An improved version of this paper appears as:
Tsang, E.P.K., No more 'Partial' and 'Full Looking Ahead', Research Note, Artificial Intelligence, Vol.98(1-2), 1998, 351-361

No more 'Partial' and 'Full Looking Ahead'

Edward Tsang

Technical Report CSM-276
Department of Computer Science, University of Essex, Colchester, Essex, CO4 3SQ, UK
Tel: +44 1206 872774
Fax: +44 1206 872788
Email: edward@essex.ac.uk
URL: http://cswww.essex.ac.uk/CSP/edward/

January 1997

Abstract

Looking ahead is a commonly used search technique in constraint satisfaction. In this paper, we examine the future role of two long established lookahead algorithms, Partial Looking Ahead (PLA) and Full Looking Ahead (FLA) [9]. We prove that PLA is inferior to Directional Arc-consistency Lookahead [3, 4, 14] in that the latter will prune at least as much as the former for no more compatibility checks in each problem reduction step. Similarly, FLA is inferior to Bi-directional Arc-consistency Lookahead, an algorithm introduced in this paper. We also point out a couple of errors in the literature.

Keywords: constraint satisfaction, looking ahead, constraint propagation, directional arc-consistency, arc-consistency

1. Introduction

A *finite constraint satisfaction problem* (CSP) is a problem which comprises a set of variables, a domain for each of the variables, and a set of constraints restricting the values that the variables can take simultaneously. The task is to assign a value to each variable from its domain satisfying all the constraints [8, 14].

We call the assignment of a value v to a variable x a *label*, denoted by <x, v>. One common way to search for a solution is to commit to one label for one variable at a time. We shall call the variables which are yet to be assigned a value *future variables*. A *redundant value* is a value in a domain which cannot be assigned to the corresponding variable to form any solution. *Lookahead* search algorithms attempt to remove redundant values through *problem reduction*. The most basic form of problem reduction is to remove all the values in the domains of future variables which are directly incompatible with the value that has just been committed to. We shall refer to this way of reducing the problem as *FC reduction*, since it is the reduction that the 'Forward Checking' algorithm [9] performs. In some lookahead algorithms, the removal of certain redundant values could lead to the removal of more redundant values or tightening of certain constraints. This is called *constraint propagation*.

Haralick & Elliott [9] introduced *Partial Looking Ahead* (which we shall refer to as PLA) and *Full Looking Ahead* (which we shall refer to as FLA) algorithms. Tsang [14] introduced the *Directional Arc-consistency Lookahead* (DAC-L) and *Arc-consistency Lookahead* (AC-Lookahead) algorithms, which reduce the remaining problem by maintain DAC and AC respectively. The objectives of this paper are:

(a) to prove that DAC-L is superior to PLA, in that the former is capable of removing more redundant values without requiring more computation than the latter;

(b) to introduce an algorithm called *Bi-directional Arc-consistency Lookahead* (BDAC-L), and show that it is superior to FLA;
(c) to point out that PLA and FLA are not the same as DAC-L and AC-Lookahead, respectively, as it was suggested by Tsang [14];
(d) to point out that maintaining DAC in both directions does not achieve AC, as it was suggested by Dechter & Pearl [3, 4].

2. PLA, FLA, DAC-L and AC-L Recapitulation

Definition 1 (Directional Arc-consistency [3, 4]):

A problem is *Directional Arc-consistent* (DAC) under an ordering of the variables if and only if for every label <x, a> which satisfies the constraints on x, there exists a compatible label <y, b> for every variable y which is after x according to the ordering.

