The Optimistic Principle and Optimistic Pruning: A Preliminary Report *

Eugene C. Freuder and Yuanlin Zhang**

Cork Constraint Computation Centre,
University College Cork, Ireland
{e.freuder, y.zhang}@4c.ucc.ie

Abstract. In contrast to the deterministic and sound inference during search, we propose to make inference in an optimistic way, to speed up the solving of problems. Specifically, we examine optimistic pruning where a value is excluded from consideration when it is close to arc inconsistent. Our preliminary empirical study shows that under a proper level of optimism, optimistic pruning improves the performance of a MAC algorithm on the hard random problem instances close to (from the satisfiable side of) the phase transition point.

1 introduction

Much human and computer search simplifies the search space by making assumptions, which can later be incrementally or totally retracted if they eliminate all satisfactory solutions. Basic backtrack search, which underlies much of CSP solving, can be viewed in these terms. However, there the expectation is that failure is likely and backtracking inevitable. Here we propose an important 'psychological shift'. Rather than take the natural, conservative attitude 'we can't do that, it could lose solutions', we propose a more 'optimistic' approach: 'let's try that, it might work'. Once we have done this basic bit of 'lateral thinking' a whole new world of possibilities opens up to us.

More specifically, we can consider situations in which conditions C allow us to conclude a useful property P, and ask: Suppose we can come 'close' to establishing C, might it prove profitable to assume P anyway? More specifically still, we can consider properties like arc-inconsistency that allows us to prune values from variable domains and ask: Suppose a value is 'close' to being arc inconsistent, might it be profitable to assume it is? At worst, we might prune values that leave us without a solution, or without the best solution if we are optimizing; but we can always 'go back' and undo our assumption if need be. The question, as with all heuristic methods, is whether the potential gains outweigh the potential losses.

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

** Current address: Department of Computer Science, Texas Tech University, Lubbock, Texas.

It is actually rather curious that a field that emphasizes backtracking in search more than any other has not explored more backtracking in inference. Of course, choice and backtrack is forced on us for search. Inference is attractive in that it can be deterministic. However, embracing the risk involved in voluntary choice, backstopped if necessary by backtracking, may prove powerful as well. Stochastic approaches, such as local search and random restart, have demonstrated that taking risks can be fruitful.

2 Optimistic pruning

Many methods have been developed to prune the search space. We restrict ourselves to a specific and well studied pruning process: maintaining arc consistency during search (MAC) [3]. In an optimistic MAC, a value is excluded from further consideration if it is *close* to be arc inconsistent with respect to a constraint. The optimism here is that if a value is close to be arc inconsistent, it stands a very small chance to be supported in the future. How do we measure the closeness of arc consistency?

Value ordering heuristics are frequently used to improve the efficiency of solving a CSP problem. When instantiating a variable, a more promising value is tried first. In other words, the less promising values are not likely to be a correct assignment for the current variable. In optimistic MAC, the closeness to arc inconsistency can be measured by any value ordering heuristic, and the less promising a value is, the closer it is to arc inconsistency. When instantiating a variable, it is cheap to calculate an ordering on its values, but it could be expensive to maintain a value ordering on *all* variables during each invocation of arc consistency. Here, we propose two ways to determine whether a value is close to be arc inconsistent.

The first is to check the number of supports of a value with respect to each constraint on it. When this number falls below a threshold, the value will be removed from further consideration. The second depends on the proportion of supports that a value has lost during a search procedure. If the proportion drops below a certain percentage, the value will be deleted optimistically because it loses supports "faster" than the other values.

Example Consider three variables x, y and z that can take values from a domain of $\{-5, -2, -1, 1, 2, 5\}$. The constraint between x and y is $|x| = |y|$ when $x = \pm 5$ and otherwise is almost "$|x| = |y| + 1$ or $|y| = |x| + 1$" (except for -2 of x and -1 of y) as shown in Fig. 1. The other constraints are $|x| = |z|$ and $|y| = |z|$. Since -2 of x has only one support in y, it can be removed optimistically, resulting in the removal of $\{-1, 1, -2, 2\}$ from the domains of x, y and z. Now, it is easy to find a solution for the problem.

