LEARNING WHILE SEARCHING IN CONSTRAINT-SATISFACTION-PROBLEMS*

Rina Dechter

Artificial Intelligence Center
Hughes Aircraft Company, Calabasas, CA 91302
and
Cognitive Systems Laboratory, Computer Science Department
University of California, Los Angeles, CA 90024

ABSTRACT

The popular use of backtracking as a control strategy for theorem proving in PROLOG and in Truth-Maintenance-Systems (TMS) led to increased interest in various schemes for enhancing the efficiency of backtrack search. Researchers have referred to these enhancement schemes by the names "Intelligent Backtracking" (in PROLOG), "Dependency-directed-backtracking" (in TMS) and others. Those improvements center on the issue of "jumping-back" to the source of the problem in front of dead-end situations.

This paper examines another issue (much less explored) which arises in dead-ends. Specifically, we concentrate on the idea of constraint recording, namely, analyzing and storing the reasons for the dead-ends, and using them to guide future decisions, so that the same conflicts will not arise again. We view constraint recording as a process of learning, and examine several possible learning schemes studying the tradeoffs between the amount of learning and the improvement in search efficiency.

I. INTRODUCTION

The subject of improving search efficiency has been on the agenda of researchers in the area of Constraint-Satisfaction-Problems (CSPs) for quite some time [Montanari 1974, Mackworth 1977, Mackworth 1984, Gaschnig 1979, Haralick 1980, Dechter 1985]. A recent increase of interest in this subject, concentrating on the backtrack search, can be attributed to its use as the control strategy in PROLOG [Matwin 1985, Bruynooghe 1984, Cox 1984], and in Truth Maintenance Systems [Doyle 1979, De-Kleer 1983, Martins 1986]. The terms "intelligent backtracking", "selective backtracking", and "dependency-directed backtracking" describe various efforts for producing improved dialects of backtrack search in these systems.

The various enhancements to Backtrack suggested for both the CSP model and its extensions can be classified as followed:

1. **Look-ahead schemes**: affecting the decision of what value to assign to the next variable among all the consistent choices available [Haralick 1980, Dechter 1985].

2. **Look-back schemes:** affecting the decision of where and how to go in case of a a dead-end situation. Look-back schemes are centered around two fundamental ideas:

   a. **Go-back to source of failure:** an attempt is made to detect and change previous decisions that caused the dead-end without changing decisions which are irrelevant to the dead-end.

   b. **Constraint recording:** the reasons for the dead-end are recorded so that the same conflicts will not arise again in the continuation of the search.

All recent work in PROLOG and truth-maintenance system, and much of the work in the traditional CSP model is concerned with look-back schemes, particularly on the go-back idea. Examples are Gaschnig's "Backmark" and "Backjump" algorithms for the CSP model [Gaschnig 1979] and the work on Intelligent-Backtracking for Prolog [Bruynooghe 1984, Cox 1984, Matwin 1985]. The possibility of recording constraints when dead-ends occur is mentioned by Bruynooghe [Bruynooghe 1984]. In truth-maintenance systems both ideas are implemented to a certain extent. However, the complexity of PROLOG and of TMS makes it difficult to describe (and understand) the various enhancements proposed for the backtrack search and, more importantly, to test them in an effort to assess their merits. The general CSP model, on the other hand, is considerably simpler, yet it is close enough to share the basic problematic search issues involved and, therefore, provides a convenient framework for describing and testing such enhancements.

Constraint-recording in look-back schemes can be viewed as a process of **learning**, as it has some of the properties that normally characterize learning in problem solving:

1. The system has a **learning module** which is independent of the problem-representation scheme and the algorithm for solving problem instances represented in this scheme.

2. The learning module works by observing the performance of the algorithm on any given input and recording some relevant information explicated during the search.