As suggested in this definition, DAC assumes an ordering of the future variables. The standard procedure to maintain DAC is as follows [3, 4]. Let us assume that k variables have been labelled. Let the future variables and their ordering be (x_{k+1}, x_{k+2}, …, x_n). To achieve DAC, variable x_n is looked at first, followed by x_{n-1}, x_{n-2}, etc. until x_{k+1} is processed. When variable x_p is looked at, each of the values v in its current domain is checked against its future variables x_{p+1}, …, x_n, to see if v can be removed; v will be removed if it has no compatible value in any of its future variables. Let D_i denote the current domain of variable x_i, and C be a set of constraints. The following reduces a problem to directional arc-consistency:

```
1   Procedure DAC((x_{k+1}, x_{k+2}, …, x_n), (D_{k+1}, D_{k+2}, …, D_n), C)
2       BEGIN
3           For p = n to k + 1 do
4               For each value v in D_p do
5                   For q = p + 1 to n do
6                       IF <x_p, v> has no compatible value in D_q
7                       THEN remove v from D_p
8       END
```

After committing to each label, the DAC-L algorithm performs FC reduction and then maintains DAC. PLA does the same, except that instead of maintaining DAC, it calls a procedure which we shall refer to as *PLA reduction*. The PLA reduction procedure is identical to DAC, except that the variables are looked at in a different ordering: (x_{k+1}, x_{k+2}, …, x_n). In other words, line 3 of procedure DAC becomes:

```
3'          For p = k + 1 to n do
```

FLA behaves exactly like PLA, except that PLA reduction is replaced by a procedure that we shall refer to as *FLA reduction*. FLA reduction differs from PLA reduction in that it examines values against past as well as future variables. To be more precise, FLA reduction calls the following procedure after committing to each label and performing FC reduction (as before, x_{k+1}, x_{k+2}, …, x_n are the future variables):

```
1   Procedure FLA_Reduction((x_{k+1}, x_{k+2}, …, x_n), (D_{k+1}, D_{k+2}, …, D_n), C)
2       BEGIN
3           For p = k + 1 to n do
4               For each value v in D_p do
5                   For q = k + 1 to n except p do
6                       IF <x_p, v> has no compatible value in D_q
7                       THEN remove v from D_p
8       END
```

Definition 2 (Arc-consistency [10]):

A problem is *Arc-consistent* (AC) if and only if for every label <x, a> which satisfies the constraints on x, there exists a compatible label <y, b> for every other variable y.

If FLA_Reduction were to be used to maintain AC, it must be called repeatedly until no value is removed. Exactly how AC could be achieved has been extensively studied but unimportant to our discussions here. Interested readers should refer to, for example, Montanari [12], Mackworth [10], Mohr & Henderson [11], Deville & Van Hentenryck [5], Van Hentenryck *et. al*. [15], Bessière *et. al*. [1, 2], van Beek [16], Falting [6], Sabin & Freuder [13].

It is worth remarking on the names of the above algorithms before we continue: the name 'Partial Looking Ahead' is quite vague. Unlike DAC-L and AC-Lookahead, it does not suggest exactly what is being achieved during problem reduction. Besides, there is nothing 'full' (or complete) about 'Full Looking Ahead'. One can attempt to reduce the problem further than making FLA reduction (by, for example, maintaining 3-consistency [7]).

3. Bi-directional AC Lookahead

In this section, we shall introduce the Bi-directional AC Lookahead (BDAC-L) algorithm. Later we shall show that it can potentially prune more redundant values than FLA. BDAC-L simply maintains DAC from both directions after each label is committed to and FC reduction is completed. The pseudo code of BDAC-L is shown below:

```
1    Procedure BDAC((x1, x2, …, xn), (D1, D2, …, Dn), C)
2        BEGIN
3            DAC((x1, x2, …, xn), (D1, D2, …, Dn), C)
4            DAC((xn, xn-1, …, x1), (Dn, Dn-1, …, D1), C)
5        END
```

Constraint propagation is built upon the notion of *support*: a value is removed if it has no support (compatible value) from another variable. So the sooner a redundant value is removed, the less support other redundant values will get, which means the higher chance of them being pruned. So as a principle, the sooner redundant values are removed the more effective and efficient a problem reduction algorithm is likely to be.

In the pseudo code shown above, DAC is maintained from x_1 to x_n first. It is also possible to maintain DAC using the reverse ordering first. Given a particular CSP, if there exists heuristics which indicate which ordering for maintaining DAC should allow more redundant values to be removed early, such ordering should be used first.

4. Example

In this section, we shall give an example of a CSP, and explain the behaviour of the above lookahead algorithms. This example will be used in subsequent sections.

