Computer Science Department Technical Report
Cognitive Systems Laboratory
University of California
Los Angeles, CA 90024-1596

EXPERIMENTAL EVALUATION OF PREPROCESSING TECHNIQUES IN CONSTRAINT SATISFACTION PROBLEMS

Rina Dechter
Itay Meiri

June 1989
CSD-890033

To appear in Proceedings, Intl. Joint Conference on Artifical Intelligence (IJCAI), Detroit, August 1989

TECHNICAL REPORT
R-121
May 1989

EXPERIMENTAL EVALUATION OF PREPROCESSING TECHNIQUES IN CONSTRAINT SATISFACTION PROBLEMS*

Rina Dechter

Computer Science Department
Technion - Israel Institute of Technology
Haifa 32000, Israel

Itay Meiri

Cognitive Systems Laboratory
Computer Science Department
University of California, Los Angeles, CA 90024

Abstract

This paper presents an evaluation of two orthogonal schemes for improving the efficiency of solving constraint satisfaction problems (CSPs). The first scheme involves a class of pre-processing techniques designed to make the representation of the CSP more explicit, including directional-arc-consistency, directional-path-consistency and adaptive-consistency. The second scheme aims at improving the order in which variables are chosen for evaluation during the search. In the first part of the experiment we tested the performance of backtracking (and its common enhancement -- backjumping) with and without each of the preprocessings techniques above. The results show that directional arc-consistency, a scheme which embodies the simplest form of constraint recording, outperforms all other preprocessing techniques. The results of the second part of the experiment suggest that the best variable ordering is achieved by the fixed **max-cardinality** search order.

1. Introduction

In this paper we report the results of two sets of experiments designed to evaluate several constraint-satisfaction algorithms. The first set is concerned with a class of **pre-processing algorithms** which transform a given constraint network into a more explicit representation before it is subjected to a backtracking algorithm for solution. The three pre-processing algorithms tested were Directional-Arc-Consistency (DAC), Directional-Path-Consistency (DPC), and Adaptive-Consistency (ADAPT), presented in [Dechter 1987]. These are variations of the well-known constraint-propagation (also called "relaxation" or "consistency-enforcing") techniques Arc-Consistency and Path-Consistency, originally advocated by [Montanari 1974], [Waltz 1975], and [Mackworth 1977], and their extension, *i*-Consistency [Freuder 1982]. The preprocessing performed by each of these consistency-enforcing algorithms amounts to **constraint recording**, i.e., the explicit recording of implicit constraints, with the aim of reducing the amount of post-processing required by the backtracking algorithm. They differ in the type and amount of implicit constraints they choose to record. The advantage of algorithms DAC, DPC, and ADAPT is that they take into account the direction in which backtracking will eventually search the problem and, as a result, they avoid processing many constraints which are unnecessary for the search and which would have been processed and recorded by the earlier consistency-enforcing algorithms.

Worst-case analysis fails to reveal the merits and drawbacks of preprocessing techniques like those described above. With the exception of ADAPT, all the consistency-enforcing algorithms mentioned are of polynomial complexity and, therefore, negligible when compared to the exponential worst-case behavior of backtracking search. ADAPT records a set of constraints sufficient to guarantee a backtrack-free search (i.e., linear post-processing complexity). Its complexity is exponential in W^* (a network parameter to be defined later, where $W^* \leq n$), which is still bounded, in the worst case, by that of unprocessed backtracking.

Thus, worst-case analysis suggests that there is "nothing to lose" and "everything to gain" by applying preprocessing before backtracking. Unfortunately, worse-case analysis does not necessarily correlate with average case performance and, in practice, straightforward backtracking often performs best without the assistance of any consistency-enforcing algorithm.

In order to shed some light on the practical utility of using algorithms DAC, DPC, and ADAPT, we conducted experiments comparing their performance to that of backtracking and its advanced version, backjumping [Dechter 1989], on a set of randomly generated CSPs. The results, in general, show that the average complexity of solving

*This work was supported in part by the National Science Foundation, Grant #IRI-8815522 and by the Air Force Office of Scientific Research, Grant #AFOSR-88-0177.

these problems by backtracking is far from exponential and, thus, the pre-processing performed by ADAPT and sometimes even by DPC, are too expensive, unless the graph is very sparse. Algorithm DAC, on the other hand, comes out as a clear winner. Apparently, it performs the right amount of pre-processing for helping backtracking and, in most cases, it even outperforms backjumping, previously shown to be a very effective dialect of backtracking [Dechter appear].

