A Comparative Study of Arc-Consistency Algorithms

Lin XU and Berthe Y. Choueiry
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln NE, 68588-0115
{lxu|choueiry}@cse.unl.edu

September 4, 2001

Abstract

Arc consistency plays an important role in Constraint Satisfaction Problems (CSPs). Several algorithms have been made to deal with arc consistency. AC-3, AC-4, AC-7 are very famous algorithms in this area. Our work is tried to figure out, which one is better to use. We judge it not only by constrain checks but also by the CPU time and so on. Based on AC algorithms, we implement Maintaining Arc consistency (MAC), and compare the performance of MAC-3, MAC-4 with FC.

1 Introduction

In this study, we implement various arc-consistency algorithms and compare their behavior for solving finite constraint satisfaction problems.

A finite CSP is defined as $\mathcal{P}$=($\mathcal{V}$, $\mathcal{D}$, $\mathcal{C}$); where $\mathcal{V}$=$\{V_1, V_2, \dots, V_n\}$ is a set of variables, $\mathcal{D}$=$\{D_{V_1}, D_{V_2}, \dots, D_{V_n}\}$ is a set of their corresponding domains (the domain of a variable is a set of possible values), and $\mathcal{C}$ a set of constraints that defines the acceptable combinations of values for variables. A solution to the CSP is to assign, to each variable, a value from its domain such as all constraints are satisfied. The question is to find one or all solutions. When all constraints are satisfied, the solution is said to be consistent (otherwise, it is inconsistent); when all variables are instantiated, the solution is said to be global (otherwise, it is partial). A binary CSP is often represented by a constraint graph in which the variables are represented by nodes, the domains by node labels, and the constraints between variables by edges linking the nodes in the scope of the corresponding constraint. We study CSPs with finite domains and binary constraints (*i.e.*, they apply to two variables).

We run two types of experiments:

1. Arc-consistency: In the first round of experiments, we compare the performance of AC-3, AC-4 and AC-7 by running them on a randomly generated CSP and making it arc-consistent. No search is conducted.

2. Search with arc-consistency: In the second round of experiments, we compare the performance of backtrack search enhanced with FC, MAC-3, MAC-4 to find the first solution to the CSP.

We report the following measures:

- Number of constraint checks
- Number of nodes visited

- CPU time
- CPU time per constraint check

We represent a constraint by its extensive definition as a list of acceptable tuples. So, a constraint check is in $O(d^2)$, more precisely $O((1-t)d^2)$, where d is the max domain size and t is the tightness of the constraint.

First, we introduce how those algorithm works.

1.1 AC-3

The algorithm of AC-3 [Mackworth, 1977] (related with AC-1 and AC-2) is based on a very simple ideal. It considers every constraint, for each constraint (I, J), we need to find if every value (such as a) in the domain of I has at least one value in the domain of J consistent with a. If there is no value consistent with a, we delete a from the domain of I and reconsider all constraints that are connected to I since the domain of I has be changed. We repeat this process until no constraint needs to be reconsidered. This algorithm is simple, so it works very fast. But since it doesn't record any message during the AC process, it may very inefficiency $O(n^2a^2)$ sometimes.

1.2 AC-4

AC-4 [Mohr and Henderson, 1986] is quite different from its ancestor. Before doing arc-consistency, AC-4 analyses all the constraints, and builds a number of data structures to store those messages. First it builds a valid bit for every value we call it M, such as $M[I, a]$=T, $M[J, b]$=nil means the value a in domain of I is valid, but value b in domain J is invalid or this value already be deleted. Then for each value, AC-4 builds a list that includes all other values which are consistent with this value, we call it S, such as $S[I, a]$ may include (J, c) (J, e) (K, a). At last, it uses a counter, called C, $C[(I, a), J]$ counts how many values in the domain of J are consistent with value a in domain of I. All those are very helpful, we will show you in the next section. But nothing is free, AC-4 requires more space than AC-3 because we need save those data structure, for example, the space for M is $O(nd)$.

