

**Weighted Super Solutions for Constraint Programs**  
**Technical Report: Number UCC-CS-2004-12-02**

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



## Abstract

Super solutions to constraint programs guarantee that if a small number of variables lose their values, repair solutions can be found by modifying only a few assignments [Hebrard *et al.*, 2004b; Ginsberg *et al.*, 1998]. In this paper we present weighted super solutions which extend the basic super solutions framework 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 that are most likely to fail. The probabilities of variables losing their values can be described using static probabilities or Weibull probability distribution functions over time. Secondly, we include a mechanism for reasoning about the cost of repair. These two extensions provide a very expressive framework for finding robust solutions that describe potential failures and repair costs in an accurate and flexible manner.

## 1 Introduction

Solutions are *robust* if they are not vulnerable to small breakages, or in other words, a repair solution is easily achieved if some assignments become invalid. The purpose of finding super solutions is that if the solution changes slightly, another solution may be found by changing a limited number of other assignments [Hebrard *et al.*, 2004a; 2004b].

The Weighted Super Solutions (WSS) framework, proposed here, allows us to capture failure characteristics such as those that underly metal fatigue, corrosion and abrasion that exhibit various probability distribution functions. Accurate failure prediction facilitates contingency planning where it is needed so that repair solutions are easily achieved should an assignment fail. This can be described in two different ways: using *static* probabilities of failure or *dynamic* failure rates. The latter describes the probability of failure over time, such as in factory scheduling problems where machines or components have probabilistic failure rates.

It is important that repair solutions are available for assignments that are likely to break within a given time-frame. We incorporate a powerful approach to modelling failures, based on the *Weibull distribution* [NIST/SEMATECH, 2004] from the field of reliability engineering, to model the failure rates of assignments in solutions to constraint programs. Using this distribution we can represent many of the most common failure distributions such as normal, lognormal and exponential. Having access to such generality is important for reasoning about robustness in a realistic manner.

It is equally important that we can model the cost of repair. We introduce a novel metric for repair costs, which we refer to as *inertia*. Once a break in a solution occurs, *e.g.* a machine in a factory breaks down, the WSS framework has a more accurate means of comparing the costs associated with alternative ways of repairing the 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. This expressiveness adds a vital layer of usability and flexibility to the super solutions framework.

This paper is organized as follows. Section 2 describes how the WSS framework considers any subset of variables whose probability of breakage is above a certain threshold to be brittle, thus requiring a repair solution. Section 3 introduces the concept of inertia as a metric for the cost of repairing a solution. Section 4 defines the framework and gives an overview of how a MAC-based [Sabin and Freuder, 1994] search algorithm establishes a WSS and also analyzes the time and space complexity of finding weighted super solutions. Section 5 conducts an experimental analysis using job-shop scheduling problems.

## 2 Probabilistic Failure

Breakages 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. We discuss each case in detail below.

*Constant probabilities of failure (Static WSS).* Hebrard *et al.* [Hebrard *et al.*, 2004a] described how some values in a solution are not subject to change, referred to as *robust* values. In a weighted super solution we propose that *assignments* have varying degrees of robustness to differentiate between those that are more or less likely to cause failure. In this manner, 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 less than 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 components of mechanical/electrical or electronic devices are typically defined in terms of probability distributions over time. In general, the shape or type of failure distribution depends upon the component's inherent failure mechanisms.

The Weibull distribution is the only one unique to the field of reliability engineering [NIST/SEMATECH, 2004]. It can be used to model a variety of distributions including normal, lognormal, exponential and Rayleigh, which are exhibited in corrosion, diffusion, fatigue, abrasion and many other degradation processes [Weibull, 1951]. It is the most widely used distribution in reliability engineering, thus ideal for simulating probabilistic failure rates in weighted super solutions, 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:

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

where  $\gamma$  and  $\eta > 0$ .  $\gamma$  is the shape parameter of the distribution. If  $\gamma$  is greater than 1, the failure rate is increasing; if  $\gamma$  is less than 1, the failure rate is decreasing;  $\gamma = 1$  implies the failure rate is constant.  $\eta$  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. Figures 1(a) and 1(b) illustrate how the various parameters effect the probability and cumulative probability distributions respectively.

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

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

We use the Weibull distribution to describe the failure rates of assignments whose probability of failure is time-dependent. When this probability is at least some threshold,  $\alpha$ , at a given time,  $\tau$ , then it is deemed brittle and requires a repair solution.

