BDI-Learning Discussion Paper: Improving Performance in Multiple Worlds

Dhirendra Singh
dhirendra.singh@rmit.edu.au

August 19, 2009

Contents

1 Motivating example

1.1 The scenario

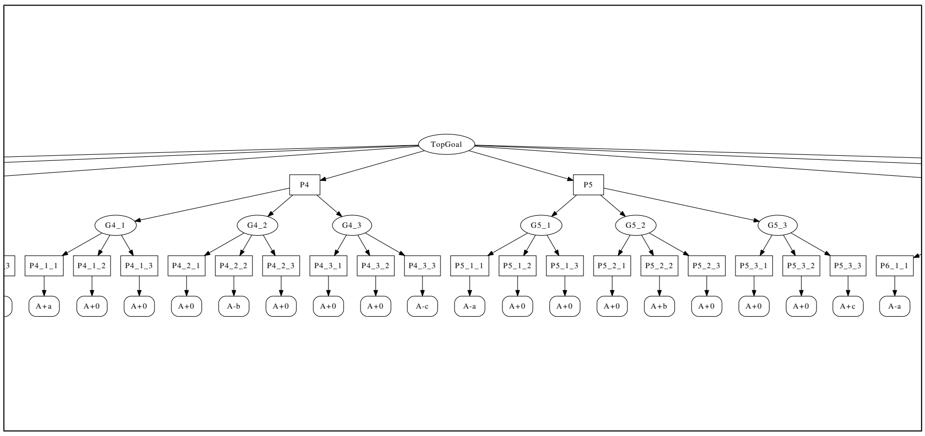


Figure 1: $testImpactvars$ input tree (sample portion only)

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 $[P1 \ldots P8]$ 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 $[P1 \ldots P8]$ 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.

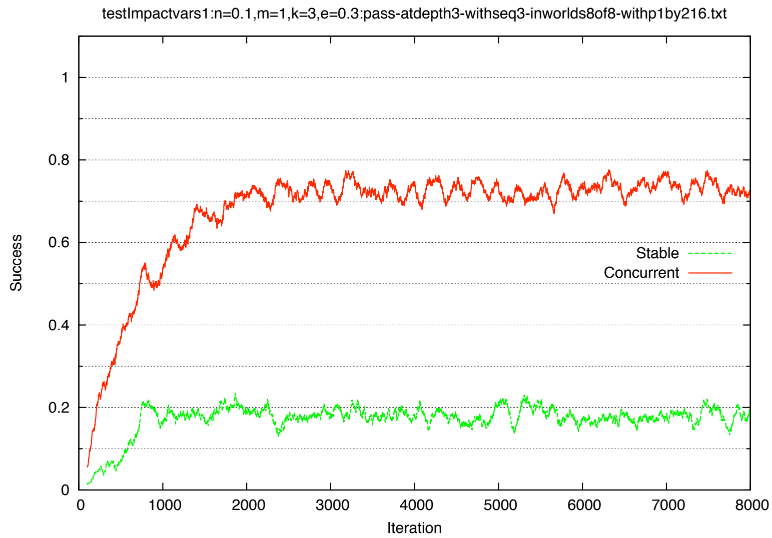


Figure 2: Performance comparison for $testImpactvars$

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.

2 Understanding the problem

2.1 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. Note that here the environment is not stochastic so if a plan fails then it is a true failure and there is no chance of it passing in that given world in the future; furthermore the single plan wired for success will succeed in *all* worlds not just a small portion of the worlds space. Even in this lenient case the problem is significant.

However, while the problem worsens as the branching factor increases, the probability of the problem state occurring in the first place decreases. In fact for M applicable plans, the probability of recovering from the problem is 1/M and the probability of witnessing the problem state is also 1/M.

- 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.

As a result the exploration of *Stable* is unfairly biased leading to a slower convergence than expected. This 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.

2.2 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 2.1 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.

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.

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]	$\cdots$	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 $[P_i, P_j, P_k]$

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.

2.3 Some insights into the workings of *Concurrent* and *Stable*

Finally, before we address the individual issues, here is some general insight into the differences between the two approaches and the scenarios where one performs better than the other.

- *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.

3 Improving *Stable* performance I

3.1 When to use a decision tree?

Section 2.1 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.

The 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.

Comment: This recommendation is declined as the $haveSeen$ check is too restrictive and defeats the purpose of using the decision tree (for generalisation) in the first place.

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

Section 2.2 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.

3.3 Test results

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

- "Repository Revision 49" shows the baseline results before any changes were applied. Notice the problematic *testImpactvars* result here.
- "Repository Revision 61" shows the results after applying the changes from Section 3.2. The change does not break any previous tests but also does not show any significant improvement in any results. Nonetheless, the change is appropriate and has been committed.

4 Improving *Stable* performance II

4.1 When to use a decision tree: The *confidence* measure

The core issue discussed in Section 2 is that of mis-classification by decision trees due to incorrect generalisation, that in turn results from the paucity of samples during the early stages of online learning. Our discussions highlight the necessity to consider two key elements in the use of decision trees in this case:

1. The current *probability* of success in a given world as predicted by the decision tree; and
2. Our *confidence* in the current prediction.

For measuring confidence we presently use a crude criterion, *that the change in probabilities be small between successive queries*. Section 2 shows that this criterion is insufficient when dealing with multiple worlds as it does not consider the specific world in question. An improvement using a $haveSeen$ check suggested in Section 3.1 is also not appropriate since it forces the strict constraint that the world be witnessed before the decision tree may be used to classify it, thereby defeating the purpose (i.e. interpolation) of using a decision tree in the first place.

In defining the characteristics of a *confidence* measure we identify the following properties:

- It must consider the world W being witnessed. Since we have not seen W before, then this becomes a function of how times have we seen *similar* worlds before. For a decision tree, we might frame this as - given a decision tree leaf node L that will classify W, how many instances of other worlds $W' \neq W$ are being classified by the same node L.
- Over time, the measure must monotonically tend from $0 \rightarrow 1$.

For instance, consider the issue described in Section 2.1 where we have 20 applicable plans such that 19 bad plans report a correct $p = 0$ while the single good plan reports an incorrect $p = 0$ (generalisation error). Here $\frac{19}{20}$ times a bad plan will be executed but the choice will be *wasted* because the result it will not change the relative probabilities (they are already all at zero). Ideally, one would expect that as the 19 other plans are getting selected and failing, that the relative probability of selection of the good plan (not selected yet) should keep improving as a consequence. And when the good plan is finally selected and succeeds, it's relative probability should increase further $\rightarrow 1$. Currently, the former does not happen - we gain no information from the failure of the other 19 plans when we should.

A well formed confidence measure would resolve this issue. In this case as each bad plan is selected and fails, this information is recorded in the confidence measure that consequently tends $\rightarrow 1$, even though the probability of success (already at zero) does not change with each failure.

Given this confidence measure, we can then use a threshold value to determine when it makes sense to use the probability given by the decision tree. For situations where the confidence measure is lower than the threshold, the default probability to use is 0.5 as described in Section 3.2.

4.1.1 A first-pass confidence measure

We use a first-pass confidence measure based on the *out-of-bag* error from a bootstrap aggregating (bagging) approach [**?**].

Given a standard training set D of size n, bagging generates p new training sets D_i of size $n' \leq n$, by sampling examples from D uniformly and with replacement. By sampling with replacement it is likely that some examples will be repeated in each D_i. If $n' = n$, then for large n the set D_i is expected to have 63.2% of the unique examples of D, the rest being duplicates. This kind of sample is known as a bootstrap sample. The p models are fitted using the above p bootstrap samples and combined for classification through a voting mechanism. The out-of-bag error here is the classification error on the remaining 36.8% of the samples.

First, we change the semantics of the original m parameter (that defines the number of instances required before a decision tree may be used) slightly so that m now determines *the number of unique instances* required before a decision tree may be utilised. This provides a more information rich sample set than originally. For a world consisting of f binary *features* (or variables), m selects a subset $S' \subset S$ where $|S| = 2^f$. This way m selects a subset (of all possible worlds) large enough to provide suitable performance.

Second, we calculate the out-of-bag error for the training set S'. The choice of whether to use the decision tree is determined by the out-of-bag error in relation to a pre-determined threshold value. For our implementation we use a threshold value of 0.5. Put simply, this means that we will choose to utilise the decision tree only when it's classification of the validation set is better than *random*.

Note here that the choice of m for a given experiment is not obvious. As a first-pass, we use the function $m = f$ for this value where f is the number of features in the world. Our experiments show that this rather simple function gives us a sufficiently rich base training set to work from.

4.2 Exploring when not using decision trees

Introducing a measure of confidence for the decision trees (Section 4.1) implies that for each node in the goal/plan hierarchy several samples may pass before we begin to rely on the respective decision tree. For the duration that the decision tree is not ready, we currently use a random exploration strategy based on the default probability of success. This strategy to explore randomly when not using a decision tree is far from ideal. A better strategy would be to utilise the information being recorded at each episode in order to make informed choices in these early stages.