3. The overall performance of the algorithm is improved when it is used in conjunction with the learning module.

*This work was supported in part by the National Science Foundation, Grant #DCR 85-01234

4. When the algorithm terminates, the information accumulated by the learning module is part of a new, more knowledgeable, representation of the same problem. That is, if the algorithm is executed once again on the same input, it will have a better performance.

Learning has been a central topic in problem solving. The task of learning is to record in a useful way some information which is explicated during the search and use it both at the same problem instance and across instances of the same domain. One of the first applications of this notion involved the creation of macro-operators from sequences and subsequences of atomic operators that have proven useful as solutions to earlier problem instances from the domain. This idea was exploited in STRIPS with MACROPS [Fikes 1971]. A different approach for learning macros was more recently offered by [Korf 1982]. Other recent examples of learning in problem solving are: the work on analogical problem solving [Carbonell 1983], learning heuristic problem-solving strategies through experience as described in the program LEX [Mitchel 1983] and developing a general problem solver (SOAR) that learns about aspects of its behavior using chunking [Laird 1984].

In this paper we examine several learning schemes as they apply to solving general CSPs. The use of the CSP model allows us to state our approach in a clear and formal way, provide a parameterized learning scheme based on the time-space trade-offs, and analyze the trade-offs involved theoretically. We evaluated this approach experimentally on two problems with different levels of difficulty.

II. THE CSP MODEL AND ITS SEARCH-SPACE

A constraint satisfaction problem involves a set of n variables $X_1, ..., X_n$, each represented by its domain values, $R_1, \dots, R_n$ and a set of constraints. A constraint $C_i(X_{i_1}, \cdots, X_{i_j})$ is a subset of the Cartesian product $R_{i_1} \times \cdots \times R_{i_j}$ which specifies which values of the variables are compatible with each other. A solution is an assignment of values to all the variables which satisfy all the constraints and the task is to find one or all solutions. A constraint is usually represented by the set of all tuples permitted by it. A **Binary CSP** is one in which all the constraints are binary, i.e., they involve only pairs of variables. A binary CSP can be associated with a **constraint-graph** in which nodes represent variables and arcs connects pairs of constrained variables. Consider for instance the CSP presented in figure 1 (from [Mackworth 1977]). Each node represent a variable whose values are explicitly indicated, and the constraint between connected variables is a strict lexicographic order along the arrows.

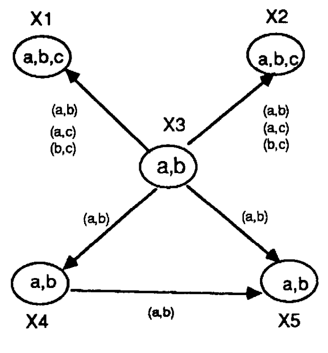

Figure 1: An example CSP

Backtracking works by provisionally assigning consistent values to a subset of variables and attempting to append to it a new instantiation such that the whole set is consistent. An assignment of values to subset of the variables is **consistent** if it satisfies all the constraints applicable to this subset. A constraint is **applicable** to a set of variables if it is defined over a subset of them.

The order by which variables get instantiated may have a profound effect on the efficiency the algorithm [Freuder 1982] since each ordering determine a different search space with different size. The ordering can be predetermined, or could vary dynamically, in which case the search space is a graph whose states are unordered subsets of consistently instantiated variables. The methods suggested in this paper are not dependent on the particular ordering scheme chosen, and we assume, without loss of generality, that the ordering is given as part of the problem input. Moreover, in Section 5 we generate different instances of a problem, for our numerical experiments, by simply changing the ordering of the variables of the same problem.