When all this done, AC-4 operates based on those data structures and doesn't need the original problem anymore. What we need to do is check each counter, if it is 0, which means there is no value in some domain consistent with this value. So, this value needs to be deleted from the domain, we set the valid bit to nil and decrease the counter of all the values inside S by one.

AC-4 normally performs worse than AC-3. It needs check more constraints when the tightness is low , and the algorithm is complexer than AC-3, which means to check the same number of constraints, AC-4 needs more time. But it still has some thing good, it build a data structure which make the further work become simple and given some useful information about the value.

1.3 AC-7

AC-7 [Bessière, 1994] consider in more detail during arc consistency, it records lots of information to reduce the constraint checks. Compare with AC-4, AC-7 has valid bit M (the same as AC-4), smallest supporter set S instead of all supporter set in AC-4, and a pointer INF. Smallest supporter set $S[(I, J), a]$ has all the values in domain of J, which regard a as their smallest supporter, for example, if we have two variable I, J, the domain of I is $\{dog, pig, cat\}$, and the domain of J is $\{I, You, He\}$ (exactly in this order), and if I like pig and cat, you like dog and pig, he like dog and cat, the $S[(I, J)dog]$ include every member in domain of J who like dog most ((J, you) (J, he)), then $S[(I, J)pig]$ is ((J, I)), $S[(I, J)cat]$ is nil. $INF[(I, J)dog]$ inform which is the first value in domain of J consistence with (I, dog), so it is

2 (the second value in the domain J is you). With all this information, AC-7 has the smallest amount of constraint checks, but AC-7 is the most complex AC algorithm too.

How does this algorithm work? We put every value into a list waiting for find a support for every related constraint. If there are some valid values already in $S[(I,J)a]$, which means (i,a) still has some support in the domain of J, so (I,a) can't be deleted. Otherwise, we need find a new support for it. So we check the domain of J one by one begin with $INF[(I,J)a]$ (before that we already checked). During this process, we need check $INF[(I,J)b]$, if the INF of b is bigger than the order number of a, that means the smallest number of value in domain I consistent with b is bigger than a, so a can't be consistent with b, this avoids some useless checks. When we find a new valid value (J,e) consistent with a, then we put a in $S[(J,I)e]$ and set the $INF[(I,J)a]$ to the number of e. If we can't find one, we need set the valid bit to nil, and delete it from the domain and put everything in $S[(I,J)a]$ to the list waiting for find a new supporter. Repeat this until this list become empty.

AC-7 keeps trace almost everything, in most case it needs much less constraint checks than the other. But since the algorithm is much more complex than AC-3 and AC-4, it needs more CPU time.

1.4 MAC

MAC $MaintainingArcConsistency$ [Sabin and Freuder, 1994], is an important kind of search algorithm. MAC uses the same basic framework as FC, alternating search and consistency inference steps, but differs conceptually in two aspects. First, the constraint network is made arc consistent initially. Second, when during the search a new variable I is instantiated to value a, all the other values in the domain are eliminated and the effects of removing them are propagated through the constraint network as necessary to restore full arc consistency.

Base on AC algorithms, we can implement different MAC (MAC-3, MAC-4). The basic ideal of MAC is the same, the only different between those MAC is they based on different AC algorithm. We find some interesting results about those MAC, we will talk about it later.

2 Experimental result

To compare those algorithms, we use the random CSP problem generator to generate sets of random CSP problem. Each problem has n variables, the domain size is uniform and equal d, density D is taken from 0.3 to 0.9 by step of 0.1 and tightness t is in the interval [0.2, …, 0.9] by step of 0.1. We generate a few problems per point and compute the average of the values obtained.

2.1 AC algorithms

Fig. 1 shows the relation of tightness and constraint checks for different AC algorithms (D=0.6, others almost the same). From this figure, we can find:

1. When tightness is small, AC-3 only needs few constraint checks. With t increasing, the number of constraint checks also increased. But when tightness become 0.6 0.7, the number of constraint checks increased very quickly, after that, the number of checks drops down.

