pyDCOP: a DCOP library for Dynamic IoT Systems

P. Rust[1,2], G. Picard[2], and F. Ramparany[1]

[1] Orange Labs, France
{pierre.rust,fano.ramparany}@orange.com
[2] MINES Saint-Etienne, Laboratoire Hubert Curien UMR CNRS 5516, France
picard@emse.fr

Abstract. This demonstration illustrates the newly developed Python-based framework, pyDCOP, which implements several state-of-the-art distributed constraint reasoning solution methods, provides utilities to deploy them over distributed infrastructures and also equip the system with resilience capabilities. The idea behind pyDCOP is to distribute agents over an Internet-of-Things infrastructure (e.g. Rapsberry Pis) to install collective decisions, as to implement Ambient Intelligence or Smart Home scenarios. Scenarios are modeled in a dedicated format, translated in a distributed constraint optimization or satisfaction problem, then pushed to the devices which coordinate using chosen protocols as to self-configure in a decentralized manner. Besides configuring the system in an optimal manner, it also provides a resilience framework, which equips the system with adaptation capabilities against unpredictable device removals. This mechanism is based on decision replication and a lightweight DCOP-based reparation mechanism. A video presenting this demonstration is available at https://www.emse.fr/~picard/demoPyDCOP.mp4.

1 pyDCOP in a Glance

pyDCOP is a new open source[3] library designed to foster the study and research on Distributed Constraints Optimization Problems (DCOP). DCOPs are a classical approach to model distributed coordination problems in multi-agent systems and have been used in many scenarios like sensors networks [4], resources allocation [17], transportation [7], smart grid [3] and smart home [5,13]. A DCOP is formally represented as a tuple $\langle \mathcal{A},\mathcal{X},\mathcal{D},\mathcal{C},\mu \rangle$, where: $\mathcal{A}=\{a_1,...,a_{|A|}\}$ is a set of agents; $\mathcal{X}=\{x_1,...,x_n\}$ are variables owned by the agents; $\mathcal{D}=\{\mathcal{D}_{x_1},...,\mathcal{D}_{x_n}\}$ is a set of finite domains, such that variable x_i takes values in $\mathcal{D}_{x_i}=\{v_1,...,v_k\}$; $\mathcal{C}=\{c_1,...,c_m\}$ is a set of soft constraints, where each c_i defines a cost $\in \mathbb{R} \cup \{\infty\}$ for each combination of assignments to a subset of variables (a constraint is initially known only to the agents involved); $\mu : \mathcal{X} \rightarrow \mathcal{A}$ is a function mapping variables to their associated agent. A *solution* to the DCOP is an assignment to all variables that minimizes the overall sum of costs $\sum_i c_i$.

pyDCOP's architecture is based on this formal representation; pyDCOP manages a set of *software agent* objects, which coordinate cooperatively, using DCOP algorithms, to assign values to the variables they are responsible for. This joint assignment should minimize the sum of the constraints defined by the problem. The problem is given is

[3] https://github.com/Orange-OpenSource/pyDcop

a problem-specific format (like for smart-lightning system in [13]) or directly given in a well-specified text-based format (using YAML[4]), as an optimization problem defining the variables and constraints. pyDCOP has been initially developed to study the use of DCOPs for implementing coordination among connected objects [13] and to distribute DCOP computations over an IoT infrastructure [14]. The resulting code has been re-designed and packaged to create pyDCOP. Special care has been given to documentation[5], notably on deploying on several machines, modeling problems as DCOPs, dynamic DCOPs, or implementing DCOP algorithms[6].

1.1 Studying DCOP Algorithms

pyDCOP provides implementation for many classical DCOP algorithms, including DSA [19], A-DSA [6], MGM, MGM2 [10], DPOP[12], ADOPT [11], and MaxSum [4], but it also allows the rapid development of new algorithms. When studying an existing algorithm or developing a new one, pyDCOP provides all the needed infrastructure: thanks to the numerous base classes and 'plumbing' utilities one can simply focus on its algorithm design. The modular architecture of pyDCOP, which decouples communication, agent managements, and algorithmic utilities to ensure that developed algorithms will be able to run in the several runtime environments and settings supported by pyDCOP. When running an algorithm, various metrics can be produced and used to benchmark algorithms or the effect of metaparameters in a specific problem topology. These metrics notably include runtime, number of cycles, number and size of messages, and cost and quality of the solution.

1.2 Runtime Environment

When working on a problem, pyDCOP runs as many agents than specified by the problem, and assigns a subset of the variables to each of them. The computations implemented the selected algorithm is then deployed on these agents. The only centralized element in the system is called the *orchestrator* and has a purely technical role. The orchestrator never takes part in the distributed coordination or decision making process. Its only responsibilities are boostraping the system and monitoring it. When bootstrapping the system, the orchestrator instanciate agents, assign variables to them (according to the problem model) and deploy the corresponding algorithm computations. At runtime, it simply monitors the system in order to produce the metrics.