In our experiments, the optimism defined above leads to too many values to be deleted, resulting in no solution for many originally satisfiable problem instances. As more variables are instantiated, the condition of optimism is easier to be satisfied in the later stage of the search. To curb this tendency, one method

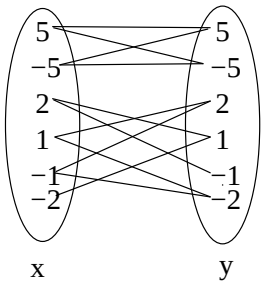

Fig. 1. A constraint between variables x and y

is to turn off the optimism at certain depth (i.e., the number of instantiated variables) of search.

2.1 Optimistic pruning algorithm

In this section, we present an optimistic MAC algorithm that removes a value optimistically in terms of the number of its supports. In the algorithm we need to maintain the number of supports of a value, implying that the techniques developed in AC-4 [2] are a good choice for this optimistic algorithm.

For each value with respect to a constraint incident on it, AC-4 maintains not only the number of, but also a list of, all supports for the value. At the initialization stage of AC-4, all the values that have zero support will be put into a queue so that later we are able to propagate the removal of them to all their supports. Specifically, when a value is taken from the queue, for each of its supports a, decrease the number of supports of a by one. The optimism comes in now. When a's number of supports falls below a threshold, a will be put into the queue, waiting to be deleted. At the same time, we can also check whether we are beyond certain depth of search and if so turn off the optimism. The propagation continues until no value is left in the queue.

The kernel of a preprocessing AC algorithm or MAC [3] is the propagation algorithm. Given a queue of removed values, the propagation algorithm of the optimistic MAC, listed in Fig. 2, popagates the values optimistically. The algorithm needs the following data structures. For any value a of a variable i and a constraint between i and j, `counter(i,a,j)` and `supports(i,a,j)` are the number, and the list respectively, of all supports of a with respect to the constraint between i and j. A list of values to be removed, each of which is denoted by `(i,b)` (a value b of variable i), are put in the queue `Q`.

The algorithm opti-propogate(`Q`) in Fig. 2 propogates the deletion of the values in `Q` and removes an affected value optimistically in terms of the optimistic condition implemented by the procdure removable-optimistically(`j, a, i`).

Line 1 in Fig. 3 ignores the optimism when the variable i is instantiated or has only one value left in its domain. In this case, any value of j has only one

```
algorithm opti-propogate(Queue Q)
    begin
        while Q not empty do
            select and delete a value (i,b) from Q;
            for each neighboring variable j of i do
                for each value a in supports(i,b,j)
                    if removable-optimistically( j, a, i)
                        delete (i,a);
                        Q ← Q ∪ { (i,a) };
                    endif
                endfor
            endfor
        endwhile
    end
```

Fig. 2. The AC-4 propagation algorithm

```
procedure removable-optimistically( j, a, i)
    begin
        counter(j,a,i) = counter(j,a,i)-1;
        if counter(j,a,i) is zero
            return true;
1.      if domain of i has only one value left
            return false;
2.      if counter(j,a,i) is smaller than a threshold
            and the current search depth is shallower than a threshold
                return true;
        else return false
    end
```

Fig. 3. procedure to check whether a value is optimistically removable

support in i and the optimism is turned off because otherwise all values of j will be excluded optimistically, resulting in a search failure. Line 2 is to apply the optimistic removal criteria to the value a of variable j.

3 Experimental results

Experiments are designed to examine the effectiveness of the optimistic MAC and to characterise the problems on which optimistic pruning is effective. Uni-

formly randomized binary constraint satisfaction problems (based on model B) serve these purposes well. To specify a set of problem instances, we need the parameters of the number of variables n, the maximum domain size d of the variables, the number of constraints e, and the tightness of the constraints t (t is the number of disallowed tuples). A tuple $< n, d, e, t >$ is used to distinguish different classes of problem instances.

