Constraint Programming Languages for Big Data Applications

Francesca Rossi[1] and Vijay Saraswat[2]

[1] University of Padova and Harvard University
[2] IBM TJ Watson Research Center

To tame the data tsunami, the fields of data mining and large-scale machine learning are developing new algorithms (e.g., clustering, regression, frequent item set mining, stochastic gradient descent), programming frameworks (such as map-reduce), and implementations (Hadoop2, [9], Google Project Sibyl). Today, programs routinely perform tasks such as personalized recommendations, multi-language translation, face recognition, speech understanding, spam filtering, fraud detection for electronic payments, that were impossible just a decade ago. For these tasks, the tools that data scientists use – systems like R or the Hadoop tool chain (Pig, Hive, DML,) – offer varying and diverse levels of abstraction.

The Constrained-X Scenarios. Recently, several research groups have pointed out a key challenge in applying these ideas and tools to real world problems. In many situations what is desired is a *constrained* version of the data analysis problem (we will call this the *constrained-X* approach).

For instance, in a particular *clustering* setting [7], the data scientist may wish to assert that some points necessarily belong in the same cluster, or must be scattered across different clusters, that the size of each cluster must be above (or below) a given threshold, that the objective of clustering is to maximize or minimize some domain-specific objective function (e.g. minimizing the maximum diameter of a cluster). Also in *item set mining* domain-specific constraints such as bounds on the size, maximality, frequency or cost of an itemset may need to be specified [8]. In regression, downstream stages of the analysis tool-chain may impose certain constraints on coefficients (constrained regression analysis). In [18], music scores are generated following the *style* of an artist. To do that, a Markov process is generated from the corpora of music scores of an artist, and then new scores are generated in that style while satisfying additional music-related constraints, such as the endpoint constraint (a melody must start and end on the same note), or the meter constraint [21].

In some cases, constraints may be soft [1], that is, better expressed as preferences, such as in personalized medical diagnosis or child tutoring systems, or in specifying ethical rules for robots's behaviour. In many of these scenarios, preferences are expressed over large collections of items (symptoms, behaviors, routes, etc) and often need to be extracted from a large corpus of actions, text, links and structured data.

Scaling Issues. A natural approach to tackle constrained-X scenarios is to use *constraint reasoning* [20]. In this approach the entire problem is formulated through a collection of constraints, perhaps with an optimization criterion, and submitted to a constraint solver, such as [12] and [14]. Such solvers are generally organized around backtrack / branch-and-bound search, use propagation rules and global constraints to efficiently propagate the consequences of choices for decision variables, learn from failures, and use (possibly user specified) heuristics for choosing the next variable to instantiate, and the value to use to instantiate it.

However, a significant challenge of this approach is that it does not yet scale, even when techniques like symmetry breaking are being used to prune the search space [19]. The current state of the

art solvers are able to handle thousands of variables, and exploit intra-node concurrency in their implementation (e.g. running on 8 threads in a single process on a single node). This is several orders of magnitude short of what is desired in many real-world settings (billions of variables, solvers capable of using hundreds of nodes).

Constraint Programming Languages. We argue that the way forward is to significantly increase the flexbility of the constraint programming toolchain by opening up the closed runtimes of existing constraint solvers so that compilers can analyze the source program and produce custom code for propagation, choices and symmetry-breaking.

Twenty years ago, the CP community was founded around a broad vision of constraint programming that included the idea that constraints could be used in general purpose programming languages as a mechanism for communication and control. The initial exemplar of constraint programming languages was CLP [15], followed soon by concurrent constraint programming (CCP) [22]. In the last twenty years, significant theoretical development on CCP has seen the emergence of a programming model founded around the use of logic [16] to combine constraints and probabilistic computation [13]. Significant progress has been made on handling reactive computation (in the context of discrete and continuous time [23]), and semantically-based techniques such as abstract interpretation [4, 10] and algorithmic debugging [11, 5].

This path was in large part eschewed by the constraint reasoning community because of the relative failure of complex, ambitious CCP language efforts such as AKL and Oz. Instead, the focus has been on developing efficient solvers applicable to a variety of real industrial problems. However, there has been some recent work on (constraint programming) languages (languages for constraint programming) such as MiniZinc [17] that are re-introducing the notion of a language (and compiler) for constraint programming, albeit in a recursion-free context where the range of user-defined types is restricted.

Programming = Logic + Constraints + Probability. We are developing a new constraint programming language, called C10, based on the CCP framework, and implemented on top of X10 [3], a language for scale-out computation. C10 is intended for use in the areas of constraint-solving, probabilistic programming, machine learning, and big data analytics. It is a pure, declarative, implicitly concurrent, statically-typed, object-oriented, timed, probabilistic [13] realization of the CCP framework. C10 is intended to be compiled to the high-performance, multi-node, concurrent programming language X10 [3], but is not itself expected to have explicit concurrency and distribution constructs. C10 permits recursive queries against the constraint store, thus subsuming pure (constraint) logic programming. It also exploits random variables ([13]to represent various probabilistic graphical models (Bayesian networks, Markov networks, probabilistic CP nets [2, 6]) directly as programs.

