CONSTRAINT SATISFACTION

by

Alan Mackworth*

Technical Report 85-15
September 1985

Laboratory for Computational Vision
Department of Computer Science
University of British Columbia
Vancouver, B.C.
Canada V6T 1W5

* Alan Mackworth is a Fellow of the Canadian Institute for Advanced Research.

1. INTRODUCTION

Constraint Satisfaction is an umbrella term for a variety of techniques in artificial intelligence and related disciplines. In this article attention is focussed on the main approaches, such as backtracking, constraint propagation and cooperative algorithms, with some consideration given to the motivations and techniques underlying other constraint-based systems.

The first class of constraint satisfaction problems considered are those in which one has a set of variables each to be instantiated in an associated domain, and a set of Boolean constraints limiting the set of allowed values for specified subsets of the variables. This general formulation has a wide variety of incarnations in various applications: it is a general search problem. One standard approach involves backtracking; various forms of "intelligent" backtracking are surveyed. A complementary approach based on the class of consistency algorithms has some nice properties which are described and illustrated.

The second class of problems considered are the numerical optimization problems that arise when one is designing a system to maximize the extent to which the solutions it provides satisfy a large number of local constraints. Algorithms for their solution are based on generalizations of the consistency algorithms, for applications primarily in computational vision. These algorithms, which have a high degree of potential parallelism, are variously known as cooperative or probabilistic relaxation algorithms.

One can call these two problem classes Boolean constraint satisfaction problems and constrained optimization problems, respectively. As with all dichotomies this one is not absolute: some approaches lie between these two poles, others combine them. There are, in fact, many other dimensions along which one could categorize the area but this is the best first cut.

2. BOOLEAN CONSTRAINT SATISFACTION PROBLEMS

A Boolean constraint satisfaction problem (CSP) is characterized as follows: given is a set V of n variables $\{v_1, v_2, \ldots, v_n\}$, associated with each variable v_i is a domain D_i of possible values. On some specified subsets of those variables, there are constraint relations given which are subsets of the Cartesian product of the domains of the variables involved. The set of solutions is the largest subset of the Cartesian product of all the given variable domains such that each n-tuple in that set satisfies all the given constraint relations. One may be required to find the entire set of solutions, one member of the set, or simply to report if the set of solutions has any members - the decision problem. If the set of solutions is empty the CSP is unsatisfiable.

A surprisingly large number of seemingly different applications can be formalised in this way. Some of them shall be enumerated in Section 3.4. One, of particular theoretical interest, is the map coloring problem. Consider, for example, the problem of deciding if three colors suffice to color a given planar map such that each region is a different color from each of its neighbors. This is formulated as a Boolean CSP by creating a variable for each region to be colored, associating with each variable the domain {Red, Green, Blue} and requiring for each pair of adjacent regions that they have different colors. Since the map coloring problem is known to be NP-complete and is therefore believed inherently to require exponential time to solve, one does not expect to find an efficient, polynomial time algorithm to determine if a general CSP is satisfiable.

Various restrictions on the general definition of a CSP are possible. For example, the domains may be required to have a finite number of discrete values. If this is the case then the constraining relations may be specified extensionally as the set of all p-tuples that satisfy the constraint. One may further require that all the relations be unary or binary, that is, that they only constrain individual variables or pairs of variables. These restrictions apply to the map coloring example above. However, they are not necessary for some of the techniques reported here to be applicable. For example, suppose one were planning the layout of furniture in an office. The position of each item of furniture would be a variable, with an associated domain that would contain an infinite number of pairs (or triples, if rotations are allowed) of real values. Those domains would have to be described intensionally by, for example, describing the boundaries of the connected subspaces permitted for that item. The constraints, such as "The wastebasket must be within three feet of the chair. The door must be unobstructed. ..." must also be specified intensionally using, perhaps, algebraic inequalities on the values of the constrained variables. Moreover, one might have p-ary relations such as: "The desk must be between the chair and the door."

Crossword puzzles will be used here as a tutorial example of the concepts of constraint satisfaction. Consider the puzzle in Figure 1. To simplify the presentation, assume that one is required to find in the given word list the eight words that correspond to 1 Across, 2 Down and so on, with duplicates allowed. The reader should try to solve this simple CSP now, introspecting on the methods used as she goes through the process of looking for a solution.

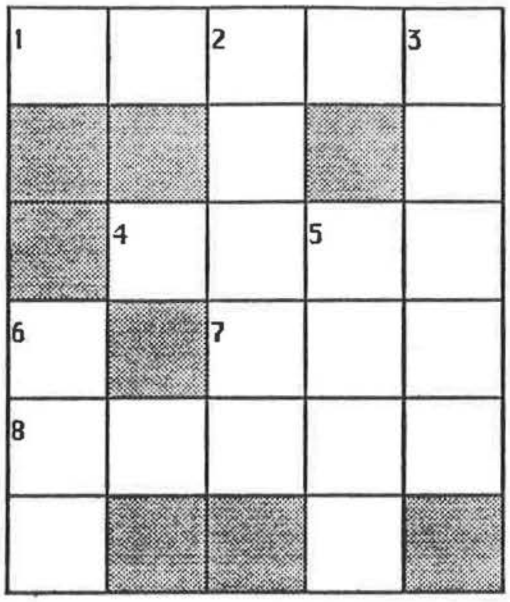
In general, one may represent the satisfiability decision problem for a CSP as equivalent to determining the truth value of a well-formed formula in first order predicate logic:

