D. Maier, *The Theory of Relational Databases*, Computer Science Press, Rockville, Md., 1983.

U. Montanari, "Networks of Constraints: Fundamental Properties and Applications to Picture Processing," *Inform. Sci.* **7**, 95–132 (1974).

P. Purdom, "Search Rearrangement Backtracking and Polynomial Average Time," *Artif. Intell.* **21**, 117–133 (1983).

F. Rossi and U. Montanari, "Exact Solution in Linear Time of Networks of Constraints Using Perfect Relaxation," in *Proceedings First International Principles of Knowledge Representation and Reasoning*, Toronto, Ont., Canada, May, 1989, pp. 394–399.

R. Seidel, "A New Method for Solving Constraint-Satisfaction Problems," in *Proceedings of the Seventh IJCAI*, Vancouver, B.C., Canada, Morgan-Kaufmann, San Mateo, Calif., 1981, pp. 338–342.

R. M. Stallman and G. J. Sussman, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," *Artif. Intell.* **9**(2), 135–196 (Oct. 1977).

H. S. Stone and J. M. Stone, *Efficient Search Techniques—An Empirical Study of the N-Queens Problem*, Technical Report RC 12057 (#54343), IBM T. J. Watson Research Center, Yorktown Heights, N.Y., 1986.

R. E. Tarjan and M. Yannakakis, "Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs and Selectively Reduce Acyclic Hypergraphs," *SIAM J. Comput.* **13**(3), 566–579 (Aug. 1984).

D. Waltz, "Understanding Line Drawings of Scenes with Shadows," in P. H. Winston, ed., *The Psychology of Computer Vision*, McGraw-Hill Book Co., Inc., New York, 1975.

R. Zabih, "Some Applications of Graph Bandwidth to Constraint Satisfaction Problems," in *Proceedings of the Ninth National Conference on AI*, Boston, 1990, AAAI, Menlo Park, Calif., 1990, pp. 46–50.

RINA DECHTER
University of California at Irvine

CONSTRAINT SATISFACTION

Constraint satisfaction is an umbrella term for a variety of techniques of AI and related disciplines. In this entry attention is focused 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 is 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 (qv) problem. One standard approach involves backtracking (qv); various forms of "intelligent" backtracking are surveyed. A complementary approach based on the class of consistency algorithms has some nice properties that are described and illustrated.

The second class of problems considered is 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 constraint 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.

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 that there 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 or 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 formalized in this way. Some of them are enumerated below. 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, 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 lay-

out 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 are 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 one goes through the process of looking for a solution.

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

$$\exists x_1 \exists x_2 \cdots \exists x_n (x_1 \in D_1) \wedge (x_2 \in D_2) \wedge \cdots \wedge (x_n \in D_n) \wedge P_1(x_1) \wedge P_2(x_2) \wedge \cdots \wedge P_n(x_n) \wedge P_{12}(x_1, x_2) \wedge P_{13}(x_1, x_3) \wedge \cdots \wedge P_{n-1,n}(x_{n-1}, x_n) \quad (1)$$

Here P_{ij} is included in the formula only 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 are 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 five 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 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 directly constrained variables. In the crossword puzzle constraint network shown in Figure 2, the initial domain of words for a variable is shown inside the vertex for that variable.

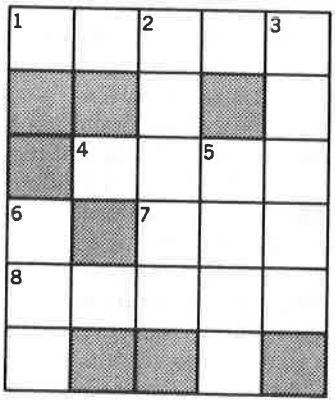

Figure 1. A constraint satifaction problem. Solve the crossword.