The second set of experiments compares the performance of backtracking and backjumping under different orderings of variables. We tested three fixed heuristics for ordering: min-width, max-degree, and max-cardinality search, and one **dynamic** ordering named dynamic search rearangement [Purdom 1983]. The results suggest that the fixed max-cardinality ordering results in the smallest average time, while dynamic ordering is most effective in pruning the search space.

Section 2 presents the constraint network model and reviews algorithms DAC, DPC and ADAPT. Section 3 discusses different ordering heuristics. Section 4 describes the methodology of the experiments and presents and analyzes results pertaining to the merits of pre-processing techniques and ordering heuristics. Section 5 provides a summary and conclusions.

2. Directional Pre-processing Algorithms

A constraint network (CN) involves a set of n variables, $X_1, \ldots, X_n$, their respective **domains**, $R_1, \ldots, R_n$, and a set of **constraints**. A constraint $C_i(X_{i_1}, \cdots, X_{i_j})$ is a subset of the Cartesian product $R_{i_1} \times \cdots \times R_{i_j}$ that specifies which values of the variables are compatible with each other. A binary constraint network is one in which all the constraints are binary, i.e., involving at most two variables. A CN may be associated with a **constraint-graph** in which nodes represent variables and arcs connect those pairs of variables which appear in the same constraints. For instance, the CN presented in Figure 1a, describes the problem of coloring the nodes in a graph s.t. adjacent nodes have different colors. This is a binary CN whose variables are the nodes and whose values are the possible colors. A link represents the set of value-pairs permitted by the constraint between the variables it connects

A **solution** of a constraint network is an assignment of values to all the variables such that all the constraints are satisfied. The classical constraint satisfaction tasks are to find one or all solutions.

The following paragraphs present the tested algorithms. We start with adaptive-consistency, then generalize its operation to describe a class of pre-processing algorithms which include DAC and DPC as special cases. Few more definitions are needed for this discussion. Given an ordering of the variables in a CN, for each variable, X, PARENTS(X) is the set of all variables connected to it and preceding it in the graph. The **width of a node** is the number of predecessors linked to that node. The **width of an ordering** is the maximum width of nodes in that ordering, and the **width of a graph** is the minimal width of all its orderings. For instance, given the ordering (E,D,C,A,B) of the graph in Figure 1a, the width of node B is 1 while the width of this ordering is 2 and so is the width of this graph (see Figure 1b). Adaptive-consistency (i.e., ADAPT) process the nodes in reverse order, i.e., each node is processed before any of its parents.

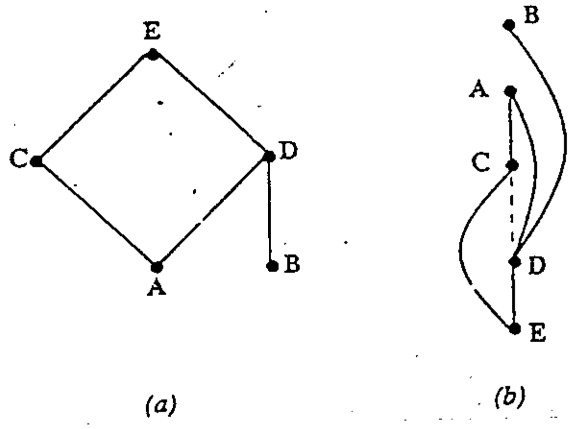

Figure 1: An example of a binary CN

```
adaptive-consistency($X_1, \ldots, X_n$)

Begin
1. for i=n to 1 by -1 do
2. Compute PARENTS($X_i$)
3. perform record-constraint($X_i$, PARENTS($X_i$))
4. connect all elements in PARENTS($X_i$)
(if they are not yet connected)
End
```

The procedure **record-constraint(*V*,SET)** generates and records those tuples of variables in SET that can be consistent with at least one value of V. For instance if, in our example, A has only the color green in its domain and C and D each have two possible colors, {red,green}, then the call for **record-constraint**(A, {C,D}) will result in recording a constraint on the variables C,D, allowing only the pair (red,red) in their relation. ADAPT may tighten existing constraints as well as impose constraints over clusters of variables. It was shown [Dechter 1987], that when the procedure terminates backtrack can solve the problem, in the order prescribed, without encountering any dead-end. The topology of the new **induced graph** can be found prior to executing the procedure, by recursively connecting any two parents sharing a common successor.