$$(\exists x_1)(\exists x_2)\cdots(\exists x_n)(x_1 \in D_1)(x_2 \in D_2)\cdots(x_n \in D_n)$$

$$P_1(x_1) \wedge P_2(x_2)\cdots \wedge P_n(x_n) \wedge P_{12}(x_1,x_2) \wedge P_{13}(x_1,x_3) \wedge \cdots \wedge P_{n-1},P_n(x_{n-1},x_n) \qquad [1]$$

P_{ij} is only included in the formula if $i < j$ since it is assumed that $P_{ji}(x_j,x_i) = P_{ij}(x_i,x_j)$. Initially here, only constraints representable as unary and binary predicates will be considered. For the crossword puzzle, the unary constraints $\{P_i\}$ specify the word length. P_1 requires that the word starting at 1 Across have 5 letters. The binary constraints arise when a word across intersects a word down. For example, P_{12} requires that the third letter of word 1 Across be the same as the first letter of word 2 Down. In general, but not for this example, p-ary predicates $(1 \leq p \leq n)$ are required.

For binary predicates another convenient problem representation is as a *network* consisting of a graph with a vertex for each variable with its associated domain attached and an edge between the vertices corresponding to each pair of



Word List

AFT	LASER
ALE	LEE
HEEL	LINE
HIKE	SAILS
HOSES	SHEET
KEEL	STEER
KNOT	

Figure 1. A constraint satisfaction problem: solve the crossword

directly constrained variables. In the crossword puzzle constraint network shown in Figure 2 the initial domain of words for each variable is shown inside the vertex for that variable. Notice that only words satisfying the unary word length constraint are shown. In general, for p-ary constraints ($p > 2$) a hypergraph representation, with a hyperedge for each constraint connecting the p vertices involved, is required.

