalent to the Set Packing Problem. The problem of bid withdrawal following winner determination in CAs constitutes a serious risk for the bid-taker because the irrevocable awarding of items to other non-reneging bidders may mean that a repair solution of sufficient revenue may be impossible to achieve (Holland & O'Sullivan 2005). A robust solution in which bid-withdrawal may be repaired with a bounded loss in revenue is particularly desirable for a risk averse bid-taker.

Some bidders may be deemed unreliable and probabilities of failure can be associated with their bids. The WSS framework permits us to find repair solutions for sets of brittle winning bids, i.e. those whose probability of withdrawal is $\geq \alpha$. Super solutions for combinatorial auctions offer the possibility of restricting the number of changes required to form a repair solution of sufficient revenue for any bid withdrawal (Holland & O'Sullivan 2004). When attempting to repair a solution after a winning bid has been withdrawn by a bidder, the bid-taker must reallocate unsold items amongst bidders. However, the bid-taker may have to pay compensation to a bidder if an item is withdrawn from him; this compensation can be regarded as a repair cost. Using the parameter β, the WSS framework can consider such repair costs so that a repair can be found to achieve a high revenue solution. The maximum cost of repair, β, may be determined by the particular break in the solution. For example, all bidders may agree that if they withdraw a bid, they pay a penalty to the bid-taker equal to a certain fraction of the bid amount. This penalty can then be used to fund a reassignment of items, i.e. provide a fund of size β that can be used to compensate winning bidders whose bids are revoked in order to find a repair solution.

We have used the Combinatorial Auction Test Suite (CATS) (Leyton-Brown, Pearson, & Shoham 2000) to generate 100 sample auction problems in which there were 20

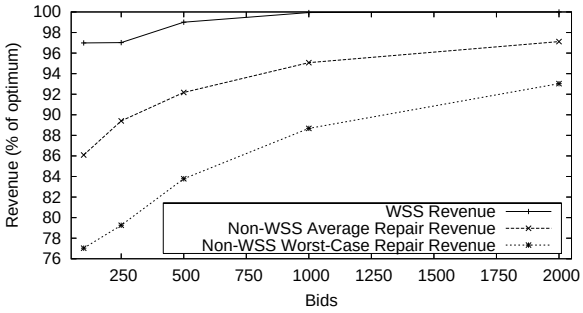

Figure 1: Average revenues for (1) WSS, (2) non-robust repair solutions and (3) worst-case non-robust repair solutions. Worst-case repair revenue for each WSS is $\geq$90%.

items for sale and 100-2000 bids[1]. CATS uses economically motivated bidding patterns to generate auction data in various real world scenarios. In these experiments we used the `arbitrary` auction distribution that simulates electronic component auctions. We established robust solutions for CAs using the WSS framework.

Combinatorial auctions are easily modeled as a constraint optimization problem which can be solved using a combination of CP and operations research techniques. Binary variables represent bids and our search mechanism used a reverse lexicographic value ordering heuristic. This complemented our dynamic variable ordering heuristic that selects the most promising unassigned variable as the next one in the search tree. We used the product of the solution of the linear programming (LP) relaxation of the problem, to give an approximate bound on revenue, and the degree of a variable to determine the likelihood of its participation in a robust solution.

Our experiments simulated an auction in which the bid-taker was willing to accept a solution that was at least 90% of optimal revenue but guaranteed a repair solution could be found, whilst still satisfying the revenue constraint, given any single bid withdrawal. Backtracking by the bid-taker on winning bids using withdrawal penalties and compensation payments to bidders whose winning bids are revoked was permissible. Prior to finding a robust solution we solved the WDP optimally using a conventional ILP solver. We then imposed the minimum tolerable revenue constraint for a solution and searched for a WSS that optimized revenue.

Figure 1 clearly illustrates the benefits of using the WSS framework for finding robust solutions to a combinatorial auction. In this figure we can see that the average revenue of a robust solution found using the WSS framework requires at most a 3% compromise on optimal revenue. However, we are always guaranteed that a solution can be found in which revenue is at least 90% of the optimum when a single bid is withdrawn because of the revenue constraint used to find robust solutions. Contrast this with the non-robust scenario, where we can see that the bid-taker is vulnerable, particularly in smaller auctions involving fewer bids. The average revenue of a repair solution varies between 86% and 97% of optimal revenue, while the average worst-case revenue is significantly lower (77%–93%). We see from Figure 1 that once the number of bids $\geq$ 1000, a robust solution found using the WSS framework can provide optimal revenue.

[1]The CATS flags included int_prices with bid_alpha set to 1000.

Conclusion

Weighted super solutions extend the basic framework (Ginsberg, Parkes, & Roy 1998; Hebrard, Hnich, & Walsh 2004b) in two important ways. Firstly, the set of variables that may lose their values is determined using a probabilistic approach enabling us to find repair solutions for assignments most likely to fail. Secondly, we include a metric for reasoning about the cost of repair. The framework provides an expressive basis for establishing robust solutions when faced with variable assignments that may lose their values when the solution breaks.This framework is practical and useful in many application domains, such as scheduling and combinatorial auctions, where reasoning about uncertainty and the cost of repair is important.

References

Fowler, D. W., and Brown, K. N. 2000. Branching constraint satisfaction problems for solutions robust under likely changes. In *Proc. of CP-2000*, LNCS 1894, 500–504.

Ginsberg, M. L.; Parkes, A. J.; and Roy, A. 1998. Supermodels and Robustness. In *Proc. of AAAI*, 334–339.

Hebrard, E.; Hnich, B.; and Walsh, T. 2004a. Robust solutions for constraint satisfaction and optimization. In *Proc. of ECAI*, 186–190.

Hebrard, E.; Hnich, B.; and Walsh, T. 2004b. Super solutions in constraint programming. In *CP-AI-OR*, LNCS 3011, 157–172.

Holland, A., and O'Sullivan, B. 2004. Super solutions for combinatorial auctions. In *ERCIM-Colognet Constraints Workshop (CSCLP 04)*, LNAI 3419, 187–200.

Holland, A., and O'Sullivan, B. 2005. Robust solutions for combinatorial auctions. In *ACM Conf. on Electronic Commerce*. (to appear).

Leyton-Brown, K.; Pearson, M.; and Shoham, Y. 2000. Towards a universal test suite for combinatorial auction algorithms. In *ACM Conf. on Electronic Commerce*, 66–76.

Rothkopf, M.; Pekeč, A.; and Harstad, R. 1998. Computationally manageable combinatorial auctions. *Management Science* 44(8):1131–1147.

Sabin, D., and Freuder, E. C. 1994. Contradicting conventional wisdom in constraint satisfaction. In *Proc. of ECAI*, 125–129.

Sandholm, T. 2002. Algorithm for optimal winner determination in combinatorial auctions. *AIJ* 135(1-2):1–54.

Weibull, W. 1951. A statistical distribution function of wide applicability. *Journal of Applied Mechanics* 293–297.

Weigel, R., and Bliek, C. 1998. On reformulation of constraint satisfaction problems. In *Proc. of ECAI*, 254–258.