Suchman, L. (1987). *Plans and situated actions: The problem of human-machine communication.* Cambridge, UK: Cambridge University Press.

Sun, C., & Chen, D. (2002). Consistency maintenance in real-time collaborative graphics editing systems. *ACM Transactions on Computer Human Interaction, 9*(1), 1–41.

Tang, J., Yankelovich, N., Begole, J., Van Kleek, M., Li, F., & Bhalodia, J. (2001). Connexus to awarenex: Extending awareness to mobile users. In *Proceedings of Human Factors in Computing Systems: CHI 2001* (pp. 221–228). New York: ACM Press.

Winograd, T. (1987/1988). A language/action perspective on the design of cooperative work. *Human-Computer Interaction, 3*(1), 3–30.

CONSTRAINT SATISFACTION

Constraint satisfaction refers to a set of representation and processing techniques useful for modeling and solving combinatorial decision problems; this paradigm emerged from the artificial intelligence community in the early 1970s. A constraint satisfaction problem (CSP) is defined by three elements: (1) a set of decisions to be made, (2) a set of choices or alternatives for each decision, and (3) a set of constraints that restrict the acceptable combinations of choices for any two or more decisions. In general, the task of a CSP is to find a consistent solution—that is, a choice for every decision such that all the constraints are satisfied. More formally, each decision is called a *variable*, the set of alternative choices for a given variable is the set of *values* or *domain* of the variable, and the *constraints* are defined as the set of allowable combinations of assignments of values to variables. These combinations can be given in extension as the list of consistent *tuples*, or defined in intention as a predicate over the variables.

The 4-Queen Problem

A familiar example of a CSP is the 4-queen problem. In this problem, the task is to place four queens on a 4×4 chessboard in such a way that no two queens attack each other. One way to model the 4-queen problem as a CSP is to define a decision variable for each square on the board. The square can be either empty (value 0) or have a queen (value 1). The constraints specify that exactly four of the decision variables have value 1 ("queen in this square") and that there cannot be two queens in the same row, column, or diagonal. Because there are sixteen variables (one for each square) and each can take on two possible values, there are a total of 2^{16} (65,536) possible assignments of values to the decision variables. There are other ways of modeling the 4-queen problem within the CSP framework. One alternative is to treat each row on the board as a decision variable. The values that can be taken by each variable are the four column positions in the row. This formulation yields 4^4 (256) possibilities. This example illustrates how the initial formulation or model affects the number of possibilities to be examined, and ultimately the performance of problem solving.

CSP Representations

A CSP is often represented as an undirected graph (or network), which is a set of nodes connected by a set of edges. This representation opens up the opportunity to exploit the properties and algorithms developed in graph theory for processing and solving CSPs. In a constraint graph, the nodes represent the variables and are labeled with the domains of the variables. The edges represent the constraints and link the nodes corresponding to the variables to which the constraints apply. The *arity* of a constraint designates the number of variables to which the constraint applies, and the set of these variables constitutes the *scope* of the constraint. Constraints that apply to two variables are called *binary* constraints and are represented as edges in the graph. Constraints that apply to more than two variables are called *nonbinary* constraints. While, early on, most research has focused on solving binary CSPs, techniques for solving nonbinary CSPs are now being investigated.

The Role of CSPs in Science

Beyond puzzles, CSPs have been used to model and solve many tasks (for example, temporal reasoning, graphical user interfaces, and diagnosis) and have been applied in many real-world settings (for example, scheduling, resource allocation, and product configuration and design). They have been used in various areas of engineering, computer science, and management to handle decision problems. A natural extension of the CSP is the constrained optimization problem (COP), where the task is to find an optimal solution to the problem given a set of preferences and optimization criteria. The problems and issues studied in the constraint processing (CP) community most obviously overlap with those investigated in operations research, satisfiability and theoretical computer science, databases, and programming languages. The 1990s have witnessed a sharp increase in the interactions and cross-fertilization among these areas.

A special emphasis is made in CP to maintain the expressiveness of the representation. Ideally, a human user should be able to naturally express the various relations governing the interactions among the entities of a given problem without having to recast them in terms of complex mathematical models and tools, as would be necessary in mathematical programming. The area of constraint *reformulation* is concerned with the task of transforming the problem representation in order to improve the performance of problem solving or allow the use of available solution techniques. Sometimes such transformations are truthful (that is, they preserve the essence of the problem), but often they introduce some sufficient or necessary approximations, which may or may not be acceptable in a particular context.

Solution Methods

The techniques used to solve a CSP can be divided into two categories: *constraint propagation* (or inference) and *search*. Further, search can be carried out as a *systematic, constructive* process (which is exhaustive) or as an *iterative repair* process (which often has a stochastic component).

