BDI-Learning Discussion Paper: Towards Improving *Stable* Performance

Dhirendra Singh
dhirendra.singh@rmit.edu.au

30 June 2009

1 Motivating example

1.1 The scenario

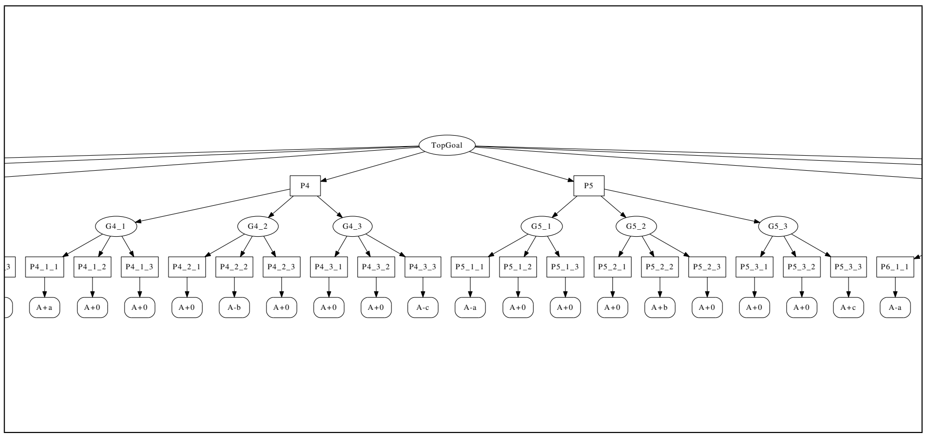


Figure 1: $testImpactvars$ input tree (sample portion only)

Continuing on with the investigation of learning context conditions in worlds with multiple variables, I created another test (referred to as *testImpactvars* in Figure 1 and herein) with a G/P tree that handles multiple variables and has the following properties.

- The tree handles 2^3 worlds described by the variables set $[a, b, c]$.
- All worlds have a unique solution in the G/P tree.
- At level one, $TopGoal$ is handled by 8 plans $[P1 \dots P8]$ such that the worlds space is evenly distributed among these sub-plans and there is no overlap.
- The plans $P1..P8$ have 3 sub-goals each, such that the sequence required to succeed is of length 3.
- At level two, each sub-goal of $[P1 \dots P8]$ is handled by 3 leaf plans, only one of which will ever succeed. So the probability of selecting a successful sequence

$p_{success}$ is given by the product of the probability of selecting a correct plan at level one and the probability of selecting 3 correct sub-plans at level two. Therefore, $p_{success} = p_{level1} * {p_{level2}}^3 = \frac{1}{8} * \frac{1}{3}^3 = \frac{1}{216}$.

- The G/P tree itself is evenly balanced i.e. the G/P hierarchy is of uniform breadth and depth.
- Finally, the distribution of the worlds within the tree is also evenly balanced i.e. each sub-tree handled the same proportion of all possible worlds $\frac{1}{8}$.

In Figure 1, the leaf nodes represent actions. Here actions with suffix $+0$ always fail. Actions with suffix $+a$ succeed when a is $true$ while those with $-a$ succeed when a is $false$. Similarly for $\pm b$ and $\pm c$. Looking at the sub-tree of plan $P4$ for instance, we can see that it will succeed only in the world $a\bar{b}\bar{c}$. In this way, the level one plans $[P1 \ldots P8]$ uniquely handle the worlds $[abc, ab\bar{c}, a\bar{b}c, a\bar{b}\bar{c}, \bar{a}bc, \bar{a}b\bar{c}, \bar{a}\bar{b}c, \bar{a}\bar{b}\bar{c}]$ respectively.

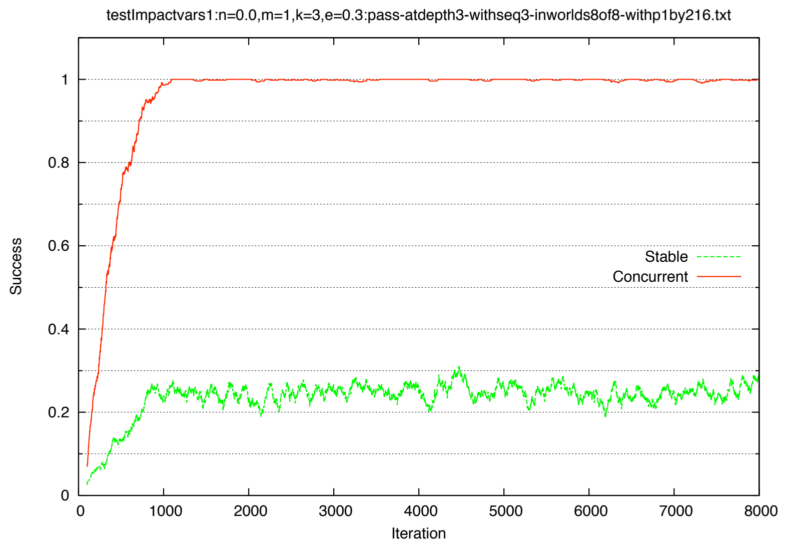


Figure 2: Performance comparison for $testImpactvars$

Figure 2 shows the performance of our *Concurrent* and *Stable* approaches in this scenario. Notice that *Stable* performance is almost four fold worse than *Concurrent* in this case.

The reason why this result is relevant is that *testImpactvars* was not purposely crafted to favour one approach over the other. Furthermore, *testImpactvars* is relatively shallow and has a low branching factor compared to tests we have performed in the past. All in all, the test is simpler in hierarchy and is arguably a better representation of a *typical* BDI tree than previously. The primary difference between this and previous tests (bar Stéphane's tree) of course is that we are experimenting here with multiple worlds.