In our experiments, $< n, d, e, t >$ is set in terms of the following rules. n, d are chosen arbitrarily and independently. To locate the hardest problems, the number of constraints is set to be about 93% of $n(n-1)/2$, the number of all possible constraints. Once n, d, e are fixed, we locate the t at the peak of the phase transition area. From the instances of $< n, d, e, t >$ we choose only those instances that are satisfiable. The reason to do so is to locate the problems where optimistic pruning is promising. For unsatisfiable instances, due to the incompleteness of optimistic pruning, finally we have to resort to a complete search algorithm to prove the unsatisfiability and thus optimistic pruning could not improve the performance of the hosting search algorithm.

After $< n, d, e, t >$ is fixed, we vary e, the number of constraints, to generate a sequence of classes of instances. In this way, we have hard problems in a relatively large range of varying e's, in contrast to generating instances by varying t with n, d, e fixed.

For example, after setting n and d to be 30 and 10 respectively, e should be 405 (about 93% of all constraints). Through experiments on various t's, we find the phase transition point where $t = 15$. With n, d, t being 30, 10 and 15 respectively, experimental data shown in Fig. 4 (The diagram is colorful and best viewed on a computer) are collected by setting e to be various values less than 405.

The optimistic pruning algorithms are parameterized by the minimum number of supports a value should have with respect to any incident constraints, and the *threshold depth* that is the search depth where the optimism is turned off. In all the experiments reported here, we set the minimum support to be 2, and vary the threshold depth from 2 to 4. In Fig. 4, optiMAC$< 1, 2 >$ means an algorithm that removes a value if it has at most 1 support and employs a threshold depth of 2. From this figure, it is observed that the optimistic algorithm with threshold depth of 4 performs better than the non-optimistic MAC, especially on harder problems close to the phase transition point.

To see whether this observation is applicable to other classes of random problems, we explored the problems whose number of variables and domain size are around 30 and 10 respectively. Fig. 5 shows the results on $n = 27$ and d from 9 to 13. The x axis is the sequence number of different settings and y axis is the average number of constraint checks used for solving the instances in each setting. The diagram contains five components. The leftmost is for $d = 9$, the second for $d = 10$, and so on. The setting for each component varies only on the numbers of constraints. For example, the left most component is obtained by varying the number of constraints from 309 to 339 when n, d, t are 27, 8 and 12 respectively. It can be regarded as the miniaturized version of a diagram like

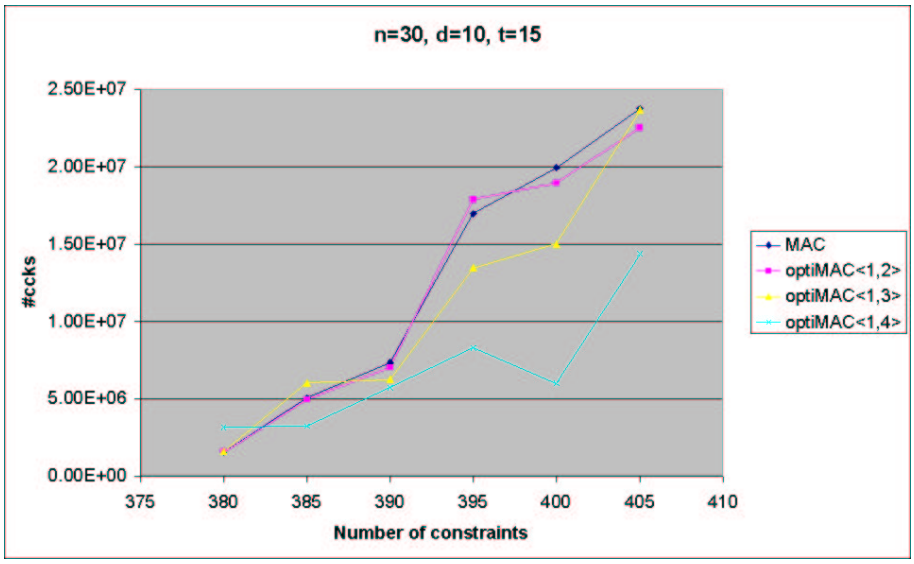

Fig. 4. Experimental results on problem instances

Fig. 4. Results on other settings are shown in Fig. 6-8 (they are colorful picures and best viewed on a computer) that use the same legends given in Fig. 4.

From these results, we can see that under a proper threshold depth, the search algorithm with optimistic pruning performs better in most cases than non-optimistic algorithm on the problems close to the peak of the phase transition area.

