Ways of Maintaining Arc Consistency in Search using the Cartesian Representation

Marius-Călin Silaghi, Djamila Sam-Haroud, and Boi Faltings

Artificial Intelligence Lab
Swiss Institute of Technology Lausanne
Lausanne, Switzerland
{silaghi,djamila,faltings}@lia.di.epfl.ch

Abstract. The search space of constraint satisfaction problems (CSPs) can be reduced by using value interchangeability. This notion consists of aggregating subsets of values that behave similarly on the future branches of the search. BT-CPR [8], is a typical backtracking algorithm using value interchangeability. It uses the Cartesian product representation of the search space (CPR) which aggregates partial solutions and proves particularly useful for finding and representing all solutions of a CSP.
It is assessed that maintaining arc-consistency (MAC) is the most efficient general algorithm for solving hard problems. A few work on combining MAC with CPR exists. In this paper we study comparatively two other possible alternatives of MAC-CPR.

1 Introduction

A lot of real world problems can be cast as Constraint Satisfaction Problems (CSP). A CSP, (V,C,D), is classically defined as a set V of variables $x_1, x_2, ..., x_n$, taking their values respectively in a set D of domains $D_1, D_2, ..., D_n$ and constrained by a set of constraints $C = \{C_1, ..., C_m\}$. If a constraint C_i links the variables $x_{i_1}, x_{i_2}, ..., x_{i_{k^i}}$ then it is defined as a subset of the Cartesian product $D_{i_1} \times D_{i_2} \times ... \times D_{i_{k^i}}$. k^i gives the arity of the constraint C_i. Two variables are neighbors if there is a constraint linking them. A tuple t satisfies a constraint C_i if the projection of t on the variables linked by C_i belongs to C_i. A tuple with values for all variables in V is a solution for (V,C,D) if it satisfies all the constraints in C.

Depending on the original problem we may need one, several, or all possible solutions. The task of extracting such solutions is NP-complete in general. An intuitive way around this complexity barrier is to structure the search space so that the exploration algorithm operates on aggregated subsets of data rather than on individual possible instantiations. This is the idea behind the Cartesian product representation (CPR) which aggregates partial solutions during backtracking. The use of CPR was shown to bring improvements, especially to the problem of finding all solutions.

MAC is one of the most powerful general search algorithms. It consists of interleaving backtracking with a notion of local consistency called Arc Consistency (AC). In [7] is presented an algorithm called *backtracking*^{dp} that combines

MAC with CPR. In that case the aggregations were computed statically prior to search and without guarantees of maximality. In this paper we study two variants of MAC-CPR that incorporate the dynamical computation of maximal aggregations.

The rest of the paper is structured as follows. We first recall the necessary background. Then we present the existing work on backtracking using CPR, as well as the two alternatives of MAC-CPR we study. The two variants are compared both theoretically and experimentally. In Section 5 we discuss our results and suggest a possible improvement of $backtracking^{dp}$.

2 Background

The search algorithms for CSPs are generally classified either as intelligent backtrackers that learn from the past, or lookahead techniques that use local consistency in order to reduce the number of future alternatives. Typical intelligent backtrackers are the Constraint-Based Back-jumping (CBJ), the Back-Marking [9], the Dynamic Backtracking [6] and the Partial Order Dynamic Backtracking [3].

The most popular lookahead strategies are Forward Checking (FC) and Maintaining Arc-Consistency (MAC) [11]. The latter has proven to outperform most existing search algorithms in practice. Once a value has been instantiated, FC prunes from the domains of its uninstantiated neighbors (future variables) the values that are inconsistent with the value chosen for the current variable. MAC enforces a form a local consistency called Arc Consistency on all the future variables.

Arc Consistency

AC has been the subject of intensive prospection. Several versions were developed, each of which stresses a particular property. Some AC algorithms deal with special cases (e.g. AC^{dp} [7]). The ones that are useful for the general case are named ACx where x stands for a number. AC3 is often the best in computational time. AC7 [1] seems to be the best in the number of constraint checks while AC6 provides a good compromise with AC3. AC7 uses the bidirectionality of the constraints in order to reduce the number of checks and its enforcement within MAC is presented as promising.

