DCOPolis: A Framework for Simulating and Deploying Distributed Constraint Optimization Algorithms

Evan A. Sultanik, Robert N. Lass and William C. Regli

Drexel University
Department of Computer Science
College of Engineering
3141 Chestnut Street
Philadelphia, PA 19104
{eas28,urlass,regli}@cs.drexel.edu

Abstract. A large class of problems in multiagent systems can be solved by distributed constraint optimization (DCOP). Until now DCOPs have exclusively been implemented in simulation—with each algorithm running in a different simulator. Furthermore, very few examples of real-world DCOP implementation exists in the literature. This paper presents DCOPolis, a framework for comparing *and deploying* DCOP software in heterogeneous environments. DCOPolis makes three main contributions to the community. Different communications platforms, DCOP algorithms and problems can be plugged in for a truly comprehensive analysis of DCOP performance. DCOPolis contributes to comparative analysis of DCOP algorithms by allowing different state-of-the-art algorithms to run in the same simulator under the same conditions or to be deployed on "real" hardware in "real" scenarios. Finally, DCOPolis introduces a new form of distributed algorithm simulation that shows promise of accurate prediction of real-world runtime.

1 Introduction

A large class of multiagent coordination and distributed resource allocation problems can be modeled as distributed constraint optimization (DCOP) problems. DCOP has generated a lot of interest in the constraint programming community and a number of algorithms have been developed to solve DCOP problems [1–4]. Evaluating the performance of these algorithms under realistic scenarios is an important and active area of research [5–7].

Evaluating and comparing DCOP algorithms is complicated by the fact that they are currently implemented in simulation; there is no record in the literature of any significant comparison of DCOP algorithms on live networks. Solely evaluating distributed systems in simulation can have a number of undesirable side effects, such as differences between the individual simulators. Most of these side effects are not unique to DCOP algorithms [8]. Ideally, an independent, community-endorsed framework should exist in which a single implementation

of a DCOP algorithm can be run both in simulation *and* deployed on a live network without modification to the underlying code [8]. This paper introduces such a framework, called DCOPolis.

2 DCOPolis

DCOPolis was designed as a framework for comparing and deploying distributed decision processes in heterogeneous environments. We are using the term "framework" in the sense that it is a standard API for interfacing different algorithms, problems, and platforms (all of these terms are defined in 2.1) to allow for a systematic evaluation of DCOP systems. In addition to the API, implementations of several common algorithms, problems, and platforms are provided.

DCOPolis currently includes implementations of three DCOP algorithms: Adopt, DPOP and NCBB. DCOPolis also has an implementation of Branch and Bound that is completely centralized, which can be used as a baseline to compare the effect of distribution on a problem.

DCOPolis differs from existing frameworks and simulators (such as FRODO [9] and those used to test Adopt and OptAPO) in three fundamental ways:

1. DCOPolis was designed to allow for both simulation of DCOPs on a single computer and full deployment of DCOP solvers on many types of live networks, including traditional wired networks and ad-hoc wireless networks;
2. DCOPolis is able to instantiate a DCOP and start the solution process completely distributedly. This means that there is no need for configuration files, nor is there any need for a central agent/server that initializes/instantiates the rest of the group; and
3. DCOPolis supports a novel type of simulation in which the runtime of any distributed algorithm can be accurately estimated on a single physical computer. This is described in detail in §3.

2.1 Architectural Overview

DCOPolis has three primary abstract components: *problems*, *algorithms*, and *platforms*. The main function of DCOPolis is to provide an interface through which the three components can interact. By writing a new instance of any of these components that properly adheres to DCOPolis' API, any algorithm should be able to solve any instance of any problem while running on any platform—even without prior knowledge of such. This makes implementation and testing of new algorithms and platforms trivial.

Problem The *problem* is a formal representation of the input to an algorithm. Currently implemented problems in DCOPolis include graph coloring, the distributed multiple knapsack problem, sensor network problems, meeting scheduling, disaster evacuation planning, and C_TÆMS scheduling. Additionally, random problems generators exist for all of these domains except for C_TÆMS scheduling. A meeting scheduling problem class is in development.