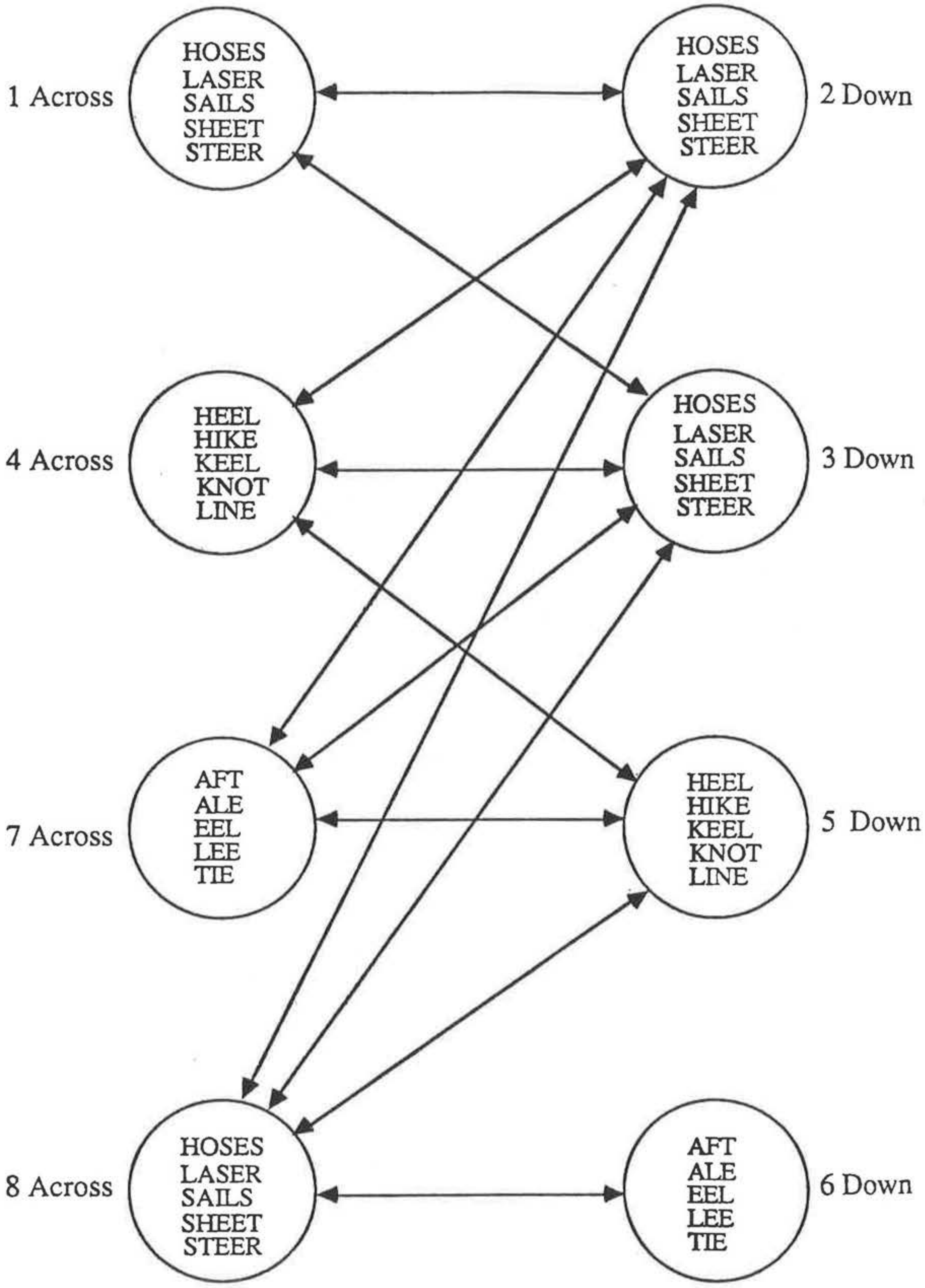


Figure 2. The crossword puzzle constraint network

3. BACKTRACKING AND CONSISTENCY ALGORITHMS FOR CONSTRAINT SATISFACTION PROBLEMS

3.1. Generate-and-Test

Assuming finite discrete domains, there is an algorithm to solve any CSP. The assignment space $D = D_1 \times D_2 \times \cdot\cdot \times D_n$ is finite and so one may evaluate the body of formula [1] on each element of D and stop if it evaluates to true. This *generate-and-test* algorithm is correct but slow. In the crossword puzzle, the number of different assignments to be tested is 5^8 or 390,625.

3.2. Backtracking Algorithms

Backtracking algorithms systematically explore D by sequentially instantiating the variables in some order. As soon as any predicate has all its variables instantiated its truth value is determined. Since the body of formula [1] is a conjunction, if that predicate is false that partial assignment cannot be part of any total valid assignment. Backtracking then fails back to the last variable with unassigned values remaining in its domain (if any) and instantiates it to its next value. The efficiency gain from backtracking arises from the fact that a potentially very large subspace of D, namely the product space of the currently unassigned variable domains, is eliminated by a single predicate failure.

The reader is invited to solve the crossword puzzle by backtracking, instantiating the words in the order 1 to 8. Start with word 1 Across as HOSES, try word 2 Down as HOSES; P_{12} is not satisfied so all potential solutions with these two choices for 1 and 2 are illegal. Next try word 2 as LASER, and so on.

The efficiency of backtracking has been investigated empirically (25, 4, 15, 16). Good analytical results are hard to come by but see (16, 13, 36, 35). Other factors being equal, it pays to pre-order the variables in terms of increasing domain size; one thereby maximizes the average size of the subspace rejected by the failure of a predicate. This principle has been extended to dynamic re-ordering (4, 37) involving 1 or 2 or more levels of lookahead search to find the variable with the smallest domain of acceptable values to instantiate next. Regardless of the order of instantiation one almost always observes thrashing behaviour in backtrack search (5). Thrashing can be defined here as the repeated exploration of subtrees of the backtrack search tree that differ only in inessential features, such as the assignments to variables irrelevant to the failure of the subtrees (43, 27). This ubiquitous phenomenon is indeed observed, in abundance, as one develops the search tree for the crossword puzzle. Many of the techniques reported in this subsection and the next are designed to reduce or eliminate thrashing, essentially by providing the algorithms with better memories.

One form of so-called intelligent backtracking uses varying degrees of look ahead to delete unacceptable values from the domains of all the uninstantiated variables (17, 16). Another form of intelligent backtracking identifies the latest instantiated variable causing the failure and fails back to it, possibly across many intervening levels (43, 15, 8). Gaschnig's (14) backmarking algorithm is another potential improvement on backtracking that looks backward to remember value combinations that guarantee failure or success so that they are not re-tried elsewhere in the tree.

Similar techniques are exploited in dependency-directed backtracking (42) and truth or belief maintenance systems (10). Those systems generally abandon the chronological stack-based control discipline of pure backtracking, allowing choices to be undone independent of the order in which they were made. The AI programming languages Micro-Planner and Prolog are based on automatic backtrack control structures. The possibility of providing some of the techniques surveyed in this article as general AI tools should not be overlooked (43, 27, 10).

3.3. Consistency Algorithms

Another family of algorithms, complementary to the class of backtracking algorithms, has been characterized as the class of consistency algorithms (27). By analyzing the various causes of thrashing behaviour in backtracking, various authors have described algorithms that eliminate those causes (45, 47, 34, 27, 12). They are most easily described in the network model of CSP's given earlier. For binary constraints each edge in the graph between vertices i and j is replaced by the arc (i,j) and arc (j,i).

Node i, composed of vertex i and the associated domain of variable v_i, is *node consistent* iff

$$[(\forall x) x \in D_i] \supset P_i(x).$$

Each node can trivially be made consistent by performing the *domain restriction operation*

$$D_i \leftarrow D_i \cap \{x \mid P(x)\}.$$

In the crossword puzzle, this corresponds to the obvious strategy of deleting from each variable's domain any word with the wrong length (and, in a real crossword puzzle, any word that does not fit the clue).

