Univ. of Kansas Libraries Interlibrary Loan (KKU)
InProcess Date: 2009/03/31
Date Printed: 04/01/2009 7:19 AM

Special Instructions: .

ODYSSEY ARIEL
ILL Number: 52614494
ILL

Patron: Choueiry, Berthe Y.

Ariel: 129.93.16.110

Paging notes:
__ Call # NOS __ Call # ≠ Title
__ Book/Volume/Issue/Series NOS (circle)
__ Year ≠ Volume (checked both)

__ Article not found as cited (check index)

Call #: **Q334 .E97 1988 -**
Location: **ANSCHUTZ LIBRARY**

Journal Title: **ECAI '88 ; proceedings of the 8th European Conference on Artificial Intelligence ... Aug. 1-5, 1988.**

Volume: **1** Issue:

Month/Year: **1988** Pages: **651-656**

Article Author: European Conference on Artificial Intelligence (8th ; 1988 ; Munich, Germany)

Article Title: Roger Mohr and Gerald Masini; Good Old Discrete Relaxation

Patron: **Choueiry, Berthe Y.**

Odyssey TN: 718662

Shipping Address for LDL
UNIVERSITY OF NEBRASKA-LINCOLN
INTERLIBRARY LOAN

Electronic transmission from:
University of Kansas
Watson Library, Main ILL Office
Phone: 785-864-3964
Email: illlend@ku.edu

For resend requests:
1) Call (785-864-3964) or email (illlend@ku.edu) within 24 hours
2) To expedite requests, please include the document ID#.

This transmission is being sent in response to this request.

Good Old Discrete Relaxation

Roger Mohr, Gérald Masini

CRIN — LORIA
B.P. 239
54506 Vandœuvre-les-Nancy Cedex, France

Abstract

This paper presents GAC4, an optimal generalization of the optimal discrete relaxation algorithm AC4. It is shown that this generalization can be used for running path consistency in a complexity equal to the best consistency algorithm already discovered, PC3. Experimentations have been done on several constrained networks in order to compare behaviours of GAC4 and AC4. They point out that GAC4 runs as fast as AC4. If the network is not ambiguous, then results from GAC4 and AC4 are similar, else GAC4 pruning is effectively better.

1 Introduction

Discrete relaxation was probably first introduced by Waltz for interpreting polyhedral scenes with shadows [Wal75]. It is a general computational technique which spreads out of computer vision applications to other domains of artificial intelligence or data bases systems.

The generic use of this technique is related to the propagation of constraints through a network of hypothesis. Each node of the network represents a given fact with several interpretations. Constraints on these nodes allow to discard some of them: for instance, if node i has interpretation x then node j cannot be y. For this reason, people usually say that relaxation is a tool for solving the labelling problem that consists in finding the admissible labels for each node when constraints on labels are added. This problem contains the graph colouring problem — labels are then colours — and therefore is a NP-hard problem. Discrete relaxation runs in polynomial time and does not solve the problem completely. It produces only a locally consistent solution.

The labelling problem can also be solved using continuous relaxation techniques [FB81]. They offer the advantage of taking into account more fuzzy constraints and, moreover, they produce a locally optimal solution. However, the process of building a solution through this approach is hard to control: the user has to hope that the various coefficients of the system are right balanced in order to produce the right solution. On the other hand, discrete relaxation handles only true/false constraints and therefore allows full control on the result produced.

The algorithm AC3[1], designed for discrete relaxation, was presented in [MF85]. An optimal algorithm for the same purpose, AC4, was presented one year later [MH86]. Section 2 of this paper presents the algorithm GAC4 which extends AC4 to make it handle n-ary constraints. This algorithm is also optimal and section 3 explains how it can be used to run path consistency [Mon74]. In the worst case, its time complexity is the same as the one of the best known algorithm, PC3. Section 4 provides some experiments with AC4 and GAC4. At last, section 5 emphasizes how to extend this kind of algorithms to make them suitable to networks having error nodes. It presents also the way to run them on parallel computers.