The inefficiency in using random exploration is apparent as we move further up the goal/plan tree. Firstly, since the stability idea effectively regulates the flow of (failure) information up the goal/plan hierarchy, then the amount of samples available at any one level is inherently less than the level below. This means that decisions have to be made using fewer samples at each higher level. Secondly, choices at higher levels have more weighing towards success. An incorrect choice at a higher level may ruin all chances of success in a given episode regardless of any bottom level decisions. Finally, it is not uncommon in our experiments for top level nodes to not become stable until late in the experiment and therefore to make decisions without help from suitably mature decision trees. This fact further motivates the requirement for a informed strategy in the absence of decision trees.

In repository revision 63, we have implemented an exploration strategy that is based on probabilities biased by the *hamming distance* to a previously seen world, when the decision tree is not ready. The heuristic first finds the previously witnessed world W' with the minimum hamming distance $h_{W'}$ to the current world W, and then biases the probability of success p_W in W with the witnessed result in W', as show in Equation 1. A maximum bias is applied when $h_{W'} = 0$, while no bias is applied for $h_{W'} = h_{max}$.

$$p_W = 0.5 + \left[\left(\frac{success_{W'}}{attemepts_{W'}} - 0.5 \right) * \left(1 - \frac{h_{W'}}{H} \right) \right] \tag{1}$$

Comment: Biasing exploration based on feature similarity between worlds (as used in the hamming distance) is not a satisfactory solution because we cannot make a general claim that such a bias even makes sense. An exploration strategy based on information gain (per feature) is preferred. Nonetheless these results are useful and may be used as a baseline measure for the information-based strategy.

4.2.1 Ideas for information-based plan choice

An informed exploration strategy might bias selection in order to maximise the *information gain* from the choice. Stéphane has proposed the following preliminary ideas for this extension:

- Given information about the number of instances n classified by leaf node L (where L is the node that also classifies W), a first simple idea would be to select the plan with the lowest n. Equally we could select a plan with a probability inversely proportional to n.

- One possible refinement to this would be to take into account the size of the input space represented by the leaf node. For example, if the leaf nodes for two plans correspond to worlds $[a]$ and $[a \cdot \bar{b} \cdot c]$ respectively and each contains the same number of instances, then the selection could favour the plan that expresses the larger portion of the worlds space, in this case $[a]$.

- Another refinement is to use *entropy*. This would take into account the number of failures/successes instead of a simple count of the instances contained in the node of the decision tree. We can compute the change in entropy if the plan succeeds and if the plan fails, and maybe go with the plan with the biggest entropy drop. (Stéphane says: I would need more time to check whether that makes sense, if observing say a failure is more promising than observing a success in terms of information gain for one DT/plan, and the opposite for another DT/plan, the decision is not that simple).

- Another option is to *simulate* the possible updates of the decision tree for each plan and base the decision on the outcome. For instance, it is possible that for the decision tree of a plan $P1$ that regardless of whether $P1$ succeeds of fails, the decision tree structure will not change (the probability in the corresponding leaf node of the decision tree only will change). For another plan $P2$ however, the new instance may trigger a change in the decision tree structure. In this case, we would favor $P2$ over $P1$.

4.3 Consequences of using a default $p = 0.5$

The decision to use a default $p = 0.5$ (refer Section 3.2) introduces a further subtlety in the meta level probabilistic plan selection (PS) mechanism. Consider the case where we have two applicable plans P_1 and P_2 whose individual probabilities of success in world w are given by the set $[0.5, 1.0]$. This represents the situation where P_1 is using a default $p = 0.5$ and P_2 is using a true $p = 1.0$. In this case, one would expect P_2 to be selected more often than not because we are confident in it's success in world W. However, the PS mechanism will use the relative probabilities $[\frac{0.5}{1.5}, \frac{1.0}{1.5}]$ which means that P_2 has a selection probability of only 0.67. Now, say P_1 was a deep subtree that does not hold a solution. Then we would be spending a third of our resources in exploring P_1 when we already have a suitable solution in P_2. The problem becomes more pronounced as the number of applicable plans grows. Consider the case of 20 applicable plans, 19 of which are bad but use a default $p = 0.5$, and only one is good (has the solution) and is using the true $p = 1.0$. In this case, the relative probability of selection of the good plan is only $\frac{1.0}{1.0+19*0.5} \approx 9.5\%$.

One way to improve the weighing of the higher probabilities during plan selection is to use a power function p^a for the individual probabilities, however it is not clear how the value a should be selected. A predefined a is not suitable as the set of applicable plans may vary significantly. Using $a = f(n)$ where n is the number of applicable plans is an option but our empirical testing shows that the plan selection (and the subsequent experiment outcome) is quite sensitive to the choice of a and the several tried options for $f(n)$ did not produce a robust enough result across all experiments.

Algorithm 1 $pSelect(P)$

1: $p_m \leftarrow max(P)$
2: $P' \leftarrow (P/p_m)^a$
3: $P'' \leftarrow [P' \cdot (P' \neq 1)] + [P' \cdot (P' = 1) \cdot |P|]$
4: return P''

Algorithm 1 shows the final heuristic we use to calculate the relative probabilities of selection. Here P is the set of probabilities of the individual plans such that p_m is the maximum probability in the set P (Line 1). We obtain a new set P' with the original probabilities scaled to 1.0 and using a power factor $a = 3$ (Line 2), and further boost the probabilities of the best candidate plans thus obtained (i.e those that have $p = 1.0$ in the set P') by the size of the set $|P|$ (Line 3). This ensures that the relative probability of the best plans is greater than the remainder by a factor of *at least* $|P|$. The final set P'' is then used to make the probabilistic selection. Our experiments show that this heuristic provides a good balance between the size $|P|$ of the set, and the maximum probability p_m in the set.

Since the choice of the probabilistic selection mechanism has a significant impact on the results, a better approach would be to formally validate this function so as to guarantee a probability profile. Algorithm 1 has only been verified empirically.

4.4 Changes to stability checking for child nodes

When doing stability checks for children of a node, a termination check was added to ensure the following:

- For a *Goal* node N, when checking the stability of the children plans set C in world W, the stability checking should stop and the node N be marked stable when a child C_i is found to be *successful* in world W. This is because if $C_i \in C$ is successful in W, then it will almost always be selected and the other plans $C_j \neq C_i$ would almost never become stable.

- For a *Plan* node N, when checking the stability of the children goals set C in world W, the stability checking should stop and the node N be marked stable when a child C_i is found to be *unsuccessful* in world W. This is because if $C_i \in C$ is unsuccessful in W, then the other subgoals $C_j \neq C_i$ would never be tried and would therefore never become stable.

This fixes an existing issue with the stability checking algorithm.

4.5 Test results

The test results from applying the changes are included in the Appendix A.

- "Repository Revision 61" is the baseline results before any changes were applied. Notice the problematic *testImpactvars* result.

- "Repository Revision 63" shows the results after applying the changes from Section 4. Noticeably, *testImpactvars* performance is now fixed. Interestingly, the change also resolves the dummy variables issue of test *testDummyvars* (using m=20, where *Stable* performs poorly when forced to consider features of the world that have no final bearing on the results (her $19/20$ features have no impact on results). The *testDummyvars* result was produced using $m = 20$ as described in Section 4.1.1

Comment: Repository revision 63 fixes all our existing tests! However we have not approved these changes because (1) the hamming distance based exploration should be replaced by an information-gain based approach; (2) the modification to the probabilistc plan selection in Section 4.3 is the result of an arbitrary formulation that works for our purpose but has not been evaluated for the general case; and (3) the confidence measure based on a modified m semantic and a out-of-bag error is not robust enough.

5 A new *Coverage*-based approach

5.1 Motivation

The latest revision of the *Stable* approach described in Section 4 works for our base tests but suffers from the genral issue of determining *when* a decision tree is ready for use. The usefulness of the confidence measure of Section 4.1.1 depends heavily on the *richness* of the sample set used in the measure, for which we use (an arbitrary) m value. Overall the performance of the *Stable* approach is controlled by three user defined parameters: m, k, and ϵ. It is difficult to see how these values should be defined for different settings.

Let us first revisit the idea of interpolation with decision trees in our context. Consider the $P4$ plan sub-tree of Figure 1 where the only solution consists of the sequential selection $[P4\_1\_1 \cdot P4\_2\_2 \cdot P4\_3\_3]$ in state $a\bar{b}\bar{c}$. Now, given that we are armed with a decision tree for $P4$ (disregard the readiness for now) and we enter the world $a\bar{b}\bar{c}$ for the first time, what should we expect from the decision tree in terms of classification of the world $a\bar{b}\bar{c}$? The answer is that we would like to classify this world as best we can based on some generalisation of worlds that we have in fact seen. Previously we have said that such classification lies at the core of our intuition to use decision trees in the first place.

However, our faith in the decision tree is slightly misplaced here because we do not consider the goal/plan hierarchy in that argument. For the decision tree to make useful predictions about the plan $P4$ in world $a\bar{b}\bar{c}$ not only should it have seen the world before, but in fact seen it a few times in different configurations i.e. different combination of choices at the child nodes (for $P4$ there are 27 configurations possible in a given world). So our expectation that the decision tree generalise between worlds makes sense only for *leaf* nodes of a goal/plan tree, but weakens as we move further up the hierarchy.

This observation leads us to think of the readiness of the decision tree of a goal/plan node n to generalise for a given world W in terms of the *coverage* of the goal/plan subtree of node n in that world, and forms the basis of the new *Coverage*-based approach.