Algorithm The *algorithm* is an implementation of a DCOP algorithm. Currently, Adopt, DPOP and NCBB are implemented. OptAPO is in development.

Platform The *platform* is the interface through which the agents communicate, providing a layer of abstraction between the agents and the underlying computer. The two platforms currently implemented in DCOPolis are a TCP-based platform and a networkless platform. The latter is able to simulate execution of the DCOP algorithms on a single computer; it uses the Macro Agent Transport Event Simulator [10] for modeling network connectivity. A platform allowing agents to be run on mobile phones using JavaME is also currently in development.

2.2 Pseudotree Generation

A similarity between many DCOP algorithms is that they assume the existence of a tree ordering over all of the variables in the problem. The pseudotree has an invariant that for each pair of variables $\langle v_i, v_j \rangle$ that are neighbors in the constraint graph, it must hold that v_i is either an ancestor or descendent of v_j in the pseudotree. The pseudotree also contains a backedge between all pairs of neighbors in the constraint graph that do not have a parent/child relationship in the pseudotree. For each variable to which an agent is to assign a value, the agent must know the relative tree position (*i.e.*, ancestor, descendent, or parent) of each constraint graph neighbor of the variable. The authors of many DCOP algorithms assume that the agents would simply elect one agent to create this ordering, which is then broadcast to the rest of the group. DCOPolis provides distributed algorithms for creating the pseudotree.

3 Simulated Time

DCOPolis' "Networkless" platform runs in simulated time using the Sefirs[1] simulation kernel. The idea behind this type of simulation is to have each agent's thread of execution run sequentially such that no two agents are competing for CPU cycles. Although the algorithms end up being executed sequentially, DCOPolis is able to keep track of which portions of the execution would have occurred concurrently in a distributed setting, and thereby can provide an accurate prediction of what the distributed runtime would be. This process is as described in Algorithm 1. Note that the EXECUTE(a) command in the algorithm will cause the thread of execution of agent a to run until it either sends a message, sleeps, performs a join on another thread, or explicitly yields its execution; the EXECUTE(a) command will block until a's thread has done so. This type of behavior is accomplished programmatically using the programming language concept known as "continuations" [11], which in the case of DCOPolis is provided through Sefirs. A sequence diagram visualizing the execution of two agents running in simulated time is given in Figure 1.

[1] `http://sefirs.sourceforge.net/`

Algorithm 1 SIMULATE-DISTRIBUTED-EXECUTION(A)

Require: A is a set of agents whose execution are to be simulated.
1: E is a map from the agents in A to execution times.
2: Initialize the execution time to zero for every agent: $(\forall a \in A : E(a) \mapsto 0)$.
3: Q is a priority queue that orders the agents by ascending execution time; the $a \in Q$ with minimum $E(a)$ is returned first.
4: $Q \leftarrow A$
5: t is the current time of the simulation
6: $t \leftarrow 0$
7: **while** $Q \neq \emptyset$ **do**
8: $a \leftarrow$ POLL(Q)
9: $t \leftarrow E(a)$
10: $c \leftarrow$ CPU-TIME()
11: EXECUTE(a)
12: **if** IS-ALIVE?(a) **then**
13: $E(a) \mapsto t + (\text{CPU-TIME}() - c)$
14: $Q \leftarrow Q \cup \{a\}$
15: **else**
16: a died during its execution
17: **if** $Q = \emptyset$ **then**
18: $t \leftarrow t + (\text{CPU-TIME}() - c)$
19: **end if**
20: **end if**
21: **end while**

3.1 Evaluation of the Simulated Time Metric

In order to evaluate the accuracy of DCOPolis' simulated time metric, we compared simulated runtime against runtime on a live network for graph coloring problems. A subset of 28 of the distributed graph coloring problems from the USC Teamcore dataset [12] were solved using the Adopt [1] algorithm. Due to a lack of hardware during the live experiments, the agents were uniformly distributed across the physical computers, resulting in some computers hosting multiple agents. This means that some of the agents were competing with each other for processor time, which was not captured in the simulation. This competition was likely negligible; calculating Pearson's linear correlation coefficient between the simulated times and the actual runtimes, we were able with 99% certainty to reject the null hypothesis that the distributions were not linearly correlated in favor of the alternate hypothesis that the simulated runtime and actual runtime are linearly correlated. Pearson's coefficient has a student's t distribution, which is what we used to test these hypotheses. The test statistic is 5.03.