2 Generalization of AC4

2.1 Some Definitions

The network has a set N of n nodes $i, j, \dots k$. Each node i has a set L_i of at most m possible labels. Constraints are relations between labels and, usually, discrete relaxation runs on binary constraints. If there is a constraint edge between i and j, a relation $R_{i,j}$ specifies the admissible labels: if $R_{i,j}(a, b)$ holds, label a for i and label b for j are admissible.

In our case, the algorithm is extended to deal with p-ary constraints. Instead of having binary constraints on pairs of nodes which define the edges in a graph of constraints, we have constraints on p nodes which define the edges in a ***hypergraph*** of constraints. So a constraint becomes a relation $R_{i,j,\dots k}$ specifying the admissible labels

[1]Arc Consistency 3

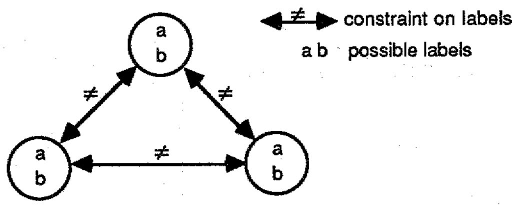


Figure 1: an arc consistent network which is not (path) consistent.

for the nodes $i, j, \dots k$. We identify $R_{i,j,\dots k}$ with the subset of p-uples of labels from $\{i, j, \dots k\}$ admissible with respect to $R_{i,j,\dots k}$. In this way, a constraint R is a set of p-uples $((i,a),(j,b),\dots (k,c))$. We assume that K is the sum of the lengths of all admissible p-uples for all constraints R. A labelling is *consistent* iff:

$$\forall i \in N, \forall a_i \in L_i, \forall j \in N, \exists\, a_j \in L_j \text{ such that } \forall R_{k,\dots j}, \; R_{k,\dots j}(a_k, \dots a_j) \text{ holds}$$

This states only the fact that, for each labelling (i,a), it exists a label for all the other nodes such that all the constraints are satisfied simultaneously.

A labelling is *arc consistent* iff:

$$\forall i \in N, \forall a_i \in L_i, \forall R_{j,\dots k} \text{ constraining } i, \forall j, \dots k, \; \exists\, a_j, \dots a_k \text{ such that } R_{j,\dots k}(a_j, \dots a_k) \text{ holds.}$$

Of course, consistency implies local consistency, but the converse is not true, as it is shown by fig. 1. This network illustrates a simple example of graph colouring which has obviously no solution. It is nevertheless consistent when each node is examined locally.

2.2 Principle of GAC4

AC4 is an optimal algorithm for building the largest locally consistent subset of an initial labelling, but it deals only with binary constraints. This is not sufficient to solve any kind of problem. For instance, in the block world problem [Wal75], it was necessary to consider ternary relations for constraining the admissible labels (see fig. 2).

GAC4 stands for *Generalized AC4*, and it is designed to solve the same kind of problems as AC4: to build the largest subsets from the initial sets of possible labels so as to constitute a locally consistent network. Both of the algorithms work in an optimal way, but GAC4 handles p-ary relations.

The process is founded on a recursive label pruning. When a label a has to be removed from the set L_i of possible labels for i, all the p-uples including the label have to be discarded. When a p-uple is discarded from a constraint R, it may happen that this p-uple was the last element of R which was supporting a particular label. So this label has to disappear at his turn, and so on.

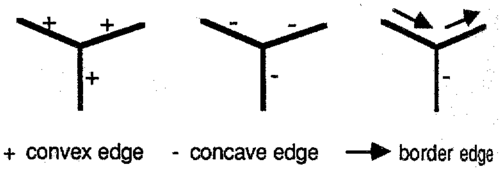


Figure 2: some possible labellings for a fork joint.