2. When tightness is small, AC-4's constraint checks much more than AC-3 or AC-7. But when the tightness more than 0.6, AC-4 has less constraint checks than AC-3, when the tightness more than 0.8, even less than AC-7.

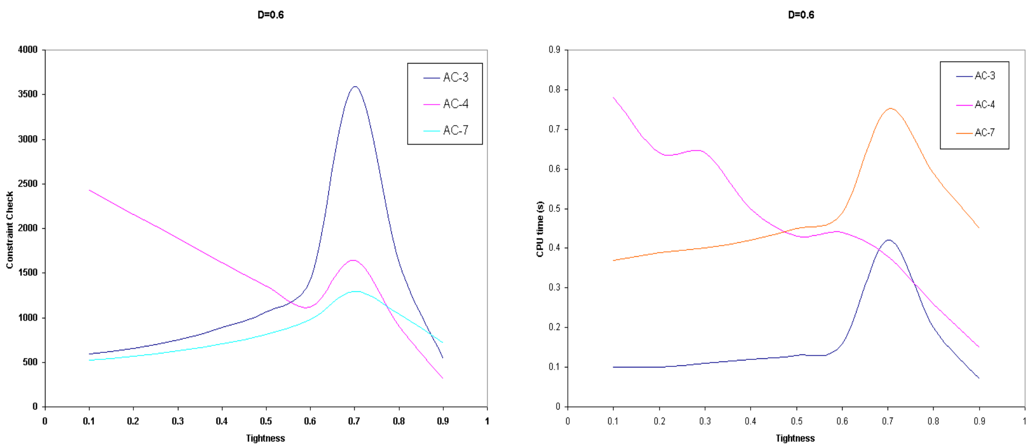

Figure 1: *The relation of t and constraint checks or CPU time for different AC. (D=0.6, n=10, d=10, 8 problems per point and computer the average values).*

3. AC-7 takes all the advantages from AC-3 and AC-4. When t is small, the constraint checks is small, and when t is high, the number of check increase slowly.So it normally has the smallest constraint checks.

4. Even AC-7 has the smallest number of checks, the CPU time is the highest in most cases.

When the tightness is low, which means there are lots of tuples in a binary constraint. Normally, we don't need eliminate any value during this case. AC-3 and AC-7 only need to find one consistent value for every constraint value. It is very easy to find when there are lots of tuples between values, so the constraint checks are quite low. But for AC-4, there are some prepare work need to check every constraints, and since the tightness is low, so the total number of constraints is big. This is the reason why AC-4 need much more constraint checks when the tightness is low.

When t become bigger, for AC-3 and AC-7, they still need to find a consistent value for every value, but since the tightness become higher, they may need to delete some value from the domain, so they need check more constraints. t is bigger, also means there are less consistent value pairs, so for AC-4, it has less constraint checks during prepare section than before. Of course, it needs delete some value, but for AC-4, only set the valid bit to nil and get everything in S, set those value's counter less one. This addition check is small, so we can find when t become bigger, the constraint checks become smaller.

t become 0.6   0.8, when we eliminated some value from the domain, may cause another value be deleted and so on. AC-3 doesn't record anything during the AC process, so each time, it needs to check every related variable. From the figure, we can see, the number of constraint checks increase very quick. For AC-4 and AC-7, they have some data structure record which value consistent with this value. when some value be deleted, they know what the exact effect to the problem. They only need to consider those related value, instead of all the value in the domain. So, in that case, AC-4 and AC-7 check much less constraints than AC-3.

When t become very high, such as 0.9, all three algorithms check very small amount of tuples. The reason is in this case, there only a few connect between two variables. The problem is likely to be not consistent, when we find it, we stop.

In order to reduce constraint checks, AC-7 records lots of information and has the most complex computation. Even if AC-7 has the smallest number of constraint checks, it needs much more CPU time than others. AC-3 still is the best choice in most case.