Consider our example of Figure 1a in the ordering (E,D,C,A,B) shown in Figure 1b. B is chosen first and since it has only one parent, D, the algorithm records a unary constraint on D's domain, eliminating any of its values which doesn't have a consistent match in B. Variable A is processed next and a binary constraint is enforced on its two parents D and C, eliminating all pairs which have no common consistent match in A. This operation may require that a constraint arc be added between C and D. The algorithm proceeds in the same manner through the rest of the variables, taking into account both old and the newly generated constraints. The resulting graph, called **induced graph**, contains the dashed arc in Figure 1b.

Let $W(d)$ be the width of the ordering d and $W^*(d)$ the width of the induced graph along this ordering. The complexity of **record-constraint**(V, PARENT(V)) (step 3) is exponential in the cardinality of V and its parents. Since the maximal size of the parent-sets is equal to the width of the induced graph and since this step dominates the whole computation, solving the problem along the ordering d is $O(n \cdot \exp(W^*(d)+1))$ [Dechter 1987].

The directional algorithms (i.e., DAC, DPC, and directional-i-consistency), differ from ADAPT only in the amount and size of constraint recording performed in step 4. Namely, instead of recording one constraint among all the parent set, they record a few, smaller constraints on subsets of the parents. Let *level* be a parameter indicating the utmost cardinality of constraints which are recorded. The class of algorithms *adaptive* (*level*) is henceforth described:

```
adaptive(level, $X_1, \ldots, X_n$)
Begin
1. for i=n to 1 by -1 do
2. Compute PARENTS($X_i$)
3. perform new-record( level, $X_i$, PARENTS($X_i$))
4. for level$\geq$2, connect all elements in PARENTS($X_i$)
(if they are not yet connected)
End
```

Procedure new-record(*level*, var, set) records only constraints of size *level* from subsets of *set* and is defined as follows:

```
new-record(level, var, set)
Begin
1. if level $\leq$ |set| then
2.    for every subset S in set, s.t |S| = level do
3.       record-constraint(var,S)
4.    end
5. else do record-constraint(var,set)
end
```

Adaptive(*level* =1) reduces to DAC while for *level* = 2 it becomes DPC. The graph induced by all these algorithms, excluding the case of *level* =1 where the graph doesn't change, has the same structure as the one generated by adaptive-consistency. clearly, adaptive(*level* = $W^*(d)$) is the same as adaptive-consistency, and thus, is guaranteed to generate a backtrack-free solution.

The complexity of adaptive(*level*) is both time and space dominated by the procedure new-record(*level*) which is $O(\binom{W^*(d)}{level} \cdot (k^{\min\{level, W^*(d)\}}))$. This bound can be tightened if the ordering d results in a smaller $W^*(d)$. However, finding the ordering which has the minimum induced width is an NP-complete problem.

Since backtracking and backjumping are used as post-processing algorithms they also deserve a word. Backtracking consistently assigns values to variables until either a solution is found or there is a deadend (i.e., a variable has no value consistent with previous values). In that case backtracking **goes back** to the most recent instantiation, changes it and continues. Backjumping improves the "go-back" phase of backtracking and whenever a dead-end occurs at variable X, it backs up to the most recent variable **connected to** X in the constraint graph. This is a graph-based variant of Gaschnig's backjumping [Gaschnig 1979], and it was shown [Dechter appear] that it outperforms backtracking on an instance by instance basis.