The correctness proof of this algorithm is obvious. It terminates because the number of labels and p-uples is finite. By simple induction, it appears that each time a label is removed, it cannot be part of a solution. By induction again, we state that a p-uple is discarded because one of the labellings it includes cannot belong to a solution. So the removed labels cannot belong to any solution, and the algorithm stops when it gets to a locally consistent solution. Here is an iterative version of this algorithm:

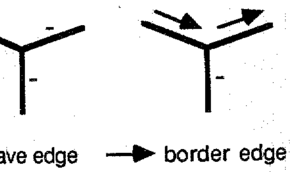
Let w_list be the list of labellings (i,a) that are discarded but not yet processed. Let $S_{i,a,R}$ be the set of p-uples from constraint R which include the labelling (i,a).

while $w\_list \neq \emptyset$ **do**
  choose one (i,a) in w_list ;
  remove (i,a) from w_list ;
  for each hyper-edge R supporting (i,a) **do**
    for each p-uple $P \in S_{i,a,R}$ **do**
      for each $(j,b) \in P$ **do**
        remove P from $S_{j,b,R}$;
        if $S_{j,b,R} = \emptyset$ **and** $j \in L_b$
        then $w\_list \leftarrow w\_list \cup (j,b)$;
          remove b from L_j ;

The most inner loop consists essentially in performing one removal of one element of a $S_{j,b,R}$ linked with the considered p-uple. This one then disappears. The complexity of GAC4 is $O(K)$ if all elementary operations can be performed in $O(1)$ and if the complexity of building the initial data structure is also at most $O(K)$. This is essentially a problem of implementation.

2.3 Implementing GAC4

The only problem in the previous loop is related to the removal of a p-uple P. We need a direct access from P



le labellings for a fork joint.

the p-uples including the label
nen a p-uple is discarded from a
pen that this p-uple was the last
upporting a particular label. So
ar at his turn, and so on.
of this algorithm is obvious. It
number of labels and p-uples is
tion, it appears that each time
nnot be part of a solution. By
e that a p-uple is discarded be-
gs it includes cannot belong to a
labels cannot belong to any so-
n stops when it gets to a locally
e is an iterative version of this

of labellings (i, a) that are dis-
essed. Let $S_{i,a,R}$ be the set of p-
which include the labelling (i, a).

n w_list ;
n w_list ;
dge R supporting (i, a) **do**
le $P \in S_{i,a,R}$ **do**
$, b) \in P$ **do**
P from $S_{j,b,R}$;
$= \emptyset$ **and** $j \in L_b$
$list \leftarrow w\_list \cup (j, b)$;
move b from L_j ;

consists essentially in performing
nent of a $S_{j,b,R}$ linked with the
one then disappears. The com-
) if all elementary operations can
nd if the complexity of building
e is also at most $O(K)$. This is
implementation.

GAC4

e previous loop is related to the
We need a direct access from P

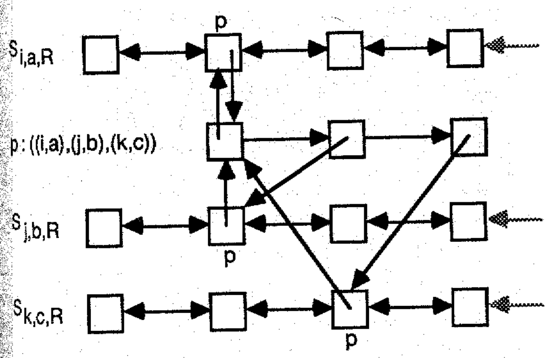


Figure 3: a data structure for $S_{i,a,R}$.

to all the $S_{i,a,R}$ which contain P. When we get it, a simple data structure, illustrated by fig. 3, allows then to delete the node corresponding to P in $S_{i,a,R}$. All $S_{i,a,R}$ are double-linked lists where each node designates one p-uple. A p-uple $P \in R$ is then just a list of p links to each $S_{i,a,R}$ such as (i, a) occurs in P. More precisely, these links designate in $S_{i,a,R}$ the element which is linked to the p-uple. This double linking allows easy removals in $S_{i,a,R}$. In the example given by fig. 3, the p-uple P has three elements (i, a), (j, b) and (k, c). The three lists $S_{i,a,R}$, $S_{j,b,R}$ and $S_{k,c,R}$ have a link pointing to P. Each element of P is only a pointer to the corresponding $S_{i,a,R}$, more precisely to the element which contains the reverse link.