2.2 MAC algorithms

MAC is the sum of a set of Arc consistency problems. Fig. 2 Fig. 3 and Fig. 4 show the effect of D, t to MAC and FC. (MAC-7's figure looks like MAC-3, but more CPU time and less constraint checks. We compare MAC-3, MAC-4 and FC)

1. Like AC-7,MAC-7 takes the advantage of MAC-3 and MAC-4. It almost always has the smallest amount of constraint checks. But consider of average CPU time per constraint check. It is about 8-10 times than MAC-3 and 4-6 times than MAC-4 (we did not show it). So, unless we can make a big improvement, MAC-7 is not a good way to deal with search problem.

2. When tightness is low, MAC-4's constraint checks more than MAC-3. But when the tightness more than 0.5, MAC-4 has less constraint checks than MAC-3.

3. We notice that the effect of D is very big to the sharp of curves and the position of the peak for MAC. D is high, MAC need more constraint checks, but this is not shown by FC.

4. Compare to FC, we find MAC needs to check much less node, but needs to check more constraints.

5. The average CPU time per constraint checks does not change much for different D and t for MAC-3.

Our MAC based on the same principal, so this difference is only caused by the different AC algorithms.

MAC-3 based on AC-3, it doesn't record anything during the MAC process. Each time we use backtracking, AC-3 begin a new AC problem, it can't find the relationship between those Arc consistency problem. To solve CSP problem, the cost will be the sum of each Arc consistency problem.

MAC-4 is different, it has a data structure to record the information. When AC-4 build the S M Counter, it doesn't need the CSP problem anymore, every work is done by modify this data. When we need backtracking, we still can use those data and don't need make a new data structure. We only need take care of the value insider S. SO when there are more backtracking and smaller size of S , the advantage of MAC-4 is more obviously. From Fig. 2, we can find that MAC-4 better than MAC-3 when t is between 0.4 and 0.7. In this range, the backtracking is higher than others.

The different between FC and MAC is very big. Since after each step of search, MAC check all the rest variables (this means more constraint checks), and eliminate as much values as possible, So it always go the better way than FC and has less backtracking (this means less nodes and less constraint checks). The problem is which one is more important?

This can explain the figure of average CPU time per constraint checks too. We know, when we meet backtracking, there are some extra work such as backup some information, reset the domain and so on. For MAC, since they almost go directly to solution (less backtracking), so the average time almost the same for MAC-3. But for FC, when there are lots of backtracking, the average CPU time become higher.

We also notice that there are some relation between D and position of the peak of our figure. It seems like when t=0.9-D/2, the CSP problem is the most complex one for MAC algorithms.

The other thing needs to be noticed is the space consume. When time to backtracking, we need backup the information. For MAC-3, it only has to backup the variables and the domain, which size is quite small. But for MAC-4 and MAC-7, since they have lots of information need to backup, so need much more space than MAC-3, this also a reason for high average CPU time per constraint checks.

3 Conclusion

From the results above, we can conclude:

- AC-3 is very simple and need less CPU time. Normally AC-3 and MAC based on AC-3 is the best choice to solve CSP problem.
- In some case, AC-4 can better than AC-3, when tightness is high (0.6   0.8).
- When density is low, the different between MAC-3 and MAC-4 is big, when density is big, the different will become small. normally, MAC-4 check less constraints than MAC-3, but consider the total CPU time, MAC-3 still is a good choice in most case.
- AC-7 (MAC-7) take all advantage from AC-3 and AC-4, when density is low, It looks like AC-3, when density is during $0.6 \cdots 0.8$, it looks like AC-4. It almost always check less constraints than AC-3 and AC-4. But the disadvantage is it need much more CPU time per constraints.
- AC-4 build some data before search, this made it bad when t is low, but it also give some good information, such as which value has the biggest chance to be deleted (if the counter =1), which value has the biggest number of consistence value, and so on. This may use in some kind of informed search.
- FC needs to visit much more nodes and than less constraint checks than MAC.