1.2 Why *Stable* performs poorly for *testImpactvars*

So what causes *Stable* to perform so poorly in this case? We can start with the obvious differences between the two approaches and see if we can eliminate this disparity. Intuitively, we know that *Concurrent* is an aggressive or *optimistic* approach compared to *Stable* that is controlled and relatively *pessimistic*.

The parameters that fine-tune *Stable* behaviour are k (the minimum number of instances *of a given world* required for a decision tree to be considered stable) and ϵ (the maximum change in probability between two instances before a decision tree can be considered stable). For our experiments we use the default values of $k = 3$ and $\epsilon = 0.3$.

Already we can appreciate that $k = 3$ imposes a strong constraint at our leaf level before failure updates can be propagated to the top. For instance in Figure 1, goal $G4\_1$ will be considered stable for say world abc when all its children $[P4\_1\_1 \ldots P4\_1\_3]$ are stable too. For $k = 3$ that will take $3 + 3 + 3 = 9$ instances. For $P4$ to be updated, all its children $[G4\_1 \ldots G4\_3]$ must be stable. That in turn will take $9 + 9 + 9 = 27$ instances. Then for $P4$ to be stable for all possible worlds will take $27 * 8 = 216$ samples. For the entire tree to be stable will require a *minimum* of $216 * 8 = 1728$ samples. The actual number will be more than that because samples are chosen randomly and will naturally result in duplicates.

Figure 2 shows a simulation of 8000 samples. Even considering the randomisation, shouldn't *Stable* be performing optimally by then? Note that $k = 3$ determines the lower bound on the number of samples. The actual number of samples required for stability of a node also depends on ϵ, and to some extent on the *noise* n in the environment.

To reduce the disparity then, we run the experiment again with $k = 1$, $\epsilon = 1.0$, and $n = 0.0$. This would make *Stable* performance almost the same as *Concurrent*. Note that *Stable* still requires the stability of each child and the entire tree still requires at least $(1 + 1 + 1) * 3 * 8 * 8 = 576$ samples. While this number is the same for *Concurrent*, the difference is that the timing of *Concurrent* updates will be different to that of *Stable*.

On conducting the experiment again with these lenient parameters the result is unexpected. Instead of *Stable* performance converging towards *Concurrent*, there is *no change* to *Stable* performance when compared to Figure 2.

This result suggests that other factors are at play here than those determined above. Debugging the implementation at length shows that some core decisions in the system introduce subtleties that eventually lead to performance degradation.

1.3 When is it ok to start using a decision tree?

The absolute minimum number of instances required to build a decision tree is 1. Currently this is the number we use to decide when to build and start using a tree, as determined by the runtime parameter $m = 1$. This decision causes several problems.

At it's core, the problem is that we are constructing a decision tree with a single sample of *one* world and then using this tree to determine the probabilities for *all* worlds. This is not reasonable.

This problem manifests itself in various symptoms, some of which are listed here.

- Consider three leaf nodes $[Pi, Pj, Pk]$. At the start, none of the nodes have trees and always return a probability of 1 by default. We are interested in world $W1$ where we know that P_j will succeed. We may see the following possible sequence of events:

1. The starting probabilities for selection in $W1$ are $[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$. Let's say Pi is randomly selected, executed, and fails in $W1$. It then builds a decision tree from this sample and will use it thereon.
2. Second time around in $W1$ the selection probabilities are $[[\approx 0, \frac{1}{2}, \frac{1}{2}]$. This time Pk is randomly selected, fails, and builds a decision tree.
3. Third time around in $W1$, the selection probabilities should be $[[\approx 0, \frac{1}{1}, \approx 0]$ and Pj should inevitably be selected. However the probabilities have somehow changed to $[\approx 0, \approx 0, \approx 0]$. Why? Because sometime between the second and third instances of $W1$, Pj was selected in *some other world* and failed. It then constructed a single-sampled decision tree in that world that it is now using in world $W1$ and returning a probability of ≈ 0.

That's the power of interpolation! The result is that the selection probabilities are now equal and back to the original value of $\frac{1}{3}$ each (but with each absolute probability ≈ 0 instead of 1 as at the start). The impact is that hereon the probability of selection of P_j will not improve doesn't matter how many times P_i and P_k may fail in between.

The problem worsens as the branching factor increases. Consider a set of 20 plans, only one of which is setup to succeed. If it's probability incorrectly reduces to ≈ 0 thanks to a misinformed decision tree, then it's chances of selection will never improve beyond $\frac{1}{20}$ *even though every other siblings may have been tried and failed numerous times*. For correct operation, the probability of this plan should gradually increase $\rightarrow 1$ as other siblings are tried and fail.

- A similar symptom is where the initial probabilities for $W1$ are all $[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$, but change to say $[\frac{1}{2}, \approx 0, \frac{1}{2}]$ because a decision tree was constructed for P_j in some other world that is returning a misguided probability for $W1$. This even before $W1$ was ever encountered.

- The reverse case is also a problem, where an ill-informed decision tree may give an incorrect probability of 1 for a world $W1$ it has never seen before. In this case, there is really no hope of improving in $W1$ because all siblings will be demoted to ≈ 0 and the false-positive incorrect choice will be made $\frac{999}{1000}$ times. The situation will improve only as the ill-informed decision tree gathers more instances in other worlds and in effect remedies the problem for $W1$. However, this may not happen at all if wrong choices are always being made in a sort of *deadlock*.

As a result the exploration of *Stable* is unfairly biased leading to a slower convergence than expected in some cases, and a sub optimal (deadlocked) convergence in others. This latter case is what I think we are seeing in *testImpactvars*. Note that this problem is evident in experiments with multiple worlds, hence why we haven't come across it earlier.