Building the initial data has also to be performed in $O(K)$. Let us suppose that the input data is the list of admissible labels for all the hyper-edges R, and that the initial possible labels as well as all the hyper-edges are known. The initial step is then:

for each node i **do**
  initialize L_i with the set of a priori admissible labels ;
  for each hyper-edge R **do**
    for each i constrained by R **do**
      for each $a \in L_i$ **do**
        $S_{i,a,R} \leftarrow empty\_list$;
      for each p-uple P in the input **do**
        let R be the corresponding hyper-edge ;
        for each $(j, b) \in P$ **do**
          insert a new node N in $S_{j,b,R}$;
          link N to P ;
          link P to the predecessor of N ;

$w\_list \leftarrow empty\_list$;
for each (i, a, R) **do**
  if $S_{i,a,R} = empty\_list$
    then $w\_list \leftarrow w\_list \cup (i, a)$;
      remove a from L_i ;

This step can even be optimized if all p-uples from each hyper-edge R are grouped together in the input. After a given R has been processed, all labels which were not supported by R can already be deleted. So if a p-uple from an other constraint R' includes this label, it has not to be taken into account. This optimization reduces further computation and saves memory space.

3 Path Consistency

Montanari has introduced a path consistency condition restrained to binary constraints [Mon74]. This condition is stronger than local consistency built by AC4, but is not yet consistency.

A labelling is *path consistent* iff:

$$\forall i, j \in N, \forall a \in L_i, \forall b \in L_j,\ R_{i,j}(a, b) \text{ holds iff } \forall k \in N, \\ \exists\, c \in L_k \text{ such that } R_{i,k}(a, c) \text{ and } R_{k,j}(c, b) \text{ hold}$$

Fig. 1 provides an example of an arc consistent network which is not path consistent. In the case of complete networks where each edge is constrainted by each other edge, Montanari has proved that each path could be labelled in a consistent way if the path consistency condition was satisfied. We have presented a path consistency algorithm that runs in $O(n^3m^3)$ in the worst case [MH86]. Path consistency condition is essentially a condition on triplets of nodes. So it can be handled by GAC4, after a simple transformation.

Let us build the network N' associated to N where each label is a couple (i, j) of nodes from N. The possible labellings are the pairs (a, b) with $a \in L_i$ and $b \in L_j$. The mapping from N to N' has obviously an inverse. On this new network, we introduce the constraints R' on each triplet of nodes using the constraint R from the initial network:

$$R'_{(i,j),(j,k),(k,i)}((a, b), (b, c), (c, a)) \text{ iff} \\ R_{i,j}(a, b) \text{ and } R_{j,k}(b, c) \text{ and } R_{k,i}(c, a)$$

Then, we can prove that N is path consistent iff N' is arc consistent for this ternary relation [Moh87]. As the construction of N' is reversible, it is easy to show by induction that pruning the labellings of N' removes only inconsistent labels from N. Therefore running GAC4 on N' provides the correct result.

The complexity of this path consistency algorithm is linear in regard to the number of admissible triplets for this new relation R. There are n^3 constraints that have

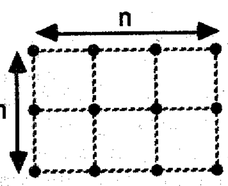


Figure 4: shape of the test networks.

at most m^3 admissible triplets: the complexity is still $O(n^3m^3)$. PC3 runs also in $O(n^3m^3)$, but we cannot derive that it is optimal on the basis of the previous result and the fact that GAC4 is optimal. Path consistency does not lead to the general case of ternary relations for which GAC4 would be optimal. So *is PC3 optimal?* remains an open question. The only lower bound known for path consistency is $O(n^2m^2)$.

4 Experiments with GAC4

4.1 Randomly generated networks

A system for running GAC4, written in Le_Lisp and compiled, has been installed on a Sun 3-260 workstation in order to test the speed of the algorithm and the quality of the results for large networks. These ones were shaped like rectangular grids of size $m \times n$ (fig. 4). Each node was provided with a set of 10 possible labels. Each horizontal and vertical pair of nodes was associated with a fixed number of binary constraints that were randomly generated. A constraint states the compatibility between some label of a node of the pair and some label of the other node. A special extra constraint was forced so that at least one global solution was always available.