3. The Effects of Variable Ordering

It is well known that the ordering of variables, be it fixed throughout search, or **dynamic**, may have a tremendous influence on the size of the search space explored by backtracking algorithms. Finding an ordering which would minimize the search space is a difficult problem and, consequently, researchers have concentrated on devising heuristics for variable ordering. The best known dynamic ordering is the **dynamic search rearangement**, which was investigated analytically via average case analysis in [Purdom 1983, Haralick 1980, Nudel 1983] and experimentally in [Stone 1986, Rosiers 1986]. This heuristic selects as the next variable to be instantiated a variable that has a minimal number of values which are consistent with the current partial solution. Heuristically, the choice of such variable minimizes the remaining search. Deeper estimates of the remaining search space were also considered [Purdom 1981, Zabih 1988].

We consider three heuristics for fixed ordering of variables: the **minimum width**, the **maximum degree**, and the **maximum cardinality** heuristics. The minimum width heuristic [Freuder 1982], orders the variables from last to first by selecting, at each stage, a node in the constraint graph which has a minimal degree in the graph remaining after deleting from the graph all nodes which have been selected already. As its name indicates, the heuristic results in a minimum width ordering. The max-degree heuristic orders the variables in a decreasing order of their degrees in the constraint graph. This heuristic also aims at (but does not guarantee) finding a minimum-width ordering.

The third heuristic is the **max-cardinality search ordering**. This ordering selects the first variable arbitrarily, then, at each stage it appends to the selected variables one which is connected to the largest group among the variables already selected. This heuristic can be thought of as the fixed version of dynamic search rearangement: the next variable to be selected is the one which constraints with the largest number of already instantiated variables, namely it is the most constrained variable.

4. Experimental Results

We compared 26 algorithm combinations on our test problems. Algorithms Backtracking (BTK) and Backjumping (BJ) were executed on each problem without any pre-processing and after pre-processing them by either directional-arc-consistency, directional-path-consistency and adaptive-consistency (8 combinations). Each such combination was tested with each one of the fixed ordering heuristics, max-degree, max-cardinality search and min-width heuristic, (yielding 24 combinations). Two more runs of backtracking and backjumping were performed in conjunction with dynamic ordering.

The test problems were selected from a randomly generated CSPs. The random problems were created by generating random graphs and associating with each arc in the graph a randomly generated binary constraint. We purposely concentrated on parameters (e.g., probability of an arc) which result in **more difficult problems** for backtracking. We chose to restrict the set of test problems to binary CSPs because problems with constraints of higher order tend to have denser constraint graphs for which the pre-processing algorithms have higher overhead. It should be pointed out, however, that adaptive consistency will add to the network non-binary constraints so the implementation of the backtracking and backjumping algorithms had to accommodate general, non-binary, CSPs.

We experimented with two sets of random problems: one including 42 problem instances having 10 variables and 5 values and the other, including 35 problem instances, with 15 variables and 5 values. This set was selected from a much larger set of instances from which all the easy problems were deleted, and therefore, the given set represent the more difficult among such randomly generated problems. Problems of larger size took too much time and space for our machine to handle, especially for algorithm adaptive-consistency.

We recorded the number of consistency-checks and the number of dead-ends (number of backtrackings) in each run. The number of consistency-checks is considered a realistic measure of the overall performance, while the number of backtrackings is indicative of the size of the search space exposed.

Each algorithm combination were run twice on each problem instance, once for finding one solution and the other for finding all solutions. The results were clustered into 6 groups corresponding to the two problem sizes (either 10 or 15 variables) and the three cases for which statistics were recorded, namely, finding one solution (called "first"), finding all solutions (called "all"), and for the cases that no solutions exist (called "failure").

4.1 Evaluation of Pre-processing Algorithms

Our first interest is to compare the effect of the three pre-processing algorithms DAC, DPC and ADAPt with back-tracking and backjumping. Since their relative behaviour w.r.t. the three fixed ordering schemes was found to be quite similar (and in order to save space) we report, in this part, the results of running them using max-degree pre-ordering only.