Another issue that have influence on the size of the search space is the CSP's input representation, i.e. a set of variables, their domains and the set of explicit constraints. It defines a relation among the variables, consisting of those tuples satisfying all the constraints, or the set of all solutions. There may be numerous equivalent CSP representations for the same set of solutions and some may be better then others since they yield a smaller search space. One way of improving the representation is by **inducing**, or **propagating** constraints [Montanari 1974, Mackworth 1977]. For example, the constraints C(X,Y) and C(Y,Z) **induce** a constraint C(X,Z) as follows: A pair (x,z) is allowed by C(X,Z) if there is at least one value y in the domain of Y such that (x,y) is allowed by C(X,Y) and (y,z) is allowed by C(Y,Z). For instance, for the problem in figure 1, a constraint between X_1 and X_2 can be induced from the binary constraints C(X_1,X_3) and C(X_3,X_2) to yield a constraint C(X_1,X_2) that disallow (among other pairs) the pair (a,a). The definition of induced constraints can be extended in a natural way to non-binary constraints.

Several schemes for improving the search efficiency by pre-processing the problem's representation have been proposed [Montanari 1974, Mackworth 1984, Dechter 1985]. These pre-processing schemes can be viewed as a mode of learning since they result in modified data structure and improved performance. However, inducing all possible constraints may involve a procedure which is exponential both in time and space [Freuder 1978].

III. LEARNING WHILE SEARCHING

The process of learning constraints need not be performed as a pre-processing exercise, but can rather be incorporated into the backtrack search. An opportunity to learn new constraints is presented each time the algorithm encounters a **dead-end** situation, i.e. whenever the current state $S = (X_1 = x_1, \dots, X_{i-1} = x_{i-1})$ cannot be extended by **any** value of the variable X_i. In such a case we say that S **is in conflict** with X_i or, in short, that S is a **conflict-set**. An obvious constraint that can be induced at that point is one that prohibits the set S. Recording this constraint, however, is of no help since under the backtrack control strategy this state will never re-occur. If, on the other hand, the set S contains one or more subsets which are also in conflict with X_i, then recording this information in the form of new explicit constraints might prove useful in future search.

One way of discovering such a subset is by removing from S all the instantiations which are **irrelevant** to X_i. A pair consisting of a variable and one of its value (X,x) in S is said to be irrelevant to X_i if it is consistent with all values of X_i w.r.t the constraints applicable to S. We denote by $Conf(S,X_i)$, or **Conf-set** in short, the conflict-set resulting by removing all irrelevant pairs from S.

The Conf-set may still contain one or more subsets which are in conflict with X_i. Some of these subsets are **Minimal conflict sets** [Bruynooghe 1981], that is, they do not contain any proper conflict-sets and, so, can be regarded as the sets of instantiations that "caused" the conflict. Since a set which contains a conflict-set is also in conflict, it is enough to explicitly discover all the minimal conflict-sets i.e., the set of smallest conflict-sets.

Consider again the problem in figure 1. Suppose that the backtrack algorithm is currently at state $(X_1 = b, X_2 = b, X_3 = a, X_4 = b)$. This state cannot be extended by any value of X_5 since none of its values is consistent with all the previous instantiations. This means, of course, that the tuple $(X_1 = b, X_2 = b, X_3 = a, X_4 = b)$ should not have been allowed in this problem. As pointed out above, however, there is no point recording this fact as a constraint among the four variables involved. A closer look reveals that the instantiation $X_1 = b$ and $X_2 = b$ are both irrelevant in this conflict simply because there is no explicit constraint between X_1 and X_5 or between X_2 and X_5. Neither $X_3 = a$ nor $X_4 = b$ can be shown to be irrelevant and, therefore, the Conf-set is $(X_3 = a, X_4 = b)$. This could be recorded by eliminating the pair (a,b) from the set of pairs permitted by $C(X_3,X_4)$. This Conf-set is not minimal, however, since the instantiation $X_4 = b$ is, by itself, in conflict with X_5. Therefore, it would be sufficient to record this information only, by eliminating the value b from the domain of X_4.