1.4 When no decision tree exists, what should the default p be?

This question impacts the performance of both *Concurrent* and *Stable*, and must be addressed carefully. So let us first ensure we understand the question clearly. Currently, the following (pseudo) code in every plan determines the likelihood of success in a given world.

```
probability = useDT(planID) ? probabilityDT(planID) : 1
```

The decision is to use a probability of success of 1 for any given world when the decision tree for the plan is not ready for use (note that $useDT$ returns $false$ when we haven't encountered the minimum number of instance i.e. $m = 1$), otherwise use the probability as determined by the decision tree. We have already seen in Section 1.3 what happens when the decision tree being used is ill-informed and returns misleading probabilities for the world in question. But what about the other part of the equation? Does it matter what we use as the default probability when we have no decision tree available? Turns out it does.

useDT	p Used	Outcome	Comment
[F F F]	[1 1 1]	Select P_j and fail	The event is recorded for P_j.
[F T F]	[1 ≈0 1]	Select P_i and fail	This time around in W, a decision tree was created for P_j and used. The decision tree returned p ≈0 for W. Subsequently P_i was randomly selected and failed. The event was recorded for P_i.
[T T F]	[≈0 ≈0 1]	Select P_k and pass	This time around in W, a decision tree was created for P_i and used along with the existing decision tree for P_j. Both returned p ≈0 for W. Subsequently P_k was inevitably selected and succeeded. The event was recorded for P_k.
[T T T]	[≈0 ≈0 1]	Select P_k and pass	All decision trees are in use. Hereon P_k will inevitably be selected most of the time which is what we expect.
[F F F]	[1 1 1]	Select P_k and pass	The event is recorded for P_k.
[F F T]	[1 1 ≈1]	···	This time around in W, a decision tree was created for P_k and used. The decision tree returned p ≈1 for W which is what we expect. However since the default p for P_i and P_j is also 1, then the selection probabilities have not changed at all. So even though we have witnessed previously that P_k succeeds in W, the probabilities used do not reflect this. This is not optimal.

Table 1: Impact of p on plan selection in W for a set of applicable plans $[P_i, P_j, P_k]$

Table 1 shows the impact of the default probability p on plan selection from a set of applicable plans $[P_i, P_j, P_k]$ in a given world W. It highlights the case when the choice of returning a default probability of 1 does not work in our favour for plan selection. This poses the question if the default $p = 1$ is the right choice and if not then what is? A default of $p \approx 0$ does not work either for the following reasons:

- If the default is $p \approx 0$ and some applicable plan is using a decision tree that returns a probability $\rightarrow 1$ for the given world, then that plan will almost always be selected. This may cause other applicable plans to never be selected and tried. As a result the parent node will *almost never* become stable (requirement for stable is that all children be stable so should have been tried in the given world at least k times).

- If the default is $p \approx 0$ then failing in a given world will not change the probabilities. The impact is that the probability of selection of the good plan will not improve doesn't matter how many times other siblings have been tried and have failed. So even though we may witness numerous times that every other applicable plan has failed in the given world, the selection probability of the good plan (that has never been tried before and has the default $p \approx 0$) will not improve.

1.5 Recommended Changes

1.5.1 When to use a decision tree?

Section 1.3 shows how the choice $m = 1$ leads to ill-informed decision trees that distort plan selection probabilities. An obvious remedy is to increase m to a suitable number that guarantees prediction within tolerance from the newly formed decision tree. However, one cannot determine this optimal number since instances are generated randomly and include duplicates. Furthermore, the higher the number the longer we have to wait to use the power of decision trees, which is also not ideal.

I function $useDT$ currently determines when we are ready to start using a decision tree as follows:

```
if(sub-treeOK && instances>=m){
```

The code first checks to see that all children have their decision trees built and then confirms that the number of instances *of any world* seen so far is greater than m.

I recommend we change this as follows:

```
if(sub-treeOK && instances>=m && (doStable?haveSeen(W):1)){
```

The recommended change is that when deciding if we are ready to use a decision tree, we include one additional check that we must have witnessed the world W in question at least once before. In effect, we are saying that we are not confident in the tree for the given world unless we have seen that world at least once before, regardless of the number of total instances m seen so far. The change applies only to *Stable* and not *Concurrent* (determined by the $doStable?$ check).

At this point one could argue that the additional check is too restrictive because you lose the case where m is large enough that the resulting tree would still give a good estimate of the probability in W even though we have never seen W before (that's the power of decision trees remember). That is true, and we could form a more complex condition as follows:

```
if(...?(haveSeen(W) || instances>=newM ):..){
```

This would allow us to start using the tree even when we have not seem W but have seen enough instances to be confident that the decision tree prediction will be meaningful. While m is a static requirement, $newM$ could be calculated dynamically based on a number of factors one of which would be the total number of worlds. For instance we could say that we are confident in a decision tree *if we have seen the world* W *before OR we have seen at least half (or any other fraction) of all possible worlds*. This decision is open for discussion, but for now I recommend only introducing the $haveSeen$ check.

1.5.2 When no decision tree exists, default p should be 0.5

Section 1.4 explains how the default values of $p = 1$ or $p = 0$ (for when no decision tree exists for a given plan) can distort plan selection probabilities. I recommend we change the default probability to $p = 0.5$ for the following reasons:

- Using a default $p = 0.5$, when a plan finally switches to using the decision tree p will start to converge towards either 0 or 1 which is the true probability for that plan in the given world. We can say that the value $p = 0.5$ is *neutral* towards the true probability of 0 or 1.
- When no other information is considered, and we have to *estimate* (i.e. by setting a default) at design time what the chances of success of a plan are, then the logical choice is $50/50$, so a $p = 0.5$ makes rational sense.