Figure 2 presents graphs of the average number of consistency checks, classified according to the width of the induced graph W^*, for four of the six groups of instances. The results of the "first" and "failure" cases for problems size of 10 are omitted, for the sake of saving space, because they exhibit the same behavior as their counter parts of size 15. Each pair of graphs describes the results of one of the groups, where the one on the left contrasts the results for algorithms ADAPT, BTK and BJ and the one on the right shows (using a different scale) the results of algorithms BJ, DAC and DPC. The results reported for DAC, DPC, and ADAPT are for the cases were these algorithms were complemented by backjumping. The results when backtracking was used were very similar to those with backjumping since after the pre-processing most of the dead-ends were eliminated.

Comparing, first, ADAPT to BTK and BJ (left column in Figure 2) we see that even on the average, adaptive-consistency has an exponential behavior as a function of W*. BTK and BJ, on the other end, exhibit a much more moderate, maybe even linear, behavior.

The average performance of ADAPT is better than BTK only for small values of W^* and when the task is to find all solutions. Evidently, the amount of preparation performed by ADAPT is too heavy to be justified by just one solution (Figure 2e), but when it is divided among several solutions, it becomes worth-while (Figure 2c). For the case of n=10, when looking for one solution, BTK outperformed ADAPT even for $W^* = 1,2$ (not shown in Figure 2).

When compared to BJ, however, ADAPT appears as a complete looser. BJ outperformed ADAPT in every instance. Clearly, BJ exploits the structure of the problem in a more efficient way than ADAPT and should be preferred, especially considering the fact that it doesn't need the additional space which is consumed by ADAPT. (Although ADAPT does not seem to be a sensible choice for a one time solution of a CSP, it still can be used in order to find a better representation of a network of constraints, for example, when the network represents some knowledge-base on which many queries are to be answered over time. In such cases the work for generating the new representation can be ignored [Dechter 1988].)

The disappointing results of ADAPT can be explained by comparing it with the two other, less ambitious, pre-processing algorithms, DAC and DPC. When we counted the number of dead-ends left after pre-processing (not shown here), we found that in almost all problem instances