Finding the conflict-sets can assist backtrack not only in avoiding future dead-ends but also by backjumping to the appropriate relevant state rather then to the chronologically most recent instantiation. If only the Conf-set is identified the algorithm should go back to the most recent variable (i.e. the deepest variable) in this set. If the minimal conflict-sets $m_1, m_2, \ldots, m_l$ are identified, and if $d(m_i)$ is the depth of the deepest variable in m_i then the algorithm should jump back to the shallowest among those deep variables, i.e. to.

$$Min\{d(m_j)\} \tag{1}$$

Discovering all minimal conflict-sets amounts to acquiring all the possible information out of a dead-end. Yet, such **deep learning** may require considerable amount of work. While the number of minimal conflict-sets is less then 2^r, where r is the cardinality of the Conf-set, we can envision a worst case where all subsets of Conf(S,X_i) having $\frac{r}{2}$ elements are in conflict with X_i. The number of minimal conflict-sets should then satisfy

$$\#\text{min-conflict-sets} = \binom{r}{\frac{r}{2}} \cong 2^r , \tag{2}$$

which is still exponential in the size of the Conf-set. If the size of this Conf-set is small it may still be reasonable to recognize all minimal conflict-sets.

Most researchers in the area of truth-maintenance-systems have adopted the approach that all the constraints realized during the search should be recorded (recording no-good sets or restriction sets), e.g., [Doyle 1979, De-Kleer 1983, Martins 1986]. However, learning all constraints may amount to recording almost all the search space explored. Every dead-end contains a new induced constraint. The number of dead-ends may be exponential in the worst case, i.e., $O(k^n)$ when n is the number of variables and k is the number of values for each variable, which presents both a storage problem and a processing problem. It seems reasonable, therefore, to restrict the information learned to items which can be stored compactly and still have a good chance for being reused. In the next section we discuss several possibilities for accomplishing these criteria.

IV. CONTROLLED LEARNING

Identifying the Conf-set is the first step in the discovery of other subsets in conflict and, by itself, it can be considered a form of **shallow learning**. It is easy to show that the Conf-set satisfies

$$Conf = \bigcup_{x_{ij}} T(x_{ij}) , \tag{3}$$

where x_{ij} is the j^{th} value in the domain of X_i and $T(x_{ij})$ is a subset of S which contains all instantiations in S that are not consistent with the assignment $X_i = x_{ij}$. Let C be the set of relevant constraints on $S \cup \{X_i\}$ which involve X_i, and let l be the size of C. The identification of a specific T-set requires testing all these constraints. An algorithm for identifying the Conf-set may work by identifying T-sets for all the values of X_i and unionize them and its complexity is $O(k \cdot l)$ when **k** is the number of values for X_i.

An approximation of the Conf-set may be obtained by removing from the set S only those variables that are not associated with any constraint involving X_i. The resulting conflict set, which contains the Conf-set, may be used as a surrogate for it. The complexity of this algorithm is just $O(l)$ but it may fail to delete an irrelevant pair which appears in some constraint but did not participate in any violation. For example, in the example CSP the state $\{X_1 = a, X_2 = c\}$ is at dead-end since it cannot be extended by any value of X_3. The approximate Conf-set in this case is the whole state since both X_1 and X_2 have constraints with X_3 however a careful look reveals that $X_2 = c$ is irrelevant to X_3 and the real Conf-set is $\{X_1 = a\}$.

Independently of the depth of learning chosen, one may restrict the **size** of the constraints actually recorded. Constraints involving only a small number of variables require less storage and have a better chance for being reused (to limit the search) than constraints with many variables. For example, we may decide to record only conflict-sets consisting of a single instantiation. this is done by simply eliminating the value from the domain of the variable. We will refer to this type of learning as **first-order learning** which amounts to making a subset of the arcs **arc-consistent** [Mackworth 1977]. It does not result in global arc-consistency because it only make consistent those arcs that are encountered during the search. First-order learning does not increase the storage of the problem beyond the size of the input and it prunes the search each time the deleted value is a candidate for assignment. For example, if we deleted a value from a variable at depth j we may prune the search in as much as k^{j-1} other states.