The agents solving the problem can run on the same machine and even in the same process, using in-memory communication, which is convenient during development but also allows large-scale system made of several hundreds of agents. They can also run on different computers, communicating over the network, for prototyping real distributed systems. pyDCOP is multi-platform and can run on Windows, Mac and Linux. Scripts are also provided to ease the deployment of agents on many computer, typically virtual machines or single-board computers like Raspberry Pis.

[4] `http://yaml.org`
[5] `https://pydcop.readthedocs.io`
[6] `https://pydcop.readthedocs.io/en/latest/tutorials.html`

1.3 pyDCOP for IoT

In addition to state-of-the-art DCOP algorithms, pyDCOP also includes novel approaches to apply the DCOP framework to dynamic systems like the IoT. The distribution of DCOP computations (corresponding to variable and potentially constraints in the original problem) is an issue that received little attention so far but is paramount when working on real-world problems [14,15]. pyDCOP provides several distribution approaches, both centralized and distributed [2]. In IoT systesm, the devices are typically very constrained (both CPU and memory wise), and the network is generally also considered to be a costly and limited resource. As a consequence, pyDCOP's distribution mechanisms take these elements into account and produce distributions that optimize for network communication while ensuring the agents capacities are respected.

Resiliency is also a key issue when building a MAS. In dynamic environments the problem may evolve at runtime and agents could join and leave the system unexpectedly at any time. In order to ensure resiliency, pyDCOP can migrate the computations needed to solve the DCOP from on agent to another. pyDCOP also implements a distributed replication mechanism inspired by distributed databases, which makes sure that the definition of the problem is not lost when some agents leave the system. Based on these two mechanisms, in case of an agent failure, remaining agents can self-repair the system by migrating orphaned computations to the remaining agents. This self-repair function is also modeled as a DCOP, where agent cooperatively agree on the best place to host the repaired computation required to solve the initial problem [14,15].

2 Demonstration Scenario

Our demonstration use pyDCOP to illustrate a distributed decision making process in an IoT system. Our scenario is based on a classical distributed weighted graph coloring problem, to which many real problems can be mapped. Each variable in the system maps to a vertex in the graph and can take one color as a value. Edges of the graph maps to binary constraints, assigning a cost for each combination of colors taken by its associated variables/vertices. The goal is to find a assignment of colors that minimize the sum of these costs.

Several graph structures can be used when generating instances of this problem. To model an IoT infrastructure, we use the Barabasi-Albert method [1], which produces graphs that follow a power-law, known to adequately model this kind of systems [18] Each agent in the system is responsible for a subset of the variables and use a DCOP algorithm to coordinate assignment of color to its variables.

The demonstrator (see Figure 1) is made of a 3×3 grid of small single-board computers (Raspberry Pis), each fitted with a small touch-screen. Each of these computers runs one pyDCOP agent and display a graphical interface presenting the current state of this agent. A central screen (an internet browser on a TV or computer screen) gives an overall view of the system and the current runtime metrics. During the demonstration, we dynamically remove random agents from the system. Remaining agents coordinate autonomously the repair process, which can be observed on their graphical interface. The self-repair it-self is also totally decentralized and is based on a distributed replication protocol followed by a host-selection mechanism modeled using a DCOP [14,15].

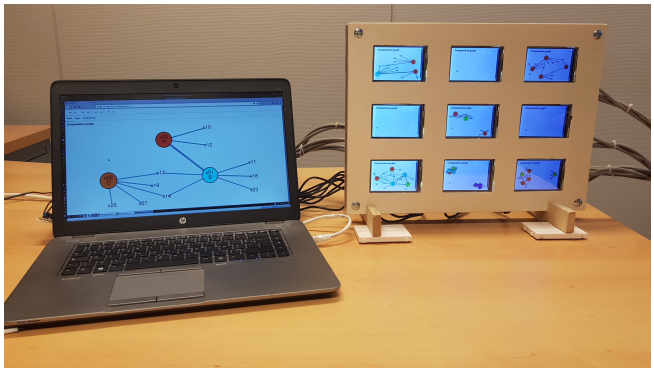

Fig. 1. A 3×3 Raspberry Pi grid to showcase pyDCOP

3 Related Works

Several other libraries currently exist for the study of DCOP: AgentZero, Frodo2 and DisChoco. AgentZero is a Java-based library developed at the Ben-Gurion University which supports a large set of functionalities for the study of distributed constraints reasoning algorithms [9]. Unfortunately documentation is scarce and the source code repository[7] has not been updated for 3 years. Frodo2 is actively developed[8] by the Artificial Intelligence Laboratory (LIA) of École Polytechnique Fédérale de Lausanne (EPFL) and is commonly used for evaluating DCOP algorithms [8]. While being very well engineered and providing numerous DCOP algorithmic implementations, it does not provide the required features to study and prototype DCOP in a dynamic system like IoT. DisChoco is also Java-based and supports real distributed settings [16]. However, the project[9] seems to be discontinued and has not been updated since 2014.