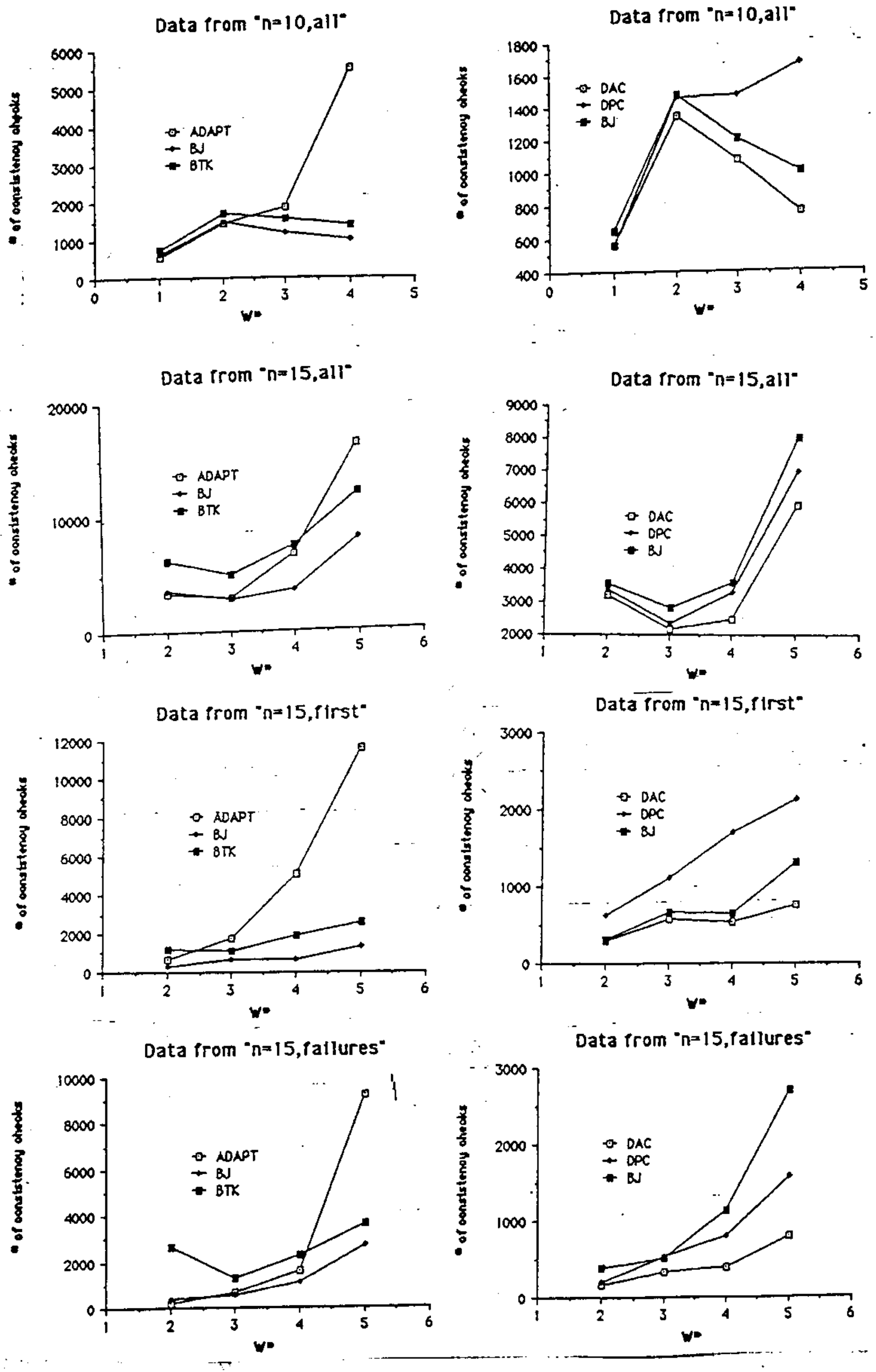

Figure 2: Effects of pre-processing algorithms

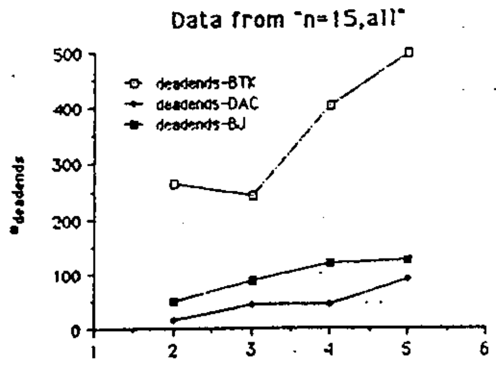

Figure 3: The effect of DAC on the search space (Case n=15)

algorithm DPC alone eliminated all future dead-ends. It is clear, therefore, that for problem instances of this type, ADAPT is doing unnecessary pre-processing. Moreover, the number of dead-ends left by DAC alone (see Figure 3) shows that most of the work is accomplished by this algorithm which performs the smallest amount of constraint recording.

In summary, the two algorithms that stand out in the experiments are BJ and DAC. Furthermore, the performance of DAC followed by BJ is better than executing just BJ. Among the other three, DPC comes next, and the relationship between BTK and ADAPT is dependent on W^*.

4.2 evaluation of the Effects of Variable Ordering

We now wish to focus on the effects of of the three fixed ordering and the dynamic ordering on various algorithm combinations, and in particular on backtracking and backjumping. Figure 4 presents the results of comparing backtracking and backjumping on (because of space limitations) four of the six groups and for each of the four ordering schemes: 1. the max-degree (**max**), 2. min-width ordering (**min**), 3. max-cardinality ordering (**card**), and 4. dynamic ordering (**dnmic**). In contrast with dynamic search rearangement, fixed ordering schemes were never evaluated experimentally. To maintain continuity we averaged the same set of instances, and therefore W* indicates the induced width of max-degree's ordering.