CPR

Interchangeability [5] provides a principled approach to simplifying the search space. Values are interchangeable if exchanging one for one another in any solution produces another solution. BT-CPR [8] is one of the first search algorithms using interchangeabilities. It uses a limited form of interchangeability called neighborhood interchangeability (NI) to aggregate partial solutions in the search space. A combination of CPR with FC (FC-CPR) was also described in [8]. Two values, a and b of a variable x_i in the CSP (V,C,D) are neighborhood interchangeable iff for any constraint $C_l \in C$ linking x_i and x_{l_j}, the two sets of values from x_{l_j} that satisfy C_l with a, respectively with b are identical. Two values a and b are partially neighborhood interchangeable (PNI) against a set of constraints $S \subset C$ iff they are neighborhood interchangeable in the CSP (V,S,D). We will also say that two values a and b are partially neighborhood interchangeable against a set of variables $N \subset V$ iff they are neighborhood interchangeable in the CSP (N,C',D) where C' is the subset of C linking only variables from N.

FC-CPR maintains at each node a partial solution represented as a Cartesian product. For example, a partial solution involving variables a and b will be represented by a Cartesian product like $(A := \{1, 2\}) \times (B := \{3, 5, 8\})$. The values of the future variables that are compatible with the current partial solution are also structured as Cartesian products. FC-CPR prunes (FC) the domains of the future variables for each possible value of the current one. Based on the obtained active domains for the future variables, it builds a structure called the discrimination tree (DT) [5] that enables a cheap merging of the result into maximal Cartesian products. These Cartesian products correspond to partial neighborhood interchangeable sets of the current variable against all future variables. The children nodes in the search tree are obtained by expanding the current partial solution with any of the obtained interchangeable sets.

We recall that the DT is a tree structure having the values in some of the nodes. The insertion of a new value is performed by following a path from the root of the tree, where each node corresponds to the next feasible tuple containing the value, in the ordered relations of the current variable. If the corresponding node did not previously exist, a new branch (set) is created. The value is placed in the set found in the node at the end of this path. The enumeration of the feasible tuples is not an overhead if it can be reused in the search process, for example when all solutions are looked for.

Backtracking^{dp}

In [7], a possible extension from FC to MAC-CPR was mentioned where the interchangeabilities were used not only for inferring partial solutions, but also to infer information during the propagation of arc consistency. For doing so, a specialized version of AC, called AC^{dp}, was introduced. However, the algorithm proposed uses only static neighborhood interchangeabilities detected before the search starts. The interchangeabilities that appear dynamically, because of pruning, were disregarded. In that version, the further computation of fully dynamic interchangeabilities can be expensive since AC^{dp} needs them for all variables.

In the *backtracking*^{dp} of [7], the partial neighborhood interchangeability sets between each pair of variables are precomputed statically before search. At each step of the search process, the neighborhood interchangeabilities between the current variable and the future ones are obtained by intersecting the precomputed sets. Note that the precomputed sets are no longer maximal in general. This is because of the pruning induced on the future variables by the current instantiations. This pruning is obtained either by forward checking or Arc Consistency.

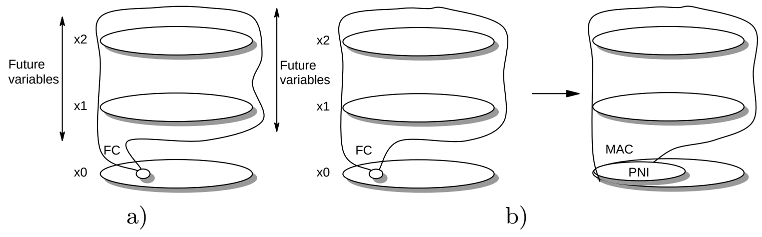

Fig. 1. a) FC-CPR: Full FC is performed on all future variables for all values in x_0 in order to compute a DT. b) MAC-CPR: Full FC performed on all future variables for all values in x_0 to build a DT, then AC is enforced on all future variables for the chosen PNI set.

No attempt of further merging is performed on the result, which means that the aggregations obtained are not minimal in number. A weakness of $backtracking^{dp}$ is the fact that it computes several times the same intersections between interchangeable sets. We will show that these operations can be avoided.

In FC-CPR [8] the partial neighborhood interchangeabilities are computed in a fully dynamic way. The possible aggregations for each node (current variable) are computed from scratch on the basis of the previous instantiations. FC-CPR does not incorporate any form of AC.

