# Constraint Satisfaction

Rina Dechter

Department of Computer and Information Science

University of California, Irvine

Irvine, California, USA 92717

dechter@@ics.uci.edu

A *constraint satisfaction problem (csp)* defined over a *constraint network* consists of a finite set of *variables*, each associated with a *domain* of values, and a set of *constraints*. A *solution* is an assignment of a value to each variable from its domain such that all the constraints are satisfied. Typical constraint satisfaction problems are to determine whether a solution exists, to find one or all solutions and to find an optimal solution relative to a given cost function. An example of a constraint satisfaction problem is the well known $k$-colorability. The problem is to color, if possible, a given graph with $k$ colors only, such that any two adjacent nodes have different colors. A constraint satisfaction formulation of this problem associates the nodes of the graph with variables, the possible colors are their domains and the inequality constraints between adjacent nodes are the constraints of the problem. Each constraint of a csp may be expressed as a relation, defined on some subset of variables, denoting their legal combinations of values. As well, constraints can be described by mathematical expressions or by computable procedures. Another known constraint satisfaction problem is *SATisfiability*; the task of finding the truth assignment to propositional variables such that a given set of clauses are satisfied. For example, given the two clauses $(A \vee B \vee \neg C), (\neg A \vee D)$, the assignment of *false* to $A$, *true* to $B$, *false* to C and *false* to $D$, is a satisfying truth value assignment.

The structure of a constraint network is depicted by a constraint graph whose nodes represents the variables and any two nodes are connected if the corresponding variables participate in the same constraint. In the $k$-colorability formulation, the graph to be colored is the constraint graph. In our SAT example the constraint graph has $A$ connected to $D$ and $A, B$ and $C$ are connected to each other.

Constraint networks have proven successful in modeling mundane cognitive tasks such as vision, language comprehension, default reasoning, and abduction, as well as in applications such as scheduling, design, diagnosis, and temporal and spatial reasoning. In general, constraint satisfaction tasks are computationally intractable (NP-hard) (see COMPUTATIONAL COMPLEXITY).

Techniques for processing constraints can be classified into two categories:(1) search and (2) consistency inference, and these techniques interact. Search algorithms traverse the space of partial instantiations while consistency-inference algorithms reason through equivalent problems. Search is either systematic and complete, or stochastic and incomplete. Likewise, consistency-inference has complete solution algorithms (e.g., variable-elimination), or incomplete versions in the form of *local consistency* algorithms.

**Consistency inference**. *Local consistency* (or consistency-enforcing, or constraint propagation) algorithms [15, 13, 6] are polynomial algorithms that transform a given constraint network into an equivalent, yet more explicit network by deducing new constraints to be added onto the network. Intuitively, a consistency-enforcing algorithm will make any partial solution of a small subnetwork extensible to some surrounding network. For example, the most basic consistency algorithm, called *arc-consistency* ensures that any legal value in the domain of a single variable has a legal match in the domain of any other selected variable. *Path-consistency* ensures that any consistent solution to a two-variable subnetwork is extensible to any third variable, and, in general, $i$*-consistency* algorithms guarantee that any locally consistent instantiation of $i-1$ variables is extensible to any $i^{th}$ variable. Enforcing $i$-consistency can be accomplished in time and space exponential in $i$. Algorithms for $i$-consistency frequently decide *inconsistency*.

A network is said to be *globally consistent*, if it is $i$-consistent for *every* $i$. This means that a solution can be assembled by assigning values using any variable ordering without encountering any dead-end, namely in a *backtrack-free* manner. However, it is enough to posses *directional* global consistency relative to a given ordering, only. Indeed, algorithm *adaptive consistency*, a *variable elimination* algorithm, enforces global consistency in a given order only, such that every solution can be extracted with no dead-ends along this ordering. Another related algorithm called *tree-clustering* compiles the given constraint problem into an equivalent tree of subproblems [4] whose respective solutions can be combined into a complete solution, efficiently. Adaptive-consistency and tree-clustering are time and space exponential in a parameter of the constraint graph called induced-width ( or tree-width) [1, 3].

When a problem is computationally hard for adaptive-consistency it can be solved by bounding the amount of consistency-enforcing (e.g. arc- or path-consistency), and augmenting the algorithm with a search component. Generally speaking, search will benefit from network representations that have a high level of consistency. However, because the complexity of enforcing $i$-consistency is exponential in $i$, there is a trade-off between the effort spent on consistency inference and that spent on search. Theoretical and empirical studies of this tradeoff, prior to- or during search, aims at identifying a problem-dependent cost-effective balance [10, 17, 21, 5].

