ECAI 2016
G.A. Kaminka et al. (Eds.)

doi:10.3233/978-1-61499-672-9-1660

Cuilt: A Scalable, Mix-and-Match Framework for Local Iterative Approximate Best-Response Algorithms

Mihaela Verman and **Philip Stutz** and **Robin Hafen** and **Abraham Bernstein** [1]

Abstract. We implement CUILT, a scalable mix-and-match framework for Local Iterative Approximate Best-Response Algorithms for DCOPs, using the graph processing framework SIGNAL/COLLECT, where each agent is modeled as a vertex and communication pathways are represented as edges. Choosing this abstraction allows us to exploit the generic graph-oriented distribution/optimization heuristics and makes our proposed framework scalable, configurable, as well as extensible. We found that this approach allows us to scale to problems more than 3 orders of magnitude larger than results commonly published so far, to easily create hybrid algorithms by mixing components, and to run the algorithms fast, in a parallel fashion.

1 Introduction

A Distributed Constraint Optimization Problem (DCOP) consists of a set of variables, a set of domains for the variables, a set of constraints over subsets of the variables, and utility functions for each of the constraints. Variables are controlled by agents, and the goal is to find the variable assignment that will maximize the global utility function, usually defined as the sum of utilities over all constraints. Chapman *et al.* [3, 4] propose a theoretical unifying framework for the class of Local Iterative Approximate Best-Response Algorithms, where agents can only be aware of their immediate neighbors' states. CUILT[2] is based on this framework. Chapman *et al.* identify three components for the algorithms, making them easily comparable and enabling the creation of hybrid algorithms: (1) **the state evaluation**, which updates an algorithm-specific **target function** to evaluate prospective states; (2) **the decision rule**, which represents how the agent decides which action to take next by using the already computed target function from the state evaluation step; (3) **the adjustment schedule**, which refers to the order in which the agents execute their processes.

There are other implemented frameworks for algorithms for DCOPs [6, 12, 10, 7], but to our knowledge, none is tailored for this class of algorithms or takes advantage of their modularity. CUILT seeks to model exactly that, and we implement it inside a graph processing framework that comes with several other benefits.

SIGNAL/COLLECT [9] is a framework for distributed large-scale graph processing implemented in Scala[3]. The programming model is vertex-centric: vertices communicate with each other via signals that are sent along directed edges (*signal*), and use the received signals to update their state (*collect*). A problem and an algorithm are specified by providing the graph structure, initial states, and the *signal* and *collect* functions. We chose SIGNAL/COLLECT not only because DCOPs can map well to graph abstractions, but also because of the capabilities and unique features of this graph processing framework: parallel and distributed execution, synchronous/asynchronous scheduling, aggregation operations and convergence detection.

2 CUILT

In CUILT, variables are represented as vertices; the neighborhood (two variables sharing at least one constraint) relationship is described by edges. The utility function of the "agent" responsible for a variable is dependent on the state of the adjacent vertices.

The **Algorithm** will be specified by the way in which different components are mixed and matched. To add different components we use Scala traits, which can be easily combined. CUILT defines the base Algorithm trait, with the types it needs and the methods that it requires. Implementations for the specified CUILT modules then get mixed and need to cover all the required methods of the Algorithm. The flexibility of CUILT comes from the fact that the components are loosely coupled and can be easily mixed and matched. The **SignalCollectAlgorithmBridge** provides the implementations for the vertices and edges, which assemble the components provided by the implementations of other modules, and describes, through its *collect* method, the behaviour of an agent at every step. The methods of Algorithm, which are used inside the *collect* step, are each provided by implementations of the several modules that extend Algorithm: the **TargetFunction**, **DecisionRule** and **AdjustmentSchedule**, providing the functionality described by Chapman *et al.*, and the added **StateModule**, **Utility** and **TerminationRule**.