A variant of FC-CPR is proposed in [10]. It allows for cheaply getting certain additional solutions once the first one has been obtained. However that algorithm is not optimal when we need an arbitrary number of additional solutions. The original FC-CPR offers a more general alternative in that case.

In [4] it was shown that the partial interchangeable sets against subsets of the future variables can be organized in a hierarchy of a tree. This proves helpful in one of the algorithms we present later.

3 MAC and CPR

We now present two other possible ways of interleaving backtracking on CPR with arc-consistency. Their main characteristics are that they:

– compute the Cartesian products in a fully dynamic way,
– guarantee that the Cartesian products obtained at each step are minimal in number

Figures 1 and 2, illustrate the main differences between the two studied variants.

3.1 MAC-CPR

It is obvious that the enhancement of FC-CPR with AC should not be done by using AC to prune the future variables for each value of the current variable. The propagations performed by AC will be identical for all the members of a partially

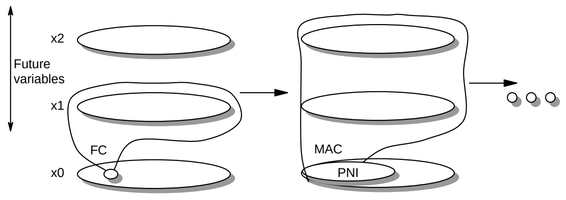

Fig. 2. QMAC-CPR: FC is performed for all values in x_0 considering only one of the future neighbor variables (here x_1) to build a DT, then AC is enforced for the chosen PNI set on all current and future variables (less the constraints already scanned in order to build the DTs). These two steps are iterated considering one by one, all the future neighbor variables.

neighborhood interchangeable set, and the work would just be replicated. The simplest way to accomplish the task is therefore to enforce AC at each node, i.e. for each interchangeable set computed. The interchangeable sets are computed dynamically as in FC-CPR. MAC-CPR is obtained by simply performing an AC propagation for each node of FC-CPR before forward check and merging. Once the interchangeable values are obtained, the Arc-Consistent domains of the future variables with any value of an interchangeable set, are consistent with all the other values of that set.

Here we have to specify that during MAC, each time that AC is reinforced there exists an ordering of the AC queue such that a prefix of the process is identical to FC. What we do in MAC-CPR is to first perform separately this prefix for all values, detecting sets of PNI values. We then perform the rest of the AC with any chosen queue ordering only once for a PNI set.

It is worth mentioning that it may be useful to merge the next levels of nodes immediately after AC is applied to all of them since the pruning can reveal stronger interchangeabilities.

3.2 QMAC-CPR

In this version our goal is still to compute maximal neighborhood interchangeable sets at each node in order to optimize the aggregations. We have however observed that performing a full forward checking on all future variables at each node, as in MAC-CPR, is not necessarily the best choice. If we only look for the first solution, a lot of useless work will be done at each node for pruning irrelevant parts of the search tree (i.e. for all the values of the current variable that will never be chosen). Performing a full FC may also entangle eventual benefits from early reductions, via Arc-Consistency, of the domains of the future variables. Such propagation can indeed reach domain wipe-outs or detect inconsistencies before the whole FC is done. A similar argument is presented in [1] where it is argued that AC can be improved by performing the propagation immediately after a value is deleted. This was presented as an AC queue ordering heuristic.

In the new algorithm we propose, we have come up with a compromise that fulfills the criteria of: building stronger aggregations at each step, early propagating AC and FC deletions and avoiding useless FC checks on postponed branches.

The idea is to enforce AC on the future variables immediately after the domain of any *single* future variable is pruned. This is done after the corresponding partial interchangeable sets I_{ij} is built (i.e. the one concerning only the current variable i and the pruned future variable j). AC is maintained after any such set $I_{ij}^k \in I_{ij}$ is chosen (figure 2). The early propagation of deletions obtained that way can also prune values from other future variables that are neighbors with the current variable. When this occurs, an additional gain, proportional with the domain size is obtained. In effect, no value in the current interchangeable set will have to be checked against the pruned value.

Moreover, the cost of finding the first solution may decrease. Performing FC on the other future variables for the values in partial interchangeable sets $I_{ij}^{l(l \neq k)}$ that are different from the one actually chosen is postponed until a backtracking occurs, with no additional cost. The gain may not appear if the difference in merging throws us on a worse branch first.