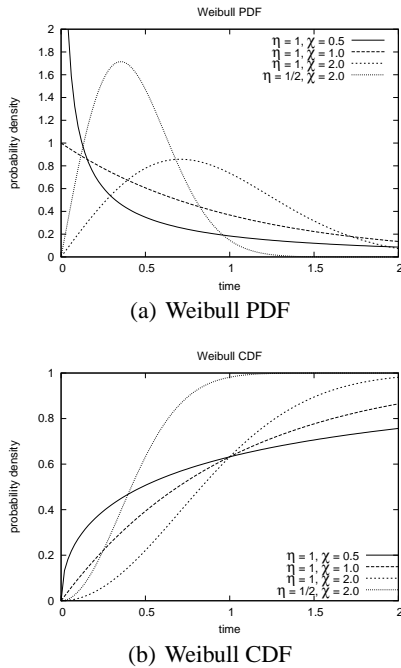


Fig. 1. Weibull Distributions

### 3 Inertia as a Metric for Repair Costs

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 change. This 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. We propose a new metric, called *inertia*, that may be viewed as a measure of an assignment's aver-

sion to change. This extension to the super solutions framework permits differentiation between the cost of alternative repairs.

The cost of changing an assignment in a solution may have multiple dependencies. We include the original and final assignments as well as the breakage variable(s) as inputs to the function to determine the cost of making a single change. 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 be different. The cost of a state transition may furthermore be influenced by the cause of the break in the solution.

The repair restrictions presented in [Hebrard *et al.*, 2004a] may be fully expressed using break and destination dependant inertia by using a value of  $\infty$  as the cost of making changes that are disallowed.

**Definition 1 (Inertia).** *Inertia 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,  $I_{v_1 \rightarrow v_2}^{(x,A)} \in \mathbb{R}_0^+$ .*

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 inertial values becomes  $O(n^{k+1}d^2)$  when represented extensionally.

#### 3.1 Example: Job Shop Scheduling

Consider an example of a Job Shop Problem (JSP) in which three jobs, each comprising two activities, need to be scheduled. Each activity requires a fixed length of time on machines  $X$  and  $Y$ . A sample solution is presented in Figure 2. The arrows indicate precedence constraints between activities. If Machine  $Y$  breaks down whilst executing the first activity, there is a period of repair time followed by the execution of a modified schedule so that the interrupted activity is re-scheduled and other activities are also re-scheduled to maintain the precedence constraints. The repair solution clearly has a longer makespan, but another difficulty is that two other activities besides the one that was interrupted were re-scheduled.

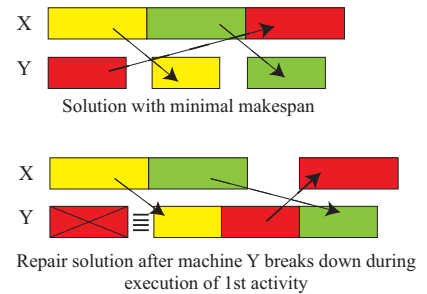


Fig. 2. JSP example

A robust solution, see Figure 3, allows a repair solution to be constructed without imposing a heavy re-scheduling cost

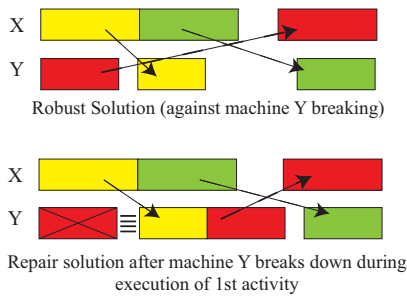


Fig. 3. Robust solution for JSP example

in the event of machine  $Y$  breaking down. However, we sacrifice makespan in favor of robustness. We shall assume that machine  $X$  is robust and no activity is likely to fail while on this machine.

Some re-scheduling decisions may incur heavier penalties than others, thus motivating inertia as a new metric for solution repair costs. The WSS framework permits an expressive means of describing the cost of repairing solutions to constraint programs. For example, rescheduling activities on one machine may be less expensive than rescheduling those on another. Fully automated machinery may be re-scheduled easily whereas manually operated machines have higher re-scheduling costs.

### 3.2 Total Cost of Repair

The total cost of repair is computed using a function,  $f$ , that combines the inertia of the individual assignments that are modified to form the repair solution. Typically, a summation of the inertia values is used to determine the overall cost of repair. However, computing the maximum of the set of the inertia values would also be useful, such as in situations where variables are associated with agents and the imposition of change upon each agent is limited.

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 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,  $\beta$ , 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 constraint on the cost function. Concurrent search for repair solutions would become computationally infeasible.