Some results are displayed in tables 1 and 2. Each test has been performed on a grid of a particuliar size which is indicated by the first column of the tables. A test includes two parts that are noted *ar2* and *ar4* in the headers of the tables. The first one consists in running GAC4 using the binary constraints while the second one consists in running GAC4 using 4-ary constraints. These constraints are computed by merging the binary constraints associated with each group of 4 connex edges constituting a square in the grid. The second column of a table gives the number of global solutions, i.e. the number of labellings that are path consistent. It is computed from the results of GAC4 by a simple depth-first search algorithm.

Table 1 shows the results of the experiments for networks with low ambiguity (10 2-uples randomly generated for each edge): the number of global solutions is generally

m×n	global labellings	initialization step				propagation step				CPU total time (seconds)	
		mean nb p-uples		CPU time (seconds)		labels left		CPU time (seconds)			
		ar2	ar4	ar2	ar4	ar2	ar4	ar2	ar4	ar2	ar4
2×90	72	3	2	0.190	0.206	187	187	0.042	0.024	0.232	0.230
2×91	4	3	2	0.196	0.204	184	184	0.046	0.024	0.242	0.228
2×92	8	3	2	0.208	0.210	188	188	0.048	0.024	0.256	0.234
2×93	12	3	2	0.194	0.212	191	191	0.044	0.026	0.238	0.238
2×94	1	2	2	0.192	0.208	188	188	0.042	0.024	0.234	0.232
2×95	16	3	2	0.204	0.216	195	195	0.044	0.024	0.248	0.240
2×96	12	3	2	0.216	0.216	196	196	0.046	0.024	0.262	0.240

Table 1: GAC4 CPU processing times for networks with low ambiguity (10 2-uples per edge).

m×n	global labellings	initialization step				propagation step				CPU total time (seconds)	
		mean nb p-uples		CPU time (seconds)		labels left		CPU time (seconds)			
		ar2	ar4	ar2	ar4	ar2	ar4	ar2	ar4	ar2	ar4
2×7	67	17	17	0.048	0.096	67	40	0.006	0.004	0.054	0.100
2×8	24	16	18	0.052	0.118	95	30	0.004	0.004	0.056	0.122
2×9	110	15	22	0.060	0.136	109	56	0.006	0.004	0.066	0.140
2×10	108	16	18	0.068	0.128	45	40	0.014	0.004	0.082	0.132
2×11	72	15	18	0.068	0.146	62	48	0.012	0.006	0.080	0.152
2×12	144	15	18	0.076	0.150	103	49	0.012	0.004	0.088	0.154
2×13	18	15	19	0.084	0.192	56	50	0.018	0.006	0.102	0.198
2×14	480	17	19	0.098	0.210	118	47	0.014	0.008	0.112	0.218
2×15	1092	16	19	0.102	0.222	92	72	0.018	0.006	0.120	0.228
2×16	1080	16	20	0.124	0.234	123	62	0.018	0.009	0.142	0.246
3×40	216	15	17	0.428	0.974	193	130	0.100	0.028	0.528	1.002
3×41	25088	15	17	0.430	0.940	177	151	0.114	0.026	0.544	0.966
3×42	512	15	16	0.448	1.026	137	137	0.120	0.034	0.568	1.060
3×43	224	15	17	0.480	1.068	139	139	0.120	0.036	0.600	1.104
3×44	12	14	16	0.460	1.044	151	138	0.110	0.032	0.570	1.076
3×45	256	15	18	0.484	1.176	195	150	0.108	0.034	0.592	1.210
3×46	138240	15	17	0.512	1.150	168	167	0.130	0.036	0.642	1.186
3×47	7680	15	17	0.508	1.186	165	164	0.120	0.032	0.628	1.218
3×48	3000	15	18	0.520	1.168	168	168	0.122	0.032	0.642	1.200
3×49	1920	15	17	0.526	1.186	165	165	0.144	0.034	0.670	1.220
3×50	13440	15	17	0.536	1.162	203	172	0.124	0.030	0.660	1.192
4×4	12	20	17	0.645	1.125	22	22	0.107	0.040	0.752	1.165
4×5	135	20	18	0.489	1.505	41	41	0.399	0.052	0.888	1.557
4×6	4	20	20	0.939	2.148	55	34	0.156	0.077	1.195	2.225
4×7	24	20	18	1.003	2.288	45	44	0.186	0.076	1.189	2.364
4×8	18	20	19	1.436	2.633	43	43	0.223	0.064	1.659	2.697
4×9	60	20	21	1.814	3.265	52	50	0.243	0.107	2.057	3.372
4×10	6	20	17	1.937	3.281	45	45	0.584	0.088	2.521	3.369
4×11	18	20	18	2.286	3.965	70	61	0.284	0.107	2.570	4.072
4×12	30	20	17	2.824	4.110	56	56	0.657	0.111	3.481	4.221