The corresponding technique is described in the algorithm 1. The function $Solve$ will recursively search the solution. The parameter $currCP$ represents the Cartesian product of the current labels of the instantiated variables. $varCrt$ is the current variable. The currently active values in its domain are in $currCP$ as well. The parameter $varFut$ points to the neighbor future variable to be analyzed in this iteration, if any. If there is no neighbor future variable to analyze, then $varFut$ is nil. $currFD$ is the Cartesian product of the currently active values in the domains of the future variables. At the beginning, AC is initialized (we have used AC6 and AC7) and the obtained AC domain for the variable i is noted D_i^0. $getFirstFutVar(v)$ returns the first future neighbor variable of v if any exists, otherwise nil. Then $Solve$ is called with $Solve(D_0^0, 0, getFirstFutVar(0), D_1^0 \times ... \times D_n^0)$. dt represents a discrimination tree computed with the function $DT()$ at line 1.2. The algorithm used is the same with the one in [7]. The iterator $next(dt)$ returns a structure containing a pair of consistent interchangeable sets in $varCrt$ and $varFut$. $intersectCP(currCP, dts, varCrt)$ computes the Cartesian product obtained from the partial solution received in $currCP$ when the domain of the current variable $varCrt$ in dts is intersected with the one in $currCP$.

At each node of the search, we first compute (line 1.2) the partial neighborhood interchangeable sets. The stuctures of AC6, respectively AC7 are used to improve the computation of the discrimination tree. The vectors $last$ are used in order to avoid checking tuples already tested during the propagation of AC. Afterwards, we iterate for each set (line 1.3) the actualization of the new current partial solution (line 1.4) and that of the future active domains (line 1.5). The propagation of the pruning of the future domains is done using AC. It occurs if any domain was changed. If one more future neighbor variable exists (line 1.6), we create for it a child node in the search tree at line 1.7. Otherwise, we choose a new variable from the future ones and we add its domain to the Cartesian product of the actual partial solution (line 1.8), before the corresponding child node is built.

As desired, at line 1.5 we propagate the Arc-Consistency earlier than if it is done after forward checking all the future variables. That would be required in order to built the discrimination trees for the partial neighborhood interchangeability against all future variables at once. However, the result was shown in [4] to be identical, since the hierarchy of partial interchangeabilities is a tree. Moreover, the computation of the interchangeable sets is performed in a more depth-first fashion than in [8]. Therefore, if we would not propagate AC, but just perform FC, we already obtain a version of FC-CPR that is improved for finding the first solution.

A drawback of the previously presented algorithm may be that the stack with the structures needed by AC increases in size for graphs of constraints of high density, especially if versions of AC requiring many structures are used (AC7, AC6). However, since many of the nodes show to not change the domains of the variables, the corresponding stack do not need to be used for those nodes. Most often, only the first variables from a set of future neighbors change the domains, therefore we could say that we perform AC after the first rather than after the last of the future neighbor variables was used for partitioning. In practice it happened sometimes that we needed similar AC structures on the stack with QMAC-CPR as with MAC-CPR, even if the technique was included in MAC-CPR as well. However, if one decides not to use the AC7 queue heuristic of [1], than he can implement the propagation as described in the section 5.1.

4 Comparing QMAC-CPR and MAC-CPR

In this section we will show that QMAC-CPR is theoretically better that MAC-CPR in term of power of aggregation. While MAC-CPR guarantees maximal partial neighborhood interchangeable sets, QMAC-CPR produces maximal aggregations based on a more global form of interchangeability. As the experiments will show, this does not mean that in practice, the difference of efficiency is significant. To give an intuition of the theoretical superiority of QMAC-CPR, we start by presenting an illustrative example. We will then give a theoretical proof before presenting a preliminary experimental evaluation.

4.1 Illustrative example

In figure 3 we present an example where QMAC-CPR gives a better aggregation that MAC-CPR. In the figure is presented the status when the x_k is the current variable. In the domain of x_k we have the active values $v_{k_{(l-1)}}$, v_{k_l} and $v_{k_{(l+1)}}$. The active values for x_{k+1} are $v_{(k+1)_{(j-1)}}$, $v_{(k+1)_j}$ and for x_{k+2}, $v_{(k+2)_i}$ and $v_{(k+2)_{(i+1)}}$. The arrows show values that are eliminated when the value at the starting point of the vectors are chosen for the corresponding variable. We see that the current problem is AC. In the next description, due to space, we will