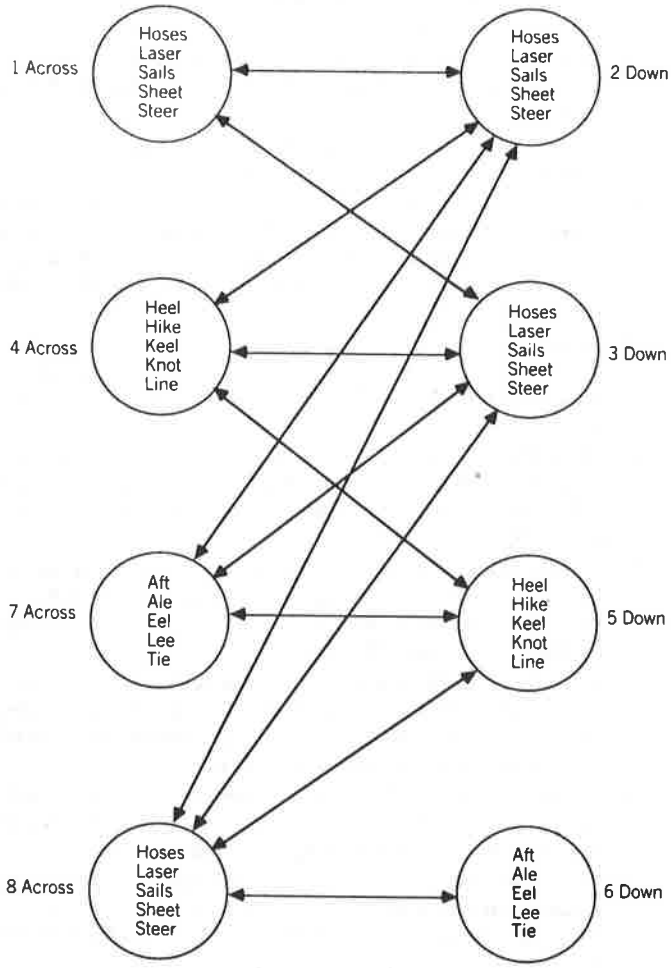

Figure 2. The crossword puzzle constraint network.

Note 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.

BACKTRACKING AND CONSISTENCY ALGORITHMS FOR CONSTRAINT SATISFACTION PROBLEMS

Generate and Test

Assuming finite discrete domains, there is an algorithm to solve any CSP. The assignment space $D = D_1 \times D_2 \times \cdots \times D_n$ 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.

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 1–8. 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 (Bitner and Reingold, 1975; Knuth, 1975; Gaschnig, 1979; Haralick and Elliot, 1980). Good analytical results are hard to come by, but see Haralick and Elliot (1980); Freuder (1982); Nudel (1982); and Purdom and Brown (1982). Other factors being equal, it pays to preorder 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 reordering (Bitner and Reingold, 1975; Purdom and co-workers, 1981) involving one or two more levels of look-ahead search to find the variable with the smallest domain of acceptable values to instantiate next. Regardless of the order of the instantiation, one almost always observes thrashing behavior in backtrack search (Bobrow and Raphael, 1974). *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 (Sussman and McDermott, 1972; Mackworth, 1977). 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 section 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 (Haralick and Shapiro, 1979; Haralick and Elliot, 1980). Another form of intelligent backtracking identifies the latest instantiated variable causing the failure and fails back to it, possibly across many intervening levels (Sussman and McDermott, 1972; Gaschnig, 1979; Bruynooghen, 1981). Gaschnig's (1977) 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 retried elsewhere in the tree.

Similar techniques are exploited in dependency-directed backtracking (Stallman and Sussman, 1977) and truth or belief maintenance systems (de Kleer, 1984) (see BACKTRACKING; TRUTH MAINTENANCE SYSTEMS). 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 are 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 entry as general AI tools should not be overlooked (Sussman and McDermott, 1972; Mackworth, 1977; de Kleer, 1984).

Consistency Algorithms. Another family of algorithms complementary to the class of backtracking has been characterized as the class of consistency algorithms (Mackworth, 1977). By analyzing the various causes of thrashing behavior in backtracking, various authors have described algorithms that eliminate those causes (Ullman, 1966; Montanari, 1974; Waltz, 1975; Mackworth, 1977; Freuder, 1978; Mohr and Henderson, 1986). They are most easily described in the network method of CSPs given earlier. For binary constraints each edge in the graph between vertices i and j is replaced by 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

$$\forall x(x \in D_i) \supset P_i(x) \tag{2}$$

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

$$D_i \leftarrow D_i \cap \{x | P_i(x)\} \tag{3}$$

In the crossword puzzle this corresponds to the obvious strategy of deleting from each variable's domain any word of 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

$$\forall x(x \in D_i) \supset \exists y(y \in D_j) \wedge P_{ij}(x, y) \tag{4}$$

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:

$$D_i \leftarrow D_i \cap \{x | \exists y(y \in D_j) \wedge P_{ij}(x, y)\} \tag{5}$$

In the language of relational database theory this operation is known as a *semijoin* (Maier, 1983). 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 (Waltz, 1975; Mackworth, 1977). The first approach is a symbolic relaxation algorithm and suggests parallel implementation techniques (Rosenfeld and co-workers, 1976). The second is usually more efficient on a single processor. The Waltz (1975) filtering algorithm uses the second approach. The arc consistency algorithm requires time linear in the number of constraints to make the network arc consistent (Mackworth and Freuder, 1984).

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 neighboring 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 behavior can be no worse and may be much better.

The result of applying algorithm AC-3, a serial arc consistency algorithm (Mackworth, 1977), 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 numbered. When words are deleted 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 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 a unique solution to the puzzle, shown in Figure 4.