Table 2: GAC4 CPU processing times for networks with high ambiguity (20 2-uples par edge).

small. We can see that computation times and numbers of remaining labels are equal. Moreover, the solutions supplied by the propagation are almost the global consistent solutions: for instance, 184 remaining labels for a 2×91 grid with 4 global labellings. In this case, AC4 or GAC4 have the same efficiency.

The results obtained with networks with high ambiguity (20 2-uples for each edge) are displayed in table 2. The number of remaining labels are smaller when dealing with 4-ary constraints but the corresponding running times are two times higher. However the most part of the computation is taken by the initialization step, especially for generating the 4-ary constraints from the binary ones. As real applications are the most often related to problems involving p-ary constraints for which this initialization step is not necessary, it appears that GAC4 is suitable to these cases. Its propagation times are three to six times lower than AC4 ones.

propagation step				CPU total time (seconds)	
labels left		CPU time (seconds)			
ar2	ar4	ar2	ar4	ar2	ar4
187	187	0.042	0.024	0.232	0.230
184	184	0.046	0.024	0.242	0.228
188	188	0.048	0.024	0.256	0.234
191	191	0.044	0.026	0.238	0.238
188	188	0.042	0.024	0.234	0.232
195	195	0.044	0.024	0.248	0.240
196	196	0.046	0.024	0.262	0.240

ssing times for networks with
er edge).

propagation step				CPU total time (seconds)	
labels left		CPU time (seconds)			
ar2	ar4	ar2	ar4	ar2	ar4
67	40	0.006	0.004	0.054	0.100
95	30	0.004	0.004	0.056	0.122
109	56	0.006	0.004	0.066	0.140
45	40	0.014	0.004	0.082	0.132
62	48	0.012	0.006	0.080	0.152
103	49	0.012	0.004	0.088	0.154
56	50	0.018	0.006	0.102	0.198
118	47	0.014	0.008	0.112	0.218
92	72	0.018	0.006	0.120	0.228
123	62	0.018	0.009	0.142	0.246
193	130	0.100	0.028	0.528	1.002
177	151	0.114	0.026	0.544	0.966
137	137	0.120	0.034	0.568	1.060
139	139	0.120	0.036	0.600	1.104
151	138	0.110	0.032	0.570	1.076
195	150	0.108	0.034	0.592	1.210
168	167	0.130	0.036	0.642	1.186
165	164	0.120	0.032	0.628	1.218
168	168	0.122	0.032	0.642	1.200
165	165	0.144	0.034	0.670	1.220
203	172	0.124	0.030	0.660	1.192
22	22	0.107	0.040	0.752	1.165
41	41	0.399	0.052	0.888	1.557
55	34	0.156	0.077	1.195	2.225
45	44	0.186	0.076	1.189	2.364
43	43	0.223	0.064	1.659	2.697
52	50	0.243	0.107	2.057	3.372
45	45	0.584	0.088	2.521	3.369
70	61	0.284	0.107	2.570	4.072
56	56	0.657	0.111	3.481	4.221

essing times for networks with
par edge).

nputation times and numbers
ual. Moreover, the solutions
on are almost the global con-
nce, 184 remaining labels for a
bellings. In this case, AC4 or
iency.
h networks with high ambigu-
dge) are displayed in table 2.
labels are smaller when deal-
but the corresponding running
er. However the most part of
by the initialization step, espe-
ary constraints from the binary
are the most often related to
constraints for which this ini-
ssary, it appears that GAC4 is
s propagation times are three
C4 ones.