There are a few remarks on the data shown in the diagrams. First, the number of settings in each component varies. This is due to the fact that we eliminate settings where there exist unsatisfiable instances. This could be remedied in the future experiments by selecting a slightly smaller tightness for those settings generating unsatisfiable instances. Second, each component is supposed to be increasing when the number of constraints increases. The reason this is not true for our data might be that we did not use a sufficient number of instances for each setting $< n, d, e, t >$. (Due to the large number of settings, we test only 5 instances for each setting. We believe larger number of instances could improve the situation.)

4 Discussion

Iterative broadening reported in [1] is a search scheme under which for each variable, there is only a fixed number of values will be tried and backtracking occurs if these values fail to be consistently extensible to a solution. In this scheme, no values in the domains of *future* variables will be excluded from consideration (if

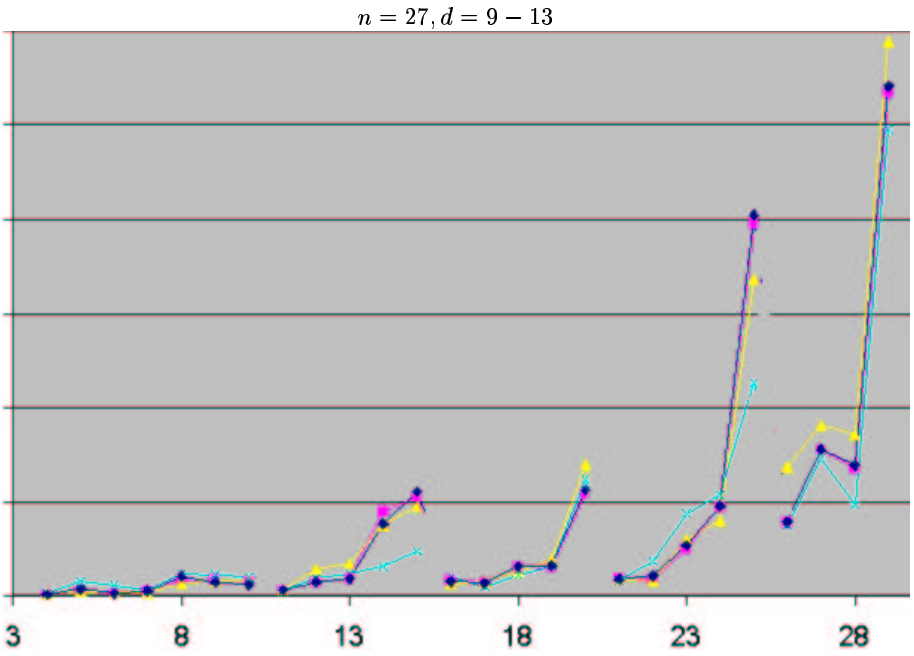

Fig. 5. Experimental results on more problem instances

they have a support with respect to every incident constraint). In the case of optimistic pruning, some values of future variables will be deleted optimistically in terms of the search depth and the number of its supports. This leads to more pruning than the iterative broadening scheme.

Some readers might have realized that the optimism MAC, especially through our experiments, is very coarse grained. For example, the optimism is turned off at a very shallow depth of 4. We had tried to increase the threshold depth of the optimistic pruning but obtained answers of unsatisfiability for some originally satisfiable instances. It seems necessary to develop more fine-grained optimism to achieve better performance. For example, when considering removing a value optimistically, we could use the information on its supports with respect to all incident constraints, rather than one constraint. In our current algorithms and implementation, the same optimism scheme is used during arc consistency after each instantiation of a variable. In this case, the optimism could be applied to the same domain repeatedly, possibly resulting in more values removed *optimistically*. To relieve this effect, we can either restrict the number of times we apply optimism to the domain of each variable in one execution of arc consistency algorithm or relate the invocation of the optimism to the size of the domain.