Similarly arc (i, j) is *arc consistent* iff

$$[(\forall x) \ \ x \in D_i] \supset (\exists y)(y \in D_j) \wedge P_{ij}(x, y)$$

that is, if for every element in D_i there is at least one element in D_j such that the pair of elements satisfy the constraining predicate. Arc (i, j) can be made arc consistent by removing from D_i all elements that have no corresponding element in D_j with the following arc consistency *domain restriction* operation:

$$D_i \leftarrow D_i \cap \{x \mid (\exists y)(y \in D_j) \wedge P_{ij}(x, y)\}. \qquad [2]$$

In the language of relational database theory this operation is known as a semi-join (31). A network is node and arc consistent iff all its nodes and arcs are consistent. A given network for a CSP can be made node consistent in a single pass over the nodes. However, a single pass of the arc consistency operation over the arcs will not guarantee that the network is arc consistent. One must either repeat that pass until there is no reduction in any domain in a complete pass or use a more selective constraint propagation technique that examines each of the arcs keeping track of the arcs that may have become inconsistent as a result of deletions from the domain at their destination node (47, 27). The first approach is a symbolic relaxation algorithm and suggests parallel implementation techniques (38). The second is usually more efficient on a single processor. The Waltz (47) filtering algorithm uses the second approach. That arc consistency algorithm requires time linear in the number of constraints to make the network arc consistent (30).

The best framework for understanding these algorithms is to see them as removing local inconsistencies from the network which can never be part of any global solution. When those inconsistencies are removed they may cause inconsistencies in neighbouring arcs that were previously consistent. Those inconsistencies are in turn removed so the algorithm eventually arrives, monotonically, at a fixed point consistent network and halts. An inconsistent network has the same set of solutions as the consistent network that results from applying a consistency algorithm to it, but if one subsequently applies, say, a backtrack search to the consistent network the resultant thrashing behaviour can be no worse and may be much better.

The result of applying algorithm AC-3, a serial arc consistency algorithm (27), to the crossword puzzle constraint graph is shown in Figure 3. The arcs to be initially examined are put on a queue in the order (12, 21, 13, 31, 42, 24, 43, ..., 86, 68) and the deleted words are italicised. When words are delected from a domain at a node all the arcs into that node not currently waiting on the queue (except the reverse of the arc causing the deletion) are added to the end of the queue. In