To illustrate the flexibility of the framework, we provide implementations (through the recombination of components) of four algorithms from the category of iterative approximate best-response algorithms for DCOPs: the Distributed Stochastic Algorithm (DSA-$\overline{A}$, DSA-$\overline{B}$)[11, 1], Distributed Simulated Annealing with Termination Detection (DSAN-TD), Joint Strategy Fictitious Play with Inertia (JSFPI) [8], Weighted Regret Matching with Inertia (WRMI) [2]. Besides these components, we also added the ϵ-Greedy Decision Rule [3]. DSAN-TD is a modification of Distributed Simulated Annealing [1]. While DSAN exhibits oscillations in under-constrained problems [5], in DSAN-TD, to enable termination detection, the probability of exploring decreases over time even when the utilities for the candidate and current value are equal.

3 Evaluation

First, we show how hybrids can be exhaustively evaluated and that they can play an important role when it comes to both speed of convergence and solution quality. *Second*, we show that our framework can handle problems with up to 1 million variables on one machine.

We chose the canonycal *vertex coloring problem*, chromatic numbers of 3, 4 and 5, average edge density of 3, constraints with maximum utility 1, and constructed five graphs per size and chromatic

[1] University of Zurich, Switzerland
[2] https://github.com/elaverman/cuilt/
[3] www.signalcollect.com

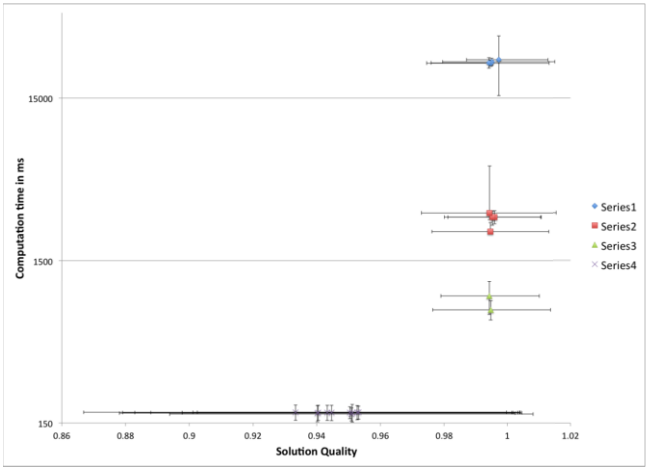

Figure 1. Solution quality vs. computation time in ms (log) for the top 10 average solution quality and top 10 computation time hybrids.

number. In the case of algorithms with the Parallel Random adjustment schedule, for the asynchronous mode we used a degree of parallelism 0.95, as asynchronous mode does not shield from thrashing.

Hybrid algorithms We generated algorithms by taking all possible combinations of the implemented components, with different parameter values. The graphs had 40 vertices and were constructed like in [4]. All the algorithms were run 5 times on these graphs. The machines that we used have 128 GB RAM and two E5-2680 v2 at 2.80GHz processors, with 10 cores per processor.

Figure 1 shows the top 10 combinations in terms of solution quality and convergence speed. We can observe four clusters. The bottom cluster in Series 4 is represented by all the top 10 convergence speed algorithms. They all ran asynchronously, with a Flood adjustment schedule, and with either argmax or ϵ-Greedy adjustment schedules. They were all new hybrids, three being modifications of Fading Memory JSFPI, with a flood schedule, and one being a modification of WRMI with a flood schedule. Most of the top 10 solution quality algorithms (Series 1-3) used the Simulated Annealing decision rule (one of them being DSAN-TD). The algorithms with lowest computation time were asynchronous hybrids with Flood schedule, most of them using the Average Weighted Expected Utility target function from Fading Memory JSFPI and either argmax or ϵ-Greedy decision rules.

Hybrid algorithms can lead to both reduced computation time and improved quality, and running the algorithms asynchronously seems to positively impact speed of convergence.

Data scalability We varied the number of vertices between 10 and 1 million. The graphs were constructed similarly to [4][4]. The timeout was 300 seconds and we executed each configuration 10 times. Convergence was detected only by using the signaling scores of the vertices, which depend on the termination condition. We ran the known algorithms DSA-$\overline{A}$, DSA-$\overline{B}$, and JSFPI on machines with two twelve-core AMD Operon™ 6174 processors and 66 GB RAM[5].

With regard to solution quality, the algorithms behave similarly, dropping from size 10 to 100, and then keeping a relatively stable quality, even when the size of the graph increases. We can see that the asynchronous variants (with probabilities 0.95) usually behave better, but suffer another drop in quality on the largest graphs.