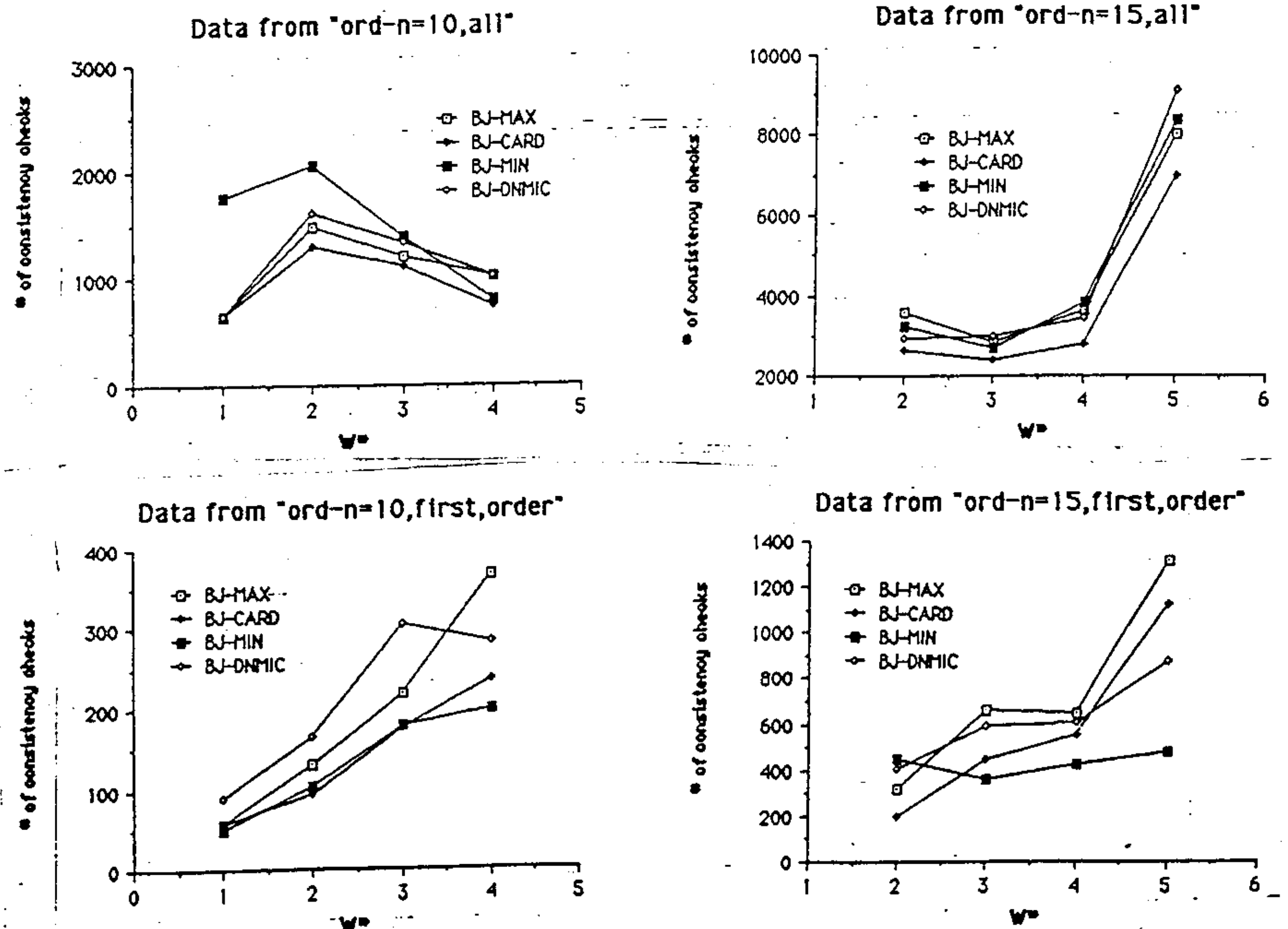

Figure 4: Effects of variable-ordering rules

in our experiments, the card ordering scheme gave the best results in most of the cases. It seems to outperform other fixed orderings and even the dynamic ordering. It is particularly clear for the task of finding all solutions, while for first solution min-width comes quite as good. However, when we compared the number of dead-ends associated with each ordering, it becomes clear that dynamic ordering is expanding the smallest search space (i.e., it has the least number of deadends on an instance by instance basis, almost (these results are not graphed here). Therefore, had we better implemented this technique we may have a better overall performance. Indeed in the current implementation no data-structure was used to alleviate redundant consistency checks as was done in other look-ahead schemes like Forward-Checking [Haralick 1980].