Second-order learning is performed by recording only conflict-sets involving either one or two variables. Since not all pairs of variables appear in constraints in the initial representation (e.g. when all pair of values are permitted nothing is written), second-order learning may increase the size of the problem. There are at most $\frac{n \cdot (n-1)}{2}$ binary constraints, each having at most k^2 pairs of values, the increase in storage is still reasonably bounded and may be compensated by saving in search. Second-order learning performs partial **path-consistency** [Montanari 1974] since it only adds and modify constraints emanating from paths discovered during the search.

When deep learning is used in conjunction with restricting the level of learning we get **deep first-order learning** (identifying minimal conflict sets of size 1) and **deep second-order learning** (i.e. identifying minimal conflict-sets of sizes 1 and 2). The complexity of **deep first-order learning** is $O(k \cdot r \cdot l)$ when r is the size of the Conf-set since each instantiation is tested against all values of X_i. The complexity of **deep second-order learning** can rise to $O(\left| \frac{r(r-1)}{2} \right| \cdot k \cdot l)$ since in this case each pair of instantiations should be checked against each value of X_i.

In a similar manner we can define and execute higher degrees of learning in backtrack. In general, an i^{th}**-order learning** algorithm will record every constraint involving i or less variables. Obviously, as i increases storage increases. The additional storage required for higher order learning can be avoided, however, by further restricting the algorithm to only **modify** existing constraint without creating new ones. This approach does not change the structure of the constraint-graph associated with the problem, a property which is sometimes desirable [Dechter 1985].

V. EXPERIMENTAL EVALUATION

The backtrack-with-learning algorithm has been tested on two classes of problems of different degrees of difficulty. The first is the **class problem**, a data-base type problem adapted* from [Bruynooghe 1984]. The problem statement is given in Appendix 1.

The second, and more difficult, problem is known as **the Zebra problem**. The problem's statement is given in Appendix 2. It can be represented as a binary CSP by defining 25 variables each having five possible values denoting the identities of the different houses.

Several instances of each problem have been generated by randomly varying the order of variables' instantiation. As explained in Section 2, each ordering results in a different search space for the problem and, therefore, can be considered as a different instance.

The mode of learning used in the experiments was controlled by two parameters: the depth of learning (i.e., shallow or deep), and the level of learning (i.e., first order or second order). This results in four modes of learning: shallow-first-order, shallow-second-order, deep-first-order, and deep-second-order. The information obtained by the learning module was utilized also for **backjumping** as discussed in Section 3.

*Our problem is an approximation of the original problem where only binary constraints are used.

Each problem instance was solved by six search strategies: naive backtrack, backtrack with backjump (no learning), and backtrack with backjump coupled with each of the four possible modes of learning. The results for six problem instances of the class problem are presented in table 1, and for six problem instances of the zebra problem in table 2. The following abbreviations are used: NB = naive backtrack, BJ = Backjump, SF = Shallow-First-Order, SS = Shallow-Second-Order, DF = Deep-First-Order, DS = Deep-Second-order.

#	NB	BJ	SF	SS	DF	DS
1	219 25	219 25	218 25 (44)	221 25 (44)	218 25 (44)	194 22 (44)
2	123 12	123 12	123 12 (43)	133 12 (43)	137 12 (42)	155 12 (42)
3	266 24	266 24	266 24 (140)	267 24 (140)	260 20 (140)	125 7 (51)
4	407 42	407 42	406 42 (108)	409 42 (108)	423 40 (91)	509 39 (50)
5	433 42	433 42	433 42 (116)	435 42 (116)	445 40 (91)	527 39 (50)
6	559 85	559 85	559 85 (441)	391 53 (55)	692 85 (461)	619 57 (49)