Figure 1 shows a graph colouring problem, which is a CSP with three variables A, B and C. The domain of variable A contains only one value "r" (which stands for red); the domains of B and C both contain values "r" and "g" (which stands for green). The constraint between A and C requires the value that A takes to be different from the value that C takes. Similarly, B is required to take a value different from C's. There exists no constraint between A and B, which means that there is no restriction on what values A and B may take simultaneously.

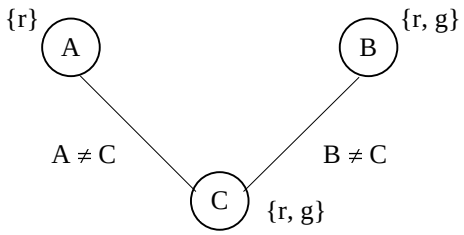

Figure 1 – A constraint satisfaction problem with variables A, B and C; their domains are indicated in { }'s and the constraints are marked next to the edges shown

Table 1: Redundant values removed by different reduction procedures

	Ordering	PLA Reduction	DAC	FLA Reduction	BDAC
1	(A, B, C)	Nil	Nil	<C,r>	<C,r>
2	(A, C, B)	Nil	Nil	<C,r>, <B,g>	<C,r>, <B,g>
3	(B, A, C)	Nil	Nil	<C,r>	<C,r>
4	(B, C, A)	<C,r>	<C,r>, <B,g>	<C,r>	<C,r>, <B,g>
5	(C, A, B)	<C,r>	<C,r>	<C,r>, <B,g>	<C,r>, <B,g>
6	(C, B, A)	<C,r>	<C,r>	<C,r>, <B,g>	<C,r>, <B,g>

Imagine A, B and C being three future variables during a search after a label is committed to and FC reduction is performed. Table 1 summarizes the redundant values that will be pruned by the above procedures. As an example, let us focus on row 4 of Table 1. Under the ordering (B, C, A), PLA will first check <B, r> against C and A. Since <C, g> is compatible with <B, r>, and no constraint exists between A and B, <B, r> gets support from both C and A. Next, <B, g> is checked against C and A, and support is found too. When <C, r> is checked against A, the value r will be removed from the domain of C because there exists no value in the domain of A which is compatible with <C, r>. Then PLA will check <C, g> against A; since it is supported by <A, r>, g will be retained in C's domain.

If DAC were to be maintained under the ordering (B, C, A), values in the domain of C will be checked against A first. <C, r> will be removed before <B, g> is checked against C, which would lead to the removal of g from the domain of B. Maintaining BDAC under this ordering will achieve the same effect due to line 3 of the above pseudo code (no further pruning will be made in line 4).

Under the same ordering, FLA will check the values in the domain of B against C and A first, which leads to no removal of redundant values. Next values for C will be checked against A and B, which leads to the removal of <C, r>. This is followed by the checking of <A, r> against B and C, which confirms that it is not redundant.

5. A couple of errors in the literature

Given the importance of lookahead algorithms in constraint satisfaction research, it is worth pausing to correct a couple of errors in the literature here. Firstly, PLA and FLA do not guarantee to maintain DAC and AC as suggested by Tsang ([14], page 136). As illustrated in the pseudo codes above, after committing to each label and performing FC reduction, PLA checks the values of variables from x_{k+1} to x_n given the variable ordering (x_{k+1}, x_{k+2}, …, x_n), whereas DAC-L checks the values for variables from x_n to x_{k+1}. Therefore, PLA does not guarantee to maintain DAC.

Row 4 in Table 1 provides a counter example to illustrate that PLA reduction does not necessarily prune enough redundant values to achieve AC: a careful check with Definition 2 (AC) and the example above should convince the readers that to maintain AC, both <B, g> and <C, r> must be removed. Row 4 in Table 1 shows that FLA only removes <C, r>, and therefore does not guarantee to achieve AC.