4.2 Application to Scene Interpretation

TRIDENT is a system that was developed as an environment for testing strategies in the frame of generic three-dimensional scene analysis [MMT85]. Scenes are artificial 3D data constructed with the help of a 3D modeler. They are segmented into surfaces [MM84] which represent the input of the interpretation process.

The a priori knowledge about the scenes to be recognized is given through a hierarchical model that describes the objects by means of their constituents. Each possible shape of an object is given by a rule associated with binary constraints on the attributes of the objects used in the rule. The attributes represent the dimensions, coordinates, colours, and so on, of the objects. Here is an example of rule for which only a set of representative constraints is displayed:

Rule # 7 : *DiningRoom* → *Chair** *Table* *Shelves* *Lamp* *Picture**

$z_{min}(Table) < z_{max}(Chair)$

colour(*Lamp*) ∈ (*white*,*yellow*,*orange*)

height(*Picture*) ≠ width(*Picture*)

...

The domains of all the attributes are predefined ranges of discrete values. The step of discretization is chosen so that the memory space for storing the domains is not tremendous.

The interpretation process is inspired from syntactic analysis techniques and it mixes both of the bottom-up and top-down approaches. However, the process is much looser than a syntactical analyzer, for it has to deal with noisy and occluded shapes. In fact, several analysis run concurrently from different confidence islands. They build partial descriptions (trees) of the scene that are progressively merged together.

All along the interpretation, the consistency of the partial descriptions is checked by running AC4. A network represents the constraints introduced by the rules used to build up the partial descriptions. Each node is a particuliar attribute of a particuliar object, and it is labelled with the potential values the attribute can take. The edges represent the relations introduced by the constraints. AC4 is performed each time new constraints are added to the network. While successive relaxations discard inconsistent values, the interpretation process is provided with more and more precise information about the domains of values of the attributes. This makes it easy to predict the accurate shapes of the objects that are not yet identified. When a constraint involving an object proves to be inconsistent with the other ones, the interpretation process eliminates the object from its sets of hypothesis.

5 Improvements

5.1 Dealing with Errors

If all labels disappear by mistake at some nodes and if the network is connex, the pruning step will discard all the labels of the other nodes. This event happens frequently in applications related to domains such as computer vision where some noisy nodes have no interpretation. A solution would be to allow a special extra label ω to be admissible with any other label of the connected nodes. Noisy nodes can therefore be labelled with ω without destroying the whole network.

Unfortunately, this is not a solution: if each node had such a label ω, the pruning would not discard any label at all, for each label would at least be admissible with ω. The true solution consists in making looser the condition that specifies that all the constraints have to be satisfied. Of course, the new condition depends on the problem and on the kind of constraints. For example, it would be something like that: constraint R_1 must always be satisfied, and a $\frac{2}{3}$ ratio of the set of constraints including each label has to be satisfied. Dealing with such a condition, or any kind of its variations, requires only a slight modification of GAC4 (or AC4 if only binary constraints are used). Instead of considering each individual constraint involved by each labelling (i, a), the whole set of constraints is examined. For instance, if we wish to have a $\frac{2}{3}$ ratio of the constraints to be satisfied, it requires only a counter indicating how many constraints are still satisfied for each (i, a). Each time a constraint is no longer satisfied, this counter is decremented. When it falls below the fixed threshold, labelling (i, a) has to disappear.

This way of taking into account weaker conditions is not equivalent to continuous relaxation [HZ80] [FB81] that deals with probabilist labels and is therefore more powerful. However, it has to be mentioned that the convergence process is hard to control in continuous relaxation and leads sometimes to surprising results. On the contrary, with the extension of GAC4, the user gets a full control of what is locally acceptable by weakening the constraints. A label is then discarded only if these weaker conditions are no longer satisfied.