## 4 The Weighted Super Solutions Framework

The WSS framework uses probabilistic failures to determine the sets of variables that require repairs and inertia to accurately measure the cost of repair. We can now define both *static* and *dynamic weighted super solutions* in terms of robustness and cost of repair.

**Definition 2 (Static WSS).** A solution to a CSP is a static weighted super solution, or  $(\alpha, \beta)$ -static WSS, if any set

of variables whose probability of losing all their values is greater than or equal to  $\alpha$ , can be repaired by reassigning other values to these and other variables with a repair cost of at most  $\beta$ .

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 can now define *dynamic weighted super solutions* in terms of robustness and cost of repair as follows:

**Definition 3 (Dynamic WSS).** A solution to a CSP is a dynamic weighted super solution, or  $(\alpha, \beta, \tau)$ -dynamic WSS, if any set of variables whose probability of losing all their values is greater than or equal to  $\alpha$  before time  $\tau$ , can be repaired by reassigning other values to these and other variables with a repair cost of at most  $\beta$ .

These definitions of static and dynamic WSS require several modifications to the super solution framework of [Hebrard *et al.*, 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 applied to both static and dynamic weighted super solutions. When  $\tau$  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 static probability of failure,  $\alpha_{(x,v)}$ , otherwise values' Weibull parameters  $\gamma_{(x,v)}$  and  $\eta_{(x,v)}$  are used to compute the probability of failure by time  $\tau$ .

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 then occur for one of two reasons: we cannot extend the assignment to satisfy the given constraints, or the current partial assignment cannot be associated with a repair solution whose inertia is less than  $\beta$  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 [Hebrard *et al.*, 2004a]; the differences being that a repair is provided for a set of breakage variables rather than a single variable and the cost of repair is considered. The procedure `check-wss-repair` determines the inertial 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 *et al.*, 2004a].

Algorithm 1: weighted-super-solve

---

```

input :  $\alpha, \beta, \tau$ , Inertia:  $I$ , CSP:  $P = \{\mathcal{X}, \mathcal{D}, \mathcal{C}\}$  // Let  $\tau = \emptyset$  for static-WSS
output:  $S$ : a  $(\alpha, \beta, \tau)$ -weighted super solution:  $R$ : 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( $P, S, Past, R, 0, \alpha, \beta, \tau, 0$ )
end

```

---

---

**Procedure** backtrack( $P, S, Past, R, lvl, \alpha, \beta, \tau, m$ ) : 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 breakage
     $S \leftarrow S \cup \{(x, v)\}$ 
    if  $AC(P, S)$  then
      foreach  $b \in \mathcal{P}(Past), |b| \leq k, failure(b, S, \tau) \geq \alpha$  do
        if  $\neg repairable(P, S, Past, R_b, 0, \beta)$  then break
        if backtrack( $P, S, Past, R, lvl + 1, \alpha, \beta, \tau, m$ ) then
          return true
      restore  $\mathcal{D}, m$  and  $R$ 
       $S \leftarrow S \setminus \{(x, v)\}$ 
   $Past[lvl] \leftarrow \emptyset$ 
  return false
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 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 and 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 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  $\beta$ .  $\square$

---

**Procedure** repairable( $P, S, Past, R_b, lvl, \beta$ ) : 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( $P, S, Past, R_b, lvl, \beta$ ) then
      if repairable( $P, S, Past, R_b, lvl + 1, \beta$ ) then return true
   $R_b[y] \leftarrow \min(\mathcal{D}(y))$ 
  return false
end

```

---



---

**Procedure** check-wss-repair( $P, S, Past, R_b, lvl, \beta$ ) : 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( $b, S, \tau$ ) : 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 ( $\tau$  undefined)
    else  $fail \leftarrow fail \times CDF(\eta_{(x,v)}, \gamma_{(x,v)}, \tau)$  // Dynamic WSS
  return  $fail$ 
end

```

---

We also show that finding weighted super solutions is  $\mathcal{NP}$ -complete in general for any fixed  $\alpha$  (and  $\tau$  for the case of dynamic WSS's).