Secondly, achieving DAC from both directions does not guarantee to achieve AC, as it was suggested by Dechter & Pearl ([3], page 1069, and [4], page 13). This can be illustrated by two counters examples in Table 1: given any ordering of variables, BDAC maintains DAC in both directions. In both rows 1 and 3 of Table 1, BDAC removes <C, r> only, but not <B, g>. This shows that by achieving DAC in both directions, one does not guarantee to achieve AC.

6. DAC-L is superior to PLA

In the following, we shall prove that maintaining DAC requires no more compatibility checks than PLA reduction, but it prunes at least as many redundant values as PLA reduction could.

Proposition 1:

Procedure DAC performs no more compatibility checks than PLA reduction under the same variable ordering.

Proof:

This is obvious in the pseudo code of DAC and line 3' in Section 2. The only difference between DAC and PLA reduction is in the ordering of label checks. In fact, when a label <x, v> is checked against variable y in PLA reduction, the domain of y contains all the values input to the PLA reduction procedure. On the other hand, when <x, v> is checked against y in the DAC procedure, it is possible, though not necessary, that the domain of y has been reduced. (This is because y could be after x under the ordering that DAC uses, and values of y are examined before <x, v> is looked at.) We know that if <x, v> were to be pruned due to its having no support from y, then all the values in the domain of y must be checked against <x, v>. So it is possible for DAC to perform fewer number of checks than PLA reduction. ■

Proposition 2:

Procedure DAC prunes at least as many redundant values as PLA reduction.

Proof:

Let DAC and PLA reduction use an ordering under which x is before y, and let D_y be the domain of variable y when a PLA reduction procedure is called. As it is explained in the proof of Proposition 1, when PLA reduction checks <x, v> against variable y, the domain of y will always be D_y. This is not necessarily the case in the DAC procedure since it checks the values of y before checking those of x. Therefore, should <x, v> be pruned by PLA reduction due to its having no support in y, it will always be pruned by procedure DAC. ■

Proposition 3:

It is possible for DAC to prune redundant values that PLA reduction does not.

Proof:

Following the analysis in the above proofs, it is possible for the only values in y which support <x, v> to be pruned before <x, v> is checked against y in achieving DAC. Row 4 in Table 1 shows an example of such a situation: under the ordering (B, C, A), DAC prunes <B, g> while PLA reduction does not. ■

Propositions 1 to 3 together suggest that PLA reduction is inferior to DAC. Table 1 shows that the DAC procedure prunes the same set of values as PLA reduction under five of the six

possible orderings of the three variables. In fact, this example illustrates some general truth: given two variables y and z, if no values in constrained variable z supports <y, b>, then:

(1) if z is *before* y under the ordering used for achieving DAC or PLA reduction, then b will not be pruned because <y, b> will not be checked against z (see rows 1 to 3 in Table 1, where A is ordered before C).

(2) if z is *after* y under the ordering used for achieving DAC or PLA reduction, then b will be pruned (see rows 4 to 6 in Table 1, where A is ordered after C).

Assume further that <y, b> is the only value that supports <x, a>. It is obvious that <x, a> will only be pruned by DAC if the ordering (x, y, z) is used. So if a random ordering is used by DAC, there is one in sixth chance that both <x, a> and <y, b> will be pruned. For reasons explained above, PLA reduction will never be able to prune both redundant values. Under the above assumptions, PLA reduction prunes less than DAC in one out of every six calls under random variable orderings. In general, there is no obvious reason why PLA should be used instead of DAC-L.

7. BDAC-L is superior to FLA

Table 1 above illustrates that BDAC reduces as many redundant values as FLA reduction in five out of the six rows; it prunes more in row 4. We shall prove in this section that it is generally true that BDAC has more pruning power than FLA reduction. We shall then prove that maintaining BDAC using the procedure shown in Section 3 requires no more compatibility checks than FLA reduction.

Proposition 4:

Any value that is removed by FLA reduction when checking forward will be removed by the BDAC procedure in line 3.

Proof:

Let variable x be before variable y under an ordering used by both FLA reduction and BDAC maintenance. Let D_y be the domain of y immediately before these procedures are called. When <x, b> is checked against y in FLA reduction, the domain of y will always be D_y, as the domain of y is not yet examined. Therefore, if y provides no support to <x, b> in FLA reduction, it will also provide no support in maintaining BDAC. ■

Proposition 5:

Any value that is removed by FLA reduction when checking backward will be removed by the BDAC procedure in line 4.