References

1. S. Bistarelli, U. Montanari, and F. Rossi. Soft Concurrent Constraint Programming. *ACM Trans. Comput. Logic*, 7(3):563–589, July 2006.
2. C. Boutilier, R. I. Brafman, C. Domshlak, H. H. Hoos, and D. Poole. CP-nets: A Tool for Representing and Reasoning with Conditional Ceteris Paribus Preference Statements. *J. Artif. Int. Res.*, 21(1):135–191, Feb. 2004.
3. P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar. X10: an Object-Oriented Approach to Non-uniform Cluster Computing. *SIGPLAN Not.*, 40(10):519–538, Oct. 2005.

4. C. Codognet and P. Codognet. A generalized semantics for concurrent constraint languages and their abstract interpretation. In M. Meyer, editor, *Constraint Processing*, volume 923 of *Lecture Notes in Computer Science*, pages 39–49. Springer Berlin Heidelberg, 1995.
5. M. Comini, L. Titolo, and A. Villanueva García. Abstract diagnosis for timed concurrent constraint programs. *Theory and Practice of Logic Progamming*, 11(4-5), 2011.
6. C. Cornelio, J. Goldsmith, N. Mattei, F. Rossi, and K. B. Venable. Updates and Uncertainity in CP-nets. In *Proceedings of the Australian Conference on Artificial Intelligence*, pages 301–312, 2013.
7. T.-B.-H. Dao, K.-C. Duong, and C. Vrain. A declarative framework for constrained clustering. In H. Blockeel, K. Kersting, S. Nijssen, and F. elezn, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 8190 of *Lecture Notes in Computer Science*, pages 419–434. Springer Berlin Heidelberg, 2013.
8. L. De Raedt, T. Guns, and S. Nijssen. Constraint programming for data mining and machine learning. In *AAAI*, 2010.
9. J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng. Large scale distributed deep networks. In *NIPS*, 2012.
10. M. Falaschi, C. Olarte, and C. Palamidessi. A framework for abstract interpretation of timed concurrent constraint programs. In *Proceedings of the 11th ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, PPDP '09, pages 207–218, New York, NY, USA, 2009. ACM.
11. M. Falaschi, C. Olarte, C. Palamidessi, and F. Valencia. Declarative diagnosis of temporal concurrent constraint programs. In V. Dahl and I. Niemel, editors, *Logic Programming*, volume 4670 of *Lecture Notes in Computer Science*, pages 271–285. Springer Berlin Heidelberg, 2007.
12. Gecode Team. Gecode: Generic constraint development environment, 2013. Available from `http://www.gecode.org`.
13. V. Gupta, R. Jagadeesan, and V. Saraswat. Probabilistic Concurrent Constraint Programming. In A. Mazurkiewicz and J. Winkowski, editors, *CONCUR '97: Concurrency Theory, 8th International Conference*, volume 1243 of *Lecture Notes in Computer Science*, pages 243–257, Warsaw, Poland, 1–4 July 1997. Springer-Verlag.
14. IBM. Ibm ilog cp optimizer, 2014. http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r6/index.jsp.
15. J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *Proceedings of the 14th Annual ACM Symposium on Principles of Programming Languages (POPL'87), Munich, Germany*, pages 111–119. ACM Press, New York (NY), USA, 1987.
16. R. Jagadeesan, G. Nadathur, and V. Saraswat. Testing Concurrent Systems: an Interpretation of Intuitionistic Logic. In *Proceedings of the 25th international conference on Foundations of Software Technology and Theoretical Computer Science*, FSTTCS '05, pages 517–528, Berlin, Heidelberg, 2005. Springer-Verlag.
17. N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. Minizinc: Towards a standard cp modelling language. In *In: Proc. of 13th International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.
18. F. Pachet, P. Roy, and G. Barbieri. Finite-length markov processes with constraints. In *Proc. IJCAi 2011*, IJCAI'11, pages 635–642. AAAI Press, 2011.
19. J. F. Puget. Automatic detection of variable and value symmetries. In *Proc. CP 2005*, pages 475–489. Springer, 2005.
20. F. Rossi, P. v. Beek, and T. Walsh. *Handbook of Constraint Programming*. Elsevier Science Inc., New York, NY, USA, 2006.
21. P. Roy and F. Pachet. Enforcing meter in finite-length markov sequences. In *Proc. AAAI 2013*. AAAI Press, 2013.
22. V. Saraswat. *Concurrent Constraint Programming*. Doctoral Dissertation Award and Logic Programming. MIT Press, 1993.
23. V. Saraswat, R. Jagadeesan, and V. Gupta. Timed Default Concurrent Constraint Programming. *Journal of Symbolic Computation*, 22(5–6):475–520, Nov.–Dec. 1996. Extended abstract appeared in the *Proceedings of the 22nd ACM Symposium on Principles of Programming Languages*, San Francisco, January 1995.