Constraint satisfaction problem

From Wikipedia, the free encyclopedia

Constraint satisfaction problems (CSPs) are mathematical problems defined as a set of objects whose state must satisfy a number of constraints or limitations. CSPs represent the entities in a problem as a homogeneous collection of finite constraints over variables, which is solved by constraint satisfaction methods. CSPs are the subject of intense research in both artificial intelligence and operations research, since the regularity in their formulation provides a common basis to analyze and solve problems of many unrelated families. CSPs often exhibit high complexity, requiring a combination of heuristics and combinatorial search methods to be solved in a reasonable time. The boolean satisfiability problem (SAT), the Satisfiability Modulo Theories (SMT) and answer set programming (ASP) can be roughly thought of as certain forms of the constraint satisfaction problem.

Examples of simple problems that can be modeled as a constraint satisfaction problem

- Eight queens puzzle
- Map coloring problem
- Sudoku, Futoshiki, Kakuro (Cross Sums), Numbrix, Hidato and many other logic puzzles.

Examples demonstrating the above are often provided with tutorials of ASP, boolean SAT and SMT solvers. In the general case, constraint problems can be much harder, and may not be expressible in some of these simpler systems.

"Real life" examples include planning and resource allocation.

Contents

Formal definition

Formally, a constraint satisfaction problem is defined as a triple $\langle X, D, C \rangle$, where [1]

$X = \{X_1, \ldots, X_n\}$ is a set of variables,
$D = \{D_1, \ldots, D_n\}$ is a set of the respective domains of values, and
$C = \{C_1, \ldots, C_m\}$ is a set of constraints.

Each variable X_i can take on the values in the nonempty domain D_i. Every constraint $C_j \in C$ is in turn a pair $\langle t_j, R_j \rangle$, where $t_j \subset X$ is a subset of k variables and R_j is an k-ary relation on the corresponding subset of domains D_j. An *evaluation* of the variables is a function from a subset of variables to a particular set of values in the corresponding subset of domains. An evaluation v satisfies a constraint $\langle t_j, R_j \rangle$ if the values assigned to the variables t_j satisfies the relation R_j.

An evaluation is *consistent* if it does not violate any of the constraints. An evaluation is *complete* if it includes all variables. An evaluation is a *solution* if it is consistent and complete; such an evaluation is said to *solve* the constraint satisfaction problem.

Resolution of CSPs

Constraint satisfaction problems on finite domains are typically solved using a form of search. The most used techniques are variants of backtracking, constraint propagation, and local search.

Backtracking is a recursive algorithm. It maintains a partial assignment of the variables. Initially, all variables are unassigned. At each step, a variable is chosen, and all possible values are assigned to it in turn. For each value, the consistency of the partial assignment with the constraints is checked; in case of consistency, a recursive call is performed. When all values have been tried, the algorithm backtracks. In this basic backtracking algorithm, consistency is defined as the satisfaction of all constraints whose variables are all assigned. Several variants of backtracking exists. Backmarking improves the efficiency of checking consistency. Backjumping allows saving part of the search by backtracking "more than one variable" in some cases. Constraint learning infers and saves new constraints that can be later used to avoid part of the search. Look-ahead is also often used in backtracking to attempt to foresee the effects of choosing a variable or a value, thus sometimes determining in advance when a subproblem is satisfiable or unsatisfiable.

Constraint propagation techniques are methods used to modify a constraint satisfaction problem. More precisely, they are methods that enforce a form of local consistency, which are conditions related to the consistency of a group of variables and/or constraints. Constraint propagation has various uses. First, it turns a problem into one that is equivalent but is usually simpler to solve. Second, it may prove satisfiability or unsatisfiability of problems. This is not guaranteed to happen in general; however, it always happens for some forms of constraint propagation and/or for some certain kinds of problems. The most known and used form of local consistency are arc consistency, hyper-arc consistency, and path consistency. The most popular constraint propagation method is the AC-3 algorithm, which enforces arc consistency.

Local search methods are incomplete satisfiability algorithms. They may find a solution of a problem, but they may fail even if the problem is satisfiable. They work by iteratively improving a complete assignment over the variables. At each step, a small number of variables are changed value, with the overall aim of increasing the number of constraints satisfied by this assignment. The min-conflicts algorithm is a local search algorithm specific for CSPs and based in that principle. In practice, local search appears to work well when these changes are also affected by random choices. Integration of search with local search have been developed, leading to hybrid algorithms.