Proof:

Let variable x be after variable w under the ordering used by both FLA reduction and BDAC maintenance. Let D_w be the domain of w immediately before these procedures are called. If the domain of w is not reduced from D_w when <x, v> is checked against w in FLA reduction, then BDAC would prune <x, v> in line 4 whenever FLA reduction does so. FLA reduction could only remove a value v' from the domain of w under two situations: (1) when <w, v'> was checked against some variable after w; and (2) when <w, v'> was checked against some variable before w. Proposition 4 suggests that pruning by FLA reduction under situation (1) would also be achieved by BDAC in line 3; so BDAC would be checking <x, v> against w's domain without v' in line 4. A careful inspection of the pseudo codes should review that line 4 in procedure BDAC performs all the backward checks of FLA reduction in exactly the same order. Therefore, pruning by FLA reduction under situation (2) will also be achieved by BDAC in line 4. So we know that when <x, v> is checked against w in BDAC, the domain of w would have been reduced as much as it could have been in FLA reduction. Therefore, BDAC will prune <x, v> whenever FLA reduction does so. ■

It is worth noting that had lines 3 and 4 swapped their places in procedure BDAC, the above proof will not be valid.

Proposition 6:

It is possible for BDAC to prune redundant values that FLA reduction does not.

Proof:

This is illustrated by the example in row 4 of Table 1. ■

Proposition 7:

Procedure BDAC performs no more compatibility checks than FLA reduction under the same variable ordering.

Proof:

The pseudo codes above show that both BDAC and FLA reduction check every label against every other variable once and once only. (Since every label is checked against every other variable, <C, r> is removed under every ordering used by FLA reduction and BDAC in our example in Section 4, as it is illustrated in Table 1. Whether <B, g> is removed or not depends on whether it is checked against C before or after <C, r> is removed.) Their difference is in the order in which the variables are processed. As explained in the proofs of Propositions 5 and 6, when <x, v> is checked against w, BDAC will be handling a smaller or equal-sized domain of w than FLA reduction. Therefore, BDAC should perform no more (possibly fewer) checks than FLA reduction. ■

Propositions 4 to 6 show that BDAC has more pruning power than FLA reduction. Proposition 7 suggests that FLA reduction requires at least as many compatibility checks as BDAC. Therefore, we can conclude that FLA reduction is inferior to BDAC.

8. Conclusion

Figure 2 summarizes the relative positions of the lookahead algorithms mentioned in this paper. Forward Checking, k-consistency [7] and adaptive consistency [4] lookahead are included in figure 2 for reference.

In this paper, we have proved that PLA reduction performs at least as many checks as DAC-L, but has less pruning power. Similarly, PLA performs at least as many checks as BDAC-L, but has less pruning power. Therefore, we argue that DAC-L and BDAC-L should be used in place of the long established PLA and FLA respectively in future research.