A generalization of this technique is to path consistency (Montanari, 1974; Mackworth, 1977). A path of length 2 from node i through node m to node j is consistent iff

$$\forall x \forall z P_{ij}(x, z) \supset \exists y(y \in D_m) \wedge P_{im}(x, y) \wedge P_{mj}(y, z) \quad (6)$$

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. If all paths of length 2 are consistent, then all paths are consistent (Montanari, 1974). In general, path consistency uses the operation of relational composition. If the relations are represented as matrices, then binary matrix multiplication implements that operation but other approaches are possible. In Allen (1983), for example, a finite number of possible relations between temporal intervals is specified and their composition table made explicit.

A further generalization to p-ary relations is the concept of k-consistency ($1 \leq p,\ k \leq n$) (Freuder, 1978). 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 (Mackworth, 1977) involves only arc consistency techniques.

Even if 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

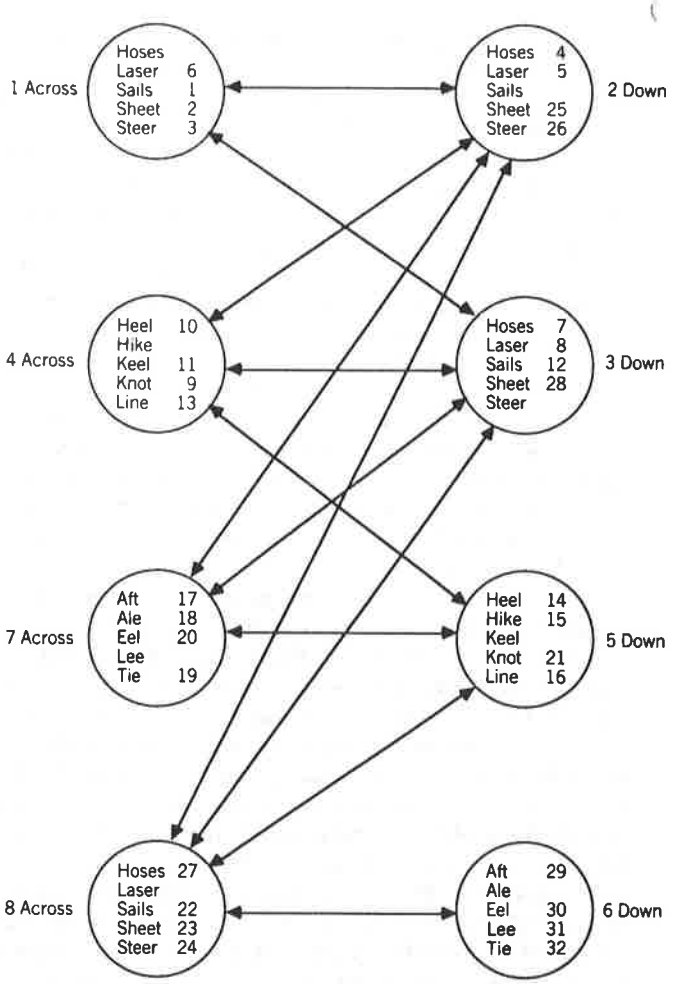

Figure 3. The arc consistent constraint network.

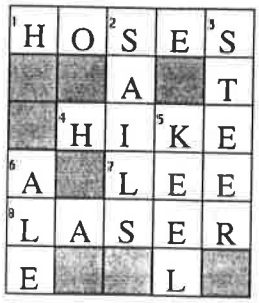

Figure 4. The crossword puzzle solution.

(Freuder, 1978), but that approach can be very inefficient as Freuder's algorithm for k-consistency is $O(n^k)$ (Seidel, 1983). A second approach is to acheive only strong arc consistency. If any node still has more than one element in its domain, choose the smallest such 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 backtracking algorithms shown above. 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.

Applications

As surveyed in Mackworth (1977) and Freuder (1980), various combinations of backtracking and consistency techniques have been 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 subgraph homomorphism and isomorphism, database retrieval for conjunctive queries, theorem proving (qv), temporal reasoning (Allen, 1983; Dechter and co-workers, 1989; Feldman and Golumbic, 1989; van Beek, 1989), and spatial layout tasks. The first application in computational vision was to edge labeling (Waltz, 1975), but there have been many others reported, including sketch map interpretation (Mackworth, 1977) and consistency for schema-based systems (Havens and Mackworth, 1983). In Woodham (1977) 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 eq. 5 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 (Waltz, 1975; Gaschnig, 1979; McGregor, 1979; Haralick and Elliot, 1980; Seidel, 1981, 1983; Mackworth and Freuder, 1984; Dechter and Meiri, 1989), but these 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 and 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.