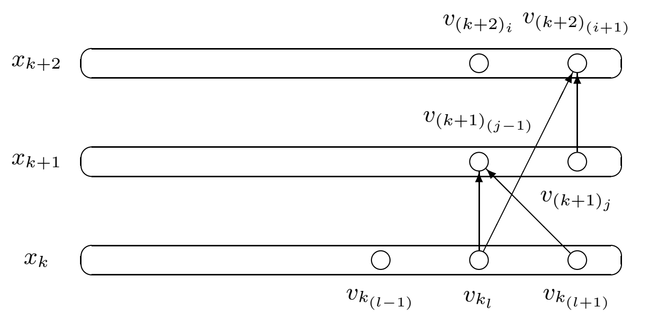

Fig. 3. Snapshot at current variable x_k.

disregard the branches where the value $v_{k_{(l-1)}}$ is chosen alone for the instantiation of the variable x_k. We refer to the pairs (currCP,currFD) as pairs of Cartesian products,

In the presented situation, MAC-CPR will first create the pairs of Cartesian products

$$(\{v_{k_{(l-1)}}\}, \{v_{(k+1)_{(j-1)}}, v_{(k+1)_j}\} \times \{v_{(k+2)_i}, v_{(k+2)_(i+1)}\}),$$

$$(\{v_{k_l}\}, \{v_{(k+1)_j}\} \times \{v_{(k+2)_i}\}),$$

$$(\{v_{k_{(l+1)}}\}, \{v_{(k+1)_j}\} \times \{v_{(k+2)_i}, v_{(k+2)_(i+1)}\}),$$

and only after the third pair is chosen and AC is enforced, will it become $(\{v_{k_{(l+1)}}\}, \{v_{(k+1)_j}\} \times \{v_{(k+2)_i}\})$. But in that moment the branch induced by the second pair will already have been solved, the solution lost, and there is no mean to infer the solution of this new branch.

When faced with the same situation, QMAC-CPR will first consider the x_{k+1} as future variable. In that step it obtains the pairs of Cartesian products

$$(\{v_{k_{(l-1)}}\}, \{v_{(k+1)_{(j-1)}}, v_{(k+1)_j}\} \times \{v_{(k+2)_i}, v_{(k+2)_(i+1)}\}),$$

$$(\{v_{k_l}, v_{k_{(l+1)}}\}, \{v_{(k+1)_j}\} \times \{v_{(k+2)_i}, v_{(k+2)_(i+1)}\}).$$

We consider now the moment when the branch for the second pair is chosen. The MAC that is performed at this moment will prune the value $v_{(k+2)_{(i+1)}}$ from x_{k+2}. Therefore, at the next step, the new Cartesian product pair found will be:

$$(\{v_{k_l}, v_{k_{(l+1)}}\}, \{v_{(k+1)_j}\} \times \{v_{(k+2)_i}\}).$$

This pair aggregates the second and the third branch of the algorithm MAC-CPR.

4.2 Theoretical results

Theorem 1. *Any two values that are aggregated at a step of MAC-CPR will also be aggregated (or both rejected) during the corresponding set of steps of QMAC-CPR.*

Proof. Two values v_{k_i} and v_{k_j} of variable x_k are aggregated by MAC-CPR only if they are neighborhood interchangeable with all the future variables given their currently active domains. We will note the future neighbor variables of the variable x_k with $\{x_{k_0}, x_{k_1}, ..., x_{k_l} ...\}$. With QMAC-CPR, when the first future neighbor variable x_{k_0} is considered, the two values v_{k_i} and v_{k_j} will also be aggregated because they needed to be neighborhood interchangeable with x_{k_0} in order to be aggregated by MAC-CPR.

By induction after the order of the considered future neighbor variable, if the values v_{k_i} and v_{k_j} were aggregated when the variable x_{k_l} considered, then we show that they will be also aggregated when the variable $x_{k_{l+1}}$ is considered as future neighbor of variable x_k. Indeed, both of them will behave identically against all the values that were active in the domain of variable $x_{k_{l+1}}$ before x_{k_0} was considered with x_k, from hypothesis. During all the process since that moment, the values of the domain of variable $x_{k_{l+1}}$ may have been only deactivated. Therefore the behavior of v_{k_i} and v_{k_j} against the values still active in $x_{k_{l+1}}$ could have remained only identical and the two values will be again aggregated. Of course, if all the supports in $x_{k_{l+1}}$ were already deactivated for the two variables, then both of them will be simultaneously rejected. □