Table 1: The Class Problem

#	NB	BJ	SF	SS	DF	DS
1	2066 180	1241 57	1234 56 (1214)	1236 56 (1214)	1272 56 (1252)	884 37 (322)
2	37541 5378	20542 1315	19498 1279 (19416)	19498 1279 (19416)	21371 1279 (19945)	4523 302 (1584)
3	241204* 31097	46771 2848	44719 2723 (44693)	44716 2721 (44693)	41911 2512 (38421)	10670 530 (1861)
4	237152 23778	21946 738	21946 738 (21946)	21205 655 (10591)	22117 738 (22117)	5521 190 (572)
5	37541 5378	30100 4159	30091 4158 (30009)	11777 1621 (11180)	30091 4158 (30009)	4604 414 (1579)
6	1153366* 105215	91026 6362	91026 6362 (33071)	93610 6362 (33126)	93610 6362 (33126)	17367 961 (9535)

Table 2: The Zebra Problem

Each of the problem instances was solved twice by the same strategy; the second run using a new representation that included all the constraints recorded in the first run. This was done to check the effectiveness of these strategies in finding a better problem representation.

Each entry in the table records three numbers: the first is the number of consistency checks performed by the algorithm, the second is the number of backtrackings, and the Parenthesized number gives the number of consistency checks in the second run.

Our experiments (implemented in LISP on a Symbolics LISP Machine) show that in most cases both performance measures improve as we move from shallow learning to deep learning and from first-order to second-order. The class problem turned out to be very easy, and is solved efficiently even by naive backtrack. The effects of backjumping and learning are, therefore, minimal except for deep-second-order learning where gains are sometimes evident. In some instances there is some deterioration due to unnecessary learning. In all these cases, the second run gave a backtrack-free performance.

The zebra problem, on the other hand, is much more difficult, and in some cases could not be solved by naive backtrack in a reasonable amount of time (in these cases the number reported are the counts recorded at time the run was stopped (*)). The enhanced backtrack schemes show dramatic improvements in two stages. First, the introduction of backjump by itself improved the performance substantially, with only moderate additional improvements due to the introduction of first-order or shallow second-order learning. Second-order-deep learning caused a second leap in performance, with gains over no-learning-backjump by a factor of 5 to 10. The experimental results for the zebra problem are depicted graphically in Figure 2 (for the number of consistency checks) and in Figure 3 (for the number of bakctrackings).