4 Example Application: Evacuation Operations Support

This section presents an application, Evacuation Operations Support (EOS) that was written using DCOPolis as the kernel for distributed decision making. The

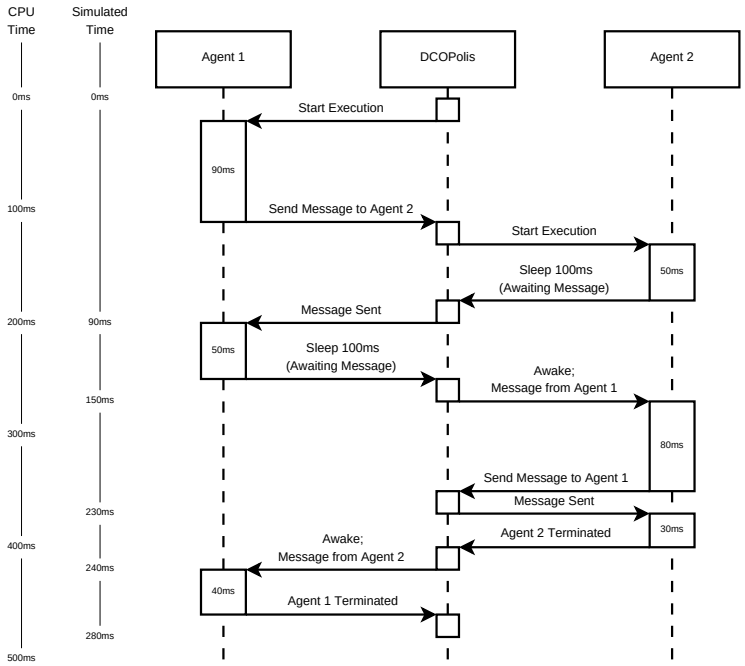

Fig. 1. A sequence diagram visualizing the execution of two agents running in simulated time on DCOPolis.

scenario for the example is presented, followed by a description of how EOS uses DCOPolis.

4.1 Scenario

Evacuation and sheltering of neighborhoods, cities, or regions is a major component of responding to any natural or other disaster. Poorly chosen and uncoordinated destinations can quickly overwhelm shelter capacities. Insufficient knowledge and decision processes may also lead to mismatches between evacuee needs and shelter capabilities, such as advanced medical units. Unfortunately, the intuitive and easy response of moving evacuees to the closest refuges can easily lead to this situation. One solution to this problem could be a tool such as EOS to help emergency personnel create a shared and accurate understanding of the situation, making the best decisions for the group, to effectively conduct disaster evacuations.

The central premise is that there are area wardens or emergency personnel leading groups of evacuees to available shelters. These authorities are equipped with handheld, networked devices to monitor and coordinate actions. Figure 2 depicts an example scenario, as it appears in EOS. There are several groups

Fig. 2. Example neighborhood sheltering scenario and possible shelter assignments.

of people and available shelters within the local area. Each group has several traits such as size and medical needs. Shelters mirror these with capacity and medical capabilities. The problem is that of assigning groups to shelters in a globally optimal fashion, i.e. not overcrowding any shelter, and allowing evacuees to access needed medical resources.

The application runs on handheld computing devices—tablets or PDAs—communicating wirelessly over a mobile, ad hoc, Wi-Fi network. Such networks enable significant data exchange without infrastructure such as wires or access points, adapt to changing conditions such as host movement, and operate over moderate geographic distances.

4.2 Using DCOPolis

EOS provides an interface for the handheld computing devices' users to communicate their status with other devices on the network, and display a graphical representation of the state of the world. After receiving status information from the other devices on the network, EOS uses DCOPolis to find a globally-optimal shelter assignment. EOS maps the state of the world to a DCOP using a built-in DCOPolis problem class: the distributed multiple knapsack problem (DMKP).