Given this situation, we believe a new DCOP library is needed, which specifically takes into account IoT and dynamic systems requirements, in order to foster research in this promising and active domain. pyDCOP has been developed and open-sourced for this purpose.

References

1. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. Science 286(5439), 509–512 (1999)
2. Bürger, M., Notarstefano, G., Bullo, F., Allgöwer, F.: A distributed simplex algorithm for degenerate linear programs and multi-agent assignments. Automatica 48(9), 2298–2304 (sep 2012)
3. Cerquides, J., Rodríguez-Aguilar, J.A., Picard, G.: Designing a marketplace for the trading and distribution of energy in the smart grid. p. 9

[7] `https://github.com/bennylut/agent-zero`
[8] `https://frodo-ai.tech/`
[9] `https://sourceforge.net/projects/dischoco/`

4. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.R.: Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: International Conference on Autonomous Agents and Multiagent Systems (AAMAS'08). pp. 639–646 (2008)
5. Fioretto, F., Pontelli, W.Y.E.: A multiagent system approach to scheduling devices in smart homes. In: Proceedings of the International Workshop on Artificial Intelligence for Smart Grids and Smart Buildings. p. 7
6. Fitzpatrick, S., Meertens, L.: Distributed coordination through anarchic optimization. In: Lesser, V., Ortiz, C.L., Tambe, M. (eds.) Distributed Sensor Networks, vol. 9, pp. 257–295. Springer US
7. Junges, R., L.C. Bazzan, A.: Evaluating the performance of DCOP algorithms in a real world, dynamic problem. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2. pp. 599–606
8. Léauté, T., Ottens, B., Szymanek, R.: FRODO 2.0: An open-source framework for distributed constraint optimization. In: Proceedings of the IJCAI'09 Distributed Constraint Reasoning Workshop (DCR'09). pp. 160–164. Pasadena, California, USA (July 13 2009), https://frodo-ai.tech
9. Lutati, B., Gontmakher, I., Lando, M., Netzer, A., Meisels, A., Grubshtein, A.: AgentZero: A framework for simulating and evaluating multi-agent algorithms. In: Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks, vol. 9783642544, pp. 309–327 (2014)
10. Maheswaran, R., Pearce, J., Tambe, M.: Distributed algorithms for dcop: A graphical-game-based approach. In: Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems (PDCS), San Francisco, CA. pp. 432–439 (2004)
11. Modi, P., Shen, W., Tambe, M., Yokoo, M.: ADOPT: Asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence Journal (2005)
12. Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. IJCAI International Joint Conference on Artificial Intelligence pp. 266–271 (2005)
13. Rust, P., Picard, G., Ramparany, F.: Using message-passing DCOP algorithms to solve energy-efficient smart environment configuration problems. In: IJCAI International Joint Conference on Artificial Intelligence. vol. 2016-Janua (2016)
14. Rust, P., Picard, G., Ramparany, F.: Self-organized and resilient distribution of decisions over dynamic multi-agent systems. In: 9th International Workshop on Optimisation in Multi-Agent Systems (OPTMAS@AAMAS 2018) (2018)
15. Rust, P., Picard, G., Ramparany, F.: Installing resilience in distributed constraint optimization operated by physical multi-agent systems. In: Autonomous Agents and Multiagent Systems (AAMAS). International Foundation for Autonomous Agents and Multiagent Systems (2019)
16. Wahbi, M., Ezzahir, R., Bessiere, C., Bouyakhf, E.H.: Dischoco 2: A platform for distributed constraint reasoning. In: Proceedings of the IJCAI'11 workshop on Distributed Constraint Reasoning. pp. 112–121. DCR'11, Barcelona, Catalonia, Spain (2011), http://dischoco.sourceforge.net/
17. Xie, J., Howitt, I., Raja, A.: Cognitive radio resource management using multi-agent systems. In: Proceedings of the 2007 4th IEEE Consumer Communications and Networking Conference. pp. 1123–1127. IEEE Computer Society
18. Yao, B., Liu, X., Zhang, W.J., Chen, X.E., Zhang, X.M., Yao, M., Zhao, Z.X.: Applying graph theory to the internet of things. Proceedings - 2013 IEEE International Conference on High Performance Computing and Communications, HPCC 2013 and 2013 IEEE International Conference on Embedded and Ubiquitous Computing, EUC 2013 pp. 2354–2361 (2014)
19. Zhang, W., Wang, G., Xing, Z., Wittenburg, L.: Distributed stochastic search and distributed breakout: Properties, comparison and applications to constraint optimization problems in sensor networks. Artificial Intelligence 161(1-2), 55–87 (2005)