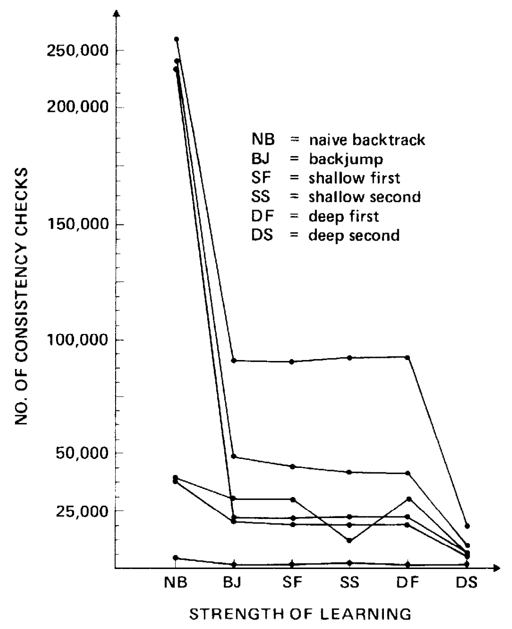

Figure 2: Number of Consistency Checks for the Zebra Problem

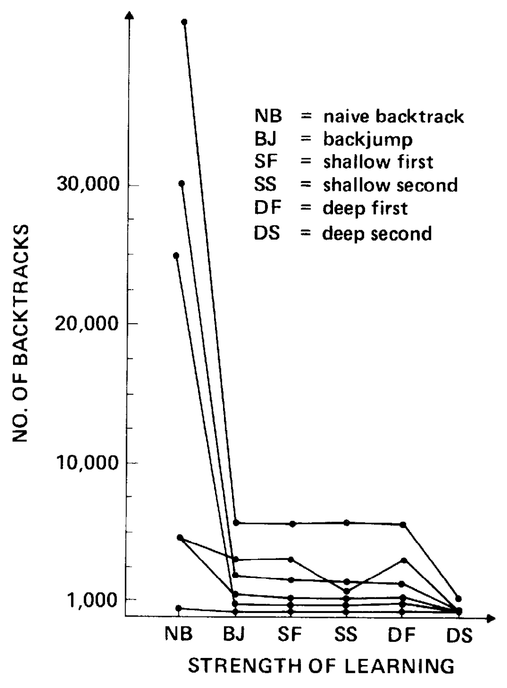

Figure 3: Numebr of Backtrackings for the Zebra Problem

VI. CONCLUSIONS

Our experiments demonstrate that learning might be very beneficial in solving CSPs. Most improvement was achieved by the strongest form of learning we have tested: deep-second-order learning. It remains to be tested whether higher degrees of learning perform even better or whether storage considerations and the amount of work invested in such learning outweigh the reduction in search.

It was also shown that the more "knowledgeable" problem representation, achieved upon termination of backtrack-with-learning, is significantly better that the original one. This feature is beneficial when a CSP model is viewed as a world representing an initial set of constraints on which many different queries can be posed. Each query assumes a world that satisfies all these static constraints and some of the additional **query constraints**. Recording all the solutions for the initial set of constraints may be too costly and may not be efficiently used when new queries arrive. In such cases it may be worthwhile to keep the world model in the form of a set of constraints enriched by those learned during past searches.

Another issue for further research is the comparison of first and second-order learning with the pre-processing approach of performing full arc and path-consistency prior to search. The pre-processing approach yield a representation which is usually better then that of second-order-learning, but the question is at what cost? Theoretical considerations reveal that pre-processing may be too costly and may perform unnecessary work. For instance, the Path-consistency algorithm is known to have a lower bound on its performance of $O(n^3k^3)$ on every problem instance. For the zebra problem this number is 1,953,125 consistency checks, which is far worse the performance of deep-second- order learning on all problem instances presented.

APPENDIX I: THE CLASS PROBLEM

Several students take classes from several professors in different days and rooms according to the following constraints:

Student(Robert,Prolog)
Student(John,Music)
Student(John,Prolog)
Student(John,surf)
Student(Mary,Science)
Student(mary,Art)
Student(Mary, Physics)

Professor(Luis,Prolog)
Professor(Luis,Surf)
Professor(Maurice,Prolog)
Professor(Eureka,Music)
Professor(Eureka,Art)
Professor(Eureka,Science)
Professor(Eureka,Physics)

Course(Prolog,Monday,Room1)
Course(Prolog,Friday,Room1)
Course(surf,Sunday,Beach)
Course(Math,Tuesday,Room1)
Course(Math,Friday,Room2)
Course(Science,Thurseday,Room1)
Course(Science,Friday,Room2)
Course(Art,Tuesday,Room1)
Course(Physics,Thurseday,Room3)
Course(Physics,Saturday,Room2)

The query is: find Student(stud,course1) and Course(course1,day1,room) and Professor(prof,course1) and Student(stud,course2) and Course(course2,day2,room) and noteq(course1,course2)

APPENDIX II: THE ZEBRA PROBLEM

There are five houses of different colors, inhabited by different nationals, with different pets, drinks, and cigarettes:

1. The Englishman lives in the red house
2. The Spaniard owns a dog.
3. Coffee is drunk in the green house.
4. The Ukranian drinks tea
5. The green house is to the right of the ivory house.
6. The old-gold smoker owns snails
7. Kools are being smoked in the yellow house.
8. Milk is drunk in thye middle house.
9. The Norwegian lives in the first house on the left.
10. The chesterfield smoker lives next to the fox owner.
11. Kools are smoked next to the house with the horse.
12. The Lucky-Strike smoker drinks orange juice.
13. The Japanese smoke Parliament
14. The Norwegian lives next to the blue house.

The question is: Who drinks water? and who owns the Zebra?