5.2 Parallelization

GAC4 can run on parallel processors on condition that conflicts are avoided when updating the data structure. The easiest way to solve the problem consists in ensuring that at most one hyper-edge is taken into account by each processor at each time. If a particuliar hyper-edge is handled by several processors at the same time, a test of mutual exclusion has to be performed before pruning any label and before deleting any element of a $S_{i,a,R}$.

However, it is possible to build networks for which the speed of arc consistency is not increased by running on a parallel computer. In such networks, any deletion of a label removes only one label among the neighbouring nodes. The behaviour of the algorithm is then essentially sequential. It has to be noticed that the potential parallelism rate was high in all the examples shown in section 4.1. At each step, the stack was containing several tens of removed labels that could have been processed in parallel.

6 Conclusion

The complexity of GAC4 has been evaluated by taking into account the number of positive constraints, i.e. the number of admissible configurations on each edge or hyper-edge. This is a good evaluation scheme because in many applications, like the block world [Wal75] or the chaining of amino acid in peptid synthesis [Vil87], the number of positive constraints is strongly lower than all the possible grouping of labels. GAC4 has a time complexity linear in the size of these positive constraints. Without changing its complexity, it can deal with unsatisfied constraints.

Experiments with GAC4 and its binary version AC4 show that their run times are very low. Randomly generated data emphasizes that both of them find an arc consistent solution which is often the consistent solution, when networks have low ambiguity. So it is not necessary to use a more complex consistency condition like path consistency. Path consistency solution can be computed using GAC4 but, in this case, space and time complexity are cubic in the size of the initial network. The algorithm remains unusable for large networks.

Building larger p-ary constraints by combining binary constraints and running GAC4 on the resulting network requires almost as much time as running directly AC4 on the initial network. The results are also quite similar when the network is not ambiguous. However the pruning performed GAC4 on the new network is better when the initial network is largely ambiguous.

Acknowledgements

This work was partially founded by P.R.C. Vision 1987. We would also like to thank Marie-Christine Vilarem for her example of peptid synthesis and Eric Thirion for his example of scene analysis.

References

[FB81] O. Faugeras and M. Berthod. Improving Consistency and Reducing Ambiguity in Stochastic Labelling: an Optimization Approach. *IEEE Trans. on PAMI*, (4):412–423, 1981.

[HZ80] A. Hummel and W. Zucker. On The Foundations of Relaxation Labelling Processes. In *Proc. 5th IJCPR*, pages 50–53, Miami, Florida, 1980.

[MF85] A.K. Mackwoth and E.C. Freuder. The Complexity of Some Polynomial Network Consistency Algorithms for the Constraint Satisfaction Problems. *Artificial Intelligence*, (25):65–74, 1985.

[MH86] R. Mohr and T.C. Henderson. Arc and Path Consistency Revisited. *Artificial Intelligence*, (28):225–233, 1986.

[MM84] Y. Muller and R. Mohr. Planes and Quadrics Detection Using Hough Transform. In *Proc. 7th IJCPR*, pages 1101–1103, Montreal, 1984.

[MMT85] G. Masini, R. Mohr, and E. Thirion. Stratégie de perception pour un modèle hiérarchique. In *Proc. 5ème Congrès AFCET RF-IA*, pages 631–640, Grenoble, 1985.

[Moh87] R. Mohr. *A Correct Path Consistency Algorithm and an Optimal Generalized Arc Consistency Algorithm*. Technical Report 87-R-030, CRIN, Vandœuvre, France, 1987.

[Mon74] U. Montanari. Networks of Constraints: Fundamental Properties and Application to Pictures. *Information Sciences*, (7):95–132, 1974.

[Vil87] M.C. Vilarem. *CINTIA : un système expert en synthèse peptidique*. Technical Report, CRIM, Université de Montpellier, 1987.

[Wal75] D. Waltz. Understanding Line Drawings of Scenes with Shadows. in *The Psychology of Computer Vision*, P.H. Winston editor, McGraw-Hill, New York, 1975.