**Search**. The most common algorithm for performing systematic search is *backtracking*, which traverses the space of partial solutions in a depth-first manner. At each step the algorithm extends a partial solution by assigning a value to one more variable. When a variable is encountered such that none of its values are consistent with the partial solution (a situation referred to as a *dead-end*), backtracking takes place. The algorithm is time exponential, but require only linear space.

Improvements of backtracking algorithm have focused on the two phases of the algorithm: moving forward (look-ahead schemes) and backtracking (look-back schemes) [2]. When moving forward, to extend a partial solution, some computation (e.g., arc-consistency) is carried out to decide which variable and value to choose next. For variable orderings, variables that maximally constrain the rest of the search space are preferred. For value selection, however, the least constraining value is preferred, in order to maximize future options for instantiations [10, 3, 18, 21].

Look-back schemes are invoked when the algorithm encounters a dead-end. These schemes perform two functions: One, decide how far to backtrack, by analyzing the reasons for the dead-end, a process often referred to as *backjumping* [8]. Two, record the reasons for the dead-end in the form of new constraints so that the same conflicts will not arise again, known as *constraint learning* and *no-good recording* [23, 2].

*Stochastic local search* strategies have been recently reintroduced into the satisfiability and constraint satisfaction literature under the umbrella name *GSAT (Greedy SATisfiability)* (see GREEDY LOCAL SEARCH). These methods move in hill-climbing manner in the space of complete instantiations to all the variables [20]. The algorithm improves its current instantiation by 'flipping' a value of a variable that maximize the number of constraints satisfied. Such search algorithms are incomplete, may get stuck in a local maxima and cannot prove inconsistency. Nevertheless, when equipped with some heuristics for randomizing the search or for revising the guiding criterion function (constraint rewighting), they prove successful in solving large and hard problems that are frequently hard for backtracking-style search [22].

*Structure-driven algorithms* cut across both search and consistency-inference algorithms. These techniques emerged from an attempt to topologically characterize constraint problems that are tractable. *Tractable classes* were generally recognized by realizing that enforcing low-level consistency (in polynomial time) guarantees global consistency for some problems. The basic network structure that support tractability is a tree [14]. In particular, enforcing arc-consistency on a tree-structured network ensures global consistency along some ordering. Most other graph-based techniques can be viewed as transforming a given network into a meta-tree. We have already seen *adaptive-consistency, tree-clustering*, and *constraint learning*, all of which are time and space exponentially bounded by the *tree-width* of the constraint graph, the *cycle-cutset scheme* combining search and inference, which is exponentially bounded by the constraint graph's *cycle-cutset*, the *bi-connected component method*, which is bounded by the size of the constraint graph's largest bi-connected component [6]; and backjumping, which is exponentially bounded by the depth of the graph's depth-first-search tree. The latter three methods require only polynomial space.

Tractable classes were also identified by the properties of the constraints themselves. Such tractable classes exploit notions such as tight domains and tight constraints [25], row-convex constraints [24], implicational and max-ordered constraints [11, 16], as well as causal networks. A connection between tractability and algebraic closure was recently discovered [16].

Finally, special classes of constraints associated with temporal reasoning has received much attention in the last decade. Tractable classes include subsets of qualitative interval algebra [9], expressing relationships such as time interval $A$ overlaps or precede time interval $B$, as well as quantitative binary linear inequalities over the Real numbers of the form $X - Y \leq a$ [19] (see TEMPORAL REASONING).

**Evaluation of algorithms.** Theoretical evaluation of constraint satisfaction algorithms is accomplished primarily by worst-case analysis, (i.e., determining a function of problem's size that upper bounds the algorithm's performance over all problems of that size) or by dominance relationships [12]. However, as worst-case analysis by its nature is too pessimistic and often does not reflect actual performance, empirical evaluation is necessary. Normally, a proposed algorithm is evaluated empirically on a set of randomly generated instances taken from the relatively "hard" "phase transition" region [22]. Other benchmarks based on real-life applications such as scheduling are also used. Currently, dynamic variable ordering and value selection heuristics that use various forms of constraint inference, backjumping as well as constraint learning, have been shown to be very effective for various problem classes. [17, 7, 21].

# References


[1] S. Arnborg and A. Proskourowski. Linear time algorithms for np-hard problems restricted to partial $k$-trees. *Discrete and Applied Mathematics*, 23:11–24, 1989.

