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 (Hadoop\(^2\), [9], Google Project Sibyl). Today, programs routinely perform tasks such as personalized recommendations, multilanguage 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 flexibility 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**