**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 the variable  $x$  for static-WSS and the probability of failure at time  $\tau$  in the dynamic case. If  $\lfloor \log_m \alpha \rfloor$  is bounded by a constant  $k$ , then 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  $\alpha$ , i.e.  $\prod_{(x,v) \in s} \alpha_{(x,v)} \geq \alpha$ . But  $\alpha_{(x,v)} \leq m$ , so we can say the following:

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

Therefore if  $s$  requires a repair we have:

$$\alpha \leq m^{|s|}$$

$$\log \alpha \leq |s| \cdot \log m$$

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

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

Since  $\lfloor \log_m \alpha \rfloor \leq k$  and  $|s| \in \mathbb{Z}$

$$k \geq |s|$$

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

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

*Proof.* To show it is 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  $\mathcal{O}(n^k)$ , polynomial for fixed  $k$ , possible breaks, the set of repair values. To show completeness we demonstrate the polynomial reducibility of binary CSP to an instance of a WSS input. Duplicates of each value in the domain of all variables are created. Constraints are added to behave equivalently on the duplicate (primed) values. This problem is satisfiable iff the original problem is also satisfiable. If a solution does exist, a weighted super solution does also because any set of  $k$  values may be primed to form a repair solution. This is a generalization of the proof of  $\mathcal{NP}$ -completeness of classical SUPER SOLUBILITY [Hebrard et al., 2004b].  $\square$

Whilst an upper bound on  $\alpha$  is fixed, there is no such constraint on  $\beta$ . If  $\alpha$  is unbounded, then  $(\alpha, \beta)$ -weighted super solubility is in PSPACE.

*Backbone variables* are variables that take the same value in all solutions. The existence of such a variable precludes the existence of weighted super solutions because no repair solution can exist such variables [Hebrard *et al.*, 2004b].

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

If  $\alpha$  is minimized, we minimize the number of possible breaks that are irreparable whilst maximizing the number of potential breaks that have repair solutions. Alternatively, when  $\beta$  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 Dynamic WSS it is also possible to maximize  $\tau$  so that the solution is repairable for as long as possible.

## 5 Experimental Analysis

We use job shop scheduling problems to demonstrate the WSS framework. Each problem consists of 3 machines and 4 jobs, each comprising a sequence of 3 activities. Each activity requires each machine for a duration chosen over a uniform random distribution [1,5]. The objective is to schedule all activities so that the precedence and resource constraints amongst activities are respected whilst minimizing the overall makespan.

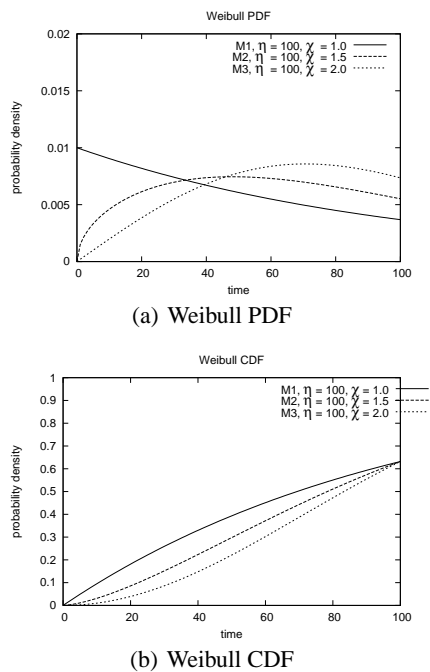


Fig. 4. Machine failure rates

In these experiments machines  $M_1$ ,  $M_2$  and  $M_3$  exhibit different failure rates and we assume Weibull distributions with  $\gamma$  as 1.0, 1.5 and 2.0, respectively. Recall that  $\gamma$  is the shape parameter and describes whether a machine is more likely to fail earlier or later in its lifespan. We let the characteristic life  $\eta = 100$  so that each machine has a cumulative probability of failure of 0.632 by this time. We model the activities as variables, whose domain values represent start times. The duration of each activity allows us to determine the end-time for each activity on each machine given the start-time. We can therefore assign a static probability of failure using the integral over the PDF on the relevant machine between these two times. This is the equivalent to the difference in the CDF between the start and end-times. Figure 4 illustrates both the PDF and the CDF for the activities on the respective machines with the above failure rates. The brittleness of a variable (activity on a particular machine), therefore, depends upon its assigned value (start time). Our solver used a dynamic minimum degree heuristic over the variables and values were chosen in lexicographical order.

Inertia permits a more expressive means of describing the cost of repairing a solution given a break. We assume that the cost of repairing a solution is machine dependant. Given a break in a solution, the cost of changing an activity on  $M_1 - M_3$ , is 25, 50 and 75 respectively. These values were chosen arbitrarily. As discussed earlier, rescheduling activities on different machines may vary because of calibration/setup/labour costs, etc.