Constraint Propagation

Constraint propagation consists in eliminating, from the CSP, combinations of values for variables that cannot appear in any solution to the CSP. Consider for example two CSP variables A and B representing two events. Assume that A occurred between 8 a.m. and 12 p.m. (the domain of A is the interval [8, 12]), B occurred between 7 a.m. and 11 a.m. (the domain of B is the interval [7, 11]), and B occurred one hour after A (B-A ≥ 1). It is easy to infer that the domains of A and B must be restricted to [8, 10] and [9, 11] respectively, because B cannot possibly occur before 9, or A after 10, without violating the constraint between A and B. This filtering operation considers every combination of two variables in a binary CSP. It is called 2-consistency. A number of formal properties have been proposed to characterize the extent to which the alternative combinations embedded in a problem description are likely to yield consistent solutions, as a measure of how "close is the problem to being solved." These properties characterize the level of *consistency* of the problem (for example, k-consistency, *minimality*, and decomposability).

Algorithms for achieving these properties, also known as constraint propagation algorithms, remain the subject of intensive research. Although the cost of commonly used constraint propagation algorithms is a polynomial function of the number of variables of the CSP and the size of their domains, solving the CSP remains, in general, an exponential-cost process. An important research effort in CP is devoted to finding formal relations between the level of consistency in a problem and the cost of the search process used for solving it. These relations often exploit the topology of the constraint graph or the semantic properties of the constraint. For example, a tree-structured constraint graph can be solved backtrack-free after ensuring 2-consistency, and a network of constraints of bounded differences (typically used in temporal reasoning) is solved by ensuring 3-consistency.

Systematic Search

In systematic search, the set of consistent combinations is explored in a tree-like structure starting from a root node, where no variable has a value, and considering the variables of the CSP in sequence. The tree is typically traversed in a depth-first manner. At a given depth of the tree, the variable under consideration (*current variable*) is assigned a value from its domain. This operation is called *variable instantiation*. It is important that the value chosen for the current variable be consistent with the instantiations of the past variables. The process of checking the consistency of a value for the current variable with the assignments of past variables is called *backchecking*. It ensures that only instantiations that are consistent (partial solutions) are explored. If a consistent value is found for the current variable, then this variable is added to the list of past variables and a new current variable is chosen from among the un-instantiated variables (future variables). Otherwise (that is, no consistent value exists in the domain of the current variable), backtracking is applied. Backtracking undoes the assignment of the previously instantiated variable, which becomes the current variable, and the search process attempts to find another value in the domain of this variable.

The process is repeated until all variables have been instantiated (thus yielding a solution) or backtrack has reached the root of the tree (thus proving that the problem is not solvable). Various techniques for improving the search process itself have been proposed. For systematic search, these techniques include *intelligent backtracking* mechanisms such as *backjumping* and conflict-directed backjumping. These mechanisms attempt to remember the reasons for failure and exploit them during search in order to avoid exploring barren portions of the search space, commonly called *thrashing*. The choices of the variable to be instantiated during search and that of the value assigned to the variable are handled, respectively, by *variable* and *value ordering heuristics*, which attempt to reduce the search effort. Such heuristics can be applied statically (that is, before the search starts) or dynamically (that is, during the search process). The general principles that guide these selections are "the most constrained variable first" and "the most promising value first." Examples of the former include the least domain heuristic (where the variable with the smallest domain is chosen for instantiation) and the minimal-width heuristic (where the variables are considered in the ordering of minimal width of the constraint graph).

Iterative-Repair Search

In iterative repair (or *iterative improvement*) search, all the variables are instantiated (usually randomly) regardless of whether or not the constraints are satisfied. This set of complete instantiations, which is not necessarily a solution, constitutes a state. Iterative-repair search operates by moving from one state to another and attempting to find a state where all constraints are satisfied. This move operator and the state evaluation function are two important components of an iterative-repair search. The move is usually accomplished by changing the value of one variable (thus the name *local* search). However, a technique operating as a multiagent search allows any number of variables to change their values. The evaluation function measures the cost or quality of a given state, usually in terms of the number of broken constraints. Heuristics, such as the min-conflict heuristic, are used to choose among the states reachable from the current state (neighboring states).

The performance of iterative-repair techniques depends heavily on their ability to explore the solution space. The performance is undermined by the existence in this space of local optima, plateaux, and other singularities caused by the nonconvexity of the constraints. Heuristics are used to avoid falling into these traps or to recover from them. One heuristic, a breakout strategy, consists of increasing the weight of the broken constraints until a state is reached that satisfies these constraints. Tabu search maintains a list of states to which search cannot move back. Other heuristics use stochastic noise such as random walk and simulated annealing.

Blending Solution Techniques

Constraint propagation has been successfully combined with backtrack search to yield effective lookahead strategies such as forward checking. Combining constraint propagation with iterative-repair strategies is less common. On the other hand, randomization, which has been for a long time utilized in local search, is now being successfully applied in backtrack search.