5. Summary and Conclusions

We evaluated the performance of several backtracking techniques for solving CSPs. First, we tested the effect of various pre-processing algorithms on backtracking and backjumping, under fixed ordering, and concluded that directional arc-consistency followed by backjumping yields the best improvement. Second, we tested the effect of four variable ordering schemes and concluded, quite convincingly that the max-cardinality order yields the least amount of computation. It is expected, therefore, that directional arc-consistency with backjumping on the max-cardinality ordering will yield the best results. Indeed, when we compared all algorithms (i.e., BTK, BJ, ADAPT, DAC, DPC) on all the three fixed orderings (i.e., MAX, MIN, CARD) the combination of DAC-CARD gave the best performance on three of the six groups tested and in the rest it came as a close second best after DAC-MIN. Averaging over the 6 groups of instances, DAC-CARD was best. This suggest that the best approach is to choose max-cardinality ordering and to perform directional arc-consistency followed by backjumping.

These results, however, should be qualified in two ways. First, the conclusions are valid only relative to problem domains with statistics similar to those used in generating our test samples. Second, the superior pruning power of dynamic ordering suggests that further performance improvements could be realized by a more sophisticated implementation of this technique.

References

[Dechter 1987]Dechter, R. and J. Pearl, "Network-based Heuristics for Constraint-Catisfaction Problems," *Artificial Intelligence*, Vol. 34, No. 1, 1987, pp. 1-38.

[Dechter 1988]Dechter, R. and J. Pearl, "Tree-clustering Schemes for Constraint-Processing," in *Proceedings AAAI-88*, St Poul, Minnesota: 1988.

[Dechter 1989]Dechter, R., "Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition," *Artificial Intelligence*, appear.

[Freuder 1982]Freuder, E.C., "A Sufficient Condition for Backtrack-free Search.," *Journal of the ACM*, Vol. 29, No. 1, 1982, pp. 24-32.

[Gaschnig 1979]Gaschnig, J., "Performance Measurement and Analysis of Certain Search Algorithms.," Carnegie-Mellon University, Pittsburg, Pennsylvania, Tech. Rep. CMU-CS-79-124, 1979.

[Haralick 1980]Haralick, R. M. and G. L. Elliott, "Increasing Tree-search Efficiency for Constraint Satisfaction Problems," *Artificial Intelligence*, Vol. 14, 1980, pp. 263-313.

[Mackworth 1977]Mackworth, A. K., "Consistency in Networks of Relations," *Artificial intelligence*, Vol. 8, No. 1, 1977, pp. 99-118.

[Montanari 1974]Montanari, U., "Networks of Constraints: Fundamental Properties and Applications to Picture Processing," *Information Science*, Vol. 7, 1974, pp. 95-132.

[Nudel 1983]Nudel, B., "Consistent-Labeling Problems and their Algorithms: Expected Complexities and Theory-based Heuristics," *Artificial Intelligence*, Vol. 21, 1983, pp. 135-178.

[Purdom 1983]Purdom, P., "Search Rearrangement Backtracking and Polynomial Average Time," *Artificial Intelligence*, Vol. 21, 1983, pp. 117-133.

[Purdom 1981]Purdom, P. W., E. L. Robertson, and C. A. Brown, "Backtracking with Multi-level Dynamic Search Rearangement," *Acta Informatica*, Vol. 15, No. 2, 1981, pp. 99-114.

[Rosiers 1986]Rosiers, W. and M. Bruynooghe, "Empirical Study of Some Constraint Satisfaction Algorithms," Katholieke Universiteit Leuven, Leuven , Belguim, Tech. Rep. CW 50, 1986.

[Stone 1986]Stone, H. S. and J. M. Stone, "Efficient Search Techniques- An Empirical Study of the N-Queens Problem.," IBM T.J. Watson Research Center, Yorktown Heights, NY, Tech. Rep. RC 12057 (#54343), 1986.

[Waltz 1975]Waltz, D., "Understanding Line Drawings of Scenes with Shadows," in *The Psychology of Computer Vision*, P. H. Winston, Ed. New York, NY: McGraw-Hill Book Company, 1975.

[Zabih 1988]Zabih, R. and D. McAllester, "A Rearrangement Search Strategy for Determining Propositional Satisfiability," in *Proceedings AAAI-88*, St. Paul, Minnesota: 1988.