Corollary 1. *With QMAC-CPR we obtain at least the same aggregations obtained with MAC-CPR. Any distinct Cartesian product obtained with MAC-CPR will be found with QMAC-CPR, eventually merged with other Cartesian products.*

The eventual additional aggregations were illustrated in figure 3. Of course, if we would decide to first perform at each MAC-CPR node the AC enforcement for all pairs of Cartesian products, and then to try again to merge the partitions, all these before any child node is visited, even stronger aggregations might be obtained. However, that may give much overhead for finding the first solution. This alternative is subject to further research.

4.3 Experiments

The most usual ways of measuring the efficiency of an algorithm searching for solutions in CSPs are by counting the number of constraint checks and by measuring the time needed for solving sets of real problems. The first measure is based on the consideration that the time needed for maintaining the data structures is more or less fixed while the time needed for checking constraints is the one that should be taken into account if such checks, with a complexity dependent on the problem, become expensive. That is the case if checking a constraint reduces to solving a problem or handling a device. If the cost of a constraint check is low, then the other costs will prevail and the time measurement gives the most important information. However, the time measurement is implementation and platform dependent.

A third way of measuring the efficiency of a backtracking, counts the number of expanded nodes in the search tree, the tree that would represent the states of the run. Even if intuitively correct, such measurements will be in certain cases cheated by algorithms that cluster several nodes into one, without reducing the overall cost. It is seldom that the work done at one node with different algorithms is identical.

variables	QChecks/Checks	First Solution	DT
20	611/624	115/116	738/762
30	508/523	211/215	764/791
40	847/851	524/543	119/123
50	120/122	720/790	107/121
60	120/123	335/354	202/209
70	397/410	139/151	571/574
80	183/185	167/179	144/147

Table 1. QMAC-CPR vs. MAC-CPR

Even if the number of expanded nodes in the QMAC-CPR algorithm is clearly higher than the one of MAC-CPR, the cost of each node is expected to be at least proportionally lower.

We perform our tests on binary constraints. The experiments were performed in a Unix environment under usual load. Therefore the comparison of running time is not relevant. We will however give an idea of time results. We have chosen to measure the performance in term of the number of checks.

We have implemented FC-CPR and MAC-CPR where we maintain AC at each step using AC6 and AC7.

As suggested before, the difference between MAC-CPR and the algorithm described in [7] is that MAC-CPR computes the interchangeabilities dynamically, but it does not use the statically computed ones during AC or for the discrimination tree. We have also implemented the QMAC-CPR algorithm using AC6 and AC7. We have generated random problems with 20 to 80 variables. Each variable participates to 3 binary constraints in average. Each variable had a domain of size 8 and each constraint a tightness of 35. Those parameters were chosen to generate the problems close to the peak of difficulty.

In table 1 we present the results obtained by comparing the algorithms on 50 examples of each type. The second column presents the ratio between the number of constraint checks needed to find all solutions with QMAC-CPR respectively with MAC-CPR. The third column presents the same ratio when only the first solution was looked for.

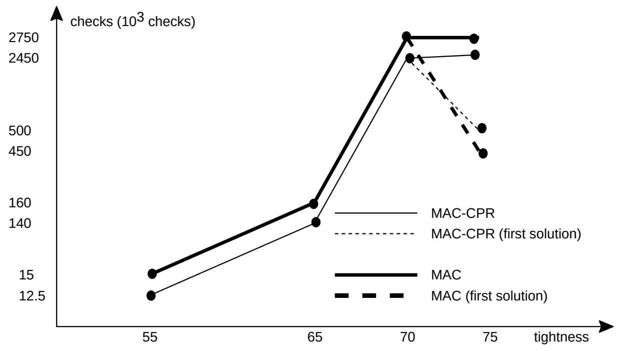

Fig. 4. The average number of constraint checks for MAC7 and MAC-CPR. The number of checks for finding only the first solution is shown with dashed lines.