Some aspects of such a theory are, however, emerging. In particular, the topology of the constraint graph itself is a crucial factor. If it is a tree, then the CSP can be solved in linear time (Freuder, 1982; Mackworth and Freuder, 1984). Other topological properties of the graph may also be exploited (Dechter and Pearl, 1987, 1989; Rossi and Montanari, 1989). Such a theory must also relate CSP consistency approaches to integer linear programming (Rivin and Zabih, 1989), ATMS formulations (de Kleer, 1989), database and graph-theoretic techniques (Dechter and co-workers, 1990), theorem proving (Bibel, 1988), dynamic programming (Seidel, 1981) and propositional satisfiability (Zabih and McAllester, 1988; Reiter and Mackworth, 1989).

The consistency algorithms are serial polynomial approximation algorithms for CSPs. Aspects of parallel complexity have been investigated. Kasif (1986) shows that arc consistency is log-space complete for P, the class of problems solvable on a single Turing machine in polynomial time. The implication of this is that it is unlikely that arc consistency can be solved in (worst-case) polylogarithmic time with a polynomial number of processors. This might be interpreted as saying that the problem of arc consistency is "inherently" sequential. Further results are presented on Kasif (1989). However, Gu and co-workers (1987) present a parallel architecture and the design for the AC Chip (Swain and Cooper, 1988) embodies a highly parallel algorithm for arc consistency implemented directly in VLSI.

Generalizations of the basic Boolean CSP have been considered. If there is no solution to a CSP, one may want to relax the constraints sufficiently to obtain a solution (Descotte and Latombe, 1985; Hertzberg and co-workers, 1988; Freuder, 1989; Freeman-Benson and co-workers, 1990). On the other hand, if there are a large number of possible solutions to the CSP one may wish to define a preference relation or metric on the solution space and find the most preferred solution(s) (Dechter and co-workers, 1990).

RELAXATION ALGORITHMS FOR CONSTRAINT OPTIMIZATION PROBLEMS

The restrictions on the Boolean CSP paradigm can be relaxed in several other 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 a paper by Zucker and co-workers (1977), 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 that adjusts the weight's previous value using the strength of the connection from this variable to each of its neighboring variables, the compatibility coefficient between this label and each of its neighbor's labels, and the previous weight of that neighboring 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-labeling algorithms* can be found in the surveys by Davis and Rosenfeld (1981) and Ballard and Brown (1982), 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 by Ballard and Brown (1982).

Algorithms in this generic class are often termed *cooperative algorithms* (Julesz, 1971; Marr, 1982). Here the sense is that compatible values in neighboring 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 neighborhood communication between uniform processors that 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 (Julesz, 1971; Zucker and co-workers, 1977; Barrow and Tenenbaum, 1978; Ikechuchi and Horn, 1981; Marr, 1982; Zucker, 1983; Hinton and co-workers, 1984).

The semantics of these algorithms—the specification of what is being computed—has been clarified (Ullman, 1979; Hummel and Zucker, 1983). 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 downward convex function of the configuration variables of the system, there is a unique global minimum, and steepest descent techniques will find it. If that requirement is not met, techniques such as simulated annealing (qv) based on the Metropolis algorithm and Boltzmann distributions are useful (Kirkpatrick and co-workers, 1983; Hinton and co-workers, 1984) (see BOLTZMANN MACHINES).

In Ikeuchi and Horn (1981) 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 gray 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 regularizing 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. Note, 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.

OTHER CONSTRAINT-BASED SYSTEMS AND LANGUAGES

The constraint satisfaction approach has considerable attraction both in AI and in other areas of computer science. In graphics and simulation, constraint propagation is the mechanism underlying two pioneering systems: Sutherland's (1965) Sketchpad and Borning's (1979) ThingLab. Stefik's (1981) Molgen system 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 (Stallman and Sussman, 1977; Kelly and Steinberg, 1982), job shop scheduling (Fox and co-workers, 1982), and mechanical design (Mittal and co-workers, 1986). Other applications in computational vision have been described (Brooks, 1981; Marr, 1982; Mackworth, 1983). Constraint propagation and data flow as the design principles for new computational architectures have also been discussed (Abelson and Sussman, 1985). Part of the appeal of logic programming (qv) (Kowalski, 1974) is that attention is focused more on the constraints of the problem and 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 descendants, 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.

A variety of systems that provide constraint-based tools in a programming environment have been proposed and implemented. REF-ARF (qv) (Fikes, 1970), Steele's (1980) constraint language, Bertrand (Leler, 1988), CON-

SAT (Güsgen, 1989), Platypus (Havens and Rehfuss, 1989), and ThingLab II (Maloney and co-workers, 1989; Freeman-Benson and co-workers, 1990), for example, cover a wide spectrum of classes of constraints, constraint-solving algorithms, and user tools.