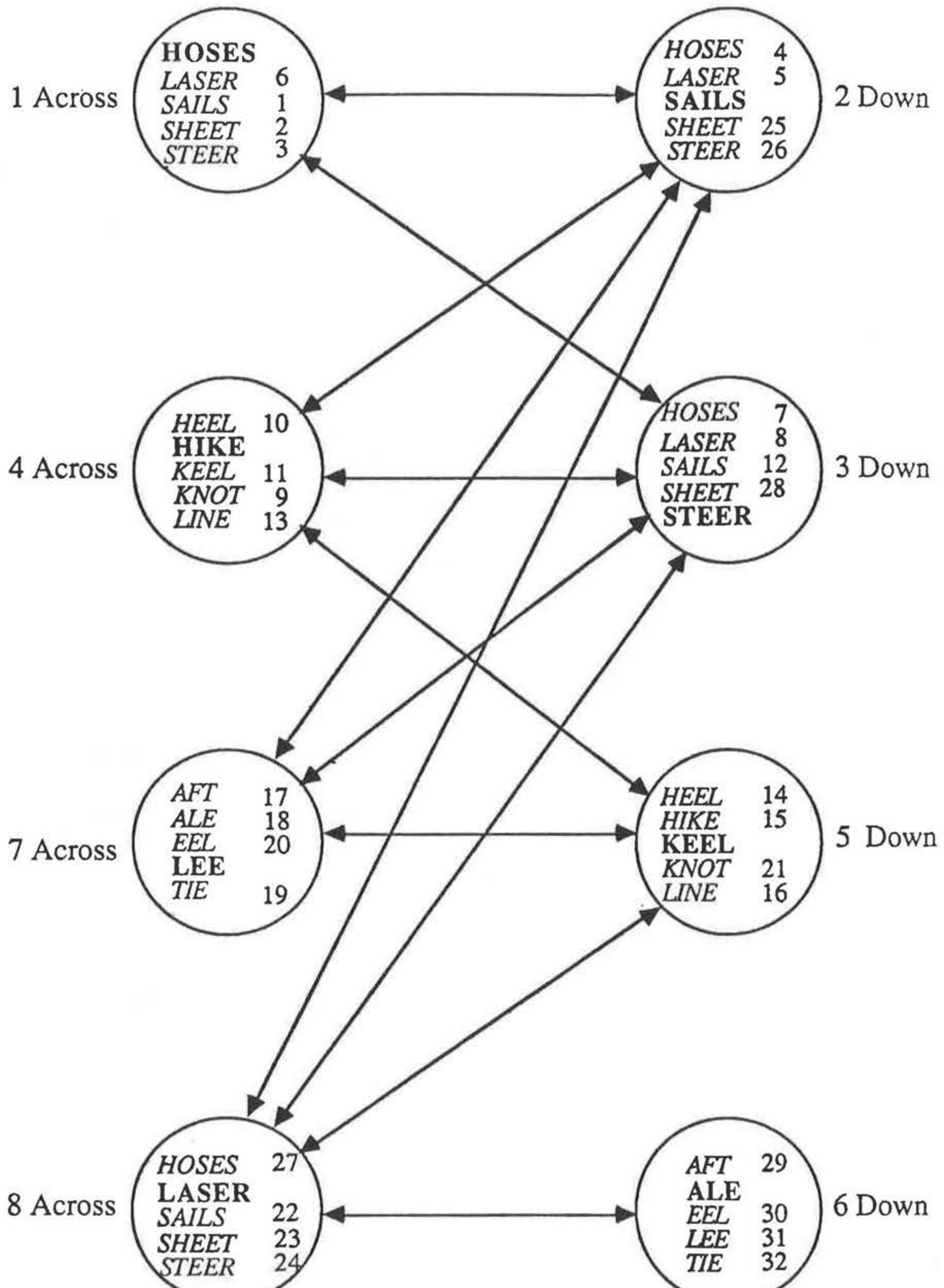


Figure 3. The arc consistent constraint network

Figure 3, the numbers following the deleted words give the order in which they are deleted. Since each domain is eventually reduced to a singleton set of one element, there is an unique solution to the puzzle, shown in Figure 4.

A generalization of this technique is to path consistency (34, 27). A path of length 2 from node i through node m to node j is consistent iff:

$$[(\forall x)(\forall z)P_{ij}(x,z)] \supset (\exists y)(y \in D_m) \wedge P_{im}(x,y) \wedge P_{mj}(y,z).$$

A path is made consistent by deleting entries in the relation matrix representing P_{ij} if it is not. Analogous relaxation and propagation techniques apply.

A further generalization to p-ary relations is the concept of k-consistency $(1 \leq p, k \leq n)$ (12). A network is k-consistent iff given any instantiation of any $(k-1)$ variables satisfying all the direct constraints among those variables it is possible to find an instantiation of any kth variable such that the k values taken together satisfy all the constraints among the k variables. Node, arc and path consistency correspond to k-consistency for $k = 1,2$ and 3, respectively. A network is strongly k-consistent iff it is j-consistent for all $j \leq k$. Another generalization to p-ary relations (28) involves only arc consistency techniques.

Even though a network is strongly k-consistent for $k < n$ there is no guarantee that a solution exists unless each domain is reduced to a singleton. One approach to finding complete solutions is to achieve strong n-consistency (12) but that approach can be very inefficient as Freuder's algorithm for k-consistency is $O(n^k)$ (44). A second approach is to achieve only strong arc consistency. If any node still has more than one element in its domain choose the smallest such

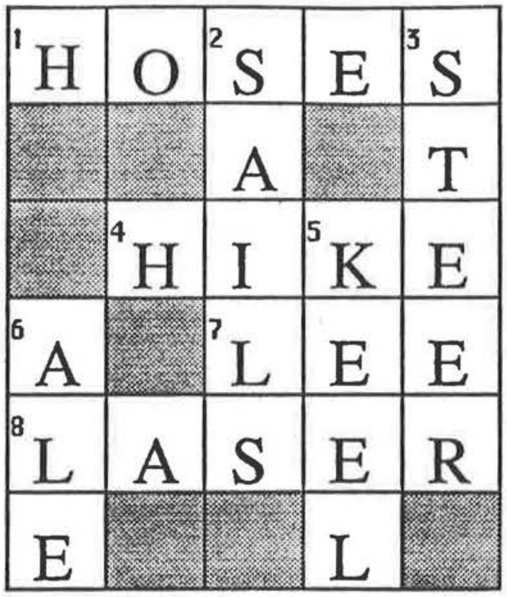


Figure 4. The crossword puzzle solution

domain and recursively apply strong arc consistency to each half of it. Only the arcs coming into that node can initially be inconsistent in the two subproblems generated. A third and related approach is to instantiate the variable with the smallest domain that has more than one value in it and repeat arc consistency recursively, backtracking on failure. Again, initially only the arcs coming into that node can be inconsistent. Or, fourth, one can simply backtrack on the consistent network using any of the techniques in section 3.2. This is the sense in which backtracking and consistency algorithms are complementary. Backtracking is a depth-first instantiation technique whereas consistency is an elimination approach ruling out all solutions containing local inconsistencies in a progressively wider context. Other names for the class of consistency algorithms include discrete relaxation, constraint propagation, domain elimination, range restriction, filtering and full forward look ahead algorithms but these terms do not properly cover the range of consistency techniques described here.

3.4. Applications

As surveyed in (27, 16) various combinations of backtracking and consistency techniques have been suggested for, or actually applied to, finite assignment space puzzles such as cryptarithmetic problems, Instant Insanity, magic and Latin squares, and the n-queens problem (not to mention crossword puzzles). Other applications reported include map coloring, Boolean satisfiability, graph and sub-graph homomorphism and isomorphism, database retrieval for conjunctive queries, theorem proving and spatial layout tasks. The first application in computational vision was to edge labelling (47) but there have been many others reported including sketch map interpretation (28) and consistency for schema-based systems (18). In (48) arc consistency is used on a vision problem in which the domains are not discrete. In that application the domains correspond to a range of allowable surface orientations at various locations in an image of a smooth surface. In general, the only requirement for using consistency is that one be able to carry out restriction operations typified by equation [2] on the descriptions of the domains and relations, which may be intensional rather than extensional.