1.5.3 Test Results

The test results from applying the recommended changes are included in the appendix.

- Page "Prior to applying suggested changes (Repository Revision 49)" shows the benchmark results before any changes were applied. Notice the problematic *testImpactvars* result on that page. These tests all used $k = 3$.
- Page "After applying suggested changes using k=3 (Repository Revision 49+)" shows the results after applying the changes. The change does not break any previous tests but shows a slight improvement in *testImpactvars* convergence from ≈ 0.25 to ≈ 0.4. These tests all used $k = 3$.
- Page "After applying suggested changes using k=1 (Repository Revision 49+)" shows the results after applying the changes but using $k = 1$. What's the point? Well we want to verify that our chances make a difference to *testImpactvars*. This time, we get the result we expected in Section 1.2.

2 A Systematic Analysis of *Stable*

2.1 Test coverage

Sebastian and I briefly discussed the *testImpactvars* result last week and agreed that it exposes some grey area about our understanding of *Concurrent* and *Stable*. Moreover, from our current tests it is difficult to understand the conditions under which one approach performs better than the other.

In considering how we can improve context learning, we need a better understanding of how the two current approaches perform in a combination of factors as listed in Table 2. So far we have experimented with some of these factors but not enough to get a good understanding of how they influence our approaches.

Our testing strategy is open for discussion. Should we conduct systematic testing to understand the influences of the various factors on the two approaches? If so, what is the best strategy here since the number of combinations is too high.

Factor	Tested
Branching factor of tree	Yes
Depth of tree	Yes
Stochastic nature of the environment	Yes
Number of worlds	Yes recently
If the G/P hierarchy is balanced (all sub-trees have the same breath/depth)	Barely (recently)
If the distribution of worlds within the G/P hierarchy is balanced (sub-trees handle an equal share of all possible worlds)	Barely (recently)
If more than one sub-tree holds a solution	No
If failure has a cost	No

Table 2: Factors that impact the performance of *Concurrent* and *Stable*

2.2 Current insights into the workings of *Concurrent* and *Stable*

Finally, here I have collected some insights into the workings of our two approaches in a generalised manner.

- *Stable* performs better when

  1. One solution exists in a deep sub-tree (note that differences between the approaches is amplified when the the probability of hitting that solution is lowered by fine-tuning the breath/depth of the sub-tree); and
  2. At least one other sub-tree is *shallower*; and
  3. the shallower sub-tree *does not* hold a solution.

  In this case, *Stable* will realise first that the shallower sub-tree does not hold the solution and that the deeper sub-tree *may*. So it will assign a lower probability to the shallower sub-tree. (*Concurrent* will assign more or less equal probability to all sub-trees since none of them seem to work). In effect the probability of picking the deeper sub-tree increases and therefore *Stable* has a better chance of finding the solution there first.

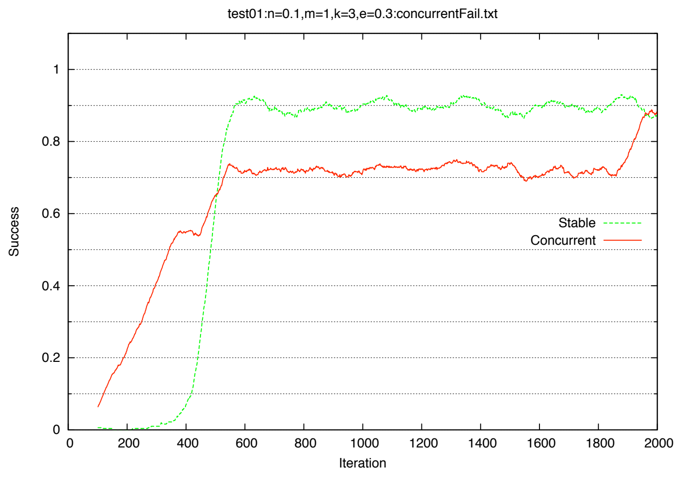
- *Concurrent* performs better when

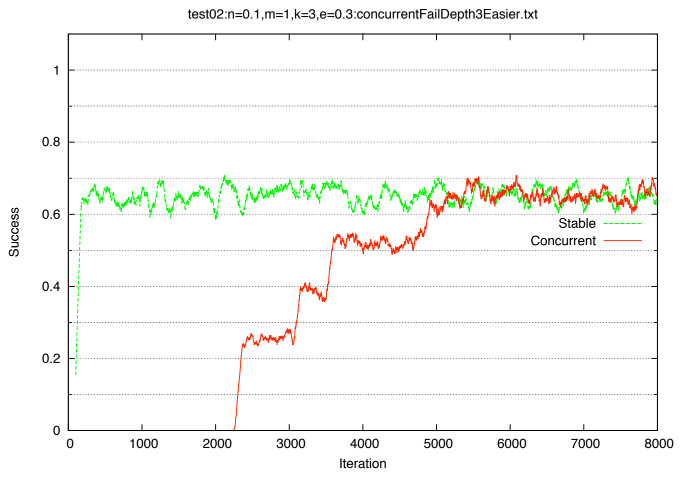
  1. One solution exists in a deep sub-tree (same as before); and
  2. At least one other sub-tree is also *deep*; and
  3. All other *deep* sub-trees *do not* have a solution (the more the number of failing deep sub-trees the more amplified the difference).