5.2 *Coverage* Implementation

Equation 2 shows how coverage is used to revise the decision tree prediction. The *Coverage* approach biases the probability $p_n(W)$ of success of a node n in a given world W by the coverage $c_n(W) \in [0, 1]$. As coverage $c_n(W) \to 0$, the revised probability $p'_n(W) \to 0.5$ and we favour the default probability of success, while as $c_n(W) \to 1$, then $p'_n(W) \to p_n(W)$ and we favour the decision tree classification.

$$p'_n(W) = 0.5 + [c_n(W) * (p_n(W) - 0.5)] \tag{2}$$

Intuitively, this approach lies somewhere between *Concurrent* that makes no consideration of coverage, and *Stable* that inherently waits for full coverage during the stability check. The immediate impact is that we no longer have to answer the question of when a decision tree is ready. We do away with the boundary calculation for readiness as in *Stable* and replace it with a smooth transition function. Furthermore, the approach is more robust since we no longer require the user defined runtime parameters m, k, and ϵ.

5.3 Test results

The test results comparing *Coverage* performance to *Concurrent* and *Stable* are included in the Appendix A.

- ”Repository Revision 63” shows the baseline results.
- ”Repository Revision 69” shows the results after applying the changes from Section 5. We see that *Coverage* has the best performance for the basic tests 1 to 5 (these are the same tests reported in the IJCAI submission and run in one world). For the multiple-worlds cases, the approach compares well to *Stable* (with hamming distance based exploration). Notably though, *Coverage* performs very poorly for *testDummyVars* that has 19 dummy variables.

6 Improving *Coverage* performance

6.1 Motivation

In Appendix A "Repository Revision 69" the poor *Coverage* performance for *testDummyVars* highlights a weakness in the approach as the number of variables grows. This is evident in the bias of Equation 2 which defaults to 0.5 for as yet unseen worlds. For *testDummyVars* we have 2^{20} different worlds and so the likelihood of having seen a given world previously is small in the early stages. As such *Coverage* is essentially guessing for a large part of the experiment. While *testDummyVars* uses dummy variables only, it is nonetheless obvious that *Coverage* performance would be similar even if all variables were in fact relevant.

6.2 Implementation changes

To improve *Coverage* performance for unseen worlds, we adjust our probability bias slightly to use the coverage of world W' that we have seen before and is the *most relevant* match to W. This allows us to exploit our existing experiences to make a decision for the unseen world W.

At this point, we may question how this is any different to just using the decision tree as is, considering this is exactly the sort of scenario we hoped to use the decision tree for in the first place. Why not simply use the decision tree prediction $p_n(W)$ as is, when faced with an unseen world W? The problem with that approach of course is that of misclassification. Instead what we would like is a smoother integration of the decision tree prediction into our decision making based on some measure of our confidence in that prediction.

One simple option we have used before to measure the relatedness of two worlds is using the *hamming distance* as described in Section 4.2. While this is not optimal (we would prefer an information-based approach), it nonetheless gives us a numeric measure to bias the probability with.

$$p'_n(W) = 0.5 + \left[\left(\frac{h_{W'}}{H} * c_n(W') \right) * (p_n(W) - 0.5) \right] \tag{3}$$

Equation 3 shows the updated probability calculation based on the shortest hamming distance $h_{W'}$ between two worlds W and W' . For $W = W'$ this reduces to the original Equation 2, but for all cases where $W \neq W'$ we now have a way to exploit our existing experiences to make a decision for the unseen world W.

6.3 Test results

The test results comparing *Coverage* performance to *Concurrent* and *Stable* are included in the Appendix A.

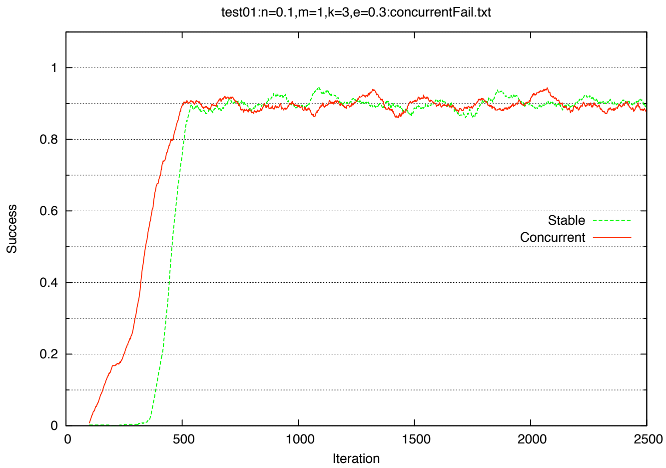
- "Repository Revision 69" shows the baseline results.
- "Repository Revision 77" shows the results after applying the changes from Section 6. Note that the tests 1 to 5 have been updated here for multiple worlds. The vNN in the test name indicates the number of variables used. Tests 1-2 use v=10 and tests 3-5 use v=5. The *Coverage* performance for all tests is now comparable to *Stable*. However, we believe *Coverage* is a more robust approach than *Stable*.

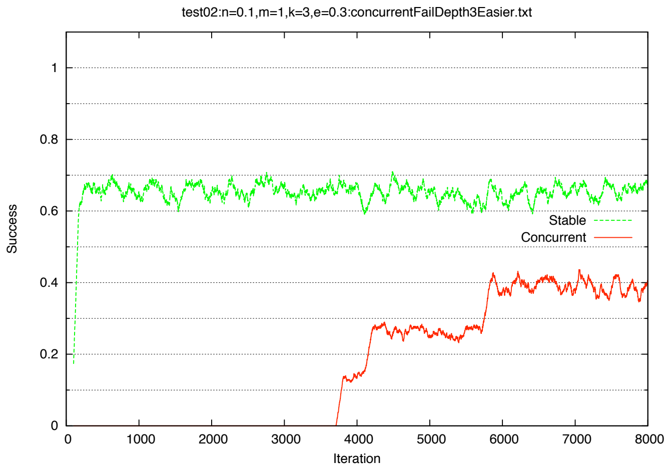
7 Acknowledgements

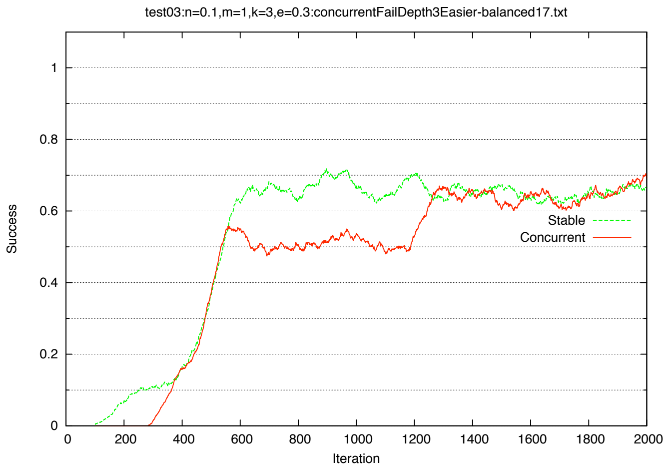
Revisions of this document are a result of an ongoing discussion and contributions from Stéphane Airiau of the University of Amsterdam, and Sebastian Sardina, Andy Song, and Professor Lin Padgham of RMIT University.

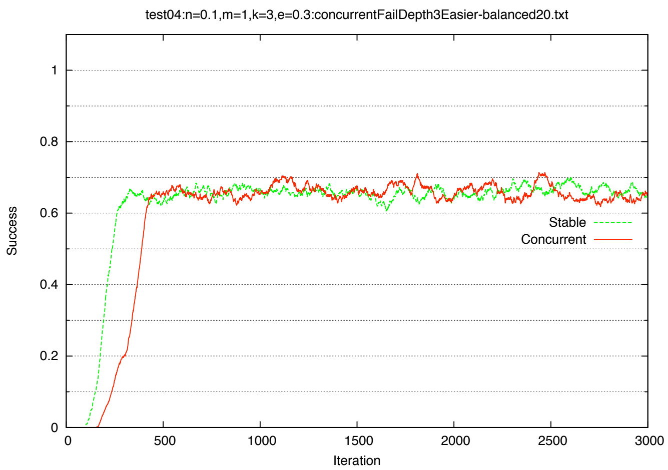
A Results

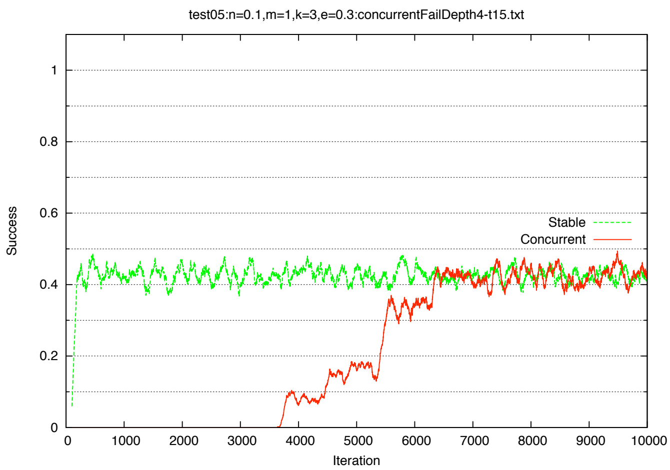
Repository Revision 49
(Baseline)