A language class of particular interest is the set of constraint logic programming (qv) (CLP) languages. These generalize the use of unification in conventional logic programming to constraint solving over various domains. For example, CLP(D) is a scheme for a family of CLP languages parameterized by D, the domain for the constraints (Jaffar and Lassez, 1987). CHIP (van Hentenryck, 1989), CLP(R) (Jaffar and Michaylov, 1987), Prolog III and Trilogy (Voda, 1988) are implemented CLP languages that allow various domains for their constraints. CHIP (qv) augments the PROLOG interpreter with arc consistency and other constraint satisfaction algorithms.

Concurrent logic programming (Foster and Taylor, 1990), another generalization of logic programming, has led to the development of the cc family of concurrent constraint programming languages (Saraswat, 1989). Saraswat advocates the use of constraints for communication and control in concurrent programming languages. The model provides a global constraint store, but this request blocks if the constraint or its negation is not yet entailed. An agent may *Tell* a constraint to the store if the constraint is consistent with the constraints already placed. An agent may also *Ask* if a constraint is entailed by the constraints in the store, but this request blocks if the constraint or its negation is not yet entailed. This protocol provides concurrency control among the agents. Approaches such as this carry the promise of making accessible constraint-based computation systems that are both simple and powerful.

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 one's knowledge of the world is best expressed in terms of what is allowed or, conversely, what is not allowed. On the other hand, most current artificial computational systems insist on a particular direction of use of that knowledge. This forces the designer or user to overspecify control information, leading to undesirable representational redundancy, a rigid input–output dichotomy, and conceptual mismatch at the human–computer interface. The constraint satisfaction 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.

BIBLIOGRAPHY

H. Abelson and G. J. Sussman, *Structure and Interpretation of Computer Programs*, MIT Press, Cambridge, Mass., 1985.

J. F. Allen, "Maintaining Knowledge about Temporal Intervals," *CACM* **26,** 832–843 (1983).

D. H. Ballard and C. M. Brown, *Computer Vision*, Prentice-Hall, Englewood Cliffs, N.J., 1982.

H. G. Barrow and J. M. Tenenbaum, "Recovering Intrinsic Scene Characteristics from Images," in E. M. Riseman and A. R. Hanson, eds., *Computer Vision Systems*, Academic Press, New York, 1978, pp. 3–26.

W. Bibel, "Constraint Satisfaction From a Deductive Viewpoint," *Artif. Intell.* **36,** 401–413 (1988).

J. R. Bitner and E. M. Reingold, "Backtrack Programming Techniques," *CACM* **18**(11), 651–656 (1975).

D. G. Bobrow and B. Raphael, "New Programming Languages for AI Research," *Comput. Surv.* **6,** 153–174 (1974).

A. Borning, *ThingLab: A Constraint-Oriented Simulation Laboratory*, Report No. CS-79-746, Computer Science Dept., Stanford University, Stanford, Calif., 1979.

R. A. Brooks, "Symbolic Reasoning among 3-D Models and 2-D Images," *Artif. Intell.* **17**(1, 3), 285–348 (1981).

M. Bruynooghe, "Solving Combinatorial Search Problems by Intelligent Backtracking," *Inform. Process. Lett.* **12**(1), 36–39 (1981).

L. S. Davis and A. Rosenfeld, "Cooperating Processes for Low-Level Vision: A Survey," *Artif. Intell.* **17,** 245–263 (1981).

R. Dechter and I. Meiri, "Experimental Evaluation of Preprocessing Techniques in Constraint Satisfaction Problems," *Proceedings of the Eleventh IJCAI*, Detroit, Mich., 1989, pp. 271–277.

R. Dechter and J. Pearl, "Network-Based Heuristics for Constraint-Satisfaction Problems," *Artif. Intell.* **34**(1), 1–38 (1987).

R. Dechter and J. Pearl, "Tree Clustering for Constraint Networks," *Artif. Intell.* **38,** 353–366 (1989).

R. Dechter, I. Meiri, and J. Pearl, *Temporal Constraint Networks, Tech. Rep. R-113-L,* Computer Science Dept., UCLA, Los Angeles, Calif., Oct. 1989.

R. Dechter, A. Dechter, and J. Pearl, "Optimization in Constraint Networks," in R. M. Oliver and J. Q. Smith, eds., *Influence Diagrams, Belief Nets and Decision Analysis*, Wiley, New York, 1990.

J. de Kleer, "Choices without Backtracking," *Proceedings of the Fourth National Conference on Artificial Intelligence*, Austin, Tex., 1984, pp. 79–85.

J. de Kleer, "A Comparison of ATMS and CSP Techniques," *Proceedings of the Eleventh IJCAI*, Detroit, Mich., 1989, pp. 290–296.