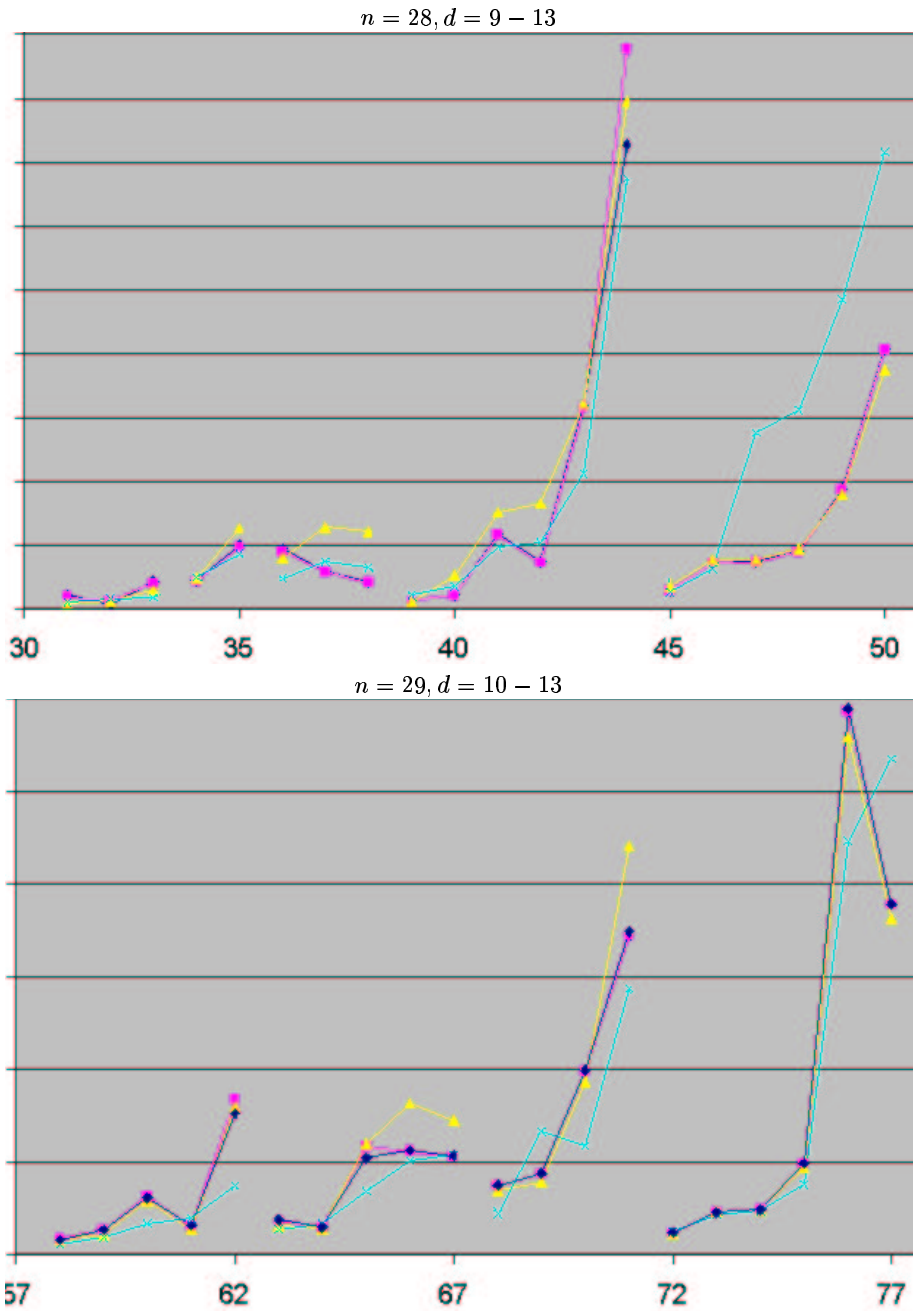

Fig. 6. Experimental results on more problem instances (continued)