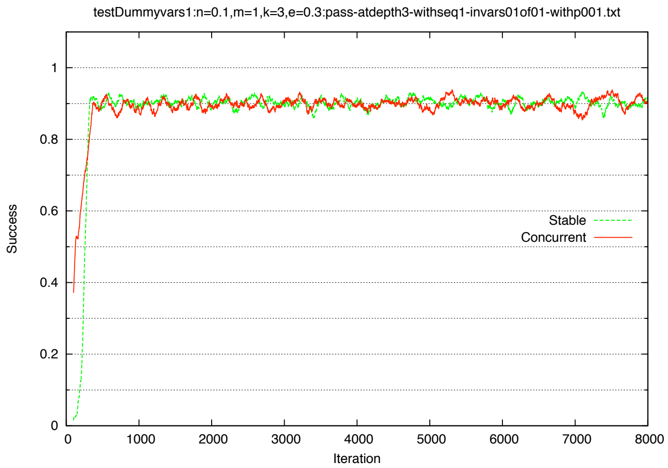


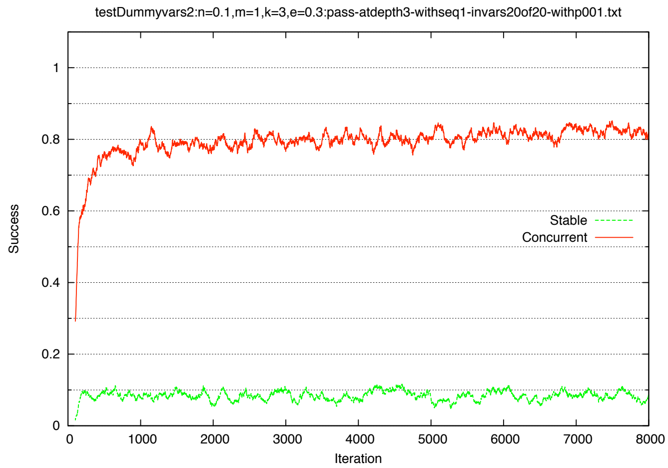


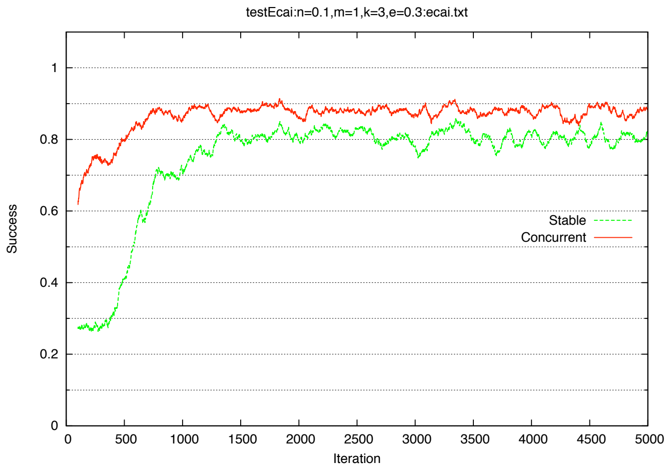


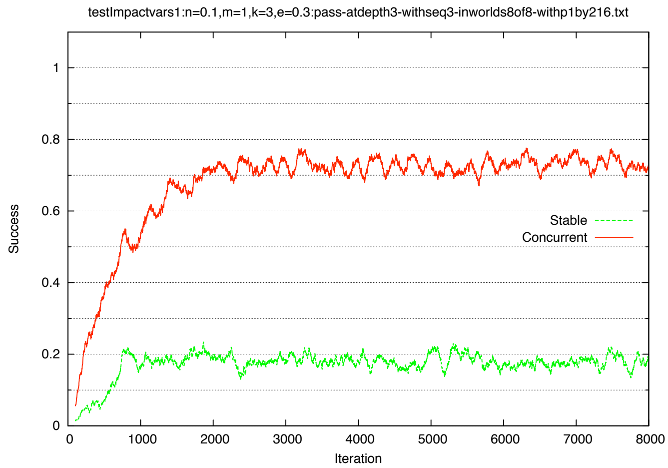





testDummyvars1:n=0.1,m=1,k=3,e=0.3:pass-atdepth3-withseq1-invars01of01-withp001.txt
Success
Iteration
Stable
Concurrent
0
0.2
0.4
0.6
0.8
1
0
1000
2000
3000
4000
5000
6000
7000
8000

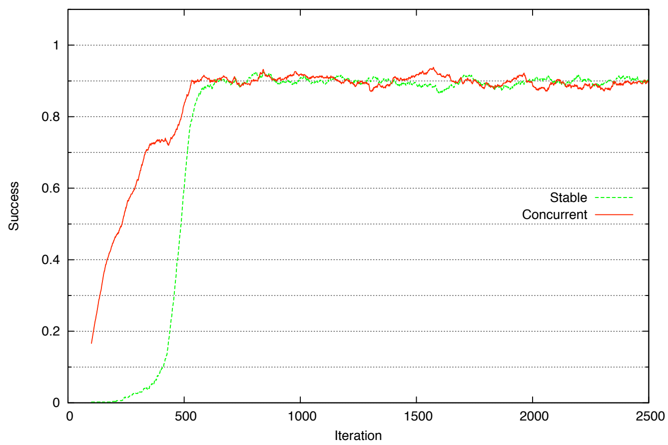

testDummyvars2:n=0.1,m=1,k=3,e=0.3:pass-atdepth3-withseq1-invars20of20-withp001.txt
Success
Iteration
Stable
Concurrent
0
0.2
0.4
0.6
0.8
1
0
1000
2000
3000
4000
5000
6000
7000
8000


testEcai:n=0.1,m=1,k=3,e=0.3:ecai.txt
Success
Iteration
Stable
Concurrent
0
0.2
0.4
0.6
0.8
1
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000

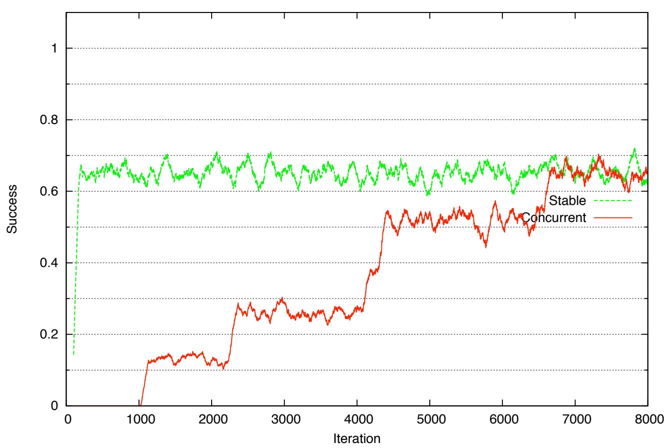

testImpactvars1:n=0.1,m=1,k=3,e=0.3:pass-atdepth3-withseq3-inworlds8of8-withp1by216.txt
Success
Iteration
Stable
Concurrent
0
0.2
0.4
0.6
0.8
1
0
1000
2000
3000
4000
5000
6000
7000
8000

Repository Revision 61
(Section 3 changes: Stable I)

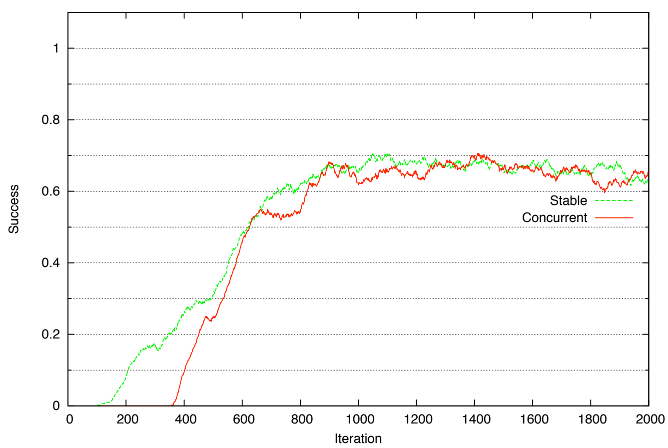
test01:n=0.1,m=1,k=3,e=0.3:concurrentFail.txt



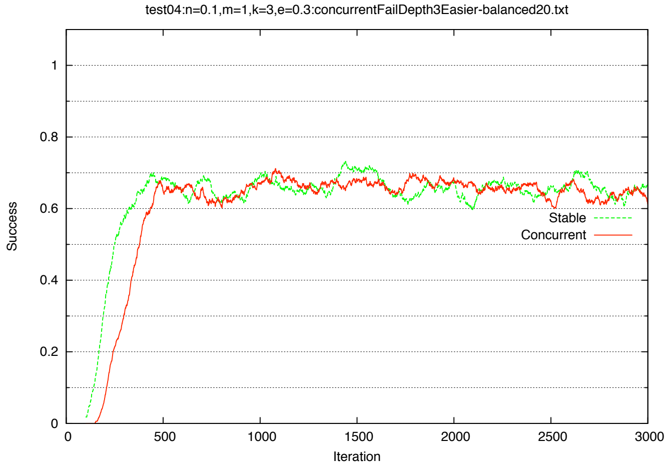
test02:n=0.1,m=1,k=3,e=0.3:concurrentFailDepth3Easier.txt