REFERENCES

[1]Bruynooghe, Maurice, "Solving combinatorial search problems by intelligent backtracking," *Information Processing Letters,* Vol. 12, No. 1, 1981.

[2]Bruynooghe, Maurice and Luis M. Pereira, "Deduction Revision by Intelligent backtracking," in *Implementation of Prolog,* J.A. Campbell, Ed. Ellis Harwood, 1984, pp. 194-215.

[3]Carbonell, J.G., "Learning by analogy: Formulation and generating plan from past experience," in *Machine Learning,* Michalski, Carbonell and Mitchell, Ed. Palo Alto, California: Tioga Press, 1983.

[4]Cox, P.T., "Finding backtrack points for intelligent backtracking," in *Implementation of Prolog,* J.A. Campbell, Ed. Ellis Harwood, 1984, pp. 216-233.

[5]Dechter, R. and J. Pearl, "The anatomy of easy problems: a constraint-satisfaction formulation," in *Proceedings Ninth International Conference on Artificial Intelligence,* Los Angeles, Cal: 1985, pp. 1066-1072.

[6]De-Kleer, Johan, "Choices without backtracking," in *Proceedings AAAI,* Washington D.C.: 1983, pp. 79-85.

[7]Doyle, Jon, "A truth maintenance system," *Artificial Intelligence,* Vol. 12, 1979, pp. 231-272.

[8]Fikes, R.E. and N.J. Nilsson, "STRIPS: a new approach to the application of theorem to problem solving.," *Artificial Intelligence,* Vol. 2, 1971.

[9]Freuder, E.C., "Synthesizing constraint expression," *Communication of the ACM,* Vol. 21, No. 11, 1978, pp. 958-965.

[10]Freuder, E.C., "A sufficient condition of backtrack-free search.," *Journal of the ACM,* Vol. 29, No. 1, 1982, pp. 24-32.

[11]Gaschnig, J., "A problem similarity approach to devising heuristics: first results," in *Proceedings 6th international joint conf. on Artificial Intelligence.,* Tokyo, Jappan: 1979, pp. 301-307.

[12]Haralick, R. M. and G.L. Elliot, "Increasing tree search efficiency for cconstraint satisfaction problems," *AI Journal,* Vol. 14, 1980, pp. 263-313.

[13]Korf, R.E., "A program that learns how to solve rubic's cube.," in *Proceedings AAAI Conference,* Pittsburg, Pa: 1982, pp. 164-167.

[14]Laird, J. E., P. S. Rosenbloom, and A. Newell, "Towards chunking as a general learning mechanism," in *Proceedings National Conference on Artificial Intelligence,* Austin, Texas: 1984.

[15]Mackworth, A.K., "Consistency in networks of relations," *Artifficial intelligence,* Vol. 8, No. 1, 1977, pp. 99-118.

[16]Mackworth, A.K. and E.C. Freuder, "The complexity of some polynomial network consistancy algorithms for constraint satisfaction problems," *Artificial Intelligence ,* Vol. 25, No. 1, 1984.

[17]Martins, Joao P. and Stuart C. Shapiro, "Theoretical Foundations for belief revision," in *Proceedings Theoretical aspects of Reasoning about knowledge,* 1986.

[18]Matwin, Stanislaw and Tomasz Pietrzykowski, "Intelligent backtracking in plan-based deduction," *IEEE Transaction on Pattern Analysis and Machine Intelligence,* Vol. PAMI-7, No. 6, 1985, pp. 682-692.

[19]Mitchel, T., P.E. Utgoff, and R. Banerji, "Learning by experimentation; acquiring and refining problem solving heuristics.," in *Machine learning,* Michalski, R.S., Carbonel, J.R., Mitchel, T.M., Ed. Palo Alto, California: Tioga publishing company, 1983.

[20]Montanari, U., "Networks of constraints :fundamental properties and applications to picture processing," *Information Science,* Vol. 7, 1974, pp. 95-132.