Appendix

There is a litter different between the algorithm of AC-4 and my code. In the algorithm, it check every possible combination of the value, so it needs check ed^2. This isn't necessary, we only need to check every available connection. In worse case, this two is the same, normally, later one will check much less constraints than the first one. Fig. 5 shows the different between this two way. It seems like the second one is better than the old one. Fig. 5 will show you the difference.

Acknowledgments

We are grateful to Rob Glaubius for his useful help. Special thanks to Amy for her good suggestions and help.

References

[Bessière, 1994] Christian Bessière. Arc-Consistency and Arc-Consistency Again. *Artificial Intelligence*, 65:179–190, 1994.

[Mackworth, 1977] Alan K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8:99–118, 1977.

[Mohr and Henderson, 1986] R. Mohr and T. C. Henderson. Arc and Path Consistency Revisited. *Artificial Intelligence*, 28:225–233, 1986.

[Sabin and Freuder, 1994] Daniel Sabin and Eugene C. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proc. of the 11 th ECAI*, pages 125–129, Amsterdam, The Netherlands, 1994.

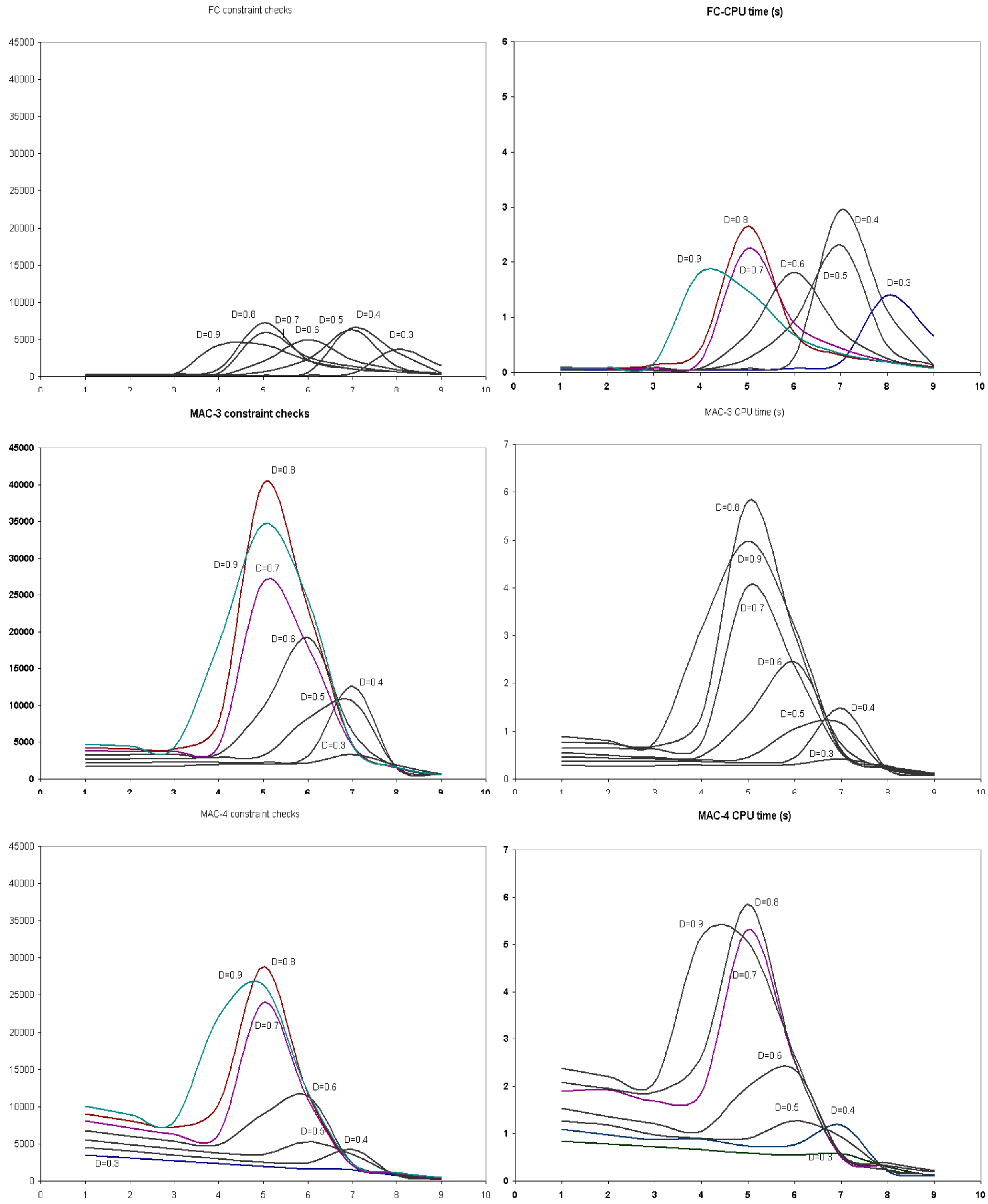

Figure 2: *The figure of constraint checks and CPU time for MAC3 MAC4 and FC (variables:10, domain size:10, 10 examples per point and get the average one).*