Various experimental and theoretical results on the running time of these algorithms have been reported (47, 15, 33, 16, 39, 40, 30) but the results must be interpreted with care since the authors are not always discussing the same algorithms, different measures of time are used, some results are task-specific, and some authors analyze the decision problem while others analyze the problem of synthesizing the global n-ary relation, reporting all solutions. More work needs to be done but at this point the situation is that arc consistency techniques can markedly improve the overall efficiency of backtracking algorithms as can the various intelligent backtracking enhancements. The general lesson is that by doing a limited amount of local computation at each level, using, say, linear, quadratic or cubic time, one can optimize backtracking search sufficiently to effect an overall substantial improvement in performance on some difficult problems; however, there is still no adequate theory of how the nature of the task constraints affects the performance of these techniques.

4. RELAXATION ALGORITHMS FOR CONSTRAINED OPTIMIZATION PROBLEMS

The restrictions on the Boolean CSP paradigm can be relaxed in several ways. In computational vision and other AI domains one is often not just *satisfying* a set of Boolean constraints but rather *optimizing* the degree to which a solution satisfies a variety of conflicting continuous constraints. Several generalizations of the consistency techniques have been invented to cope with that problem. In (49) the labels in the discrete domains have associated weights in the unit interval [0,1] and the relation matrices are allowed to have entries from [-1,1]. These entries measure the extent to which two values from related domains are *compatible*. The algorithm looks at each variable domain in parallel adjusting the weight of each label based on an updating rule which adjusts the weight's previous value using the strength of the connection from this variable to each of its neighbouring variables, the compatibility coefficient between this label and each of its neighbour's labels, and the previous weight of that neighbouring label. This process iterates until a fixed point is reached when no significant change occurs in any weight or until some other stopping criterion applies. The details of the various updating and stopping rules used by these so-called *relaxation labelling algorithms* can be found in the surveys in (9, 2) where applications and other variations on this formulation are also given. An interpretation of the weights as probabilities and the compatibilities as Bayesian conditional probabilities was suggested, hence the term *probabilistic relaxation algorithms*. The term relaxation was suggested by the loose analogy with the numerical methods used to solve, say, the heat equation for a steel plate. However, the probabilistic interpretation has several problems of semantics and convergence and other interpretations are now preferred. For example, this class of algorithms can be seen as finding the optimal solution to a linear programming problem as surveyed in (2).

Algorithms in this generic class are often termed *cooperative algorithms* (22, 32). Here the sense is that compatible values in neighbouring domains can cooperatively reinforce each other by increasing each other's weight. Simultaneously, incompatible values compete, trying to suppress each other. Each value in a domain is competing with each of the other values in that domain. This general class of algorithms is attractive because they are highly parallel, requiring only local neighbourhood communication between uniform processors which need only simple arithmetic operations and limited memory. These features suggest various implementations for low level perception (such as stereo vision) in artificial and biological systems, which are being explored (22, 32, 49, 3, 21, 50, 19).

The semantics of these algorithms - the specification of what is being computed - has been clarified (46, 20). The best formal analysis and design of these algorithms is based on the concept of minimization of a figure-of-merit (or "energy") of the system under study. If that surface is everywhere a downwards-convex function of the configuration variables of the system then there is an unique global minimum and steepest descent techniques will find it. If that requirement is not met then techniques such as simulated annealing based on the Metropolis algorithm and Boltzmann distributions (24) are useful.

In (21) an iterative shape-from-shading algorithm is proposed in which a specific figure-of-merit is minimized. The algorithm is given an image of a smooth surface for which the dependence of the grey value on surface orientation is known. Since surface orientation at a point has two degrees of freedom that single constraint is not sufficient. Accordingly, the additional requirement that the surface be as smooth as possible is introduced. The figure-of-merit is a weighted sum of measures of the extent to which these two constraints are violated. The requirement that it be minimized translates analytically to a very large, sparse set of equations on the values of surface orientation at each pixel in the image. That set of equations is solved by standard numerical iterative relaxation techniques using gradient descent, yielding a simple updating rule for approximations to the surface orientation values. Notice, here, however, that the domains no longer consist of a discrete set of possible values with associated weights but simply the best current approximation to the value.

5. OTHER CONSTRAINT-BASED SYSTEMS

The constraint satisfaction approach has considerable attraction both in artificial intelligence and other areas of computer science. In graphics and simulation, constraint propagation is the mechanism underlying two pioneering systems: Sutherland's Sketchpad (44) and Borning's Thinglab (6). Stefik's Molgen system (41) propagates constraints arising at different levels of planning abstraction to generate plans for gene-splicing experiments. Various systems have been implemented for domains such as circuit analysis (42, 23) and job shop scheduling (11). Other applications in computational vision are described in (32, 7, 29). Constraint propagation and dataflow as the design principles for new computational architectures are discussed in (1). Part of the appeal of logic programming (26) is that attention is focussed more on the constraints of the problem, less on the way they are used. There is, for example, less of a distinction between input and output variables in a relational language like Prolog than in a functional language like LISP. Personal computer spreadsheet systems based on Visicalc and its descendant already embody some of these constraint-based ideas. There the variables take only numeric values and the constraints are simple algebraic formulas but some of the latest systems allow relaxation for the solution of mutually dependent constraint sets.