Theoretical aspects of CSPs

Decision problems

CSPs are also studied in computational complexity theory and finite model theory. An important question is whether for each set of relations, the set of all CSPs that can be represented using only relations chosen from that set is either in P or NP-complete. If such a dichotomy theorem is true, then CSPs provide one of the largest known subsets of NP which avoids NP-intermediate problems, whose existence was demonstrated by Ladner's theorem under the assumption that P ≠ NP. Schaefer's dichotomy theorem handles the case when all the available relations are boolean operators, that is, for domain size 2. Schaefer's dichotomy theorem was recently generalized to a larger class of relations.[2]

Most classes of CSPs that are known to be tractable are those where the hypergraph of constraints has bounded treewidth (and there are no restrictions on the set of constraint relations), or where the constraints have arbitrary form but there exist essentially non-unary polymorphisms of the set of constraint relations.

Every CSP can also be considered as a conjunctive query containment problem.[3]

Function problems

A similar situation exists between the functional classes FP and #P. By a generalization of Ladner's theorem, there are also problems in neither FP nor #P-complete as long as FP ≠ #P. As in the decision case, a problem in the #CSP is defined by a set of relations. Each problem takes as input a Boolean formula as input and the task is to compute the number of satisfying assignments. This can be further generalized by using larger domain sizes and attaching a weight to each satisfying assignment and computing the sum of these weights. It is known that any complex weighted #CSP problem is either in FP or #P-hard.[4]

Variants of CSPs

The classic model of Constraint Satisfaction Problem defines a model of static, inflexible constraints. This rigid model is a shortcoming that makes it difficult to represent problems easily.[5] Several modifications of the basic CSP definition have been proposed to adapt the model to a wide variety of problems.

Dynamic CSPs

Dynamic CSPs[6] (*DCSP*s) are useful when the original formulation of a problem is altered in some way, typically because the set of constraints to consider evolves because of the environment.[7] DCSPs are viewed as a sequence of static CSPs, each one a transformation of the previous one in which variables and constraints can be added (restriction) or removed (relaxation). Information found in the initial formulations of the problem can be used to refine the next ones. The solving method can be classified according to the way in which information is transferred:

- Oracles: the solution found to previous CSPs in the sequence are used as heuristics to guide the resolution of the current CSP from scratch.
- Local repair: each CSP is calculated starting from the partial solution of the previous one and repairing the inconsistent constraints with local search.
- Constraint recording: new constraints are defined in each stage of the search to represent the learning of inconsistent group of decisions. Those constraints are carried over the new CSP problems.

Flexible CSPs

Classic CSPs treat constraints as hard, meaning that they are *imperative* (each solution must satisfy all them) and *inflexible* (in the sense that they must be completely satisfied or else they are completely violated). **Flexible CSP**s relax those assumptions, partially *relaxing* the constraints and allowing the solution to not comply with all them. This is similar to preferences in preference-based planning. Some types of flexible CSPs include:

- MAX-CSP, where a number of constraints are allowed to be violated, and the quality of a solution is measured by the number of satisfied constraints.
- Weighted CSP, a MAX-CSP in which each violation of a constraint is weighted according to a predefined preference. Thus satisfying constraint with more weight is preferred.
- Fuzzy CSP model constraints as fuzzy relations in which the satisfaction of a constraint is a continuous function of its variables' values, going from fully satisfied to fully violated.

Decentralized CSPs

In DCSPs[8] each constraint variable is thought of as having a separate geographic location. Strong constraints are placed on information exchange between variables, requiring the use of fully distributed algorithms to solve the constraint satisfaction problem.

See also

- Constraint programming
- Declarative programming
- Distributed Constraint Satisfaction Problem (DisCSP)
- Unique games conjecture
- Weighted constraint satisfaction problem (WCSP)