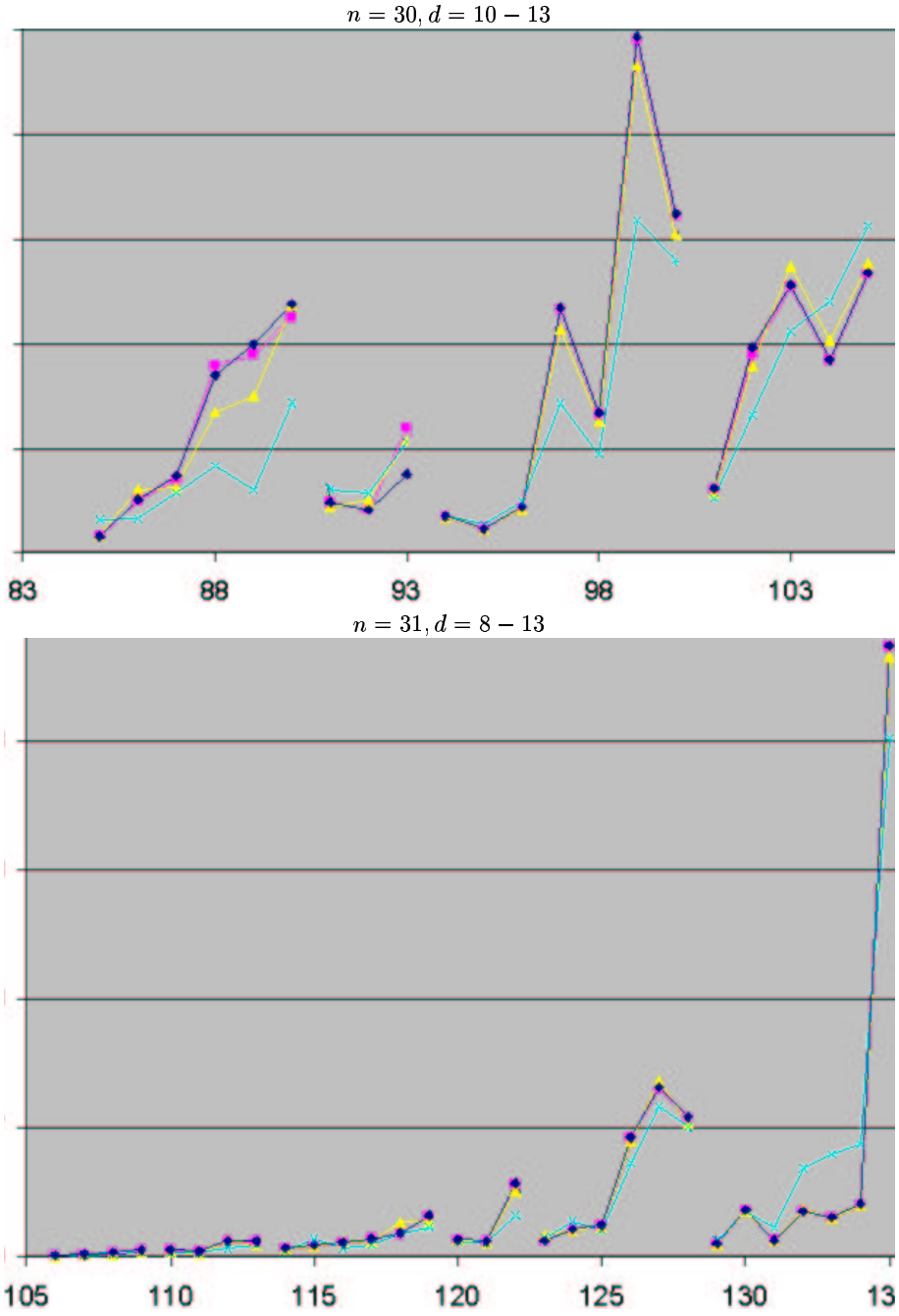

Fig. 7. Experimental results on more problem instances (continued)