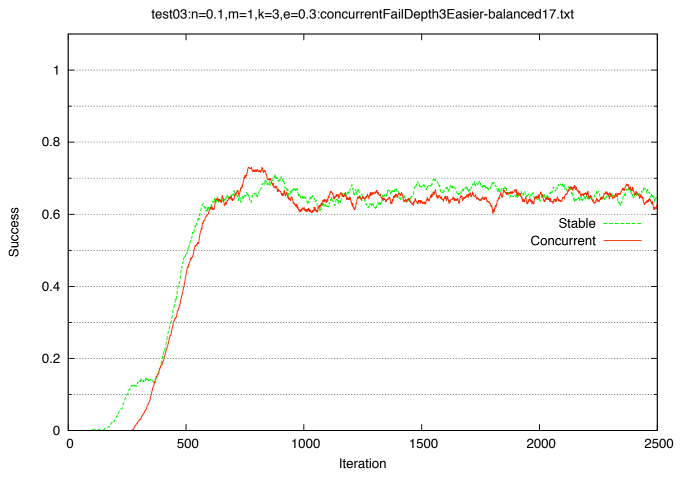
  In this case, *Concurrent* performs the same as before. *Stable* however takes a long time to be confident that the failing deep sub-trees are in fact fruitless so it does not change their probabilities for a long time. When a solution is finally found, *Concurrent* favours that sub-tree whereas *Stable* still devotes exploration to the fruitless sub-trees until it is confident that no solution exists there.

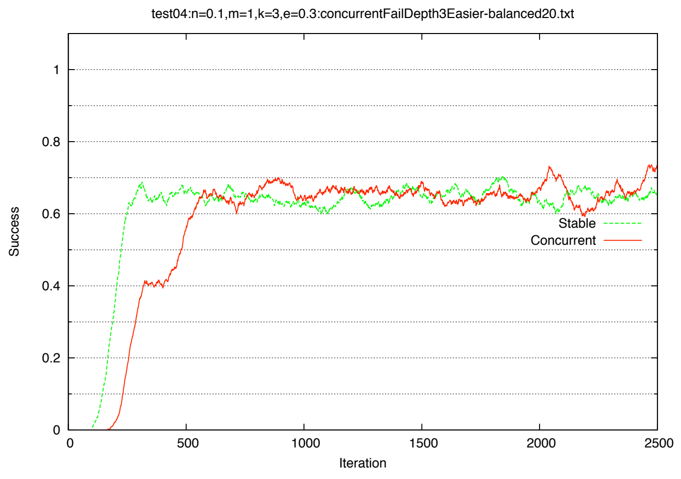
- At the leaf nodes, the differences between *Concurrent* and *Stable* are minimal, but *Stable* takes longer to be confident that an observation of failure is in fact a true failure (and not due to a stochastic environment).

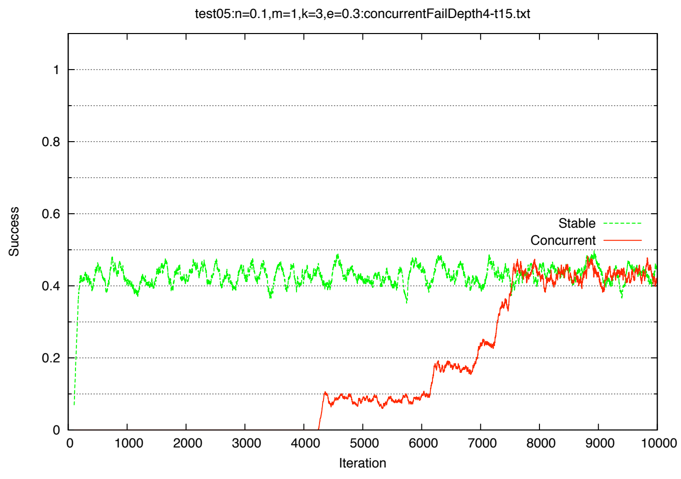
Prior to applying suggested changes
(Repository Revision 49)