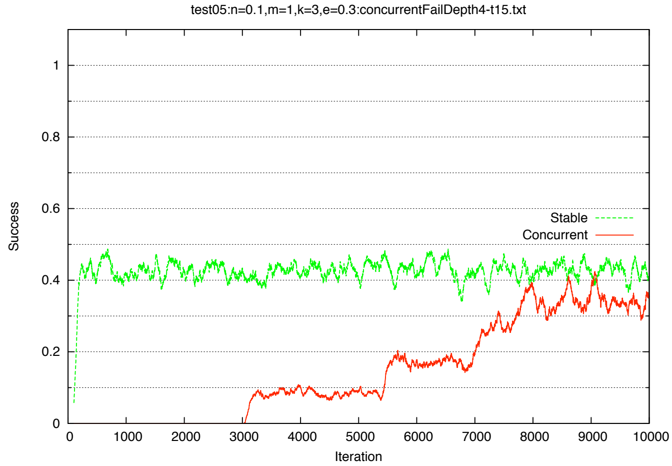
test03:n=0.1,m=1,k=3,e=0.3:concurrentFailDepth3Easier-balanced17.txt

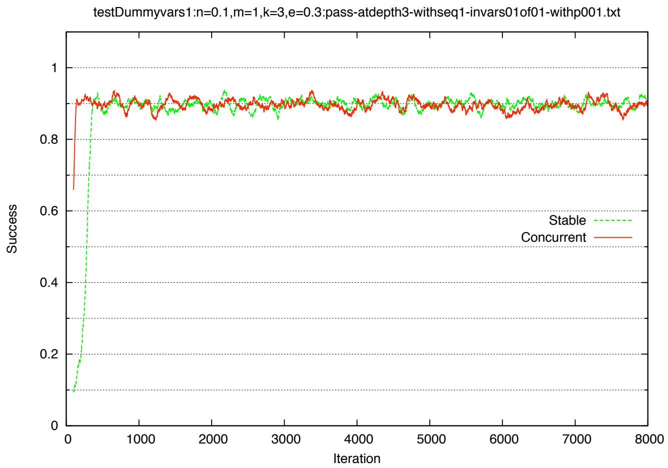


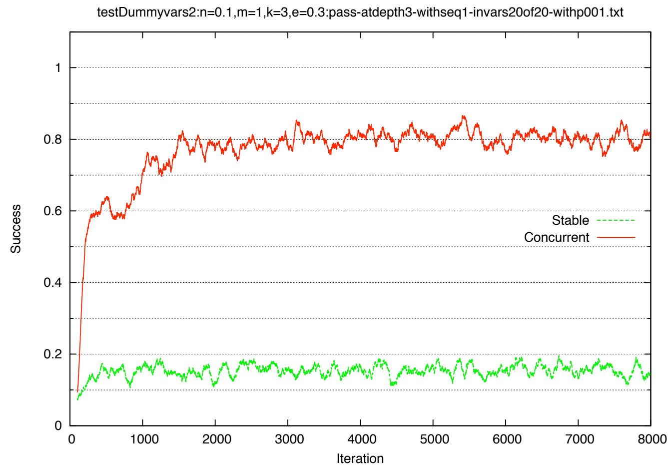
test04:n=0.1,m=1,k=3,e=0.3:concurrentFailDepth3Easier-balanced20.txt

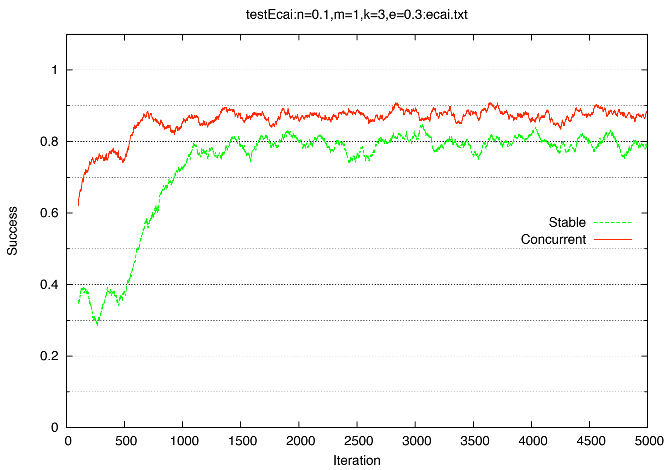


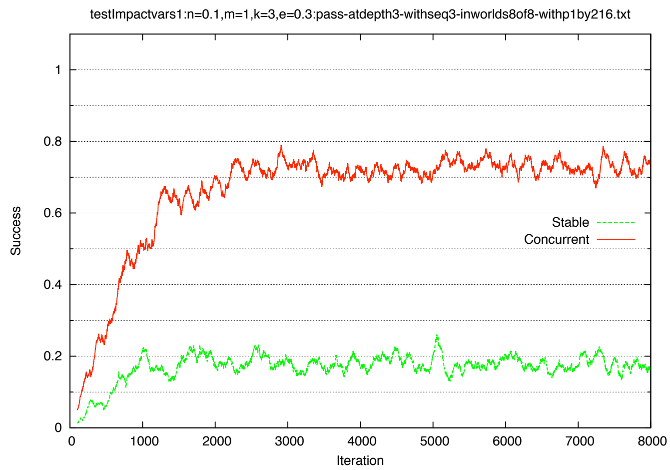
test05:n=0.1,m=1,k=3,e=0.3:concurrentFailDepth4-t15.txt



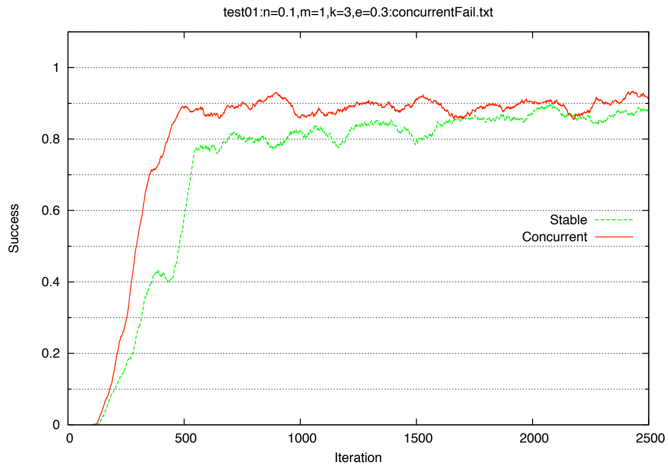

testDummyvars1:n=0.1,m=1,k=3,e=0.3:pass-atdepth3-withseq1-invars01of01-withp001.txt
Success
Iteration
Stable
Concurrent
0
0.2
0.4
0.6
0.8
1
0
1000
2000
3000
4000
5000
6000
7000
8000

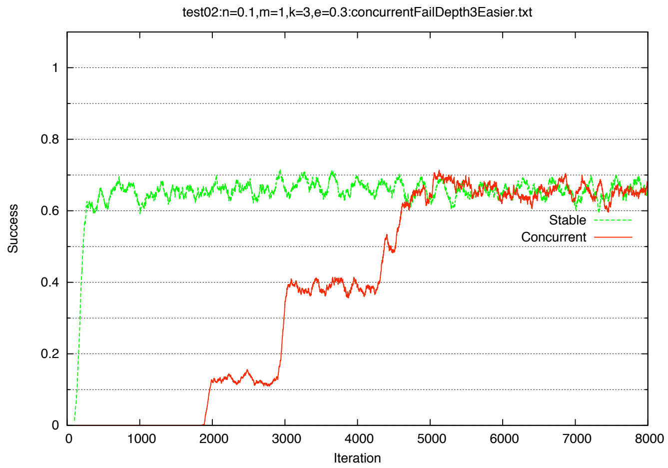

testDummyvars2:n=0.1,m=1,k=3,e=0.3:pass-atdepth3-withseq1-invars20of20-withp001.txt
Success
Iteration
Stable
Concurrent
0
0.2
0.4
0.6
0.8
1
0
1000
2000
3000
4000
5000
6000
7000
8000

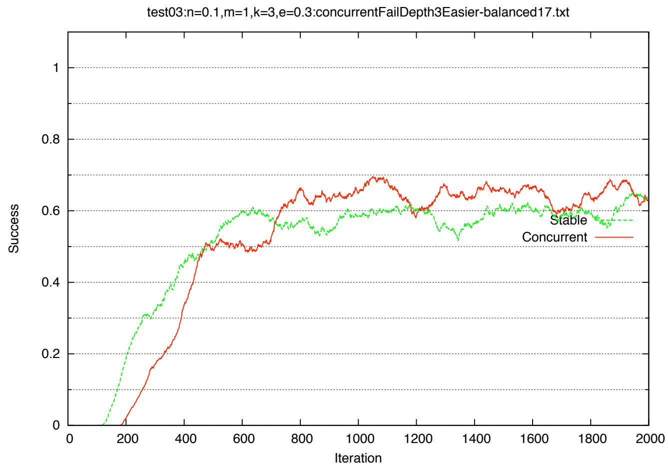

testEcai:n=0.1,m=1,k=3,e=0.3:ecai.txt
Success
Iteration
Stable
Concurrent
0
0.2
0.4
0.6
0.8
1
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000