DSA-$\overline{A}$ appears to have the best convergence speed. One possible explanation for the non-linearity of the computation time is reaching the memory limit of the available machines, which makes the computation slower. This could be addressed by using a machine with more memory, or by distributing the framework across multiple machines.

4 Conclusions and Future Work

We introduced CUILT, a software framework based on a mapping of Chapman *et al.*'s theoretical framework to a graph processing model. CUILT is flexible and configurable, allowing to easily create and mix components into hybrid algorithms that can then be exhaustively evaluated to determine a good balance between speed of convergence and solution quality. It has automatic convergence detection, and it enables scaling to problems with 1 million variables and exploiting the advantages of asynchronous execution.

In the future, we intend to implement more components, evaluate the hybrids on different scales and types of problems, including real-world ones, and investigate the behaviour of the algorithms on even larger problems, using the distributed version of SIGNAL/COLLECT.

We believe that our findings show how CUILT enables the systematic exploration of new hybrid algorithms and opens the possibility of applying them to big real-world constraint problems.

ACKNOWLEDGEMENTS

We would like to thank the Hasler Foundation for their generous support, and Dr. J. Enrique Munoz de Cote for his advice.

REFERENCES

[1] M. Arshad and M.C. Silaghi, 'Distributed simulated annealing and comparison to dsa', in *Proceedings of the fourth international workshop on distributed constraint reasoning (DCR-03)*, (2003).

[2] G. Arslan, J.R. Marden, and J.S. Shamma, 'Autonomous vehicle-target assignment: A game-theoretical formulation', *Journal of Dynamic Systems, Measurement, and Control*, **129**, 584, (2007).

[3] A.C. Chapman, A. Rogers, N.R. Jennings, and D.S. Leslie, 'A unifying framework for iterative approximate best-response algorithms for distributed constraint optimization problems', *Knowledge Engineering Review*, **26**(4), 411–444, (2011).

[4] Archie Chapman, Alex Rogers, and Nicholas Jennings, 'Benchmarking hybrid algorithms for distributed constraint optimisation games', *Autonomous Agents and Multi-Agent Systems*, **22**, 385–414, (2011).

[5] A. Flueckiger, M. Verman, and A. Bernstein, 'Improving approximate algorithms for dcops using ranks', in *OptMAS*, (2016).

[6] Thomas Léauté, Brammert Ottens, and Radoslaw Szymanek, 'FRODO 2.0: An open-source framework for distributed constraint optimization', in *Distributed Constraint Reasoning Workshop*, Pasadena, California, USA, (2009). http://frodo2.sourceforge.net.

[7] B. Lutati, I. Gontmakher, M. Lando, A. Netzer, A. Meisels, and A. Grubshtein, 'Agentzero: A framework for simulating and evaluating multi-agent algorithms', in *Agent-Oriented Software Engineering*, pp. 309–327. Springer Berlin Heidelberg, (2014).

[8] J. R. Marden, G. Arslan, and J. S. Shamma, 'Joint strategy fictitious play with inertia for potential games', *IEEE Transactions on Automatic Control*, **54**(2), 208–220, (2009).

[9] P. Stutz, D. Strebel, and A. Bernstein, 'Signal/Collect: Processing Large Graphs in Seconds', *Semantic Web Journal - forthcoming*, (2015).

[10] E. Sultanik, R. Lass, and W. Regli, 'Dcopolis: A framework for simulating and deploying distributed constraint reasoning algorithms', in *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, (2008).

[11] G. Tel, *Introduction to distributed algorithms*, Cambridge Univ Press, 2000.

[12] Mohamed Wahbi, Redouane Ezzahir, Christian Bessiere, and El Houssine Bouyakhf, 'Dischoco 2: A platform for distributed constraint reasoning', in *Distributed Constraint Reasoning Workshop*, (2011).

[4] We picked colors for each vertex, we randomly added edges between vertices with different colors, and then randomly edited edges to connect isolated vertices.

[5] Full results at: `https://docs.google.com/spreadsheets/d/1A0auNnM0MedEgu0SjNk9OjWIn9wdyefMLhmnfcanvwQ/edit#gid=2036767986`