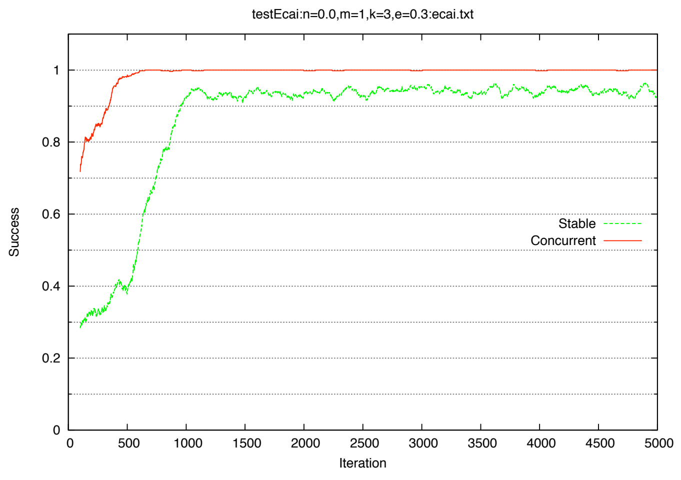


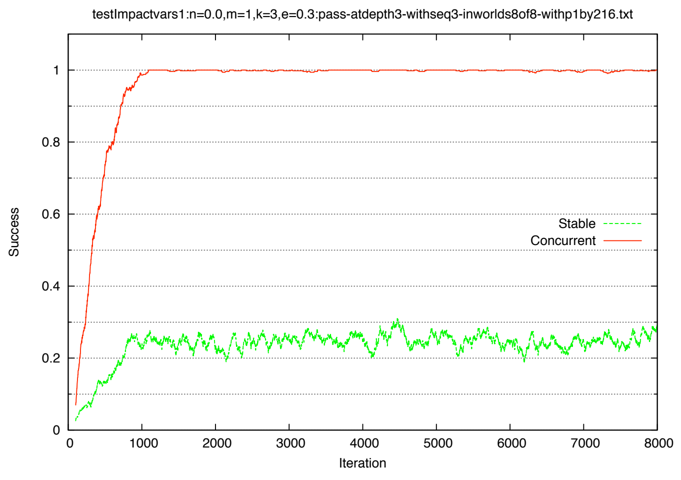




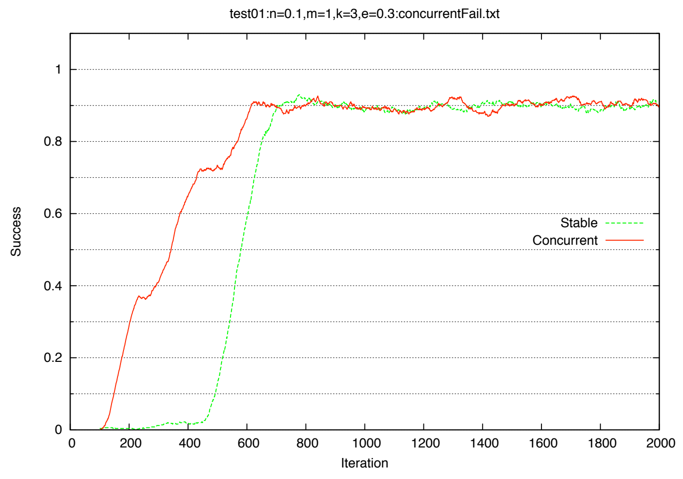


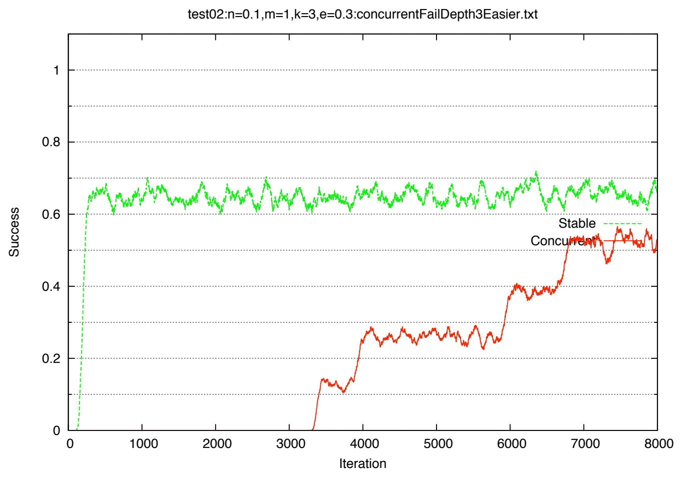


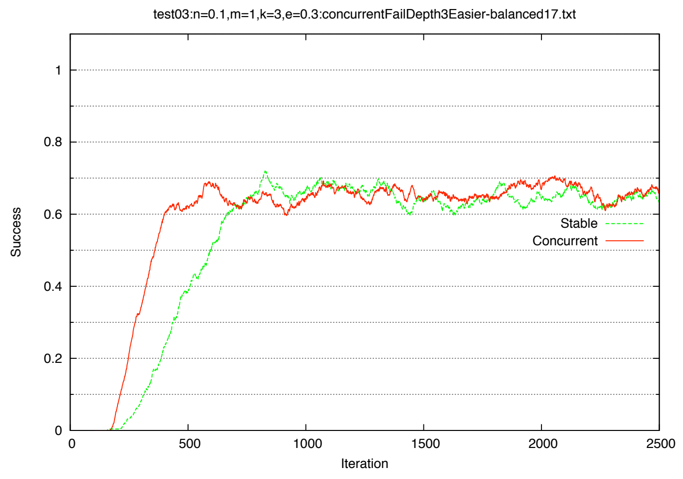


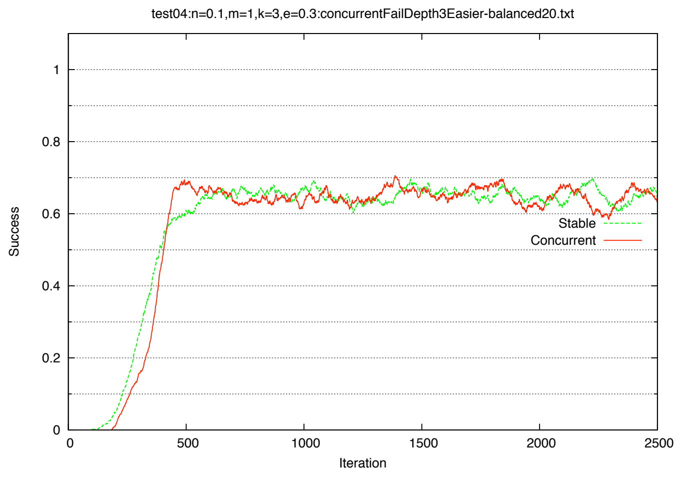


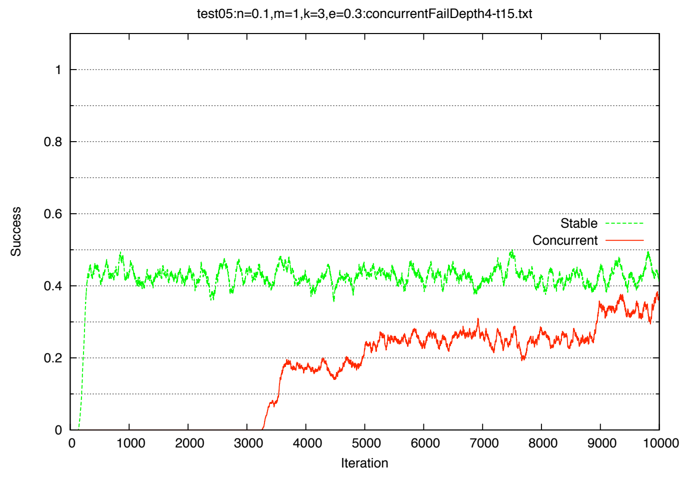
After applying suggested changes using k=3
(Repository Revision 49+)