6. CONCLUSIONS

The definition of the word constraint varies enormously. It has been taken to mean a relation over a Cartesian product of sets, a Boolean predicate, a fuzzy relation, a continuous figure-of-merit analogous to energy, an algebraic equation, an inequality, a Horn clause in Prolog and various other arbitrarily complex symbolic relationships. Nevertheless, underlying this variety, a common constraint satisfaction paradigm is emerging. Much of our knowledge of the world is best expressed in terms of what is allowed or, conversely, what is not allowed. Most current artificial computational systems on the other hand, insist on a particular direction of use of that knowledge. This forces the designer or user to over-specify control information leading to undesirable representational redundancy, a rigid input/output dichotomy and conceptual mismatch at the human-computer interface. The constraint propagation paradigm allows the system designer to concentrate on *what* not *how*. In computational vision, for example, it is crucial to determine precisely how an image constrains the equivalence class of scenes that could produce it, and to identify other constraints that will further constrain the scene. The constraints implicit in other knowledge and data sources can be analyzed and represented. These constraints may be uniformly introduced and used in various directions depending on the current availability to the system of specific data and knowledge.

7. BIBLIOGRAPHY

1. Abelson, Harold and Sussman, Gerald Jay, *Structure and Interpretation of Computer Programs*, MIT Press, Cambridge, Mass., 1985.

2. Ballard, Dana H. and Brown, Christopher M., *Computer Vision*, Prentice-Hall Inc., Englewood Cliffs, N.J., 1982.

3. Barrow, Harry G. and Tenenbaum, Jay M., "Recovering intrinsic scene characteristics from images", in E.M. Riseman and A.R. Hanson, (eds.), *Computer Vision Systems*, New York: Academic Press, 3-26, (1978).

4. Bitner, J.R. and Reingold, E.M., "Backtrack programming techniques", *Commun. ACM* 18, 11, 651-656, (Nov. 1975).

5. Bobrow, Daniel G. and Raphael, B., "New programming languages for AI research", *Comput. Surv.* 6, 153-174, (1974).

6. Borning, Alan, "Thinglab: A constraint-oriented simulation laboratory", *Rep. No. CS-79-746, Computer Science Department, Stanford University*, California, (1979).

7. Brooks, Rodney A., "Symbolic reasoning among 3-D models and 2-D images", *Artificial Intelligence* 17(1-3), 285-348, (1981).

8. Bruynooghe, Maurice, "Solving combinatorial search problems by intelligent backtracking", *Information Processing Letters* 12, 1, 36-39, (1981).

9. Davis, Larry S. and Rosenfeld, Azriel, "Cooperating processes for low-level vision: a survey", *Artificial Intelligence* 17, 245-263, (1981).

10. de Kleer, Johan, "Choices without backtracking", *AAAI-84 Proceedings of the National Conference on Artificial Intelligence*, (1984), 79-85.

11. Fox, Mark S., Allen, Brad and Strohm, Gary, "Job-shop scheduling: an investigation in constraint-directed reasoning", *AAAI-82 Proceedings of the National Conference on Artificial Intelligence*,(1982), 155-158.

12. Freuder, Eugene C., "Synthesizing constraint expressions", *Comm. ACM* 21, 958-966, (1978).

13. Freuder, Eugene C., "A sufficient condition for backtrack-free search" *J. ACM* 19, 24-32, (1982).

14. Gaschnig, John A., "A general backtrack algorithm that eliminates most redundant tests", *Proc. International Conference on Artificial Intelligence*, 457, (August 1977), Cambridge, MA.

15. Gaschnig, John, *Performance Measurement and Analysis of Certain Search Algorithms*, (Thesis) CMU-CS-79-124, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Penn., 1979.

16. Haralick, Robert M. and Elliott, G.L., "Increasing tree search efficiency for constraint satisfaction problems", *Artificial Intelligence* 14, 263-313, (1980).

17. Haralick, Robert M. and Shapiro, Linda, "The consistent labeling problem: Part 1", *IEEE Trans. Pattern Anal. Machine Intell.*, PAMI-1, 173-184, (1979).

18. Havens, William S. and Mackworth, Alan K., "Representing knowledge of the visual world", *IEEE Computer* 16, 10, 90-96, (1983).

19. Hinton, Geoffrey E., Sejnowski, Terrence J., and Ackley, David H., "Boltzmann machines: constraint satisfaction networks that learn", *Technical Report CMU-CS-84-119, Department of Computer Science, Carnegie-Mellon University*, Pittsburgh, Pennsylvania, 1984.