The structure of QMAC-CPR reduces also the number of comparisons needed in order to built the discrimination trees during the search for all solutions. The ratio of their number for QMAC-CPR and MAC-CPR was given in the fourth column and suggests a very small improvement that would be brought to FC-CPR in the search for all solutions. We have also verified that both versions of MAC-CPR bring strong improvements compared with FC-CPR. This is very clear in the corresponding tests, but it does not make the subject of this paper. Preliminary comparisons of the two new algorithms versus MAC6 and MAC7 were also performed for 800 random problems with 40 variables, density of 30% and 8 values per domain. The improvements for either finding all solutions or for finding that no solution exists was of up to 20% in average for both time and number of constraints checks, as shown in figure 4. The first solution for loosely constraint problems continues to be found quicker with MAC.

The number of the Cartesian products used to represent all solutions was sometimes different for the two algorithms. This was due to the additional interchangeabilities that appear by maintaining AC after each future variable is pruned. The QMAC-CPR may need less Cartesian products than MAC-CPR as it succeeds to merge some of them. The inverse is never possible. However, the averaged reduction has reveled to be seldom above one percent. Concerning the time comparisons, we mention that the ratio of the averaged observed time oscillated of a few percents around 1. Certain single cases were at maximum one order of magnitude better in time with one algorithm than with the other.

5 Discussions

We have presented and evaluated two possible variants of backtracking on CPR using AC. MAC-CPR is a straightforward extension. QMAC-CPR is more elaborated and exhibit better features in theory. The empirical evaluation shows almost similar performances on random problems. The empirical comparison against MAC7 shows some advantage of the new algorithms for over-constrained problems.

CPR infers from the partial solution obtained for one value, partial results for other values of the same variable. From this point of view, CPR belongs to the class of intelligent backtrackers. Several researchers have tried to combine lookahead strategies with some learning techniques. While CBJ [9] brings some improvements to FC, it seems to alter MAC [2]. The explanation can be that CBJ brings pruning at an average cost superior to the average efficiency of MAC. But CPR offers high gains at low cost and therefore, integrating it with lookahead strategies like MAC can be more successful than it was for the other learning techniques.

Note also that a version of FC-CPR improved for finding the first solution is obtained by eliminating the AC propagations from QMAC-CPR.

5.1 EMAC-CPR

We present a possible improvement of $backtracking^{dp}$ that incorporates features of QMAC-CPR. This algorithm has not been implemented.

We note with K the number of future neighbors of the current variable and with d the maximum size of a currently active domain. As previously mentioned, the algorithm presented in [7] can be enhanced by using the structure of the new algorithm QMAC-CPR. Indeed, even if AC is maintained only after pruning all the future neighbors of the current variable, that could be done with the procedure described in algorithm Extended MAC-CPR (EMAC-CPR) presented in algorithm 2. Doing like QMAC-CPR, the cost of computing the intersection of the partial interchangeable sets is reduced from the combinatorial worst case when all tuples of sets from all partial interchangeable sets are considered, Kn^K, to n^K by maintaining on the stack the partial intersection of several such sets. In the presented algorithm, we propose to try to merge the statically computed partially interchangeable sets. That can be accomplished using a discrimination tree algorithm (DT^{enhanced}) over only one representative value of each of the precomputed sets.

The algorithm DT^{enhanced} is the same as the one described in [5] with the only difference that the substitutability that was obtained statically, like in [7], is recursively used to obtain a dynamically computed one, as previously mentioned. It is obvious that if two values belonging to two different neighborhood interchangeable sets can be merged (are interchangeable), than the two sets can be merged. $getFD$ will prune from the active domain of the future variable implied in dts the values deactivated in dts. The changes to domains are recorded whenever done (at lines 2.2,2.3) and they can launch an AC maintenance (at line 2.4). The other functions of EMAC-CPR are the same with those in QMAC-CPR.

6 Conclusion

Two variants of MAC with CPR, called MAC-CPR and QMAC-CPR, were developed and presented in this paper. The corresponding advantages and drawbacks as well as their sources were discussed. QMAC is theoretically better but shows no significant gains on random problems. The result of the theoretical analysis remains interesting.

QMAC can be seen simultaneously as both primal and dual space oriented. Indeed, the search is performed variable after variable, and in the same time it is performed constraint after constraint. For example, in the case of binary constraints, considering a pair of variables as QMAC-CPR does with the current and a future neighbor variable, reduces to considering the constraint linking them. This can help in the understanding of the two streams of approaches. By studying it, further algorithms could be transferred in an improved way from one of them to the other one. An example of such an application is suggested in section 5.1.