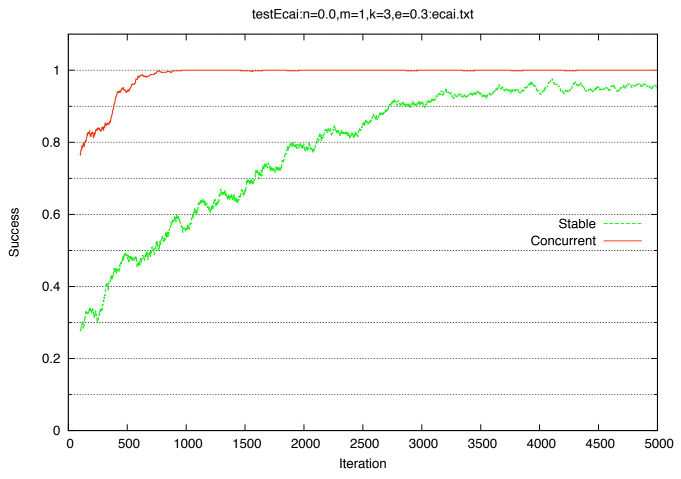


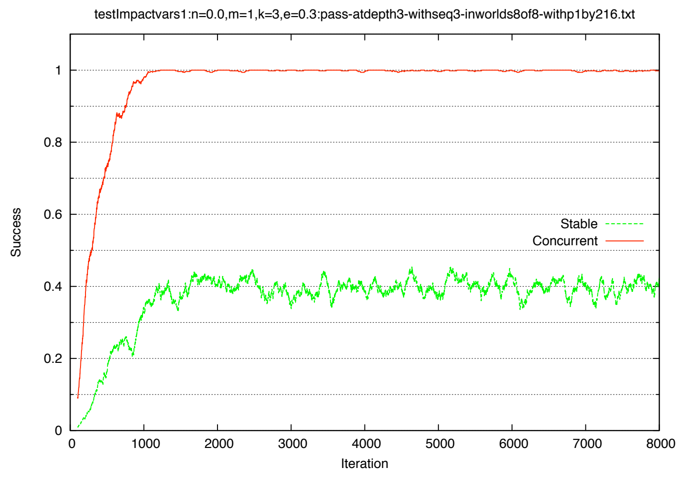




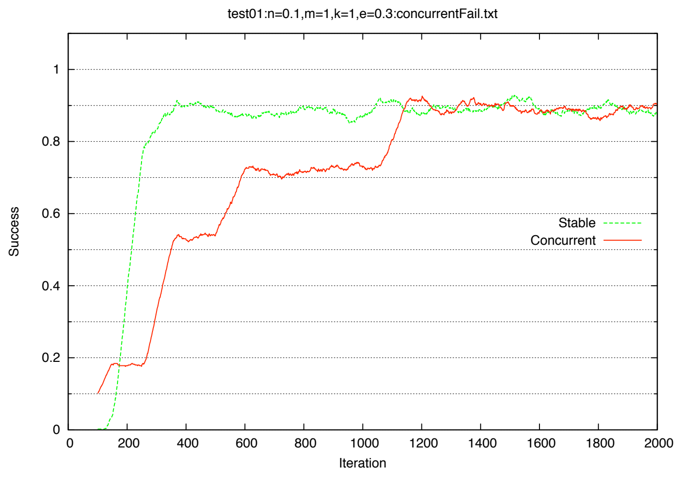


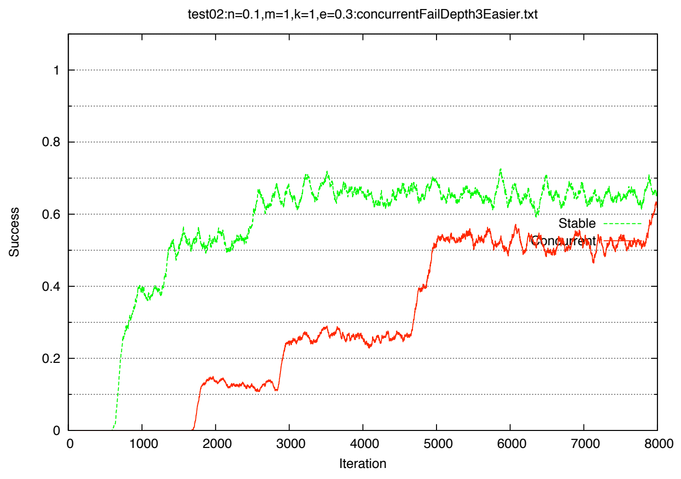


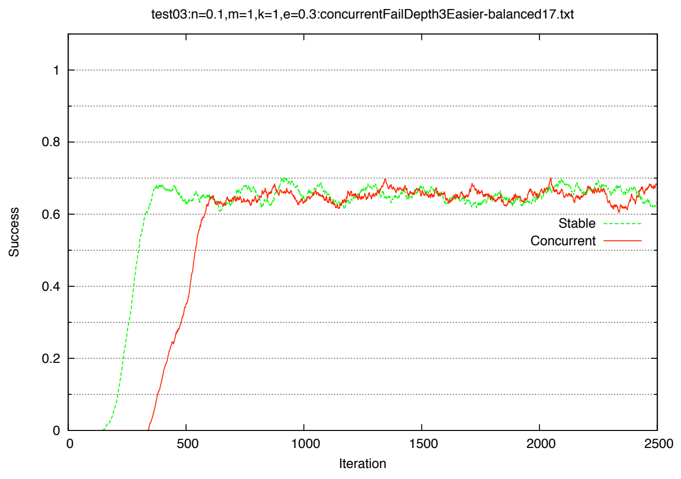


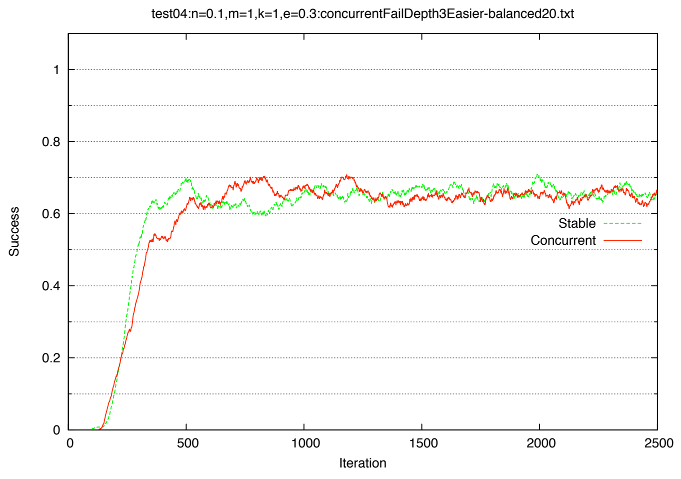


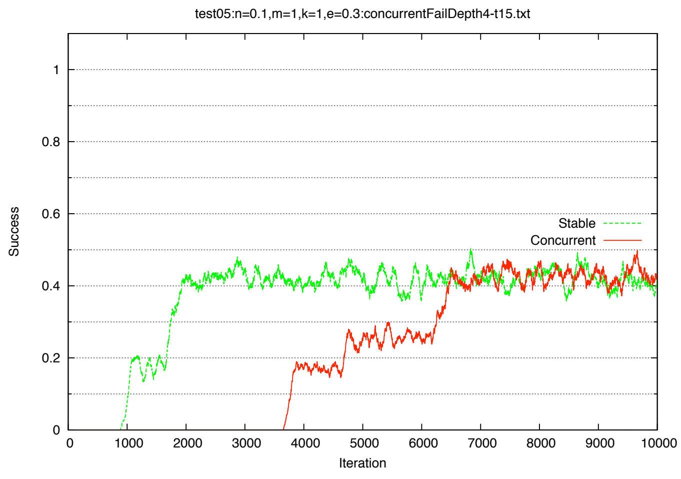
After applying suggested changes using k=1
(Repository Revision 49+)

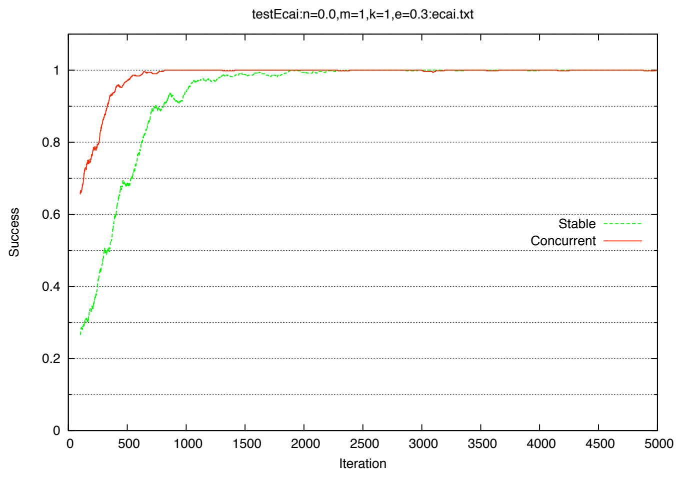

test01:n=0.1,m=1,k=1,e=0.3:concurrentFail.txt
Success
Iteration
Stable
Concurrent

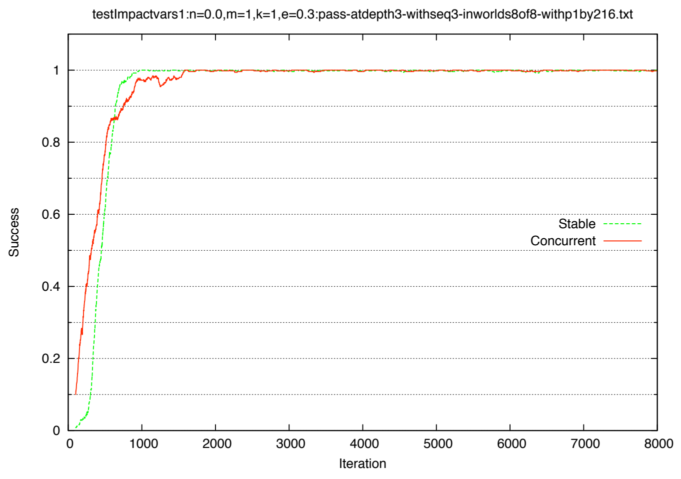

test02:n=0.1,m=1,k=1,e=0.3:concurrentFailDepth3Easier.txt
Success
Iteration
Stable
Concurrent


test03:n=0.1,m=1,k=1,e=0.3:concurrentFailDepth3Easier-balanced17.txt
Success
Iteration
Stable
Concurrent


test04:n=0.1,m=1,k=1,e=0.3:concurrentFailDepth3Easier-balanced20.txt
Success
Iteration
Stable
Concurrent


test05:n=0.1,m=1,k=1,e=0.3:concurrentFailDepth4-t15.txt
Success
Iteration
Stable
Concurrent


testEcai:n=0.0,m=1,k=1,e=0.3:ecai.txt
Success
Iteration
Stable
Concurrent


testImpactvars1:n=0.0,m=1,k=1,e=0.3:pass-atdepth3-withseq3-inworlds8of8-withp1by216.txt
Success
Iteration
Stable
Concurrent