Y. Descotte and J. C. Latombe, "Making Compromises Among Antagonist Constraints in a Planner," *Artif. Intell.* **27,** 183–217 (1985).

R. Feldman and M. C. Golumbic, "Constraint Satisfiability Algorithms for Interactive Student Scheduling," *Proceedings of the Eleventh IJCAI*, Detroit, Mich., 1989, pp. 1010–1016.

R. E. Fikes, "REF-ARF: A System for Solving Problems Stated as Procedures," *Artif. Intell.* **1,** 27–120 (1970).

I. Foster and S. Taylor, *Strand: New Concepts in Parallel Programming*, Prentice-Hall, Englewood Cliffs, N.J., 1990.

M. S. Fox, B. Allen, and G. Strohm, "Job-Shop Scheduling: An Investigation in Constraint-Directed Reasoning," *Proceedings of the Second National Conference on Artificial Intelligence*, Pittsburgh, Penn., 1982, pp. 155–158.

B. N. Freeman-Benson, J. Maloney, and A. Borning, "An Incremental Constraint Solver," *CACM* **33**(1), 54–63 (1990).

E. C. Freuder, "Synthesizing Constraint Expressions," *CACM* **21,** 958–966 (1978).

E. C. Freuder, "A Sufficient Condition for Backtrack-Free Search," *J. ACM* **19,** 24–32 (1982).

E. C. Freuder, "Partial Constraint Satisfaction," *Proceedings of the Eleventh IJCAI*, Detroit, Mich., 1989, pp. 278–283.

J. A. Gaschnig, "A General Backtrack Algorithm That Eliminates Most Redundant Tests," *Proceedings of the Fifth IJCAI*, Cambridge, Mass., Aug. 1977, p. 457.

J. Gaschnig, Performance Measurement and Analysis of Certain Search Algorithms, Ph.D. dissertation, CMU-CS-79-124, Dept. of Computer Science, Carnegie Mellon University, Pittsburgh, Penn., 1979.

J. Gu, W. Wang, and T. C. Henderson, "A Parallel Architecture for Discrete Relaxation Algorithm," *IEEE Trans. Patt. Anal. Mach. Intell.*, **PAMI-9,** 816–831, 1987.

H. W. Güsgen, *CONSAT: A System for Constraint Satisfaction*, Pitman Publishing, London, 1989.

R. M. Haralick and G. L. Elliott, "Increasing Tree Search Efficiency for Constraint Satisfaction Problems," *Artif. Intell.* **14,** 263–313 (1980).

R. M. Haralick and L. Shapiro, "The Consistent-Labeling Problem: Part 1," *IEEE Trans. Patt. Anal. Mach. Intell.* **PAMI-1,** 173–184 (1979).

W. S. Havens and A. K. Mackworth, "Representing Knowledge of the Visual World," *IEEE Trans. Comput.* **C-16**(10), 90–96 (1983).

W. S. Havens and P. S. Rehfuss, "Platypus: A Constraint-Based Reasoning System," *Proceedings of the Eleventh IJCAI*, Detroit, Mich., 1989, pp. 48–53.

J. Hertzberg, H. W. Güsgen, A. Voss, M. Fidelak, and H. Voss, "Relaxing Constraint Networks to Resolve Inconsistencies," in W. Hoeppner, ed., *Kuenstliche Intelligenz—GWAI-88*, Springer, Berlin, 1988, pp. 61–65.

G. W. Hinton, T. J. Sejnowski, and D. H. Ackley, *Boltzmann Machines: Constraint Satisfaction Networks That Learn*, Technical Report CMU-CS-84-119, Dept. of Computer Science, Carnegie Mellon University, Pittsburgh, Penn., 1984.

R. A. Hummel and S. W. Zucker, "On the Foundations of Relaxation Labeling Processes," *IEEE Trans. Patt. Anal. Mach. Intell.* **PAMI-5**(3), 267–287 (1983).

K. Ikeuchi and B. K. P. Horn, "Numerical Shape from Shading and Occluding Boundaries," *Artif. Intell.* **17,** 141–184 (1981).

J. Jaffar and J.-L. Lassez, "Constraint Logic Programming," *Proceedings of the Fourteenth ACM Principles of Programming Languages Conference*, Munich, 1987, pp. 111–119.

J. Jaffar and S. Michaylov, "Methodology and Implementation of a CLP System" in *Proceedings of the Fourth International Conference on Logic Programming*, Melbourne, 1987, pp. 196–218.

B. Julesz, *Foundations of Cyclopean Perception*, University of Chicago Press, Chicago, 1971.

S. Kasif, "On the Parallel Complexity of Some Constraint Satisfaction Problems," *Proceedings of the Fifth National Conference on AI*, 1986, pp. 349–353.

S. Kasif, "Parallel Solutions to Constraint Satisfaction Problems," *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, 1989, pp. 180–188.