This problem is a variant of the knapsack problem that uses multiple knapsacks and is distributed.

The evacuation problem is mapped to a DCOP as follows. The agents are individual instances of the EOS program. One DMKP variable is created for each shelter. The domain of each variable is the powerset of the evacuee groups (*i.e.*, all possible assignments of evacuee groups to that shelter). This mapping creates both n-ary and unary constraints. The shelters capacities are considered to be n-ary constraints—a constraint is violated if the sum of the group sizes assigned to any shelter exceeds its capacity, with the cost computed using the DMKP cost function, or if a group is assigned to more than one shelter. This is all encoded within the built-in DCOPolis problem class.

Medical requirements are mapped to unary constraints—a constraint is violated if a group is assigned to a shelter that does not meet its requirements for first aid or trauma. This constraint was not in DCOPolis, so a custom constraint was written to represent it.

5 Conclusions and Future Work

DCOPolis provides a framework for distributed constraint reasoning researchers to, with relatively little effort, implement and test new algorithms, implement and test new DCOP problems, and simulate and deploy DCOP systems. DCOPolis also provides a novel simulation environment that can accurately generate lower-bounds on and asymptotic behavior of distributed runtime.

In the future, we plan to add more DCOP algorithms to the system and more problem instantiations. Implementations of OptAPO [2] and Distributed Breakout [13] are in progress, as is a Java ME platform that will allow the algorithms to run on extremely constrained devices (such as cellular phones). We are also interested in different messaging paradigms, such as multicast. We have open-sourced DCOPolis[2] in the hopes that other researchers will be interested in adding to it and using it for their own work. Any interested people from the community are welcome to contribute algorithms, optimizations or other relevant additions.

References

1. Modi, P.J., Shen, W.M., Tambe, M., Yokoo, M.: An asynchronous complete method for distributed constraint optimization. In: AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM Press (2003) 161–168
2. Mailler, R., Lesser, V.: Solving distributed constraint optimization problems using cooperative mediation. In: AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, Washington, DC, USA, IEEE Computer Society (2004) 438–445

[2] `http://dcopolis.sourceforge.net`

3. Petcu, A., Faltings, B.: A distributed, complete method for multi-agent constraint optimization. In: CP 2004 - Fifth International Workshop on Distributed Constraint Reasoning (DCR2004), Toronto, Canada (September 2004)
4. Chechetka, A., Sycara, K.: No-commitment branch and bound search for distributed constraint optimization. In: AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM Press (2006) 1427–1429
5. Meisels, A., Kaplansky, E., Razgon, I., Zivan, R.: Comparing performance of distributed constraints processing algorithms. In: Workshop on Distributed Constraint Reasoning (AAMAS 2002). (2002)
6. Davin, J., Modi, P.J.: Impact of problem centralization in distributed constraint optimization algorithms. In: AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM Press (2005) 1057–1063
7. Zivan, R., Meisels, A.: Message delay and discsp search algorithms (2006)
8. Haeberlen, A., Mislove, A., Post, A., Druschel, P.: Fallacies in evaluating decentralized systems. In: Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS'06). (February 2006)
9. Petcu, A.: Frodo: A framework for open/distributed constraint optimization. Technical Report No. 2006/001 2006/001, Swiss Federal Institute of Technology (EPFL), Lausanne (Switzerland) (2006) http://liawww.epfl.ch/frodo/.
10. Sultanik, E.A., Peysakhov, M.D., Regli, W.C.: Agent transport simulation for dynamic peer-to-peer networks. Technical Report DU-CS-04-02, Drexel University (2004)
11. Reynolds, J.C.: The discoveries of continuations. LISP and Symbolic Computation **6**(3–4) (1993) 233–247
12. Pearce, J.P.: University of southern california DCOP repository (2007) http://teamcore.usc.edu/dcop/.
13. Hirayama, K., Yokoo, M.: The distributed breakout algorithms. Artif. Intell. **161**(1-2) (2005) 89–115