References

1. Stuart Jonathan Russell, Peter Norvig (2010). *Artificial Intelligence: A Modern Approach*. Prentice Hall. p. Chapter 6. ISBN 9780136042594.
2. Bodirsky, Manuel; Pinsker, Michael (2011). "Schaefer's theorem for graphs" (http://arxiv.org/pdf/1011.2894v3). *Proceedings of the 43rd Annual Symposium on Theory of Computing (STOC '11)*. Association for Computing Machinery. pp. 655–664. arXiv:1011.2894 (https://arxiv.org/abs/1011.2894). Bibcode:2010arXiv1011.2894B (http://adsabs.harvard.edu/abs/2010arXiv1011.2894B). doi:10.1145/1993636.1993724 (https://dx.doi.org/10.1145%2F1993636.1993724). ISBN 978-1-4503-0691-1.
3. Kolaitis, Phokion G.; Vardi, Moshe Y. (2000). "Conjunctive-Query Containment and Constraint Satisfaction". *Journal of Computer and System Sciences* **61** (2): 302–332. doi:10.1006/jcss.2000.1713 (https://dx.doi.org/10.1006%2Fjcss.2000.1713).
4. Cai, Jin-Yi; Chen, Xi (2012). *Complexity of counting CSP with complex weights*. pp. 909–920. arXiv:1111.2384 (https://arxiv.org/abs/1111.2384). doi:10.1145/2213977.2214059 (https://dx.doi.org/10.1145%2F2213977.2214059). ISBN 978-1-4503-1245-5.
5. Miguel, Ian (July 2001). *Dynamic Flexible Constraint Satisfaction and its Application to AI Planning* (Ph.D. thesis). University of Edinburgh School of Informatics. hdl:1842/326 (http://hdl.handle.net/1842%2F326); CiteSeerX: 10.1.1.9.6733 (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.9.6733).
6. Dechter, R. and Dechter, A., Belief Maintenance in Dynamic Constraint Networks In Proc. of AAAI-88, 37-42. [1] (http://www.ics.uci.edu/%7Ecsp/r5.pdf)
7. Solution reuse in dynamic constraint satisfaction problems (http://www.aaai.org/Papers/AAAI/1994/AAAI94-302.pdf), Thomas Schiex
8. Duffy, K.R.; Leith, D.J. (August 2013), "Decentralized Constraint Satisfaction" (http://dx.doi.org/10.1109/TNET.2012.2222923), *IEEE/ACM Transactions on Networking, 21(4)*, pp. 1298–1308

Further reading

- Steven Minton, Andy Philips, Mark D. Johnston, Philip Laird (1993). "Minimizing Conflicts: A Heuristic Repair Method for Constraint-Satisfaction and Scheduling Problems" (https://eprints.kfupm.edu.sa/50799/1/50799.pdf) (PDF). *Journal of Artificial Intelligence Research* **58**: 161–205. doi:10.1016/0004-3702(92)90007-k (https://dx.doi.org/10.1016%2F0004-3702%2892%2990007-k).

- CSP Tutorial (http://4c.ucc.ie/web/outreach/tutorial.html)

- Tsang, Edward (1993). *Foundations of Constraint Satisfaction* (http://www.bracil.net/edward/FCS.html). Academic Press. ISBN 0-12-701610-4
- Chen, Hubie (December 2009). "A Rendezvous of Logic, Complexity, and Algebra". *ACM Computing Surveys* (ACM) **42** (1): 1–32. arXiv:cs/0611018v1 (https://arxiv.org/abs/cs/0611018v1). doi:10.1145/1592451.1592453 (https://dx.doi.org/10.1145%2F1592451.1592453).
- Dechter, Rina (2003). *Constraint processing* (http://www.ics.uci.edu/~dechter/books/index.html). Morgan Kaufmann. ISBN 1-55860-890-7
- Apt, Krzysztof (2003). *Principles of constraint programming*. Cambridge University Press. ISBN 0-521-82583-0
- Lecoutre, Christophe (2009). *Constraint Networks: Techniques and Algorithms* (http://www.iste.co.uk/index.php?f=a&ACTION=View&id=250). ISTE/Wiley. ISBN 978-1-84821-106-3
- Tomás Feder, *Constraint satisfaction: a personal perspective* (http://theory.stanford.edu/~tomas/consmod.pdf), manuscript.
- Constraints archive (http://4c.ucc.ie/web/archive/index.jsp)
- Forced Satisfiable CSP Benchmarks of Model RB (http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/benchmarks.htm)
- Benchmarks -- XML representation of CSP instances (http://www.cril.univ-artois.fr/~lecoutre/research/benchmarks/benchmarks.html)
- Dynamic Flexible Constraint Satisfaction and Its Application to AI Planning (http://www.cs.st-andrews.ac.uk/~ianm/docs/Thesis.ppt), Ian Miguel - slides.
- Constraint Propagation (http://www.ps.uni-sb.de/Papers/abstracts/tackDiss.html) - Dissertation by Guido Tack giving a good survey of theory and implementation issues

Categories: Constraint programming