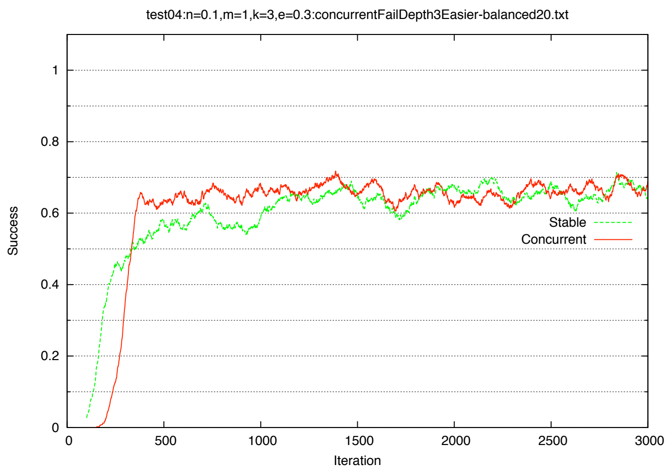

testImpactvars1:n=0.1,m=1,k=3,e=0.3:pass-atdepth3-withseq3-inworlds8of8-withp1by216.txt
Success
Iteration
Stable
Concurrent
0
0.2
0.4
0.6
0.8
1
0
1000
2000
3000
4000
5000
6000
7000
8000

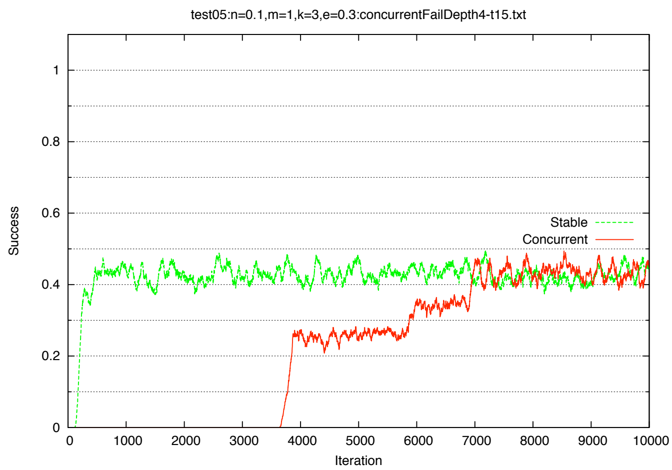
Repository Revision 63
(Section 4 changes: Stable II)

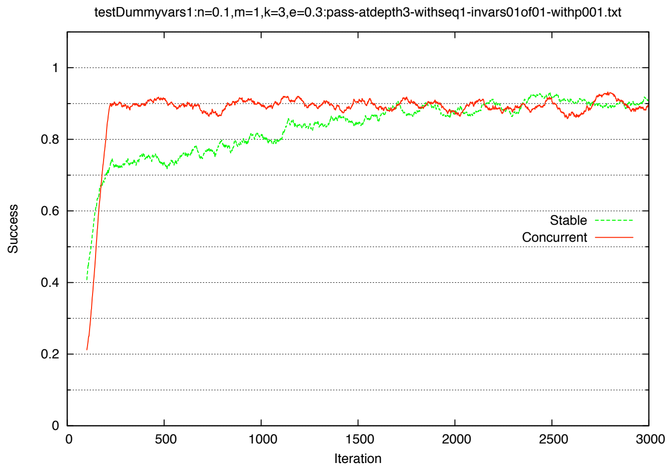


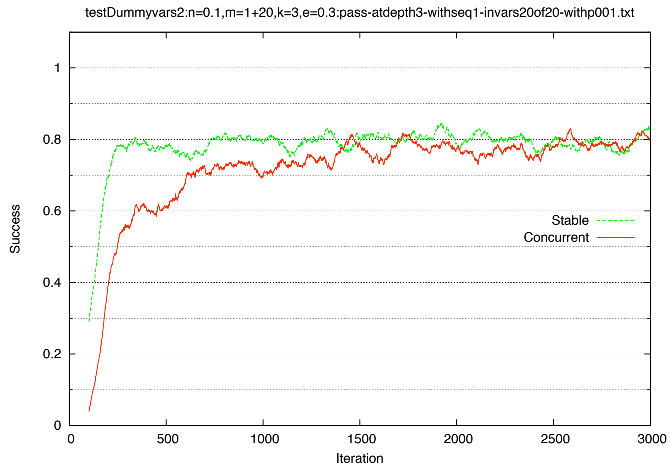


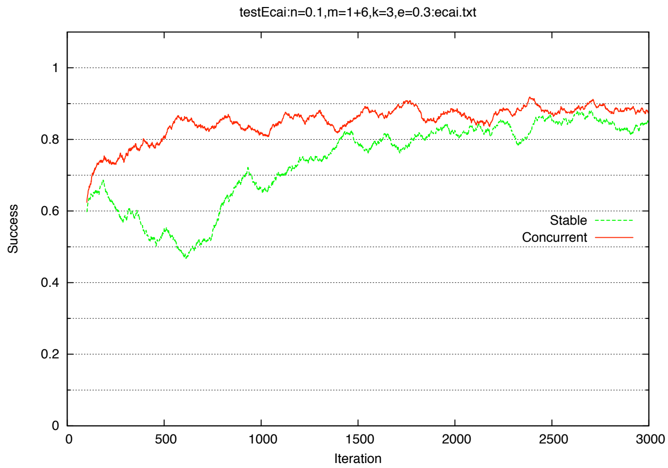


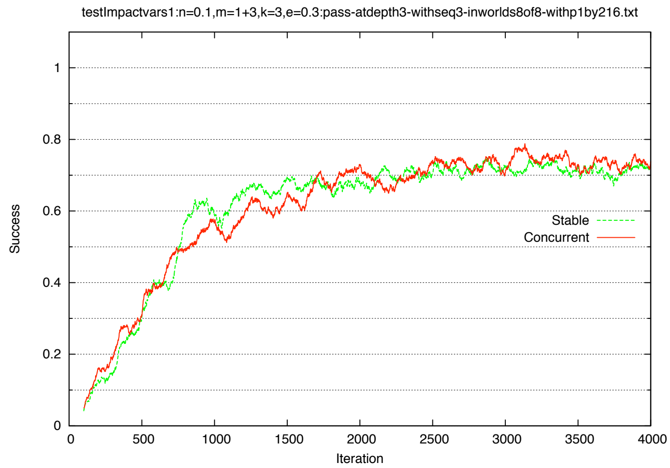




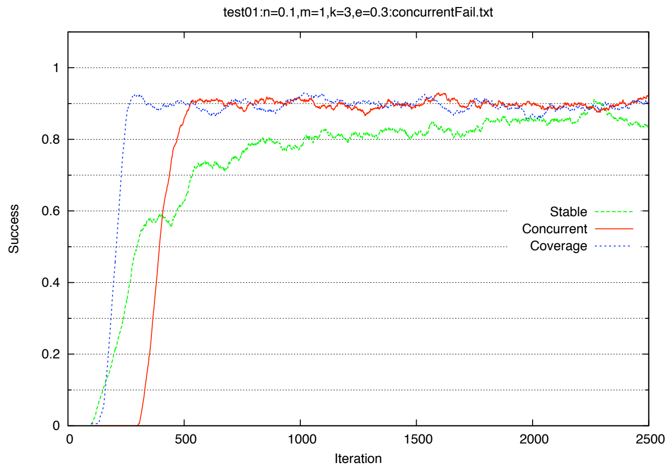

testDummyvars1:n=0.1,m=1,k=3,e=0.3:pass-atdepth3-withseq1-invars01of01-withp001.txt
Success
Iteration
Stable
Concurrent
0
0.2
0.4
0.6
0.8
1
0
500
1000
1500
2000
2500
3000

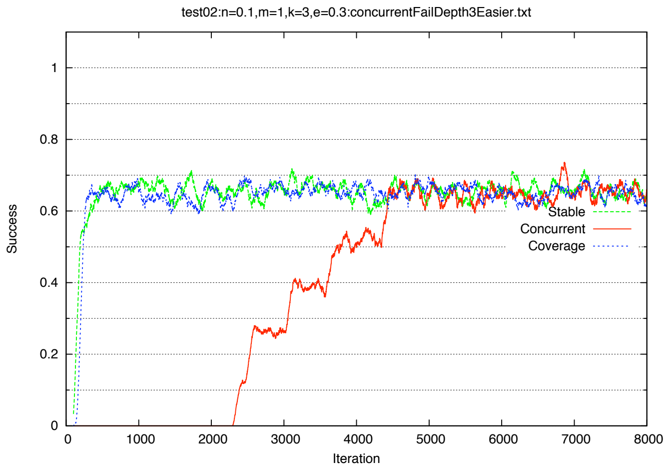

testDummyvars2:n=0.1,m=1+20,k=3,e=0.3:pass-atdepth3-withseq1-invars20of20-withp001.txt
Success
Iteration
Stable
Concurrent
0
0.2
0.4
0.6
0.8
1
0
500
1000
1500
2000
2500
3000