We examined the trade-off between robustness and makespan for over 50 randomly generated instances. Problems have 12 variables, whose domain sizes contain approximately 15-20 values. The last value in the domain is an upper bound on the latest start time required for that activity. A solution is found by bounding the makespan to an initial lower bound. If a solution is not found, this makespan is incremented by one and the problem is resolved. This process is 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  $\alpha$  and  $\beta$ . It is noticeable that the optimal makespan decreases as  $\beta$  increases and is more pronounced when  $\beta > 200$ . When  $\beta$  is low, reparability is inhibited because fewer repair solutions can be considered, so the makespan increases. It is also clear that as  $\alpha$  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  $\alpha$  is set very low ( $\alpha = 0.01$ ), 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 visited 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 breakages checked are not tightly correlated with  $\beta$  for the following reason. When  $\beta$  is low, searches for repair solutions may fail quickly whereas

**Table 1. Results (4x3 JSP)**

(a) $\alpha = 0.01$				(b) $\alpha = 0.02$			
$\beta$	makespan	nodes	breaks	$\beta$	makespan	nodes	breaks
0	18.82	67,624,710	303,848	0	18.28	15,878,868	53,369
50	18.92	638,739,598	743,005	50	18.32	354,186,195	128,888
100	19.26	533,368,957	298,325	100	18.32	290,072,439	78,951
150	18.88	500,723,802	234,598	150	18.28	412,006,955	105,511
200	18.32	383,725,912	136,273	200	18.12	415,698,731	74,169
250	18.42	577,310,412	159,820	250	17.32	243,148,916	45,748
300	18.2	449,798,262	227,233	300	17.8	360,334,811	53,963
350	18.48	821,485,685	174,816	350	17.5	281,017,558	60,642
400	17.84	525,212,354	209,574	400	16.98	421,379,182	63,510

(c) $\alpha = 0.03$				(d) $\alpha = 0.04$			
$\beta$	makespan	nodes	breaks	$\beta$	makespan	nodes	breaks
0	18.40	18,915,467	41,493	0	16.96	3,748,352	5,329
50	17.68	33,864,950	10,959	50	16.98	6,005,338	1,159
100	17.8	175,650,706	40,622	100	17.06	473,647	163
150	18.24	182,240,874	38,803	150	17.14	1,517,335	362
200	17.74	193,117,157	24,728	200	17.08	10,853,564	1,126
250	18.18	257,233,428	73,544	250	16.84	3,445,032	639
300	17.68	290,801,620	52,235	300	16.2	66,100,351	6,260
350	17.7	370,700,646	44,384	350	15.96	249,953	33
400	18.16	318,035,915	59,601	400	15.82	8,158,838	1,123

when it is high the repair searches visit more nodes but the success rate is increased thereby leading to initial solutions more quickly. These two effects counteract one another as  $\beta$  is increased.

Table 1(a)-1(d) also shows how many breaks were checked if they were repairable before extending the search tree (column breaks). This corresponds to the number of calls made to the `repairable` procedure described in Section 4. 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.

## 6 Conclusion

The Weighted Super Solutions (WSS) framework extends the basic framework [Hebrard *et al.*, 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 value when the solution breaks. This framework is practical and useful in many application domains, such as scheduling or combinatorial auctions [Holland and O’Sullivan, 2004], where reasoning about uncertainty and the cost of repair is important.

## References

[Ginsberg *et al.*, 1998] Matthew L. Ginsberg, Andrew J. Parkes, and Amitabha Roy. Supermodels and Robustness. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 334–339, Madison, WI, 1998.

[Hebrard *et al.*, 2004a] Emmanuel Hebrard, Brahim Hnich, and Toby Walsh. Robust solutions for constraint satisfaction and optimization. In *Proceedings of the European Conference on Artificial Intelligence*, pages 186–190, 2004.

[Hebrard *et al.*, 2004b] Emmanuel Hebrard, Brahim Hnich, and Toby Walsh. Super solutions in constraint programming. In *Proceedings of CP-AI-OR 2004*, pages 157–172, 2004.

[Holland and O’Sullivan, 2004] Alan Holland and Barry O’Sullivan. Super solutions for combinatorial auctions. In *Ercim-Colognet Constraints Workshop (CSCLP 04)*. Springer LNAI, Lausanne, Switzerland, 2004.

[NIST/SEMATECH, 2004] NIST/SEMATECH. E-Handbook of Statistical Methods. [www.itl.nist.gov/div898/handbook/](http://www.itl.nist.gov/div898/handbook/), 2004.

[Sabin and Freuder, 1994] Daniel Sabin and Eugene C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In A. Cohn, editor, *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 125–129, 1994.

[Weibull, 1951] Waloddi Weibull. A statistical distribution function of wide applicability. *Journal of Applied Mechanics*, pages 293–297, 1951.