7 Acknowledgments

This work was performed at the Artificial Intelligence Laboratory of the Swiss Federal Institute of Technology in Lausanne and was sponsored by the Swiss National Science Foundation under project number 21-52462.97.

References

1. Bessière C., Freuder E.C., and Régin J.-C.: Using constraint metaknowledge to reduce arc consistency computation. AI, (107):125–148, 99
2. Bessière C. and Régin J.-C.: MAC and combined heuristics: Two reasons to forsake FC(and CBJ?) on hard problems. In CP96, pages 61–75. CP, 96
3. Bliek C.: Generalizing partial order and dynamic backtracking. In AAAI-98 Proceedings, pages 319–325, Madison,Wisconsin, July 98. AAAI
4. Choueiry B.Y. and Noubir G.: On the computation of local interchangeability in discrete constraint satisfaction problems. Technical Report KSL-98-24, Standford, 98
5. Freuder E.C.: Eliminating interchangeable values in constraint satisfaction problems. In Proceedings of AAAI'91, pages 227–233, 91
6. Ginsberg M. and McAllester D.: Gsat and dynamic backtracking. In J.Doyle, editor, Proceedings of the 4th IC on PKRR, pages 226–237. KR, 94
7. Haselböck A.: Exploiting interchangeabilities in constraint satisfaction problems. In Proceedings of IJCAI'93, pages 282–287, 93
8. Hubbe P.D. and Freuder E.C.: An efficient cross product representation of the constraint satisfaction problem search space. In Proc. of AAAI-92, pages 421–427, July 92
9. Kondrak G.: A theoretical evaluation of selected backtracking algorithms. Master's thesis, Univ. of Alberta, 94
10. Lesaint D.: Maximal Sets of Solutions for Constraint Satisfaction Problems. In A. Cohn, editor, Proceedings of ECAI'94, pages 110–114. John Wiley & Sons,Ltd, 94
11. Sabin D. and Freuder E.C.: Contradicting conventional wisdom in constraint satisfaction. In Proceedings ECAI-94, pages 125–129, 94

```
    procedure Solve(currCP, varCrt, varFut, currFD)
        if (varFut==nil) then
            if (existsNextVar(varCrt)) then
                v=getNextVar(varCrt)
1.1             add(currCP,domain(v))
                newFD=project(currFD,v)
                solve(currCP,v,getFirstFutVar(v),newFD)
            else
                ReportSolution(currCP)
            end
            return
        end
1.2     dt=DT(varCrt, varFut, currFD)
        while (!empty(dt)) do
1.3         dts=next(dt)
1.4         newCP=intersectCP(currCP, dts, varCrt)
            if (modifies(dts,currCP,currFD)) then
1.5             newFD=propagate(currFD,dts,varFut)
                if empty(newFD) then
                    continue
                end
            else
                newFD=currFD
            end
1.6         if (existFutureVariable(varCrt, varFut)) then
                newFutureVar=getNextFutVar(varCrt,varFut)
1.7             solve(newCP,varCrt,newFutureVar,newFD)
            else
                v=getNextVar(varCrt)
1.8             add(newCP,domain(v))
                newFD=project(newFD,v)
                solve(newCP,v,getFirstFutVar(v),newFD)
            end
        end
    end.
```

Algorithm 1: *QMAC-CPR*

```
    procedure Solve(currCP,varCrt, varFut, currFD)
      if (varFut==nil) then
        if (existsNextVar(varCrt)) then
          v=getNextVar(varCrt)
2.1       add(currCP,domain(v))
          newFD=project(currFD,v)
          solve(currCP,v,getFirstFutVar(v),newFD)
        else
          ReportSolution(currCP)
        end
        return
      end
      dt=DT$^{enhanced}$(varCrt, varFut, currFD)
      while (!empty(dt)) do
        dts=next(dt)
2.2     newCP=intersectCP(currCP,dts,varCrt)
2.3     newFD=getFD(currFD,dts,varFut)
        if (existFutureVariable(varCrt,varFut)) then
          solve(newCP,varCrt,getNextFutVar(varCrt),newFD)
        else
2.4       if (changesToDomains()) then
            AC$^{dp}$()
            if (empty(newFD)) then
              continue
            end
          end
          v=getNextVar(varCrt)
          add(newCP,domain(v))
          newFD=project(currFD,v)
          solve(newCP,v,getFirstFutVar(v),newFD)
        end
      end
    end.
```

Algorithm 2: *EMAC-CPR*