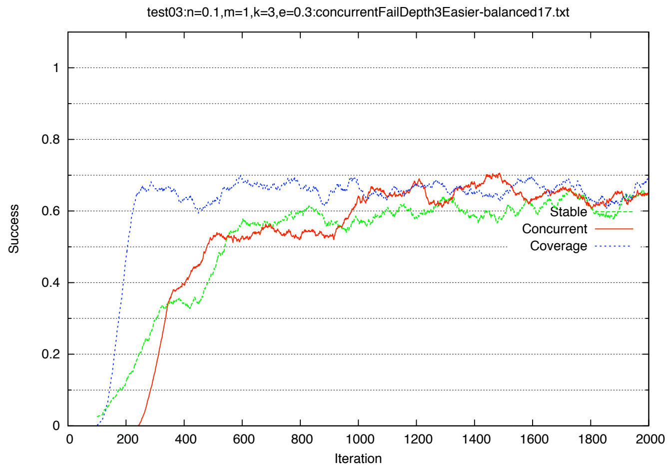

testEcai:n=0.1,m=1+6,k=3,e=0.3:ecai.txt
Success
Iteration
Stable
Concurrent
0
0.2
0.4
0.6
0.8
1
0
500
1000
1500
2000
2500
3000

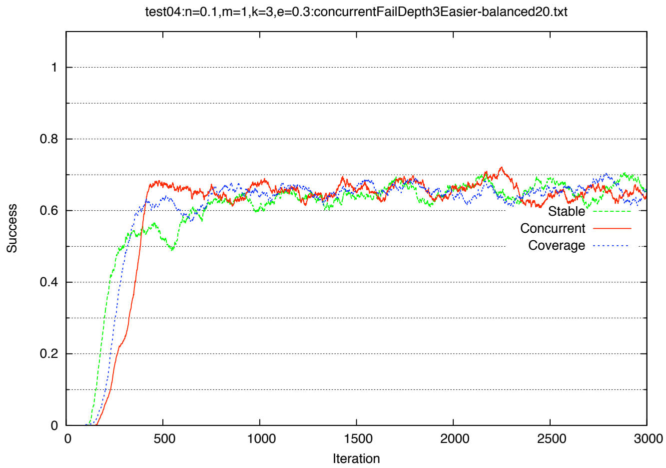

testImpactvars1:n=0.1,m=1+3,k=3,e=0.3:pass-atdepth3-withseq3-inworlds8of8-withp1by216.txt
Success
Iteration
Stable
Concurrent
0
0.2
0.4
0.6
0.8
1
0
500
1000
1500
2000
2500
3000
3500
4000

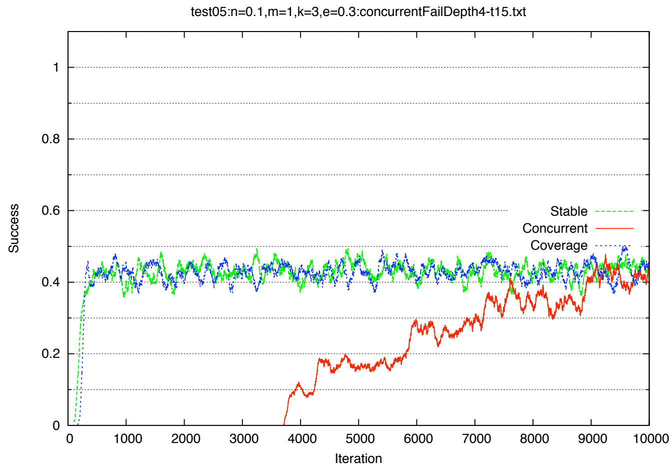
Repository Revision 69
(Section 5 changes: Coverage I)

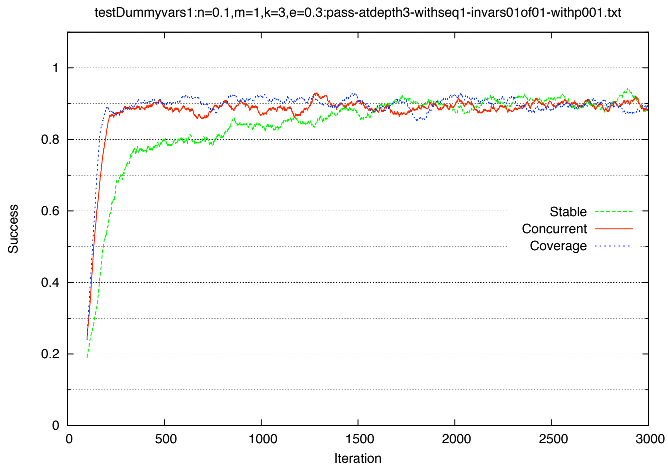


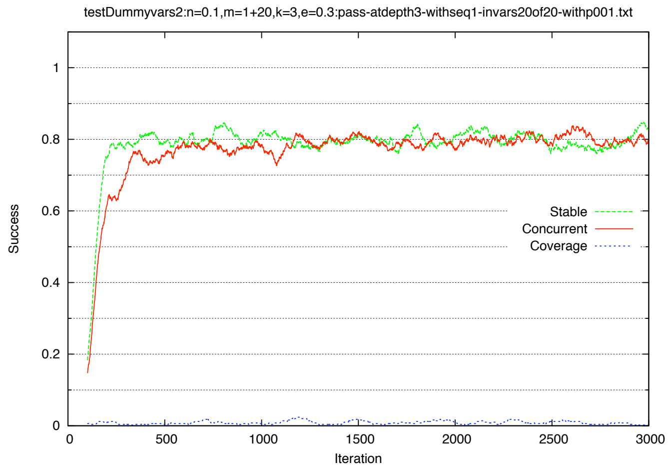


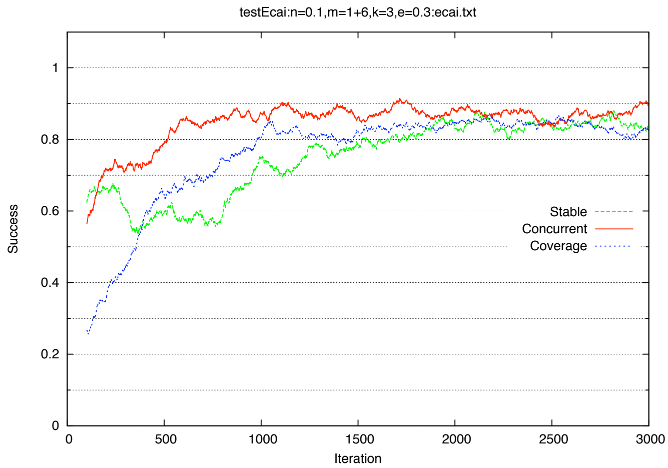


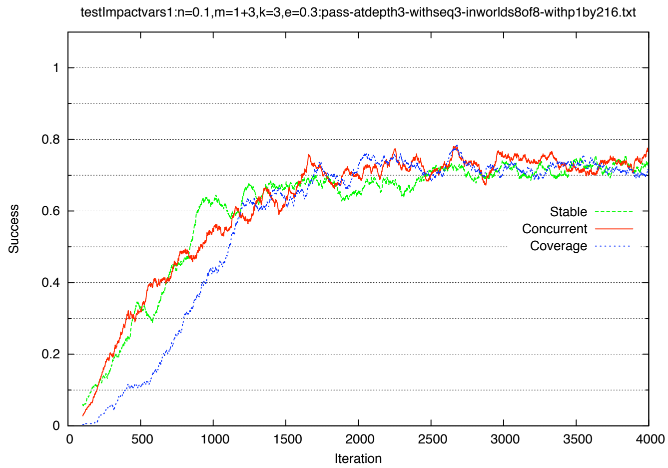




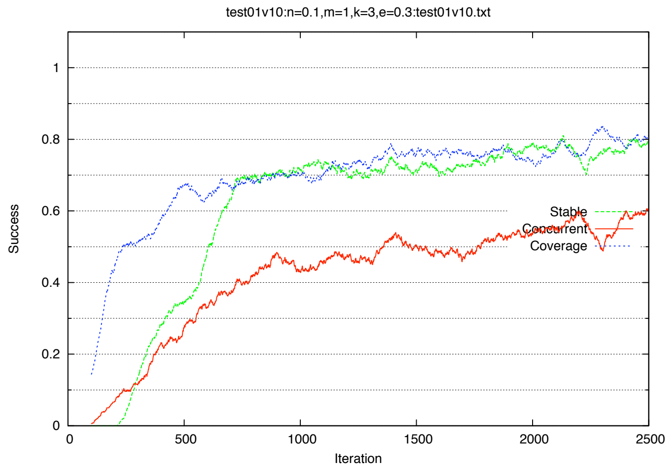

testDummyvars1:n=0.1,m=1,k=3,e=0.3:pass-atdepth3-withseq1-invars01of01-withp001.txt
Success
Iteration
Stable
Concurrent
Coverage
0
0.2
0.4
0.6
0.8
1
0
500
1000
1500
2000
2500
3000

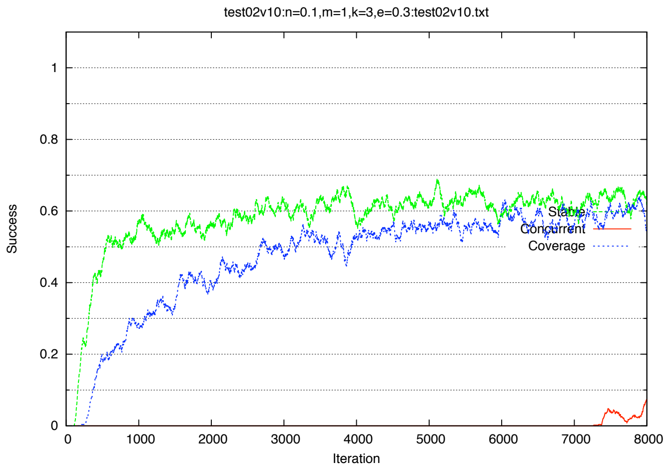

testDummyvars2:n=0.1,m=1+20,k=3,e=0.3:pass-atdepth3-withseq1-invars20of20-withp001.txt
Success
Iteration
Stable
Concurrent
Coverage
0
0.2
0.4
0.6
0.8
1
0
500
1000
1500
2000
2500
3000