Research Directions

The use of constraint processing techniques is widespread due to the success of the constraint programming paradigm and the increase of commercial tools and industrial achievements. While research on the above topics remains active, investigations are also invested in the following directions: user interaction; discovery and exploitation of symmetry relations; propagation algorithms for high-arity constraints and for continuous domains; preference modeling and processing; distributed search techniques; empirical assessment of problem difficulty; and statistical evaluation and comparison of algorithms.

Berthe Y. Choueiry

See also Artificial Intelligence; N-grams

FURTHER READING

Bistarelli, S., Montanari, U., & Rossi, F. (1997). Semiring-based constraint satisfaction and optimization. *Journal of the ACM, 44*(2), 201–236.

Borning, A., & Duisberg, R. (1986). Constraint-based tools for building user interfaces. *ACM Transactions on Graphics, 5*(4), 345–374.

Cohen, P. R. (1995). *Empirical methods for artificial intelligence.* Cambridge, MA: MIT Press.

Dechter, R. (2003). *Constraint processing.* San Francisco: Morgan Kaufmann.

Ellman, T. (1993). Abstraction via approximate symmetry. In *Proceedings of the 13th IJCAI* (pp. 916–921). Chambéry, France.

Freuder, E. C. (1982). A sufficient condition for backtrack-free search. *JACM, 29*(1), 24–32.

Freuder, E. C. (1985). A sufficient condition for backtrack-bounded search. *JACM, 32*(4), 755–761.

Freuder, E. C. (1991). Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of AAAI-91* (pp. 227–233). Anaheim, CA.

Gashnig, J. (1979). *Performance measurement and analysis of certain search algorithms.* Pittsburgh, PA: Carnegie-Mellon University.

Glaisher, J. W. L. (1874). On the problem of the eight queens. *Philosophical Magazine, 4*(48), 457–467.

Glover, F. (1989). Tabu Search—Part I. *ORSA Journal on Computing, 1*(3), 190–206.

Gomes, C. P. (2004). Randomized backtrack search. In M. Milano (Ed.), *Constraint and Integer Programming: Toward a Unified Methodology* (pp. 233–291). Kluwer Academic Publishers.

Haralick, R. M., & Elliott, G. L. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence, 14*, 263–313.

Hogg, T., Huberman, B. A., & Williams, C. P. (Eds.). (1996). Special volume on frontiers in problem solving: Phase transitions and complexity. *Artificial Intelligence, 81*(1–2). Burlington, MA: Elsevier Science.

Hooker, J. (2000). *Logic-based methods for optimization: Combining optimization and constraint satisfaction.* New York: Wiley.

Hoos, H. H., & Stützle, T. (2004). *Stochastic local search.* San Francisco: Morgan Kaufmann.

Kirkpatrick, S., Gelatt, J. C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680.

Liu, J., Jing, H., & Tang, Y. Y. (2002). Multi-agent oriented constraint satisfaction. *Artificial Intelligence, 136*(1), 101–144.

Minton, S., et al. (1992). Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence, 58*, 161–205.

Montanari, U. (1974). Networks of constraints: Fundamental properties and application to picture processing. *Information Sciences, 7*, 95–132.

Prosser, P. (1993). Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence, 9*(3), 268–299.

Régin, J.-C. (1994). A filtering algorithm for constraints of difference in constraint satisfaction problems. In *Proceedings from the National Conference on Artificial Intelligence (AAAI 1994)* (pp. 362–437). Seattle, WA.

Revesz, P. (2002). *Introduction to constraint databases.* New York: Springer.

Stuckey, K. M. (1998). *Programming with constraints: An introduction.* Cambridge, MA: MIT Press.

Tsang, E. (1993). *Foundations of constraint satisfaction.* London, UK: Academic Press.

Yokoo, M. (1998). *Distributed constraint satisfaction.* New York: Springer.

CONVERGING TECHNOLOGIES

Human-computer interaction (HCI) is a multidisciplinary field arising chiefly in the convergence of computer science, electrical engineering, information technology, and cognitive science or psychology. In the future it is likely to be influenced by broader convergences currently in progress, reaching out as far as biotechnology and nanotechnology. Together, these combined fields can take HCI to new levels where it will unobtrusively but profoundly enhance human capabilities to perceive, to think, and to act with maximum effectiveness.

The Basis for Convergence

During the twentieth century a number of interdisciplinary fields emerged, bridging the gaps between separate traditionally defined sciences. Notable examples are astrophysics (astronomy plus physics), biochemistry (biology plus chemistry), and cognitive science (psychology plus neurology plus computer science). Many scientists and engineers believe that the twenty-first century will be marked by a broader unification of all of the sciences, permitting a vast array of practical breakthroughs—notably in the convergence of nanotechnology, biotechnology, information technology, and cognitive technology—