V. E. Kelly and L. I. Steinberg, "The Critter System: Analyzing Digital Circuits by Propagating Behaviors and Specifications," *Proceedings of the Second National Conference on Artificial Intelligence*, Pittsburgh, Penn., 1982, pp. 284–289.

S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," *Science* **220,** 671–680 (1983).

D. E. Knuth, "Estimating the Efficiency of Backtrack Programs," *Math. Comput.* **29,** 121–136 (1975).

R. Kowalski, *Predicate Logic as a Programming Language*, IFIP 74, North-Holland, Amsterdam, 1974, pp. 569–574.

W. Leler, *Constraint Programming Languages: Their Specification and Generation*. Addison-Wesley, Reading, Mass., 1988.

A. K. Mackworth, "Consistency in Networks of Relations," *Artif. Intell.* **8**(1), 99–118 (1977).

A. K. Mackworth, "On Reading Sketch Maps," *Proceedings of the Fifth IJCAI*, Cambridge, Mass., 1977, pp. 598–606.

A. K. Mackworth, "On Seeing Things, Again," *Proceedings of the Eighth IJCAI*, Karlsruhe, FRG, 1983, pp. 1187–1191.

A. K. Mackworth and E. C. Freuder, "The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems," *Artif. Intell.* **25**(1), 65–74 (1984).

D. Maier, *The Theory of Relational Databases*, Computer Science Press, Rockville, Md., 1983.

J. Maloney, A. Borning, and B. Freeman-Benson, "Constraint Technology for User-Interface Construction in ThingLab II," *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*, New Orleans, 1989, pp. 381–388.

D. Marr, *Vision*, W. H. Freeman, San Francisco, 1982.

J. J. McGregor, "Relational Consistency Algorithms and Their Application in Finding Subgraph and Graph Isomorphisms," *Inform. Sci.* **19,** 229–250 (1979).

S. Mittal, C. L. Dym, and M. Morjaria, "PRIDE: An Expert System for the Design of Paper Handling Systems" *IEEE Comput.* 102–114 (July 1986).

R. Mohr and T. C. Henderson, "Arc and Path Consistency Revisited," *Artif. Intell.* **28**(2), 225–233 (1986).

U. Montanari, "Networks of Constraints: Fundamental Properties and Applications to Picture Processing," *Inform. Sci.* **7,** 95–132 (1974).

B. Nudel, "Consistent-Labeling Problems and Their Algorithms," *Proceedings of the Second National Conference on Artificial Intelligence*, Pittsburgh, Penn., 1982, pp. 128–132.

P. Purdom, C. Brown, and E. Robertson, "Multi-Level Dynamic Search Rearrangements," *Acta Inform.* **15,** 99–114 (1981).

P. W. Purdom, Jr., and C. A. Brown, "Evaluating Search Methods Analytically," *Proceedings of the Second National Conference on Artificial Intelligence*, Pittsburgh, Penn., 1982, pp. 124–127.

R. Reiter and A. Mackworth, "A Logical Framework for Depiction and Image Interpretation," *Artif. Intell.* **41,** 125–155 (1989).

I. Rivin and R. Zabih, "An Algebraic Approach to Constraint Satisfaction Problems," *Proceedings of the Eleventh IJCAI*, Detroit, 1989, pp. 284–289.

A. Rosenfeld, R. A. Hummel, and S. W. Zucker, "Scene Labeling by Relaxation Operations," *IEEE Trans. Syst. Man Cybern.* **SMC-6,** 420–433 (1976).

F. Rossi and U. Montanari, "Exact Solution in Linear Time of Networks of Constraints Using Perfect Relaxation," *Proceedings First Int. Conf. on Principles of Knowledge Representation*, Toronto, 1989, pp. 394–399.

V. A. Saraswat, *Concurrent Constraint Programming Languages*, Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, Penn., Jan 1989.

R. Seidel, "A New Method for Solving Constraint Satisfaction Problems," *Proceedings of the Seventh IJCAI*, Vancouver, B.C., Canada, 1981, pp. 338–342.

R. Seidel, *On the Complexity of Achieving k-Consistency, Tech. Report 83-4*, University of British Columbia, Dept. of Computer Science, Vancouver, B.C., Canada, 1983.

R. M. Stallman and G. J. Sussman, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," *Artif. Intell.* **9**(2), 135–196 (1977).

G. L. Steele, *The Definition and Implementation of a Computer Programming Language Based on Constraints*, Technical Report AI-TR 595, MIT, Cambridge, Mass., 1980.

M. Stefik, "Planning with Constraints," *Artif. Intell.* **16**, 111–140 (1981).

G. I. Sussman and D. V. McDermott, "Why Conniving is Better than Planning," *Artificial Intelligence Memo No. 255A*, MIT, Cambridge, Mass., 1972.