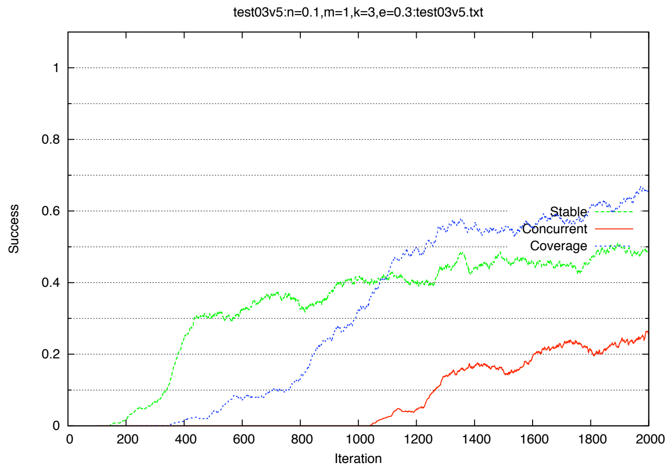

testEcai:n=0.1,m=1+6,k=3,e=0.3:ecai.txt
Success
Iteration
Stable
Concurrent
Coverage
0
0.2
0.4
0.6
0.8
1
0
500
1000
1500
2000
2500
3000

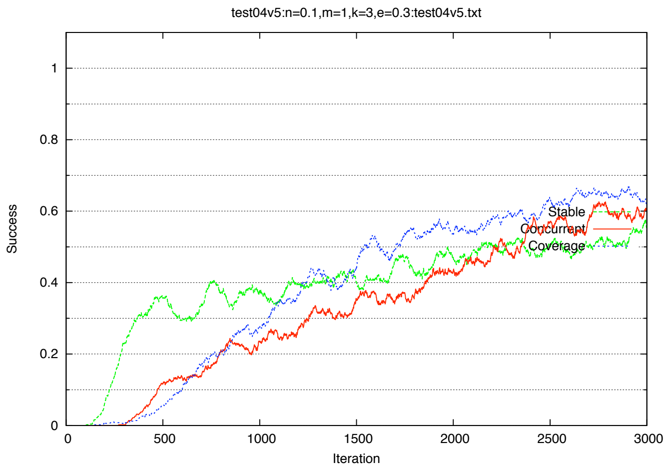

testImpactvars1:n=0.1,m=1+3,k=3,e=0.3:pass-atdepth3-withseq3-inworlds8of8-withp1by216.txt
Success
Iteration
Stable
Concurrent
Coverage
0
0.2
0.4
0.6
0.8
1
0
500
1000
1500
2000
2500
3000
3500
4000

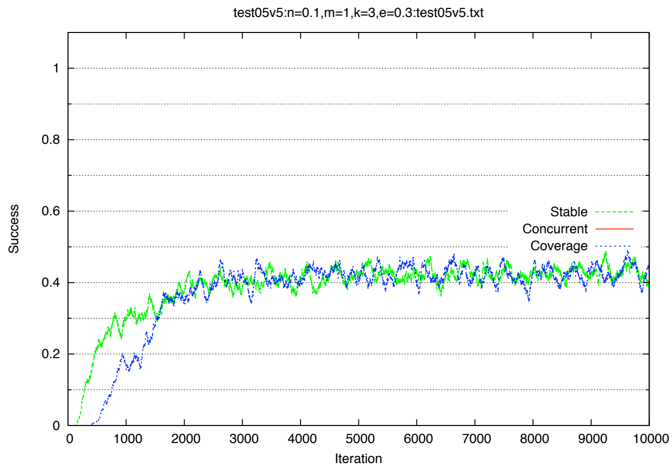
Repository Revision 77
(Section 6 changes: Coverage II)

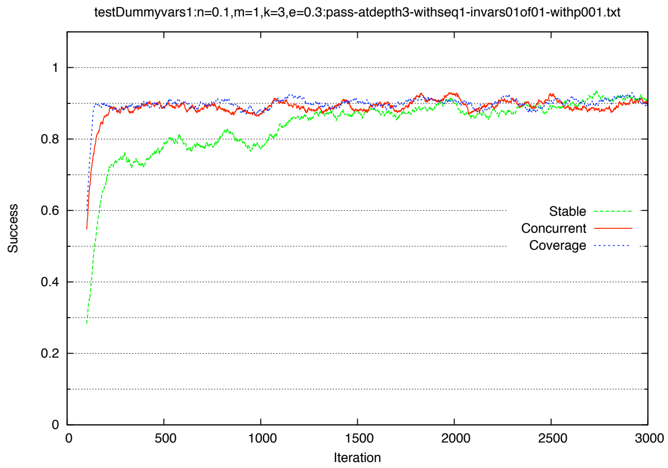


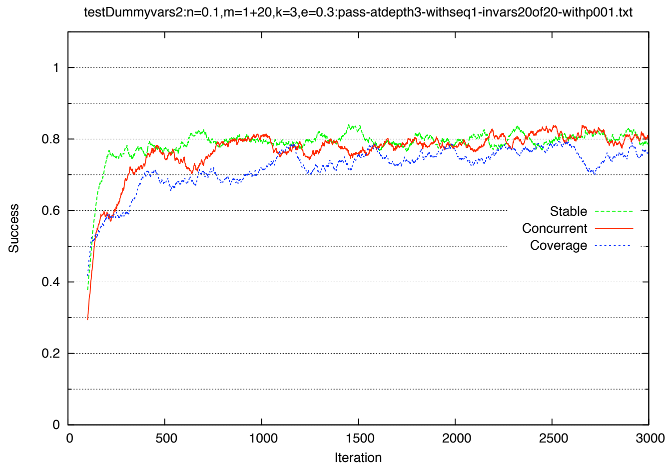


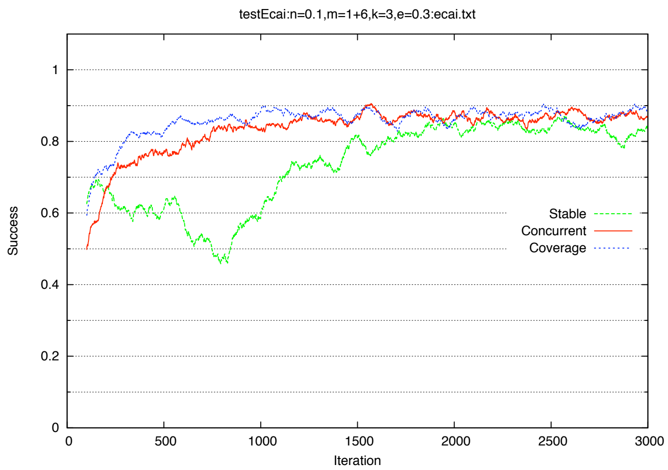


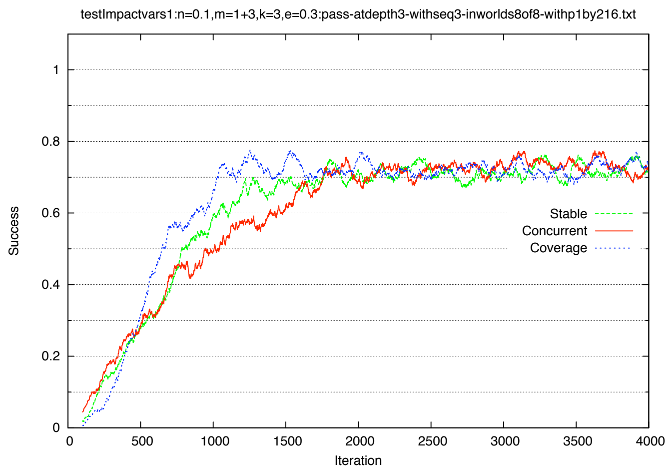





testDummyvars1:n=0.1,m=1,k=3,e=0.3:pass-atdepth3-withseq1-invars01of01-withp001.txt
Success
Iteration
Stable
Concurrent
Coverage
0
0.2
0.4
0.6
0.8
1
0
500
1000
1500
2000
2500
3000


testDummyvars2:n=0.1,m=1+20,k=3,e=0.3:pass-atdepth3-withseq1-invars20of20-withp001.txt
Success
Iteration
Stable
Concurrent
Coverage
0
0.2
0.4
0.6
0.8
1
0
500
1000
1500
2000
2500
3000


testEcai:n=0.1,m=1+6,k=3,e=0.3:ecai.txt
Success
Iteration
Stable
Concurrent
Coverage
0
0.2
0.4
0.6
0.8
1
0
500
1000
1500
2000
2500
3000


testImpactvars1:n=0.1,m=1+3,k=3,e=0.3:pass-atdepth3-withseq3-inworlds8of8-withp1by216.txt
Success
Iteration
Stable
Concurrent
Coverage
0
0.2
0.4
0.6
0.8
1
0
500
1000
1500
2000
2500
3000
3500
4000