20. Hummel, Robert A., and Zucker, Steven W., "On the foundations of relaxation labeling processes", *IEEE Trans. on Pattern Analysis and Machine Intelligence* PAMI-5, 3, 267-287 (1983).

21. Ikeuchi, Katsushi and Horn, Berthold K.P., "Numerical shape from shading and occluding boundaries", *Artificial Intelligence*, 17, 141-184, (1981).

22. Julesz, Bela, *Foundations of Cyclopean Perception*, University of Chicago Press, Chicago (1971).

23. Kelly, Van E. and Steinberg, Louis I., "The Critter system: analyzing digital circuits by propagating behaviors and specifications", *AAAI-82 Proc. of the*

National Conference on Artificial Intelligence, (1982), 284-289.

24. Kirkpatrick, S., Gelatt, C.D., Jr., and Vecchi, M.P., "Optimization by simulated annealing", *Science*, 220, 671-680, (1983).

25. Knuth, Donald E., "Estimating the efficiency of backtrack programs", *Math. Comput.* 29, 121-136, (1975).

26. Kowalski, Robert, "Predicate logic as a programming language", IFIP 74, North-Holland, Amsterdam, 569-574, (1974).

27. Mackworth, Alan K., "Consistency in networks of relations", *Artificial Intelligence*, 8, 1, 99-118, (1977).

28. Mackworth, Alan K., "On reading sketch maps", *Proc. IJCAI-5* , MIT, Cambridge, MA, (1977), 598-606.

29. Mackworth, Alan K., "On seeing things, again", *Proc. Eight International Joint Conference on Artificial Intelligence*, Karslruhe (1983), 1187-1191.

30. Mackworth, Alan K. and Freuder, Eugene C., "The complexity of some polynomial network consistency algorithms for constraint satisfaction problems", *Artificial Intelligence*, 25, 1, 65-74, (1984)

31. Maier, David, *The Theory of Relational Databases*, Computer Science Press, Rockville, Maryland, 1983.

32. Marr, David, *Vision*, San Francisco: W.H. Freeman, 1982.

33. McGregor, J.J., "Relational consistency algorithms and their application in finding subgraph and graph isomorphisms", ***Information Sciences*** **19, 229-250, (1979).**

34. Montanari, Ugo, "Networks of constraints: fundamental properties and applications to picture processing". ***Inform. Sci.*** **7, 95-132, (1974).**

35. Nudel, Bernard, "Consistent-labeling problems and their algorithms", *AAAI-82 Proc. of the National Conference on Artificial Intelligence*, (1982), 128-132.

36. Purdom, Paul W., Jr., and Brown, Cynthia, A., "Evaluating search methods analytically", *AAAI-82 Proc. of the National Conference on Artificial Intelligence*, (1982), 124-127.

37. Purdom, Paul, Brown, Cynthia, and Robertson, Edward, "Multi-level dynamic search rearrangement", *Acta Informatica* 15, 99-114, (1981).

38. Rosenfeld, Azriel, Hummel, Robert A. and Zucker, Steven, W., "Scene labelling by relaxation operations", IEEE Trans. SMC 6, 420-433, (1976).

39. Seidel, Raimund, "A new method for solving constraint satisfaction problems", *Proc. IJCAI-81*, Vancouver, British Columbia, Canada, (1981), 338-342.

40. Seidel, Raimund, "On the complexity of achieving k-consistency", *Technical*

Report 83-4, University of British Columbia, Department of Computer Science, Vancouver, British Columbia, Canada, (1983).

41. Stefik, Mark, "Planning with constraints", *Artificial Intelligence* 16, 111-140, (1981).

42. Stallman, R.M. and Sussman, Gerald J., "Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis", *Artificial Intelligence* 9, 2, 135-196, (1977).

43. Sussman, Gerald J. and McDermott, Drew V., "Why Conniving is better than Planning", *Artificial Intelligence Memo* No. 255A, MIT (1972).

44. Sutherland, Ivan E., "Sketchpad: a man-machine graphical communication system", *MIT Lincoln Lab. Tech. Rep. 296*, Cambridge, MA, 1965.

45. Ullman, Julian R., "Associating parts of patterns", *Information and Control* 9, (6) 583-601, (1966).

46. Ullman, Shimon "Relaxation and constrained optimization by local processes", *Comput. Graphics Image Processing*, 10, 115-125, (1979).

47. Waltz, David "Understanding line drawings of scenes with shadows", in P.H. Winston, (ed.), *The Psychology of Computer Vision*, McGraw-Hill, NY, 1975, 19-91.

48. Woodham, Robert J., "A cooperative algorithm for determining surface

orientation from a single view", *Proc. IJCAI-5*, MIT, Cambridge, MA, (1977), 635-641.

49. Zucker, Steven W., Hummel, Robert A. and Rosenfeld, Azriel, "An application of relaxation labeling to line and curve enhancement", *IEEE Trans. Comput.*, vol. C-26, 394-403, 922-929, (1977).

50. Zucker, Steven W., "Cooperative grouping and early orientation selection", in O.J. Braddick and A.C. Sleigh (eds.), *Physical and Biological Processing of Images*, Springer-Verlag, Berlin, 1983, 326-334.