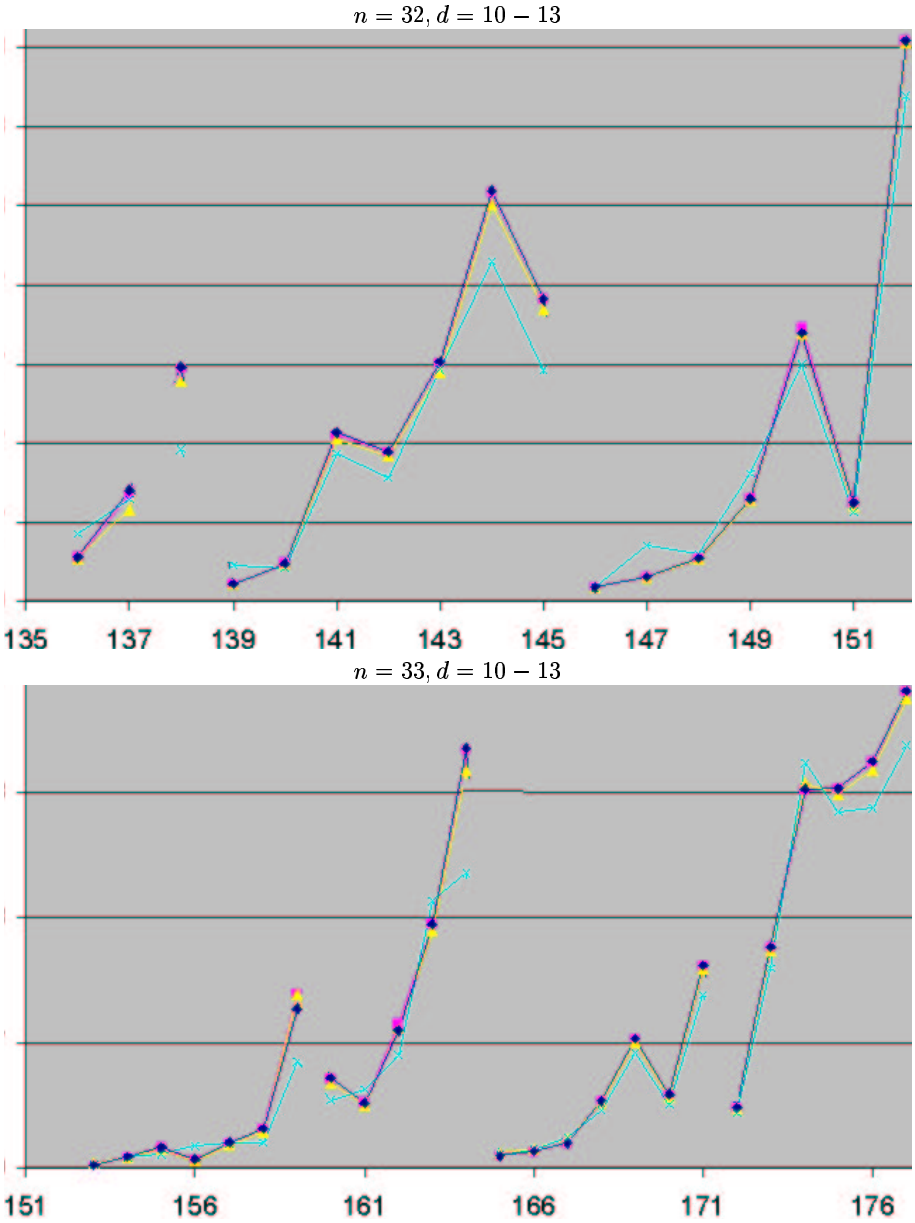

Fig. 8. Experimental results on more problem instances (continued)

5 Conclusion

We propose to make optimistic inference during search and examine a specific optimistic pruning – optimistic MAC. Preliminary experiments show that there exist problems for which optimistic pruning is very promising. With proper control of optimism (through number of supports and depth of search), this approach improves the performance of a search procedure on most problem instances close (from the satisfiable side) to the phase transition point. It is also observed that optimistic pruning makes some easy problems harder to solve. In the future we will explore the potential of optimistic pruning by designing more fine-tuned types of optimism and more importantly identifying the problems in specific application domains that can be efficiently solved by optimistic pruning.

6 Acknowledgement

We thank Barry O'Sullivan and Tudor Hulubei for many disscussions on the material presented here.

References

1. Matthew L. Ginsberg and William D. Harvey. Iterative broadening. *Artificial Intelligence*, 55(2):367–383, 1992.
2. R. Mohr and T.C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.
3. D. Sabin and E. Freuder. Contradicting conventional wisdosdom in constraint satisfaction. In *Proceedings of the Second Workshop on Principles and Practice of Constraint Programming*, pages 10–20, Rosario, Orcas Island, Washington, 1994.