[2] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[3] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.

[4] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, pages 353–366, 1989.

[5] R. Dechter and I. Rish. Directional resolution: The davis-putnam procedure, revisited. In *Principles of Knowledge Representation and Reasoning (KR-94)*, pages 134–145, 1994.

[6] E. C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982.

[7] D. Frost and R. Dechter. In search of best search: An empirical evaluation. In *AAAI-94: Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 301–306, Seattle, WA, 1994.

[8] J. Gaschnig. Performance measurement and analysis of search algorithms. Technical Report CMU-CS-79-124, Carnegie Mellon University, 1979.

[9] M. C. Golumbic and R. Shamir. Complexity and algorithms for reasoning about time: a graph-theoretic approach. *Journal of the ACM*, 40:1108–1133.

[10] M. Haralick and G. L. Elliot. Increasing tree-search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[11] L. M. Kirousis. Fast parallel constraint satisfaction. *Artificial Intelligence*, 64:147–160, 1993.

[12] G. Kondrak and P. van Beek. A theoretical valuation of selected algorithms. *Artificial Intelligence*, 89:365–387, 1997.

[13] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[14] A. K. Mackworth and E. C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25, 1985.

[15] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7(66):95–132, 1974.

[16] D. Cohean P. Jeavons and M. Gyssens. A unifying framework for tractable constraints. In *Constraint Programming (CP95)*, France, 1995.

[17] P. Prosser. Hybrid algorithms for constraint satisfaction problems. *Computational Intelligence*, 9(3):268–299, 1993.

[18] P. W. Purdom. Search rearangement backtracking and polynomial average time. *Artificial Intelligence*, 21:117–133, 1983.

[19] I. Meiri R. Dechter and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1990.

[20] A.B. Philips S. Minton, M.D. Johnston and P. Laired. Solving large scale constraint satisfaction and scheduling problems using heuristic repair methods. In *National Conference on Artificial Intelligence (AAAI-90)*, pages 17–24, Anaheim, CA, 1990.

[21] D. Sabin and E. C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *ECAI-94*, pages 125–129, Amsterdam, 1994.

[22] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, 1992.

[23] M. Stallman and G. J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9(2):135–196, 1977.

[24] P. van Beek and R. Dechter. On the minimality and decomposability of row-convex constraint networks. *Journal of the ACM*, 42:543–561, 1995.

[25] P. van Beek and R. Dechter. Constraint tightness and looseness versus local and global consistency. *Journal of the ACM (in press)*, 1997.

# Further readings

# References

[1] A. B. Baker. Intelligent Backtracking on constraint satisfaction problems: experimental and theoretical results. *Phd thesis, Graduate school of the university of Oregon*, Oregon 1995.

[2] R. Bayardo and D. Mirankar. A complexity analysis of space-bound learning algorithms for the constraint satisfaction problem. In *AAAI-96: Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 298–304, 1996.

[3] S. Bistarelli and U. Montanari and F Rossi. Semiring-based Constraint Satisfaction and Optimization. *journal of the Association of Computing Machinery (JACM)*, volume = "to appear", 1998.

[4] D. A. Cohen M. C. Cooper and P.G. Jeavons. Characterizing tractable constraints. *Artificial Intelligence*, 65:347–361, 1994.

[5] R. Dechter. Constraint networks. *Encyclopedia of Artificial Intelligence*, pages 276–285, 1992.

[6] R. Dechter and I. Meiri. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. *Artificial Intelligence*, 68:211–241, 1994.

[7] R. Dechter and P. van Beek. Local and global relational consistency. *Theoretical Computer Science*, pages 283–308, 1997.

[8] R. Dechter and D. Frost. Backtracking algorithms for constraint satisfaction problems; a survey. *Constraints*, International Journal, to appear 1998.

[9] D. Frost Algorithms and Heuristics for Constraint Satisfaction Problems. *Phd Thesis, Information and Computer Science*, University of California, Irvine, 1997.

[10] , M. L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25-46, 1993.

[11] , V. Kumar. Algorithms for constraint satisfaction problems: a survey, *AI magazine*, 13(1), 1992, 32-44.

[12] A. K. Mackworth. Constraint Satisfaction. *Encyclopedia of Artificial Intelligence*, pages 285–293, 1992.

[13] E. Schwalb, and R. Dechter. Processing disjunctions in temporal constraint networks. *Artificial Intelligence*, 93(1-2), 29–61, 1997.

[14] E. Tsang. Foundation of Constraint Satisfaction. *Academic press*, 1993.