I. E. Sutherland, *Sketchpad: A Man-Machine Graphical Communication System*, MIT Lincoln Laboratory Technical Report 296, Cambridge, Mass., 1965.

M. J. Swain and P. R. Cooper, "Parallel Hardware for Constraint Satisfaction," *Proceedings of the Seventh National Conference on AI*, St. Paul, Minn., 1988, pp. 682–686.

J. R. Ullman, "Associating Parts of Patterns," *Inform. Contrl.* **9**(6), 583–601 (1966).

S. Ullman, "Relaxation and Constrained Optimization by Local Processes," *Comput. Graph. Image Proc.* **10**, 115–125 (1979).

P. van Beek, "Approximation Algorithms for Temporal Reasoning" *Proceedings of the Eleventh IJCAI*, Detroit, 1989, pp. 1291–1296.

P. van Hentenryck, *Constraint Satisfaction in Logic Programming*, MIT Press, Cambridge, Mass., 1989.

P. Voda, *The Constraint Language Trilogy: Semantics and Computations*, Technical Report, Complete Logic Systems, North Vancouver, B.C., Canada, 1988.

D. Waltz, "Understanding Line Drawings of Scenes with Shadows," in P. H. Winston, ed., *The Psychology of Computer Vision*, McGraw-Hill, New York, 1975, pp. 19–91.

R. J. Woodham, "A Cooperative Algorithm for Determining Surface Orientation From a Single View," *Proceedings of the Fifth IJCAI*, Cambridge, Mass., 1977, pp. 635–641.

R. Zabih and D. McAllester, "A Rearrangement Search Strategy for Determining Propositional Satisfiability," *Proceedings of the Seventh National Conference on AI*, St. Paul, Minn., 1988, pp. 155–160.

S. W. Zucker, "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, pp. 326–334.

S. W. Zucker, R. A. Hummel, and A. Rosenfeld, "An Application of Relaxation Labeling to Line and Curve Enhancement," *IEEE Trans. Comput.* **C-26**, 394–403, 922–929 (1977).

ALAN K. MACKWORTH
University of British Columbia

CONTROL STRUCTURES

It is necessary to distinguish control structures from algorithms and from virtual machines. Control concerns what happens in a computational process. The word *control* has two different meanings. On the one hand, there is the problem of ensuring that a process does as little work as necessary. Thus, the term *search control* is applied to a search that explores a minimum of wrong paths. This is more properly a matter for the study of algorithms. On the other hand, there is the problem of specifying clearly what should happen in a computational process. The notion of a control structure concerns this problem of specification. It is a more general notion than that of an algorithm, although no precise line can be drawn.

The notion of a control structure must also be distinguished from the notion of a virtual machine. For simplicity, this article assumes that a virtual machine is defined by a programming language, which might be a machine language. A virtual machine provides the programmer with a collection of primitive operations (no matter if they have a direct physical embodiment in the architecture of a real machine). More important, though, a virtual machine provides an ontology of objects and processes, on top of which programmers can build their own abstractions. The notion of control originated in the days when computers all had simple von Neumann architectures and a programmer needed no more sophisticated a metaphor for control than simply running a finger through the code. The conclusion of this article suggests that the notion of control can become inappropriate on a virtual machine that departs substantially from this model.

Finally, it is necessary to distinguish between a particular control structure and a whole philosophy and style of programming. For example, object-oriented programming is a style that requires a particular control structure, the familiar type-dispatching procedure call. [With a sufficiently rigid model of types, the outcome of this dispatch can be determined at compile time (Liskov and co-workers, 1981). Thus, a control structure can be entirely a fiction of the virtual machine.] This distinction is particularly important for the history of AI because of the frequency with which subtle and profound philosophies of programming are melted down to catalogs of control and data structures. A control structure must be analyzed in the context of a coherent philosophy of programming.

A control structure is a technique, especially one set down as a linguistic construct, that an algorithm can use in determining what happens when on some virtual machine. This article does not exhaustively treat all the different control structures, because many of them are treated in their own articles in this encyclopedia. Instead, this article outlines current issues and describes the history of AI researchers' attitudes toward process organization in general. (For discussion of particular control structures, see AGENDA-BASED SYSTEMS; BACKTRACKING; BLACKBOARD SYSTEMS; COROUTINES; DISTRIBUTED PROBLEM SOLVING; LANGUAGES, OBJECT-ORIENTED; LOGIC PROGRAMMING; MEANS-ENDS ANALYSIS; META-KNOWLEDGE, META-RULES, AND META-REASONING; PARSING, WORD-EXPERT; PROCESSING, BOTTOM-UP AND TOP-DOWN; and RULE-BASED SYSTEMS. For discussion of languages, sys-