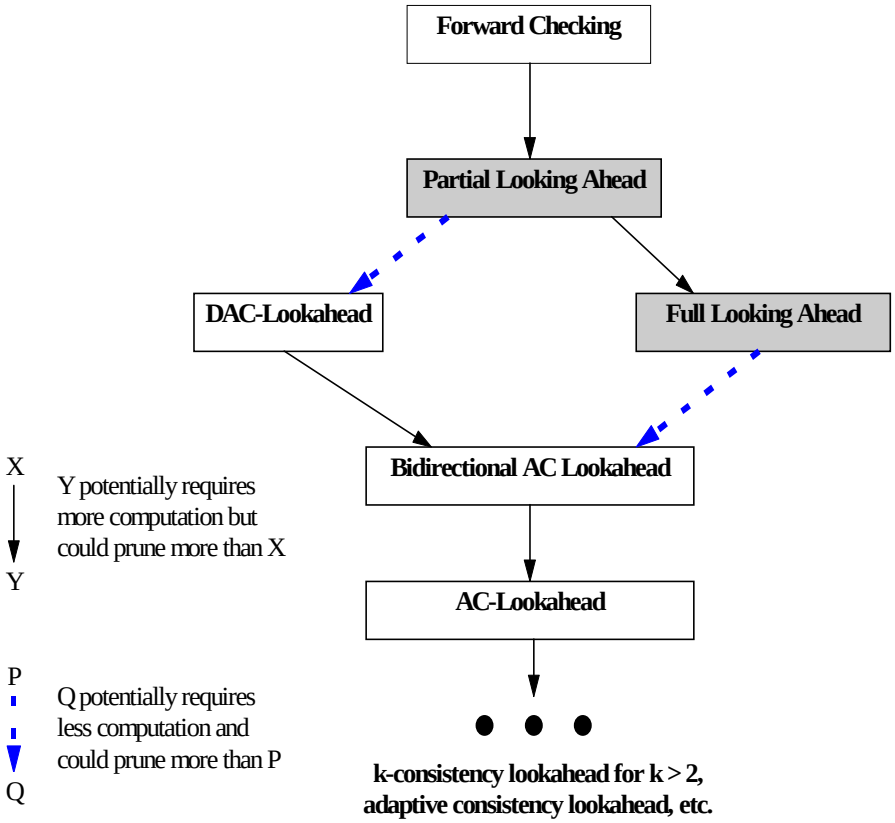

Figure 2 – Relative positions of selected lookahead algorithms (inferior algorithms are shaded, see text)

Acknowledgments

The author would like to thank James Borrett for valuable discussions on this topic. This research is partly supported by EPSRC grant GR/J/42878.

References

[1] Bessière, C., *Arc-consistency and arc-consistency again*, Research Note, Artificial Intelligence, Vol.65, 1994, 179-190
[2] Bessière, C., Freuder, E.C. & Regin, J-C., Using inference to reduce arc consistency computation, in Mellish, C. (ed.), Proc., 14th International Joint Conference on AI, Montreal, Canada, August, 1995, 592-598
[3] Dechter, R. & Pearl, J., The anatomy of easy problems: a constraint satisfaction formulation, Proc., 9th International Joint Conference on AI, 1985, 1066-1072
[4] Dechter, R. & Pearl, J., Network-based Heuristics for constraint-satisfaction problems, Artificial Intelligence, Vol.34, 1988, 1-38
[5] Deville, Y. & Van Hentenryck, P., An efficient arc consistency algorithm for a class of CSP problems, Proc., 12th International Joint Conference on AI, 1991, 325-330
[6] Faltings, B., Arc-consistency for continuous variables, Research Note, Artificial Intelligence, Vol.65, 1994, 363-376
[7] Freuder, E.C., A sufficient condition for backtrack-free search, J ACM Vol.29 No.1 January, 1982, 24-32
[8] Freuder, E.C. & Mackworth, A., (ed.), Constraint-based reasoning, MIT Press, 1994
[9] Haralick, R.M. & Elliott, G.L., Increasing tree search efficiency for constraint satisfaction problems, Artificial Intelligence, Vol.14, 1980, 263-313
[10] Mackworth, A.K., Consistency in networks or relations, Artificial Intelligence, Vol.8, No.1, 1977, 99-118
[11] Mohr, R. & Henderson, T.C., Arc and path consistency revisited, Artificial Intelligence, Vol.28, 1986, 225-233
[12] Montanari, U., Networks of constraints fundamental properties and applications to picture processing, Information Sciences 7 (1974), 95-132
[13] Sabin, D. & Freuder, E.C., Contradicting conventional wisdom in constraint satisfaction, Proc., 11th European Conference on Artificial Intelligence, 1994, 125-129
[14] Tsang, E.P.K., Foundations of constraint satisfaction, Academic Press, London and San Diego, 1993
[15] Van Hentenryck, P., Deville, Y. & Teng, C-M., A generic arc-consistency algorithm and its specializations, Artificial Intelligence, Vol.57, 1992, 291-321
[16] van Beek, P., On the inherent level of local consistency in constraint networks, Proc., 12th National Conference for Artificial Intelligence (AAAI), 1994, 368-373