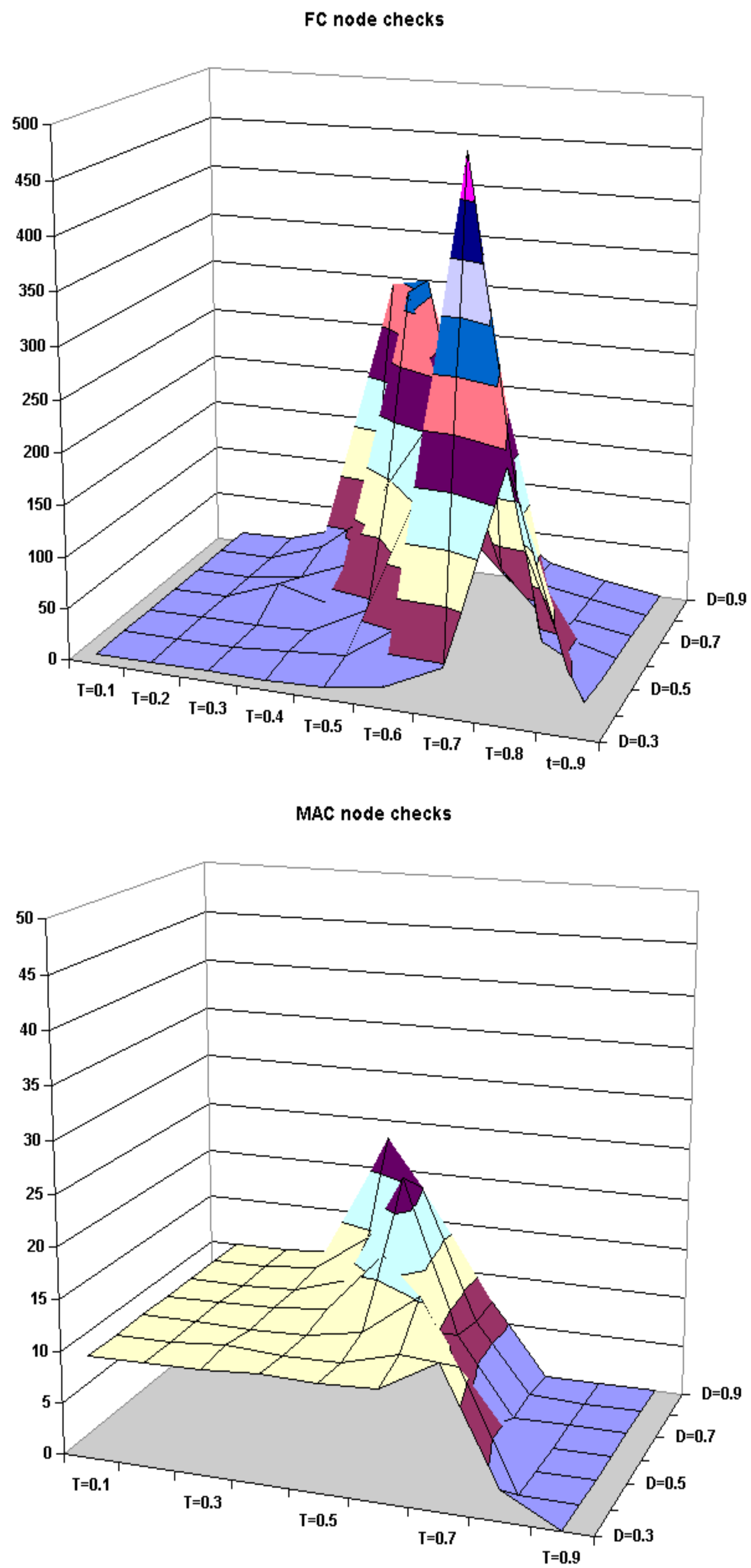

Figure 3: *The surface figure of nodes visted for MAC and FC (variables:10, domain size:10, 10 examples per point and get the average one).*

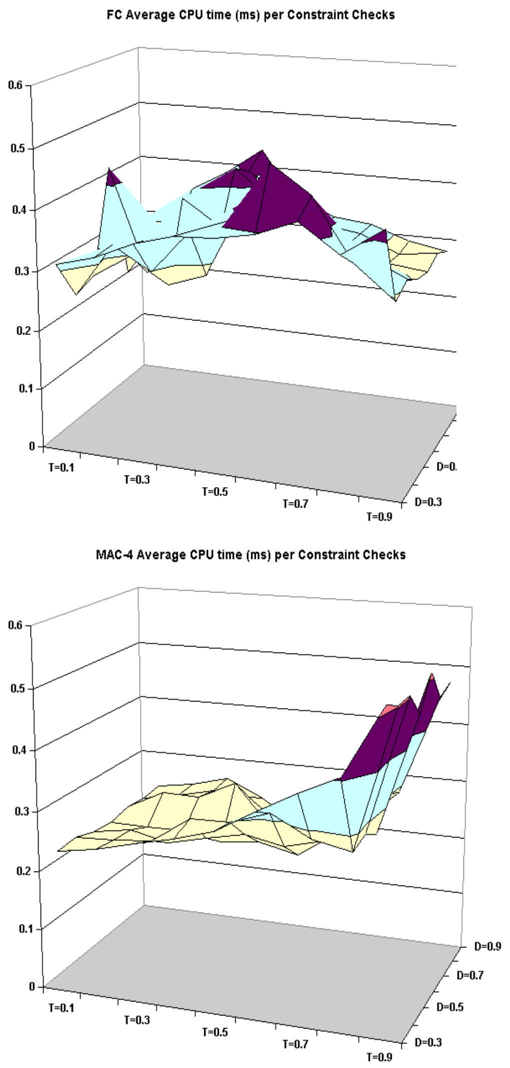

Figure 4: *The figure of average cpu time per constraint check for MAC3 MAC4 and FC (variables:10, domain size:10, 10 examples per point and get the average one).*

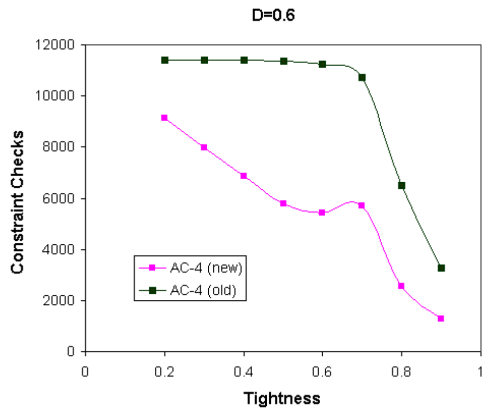

Figure 5: *The different between two AC-4 algorithms. (right: old, left: new) (n=20, d=10, 20 problems per point and compute the average of the values obtained*