ASP-DPOP: Solving Distributed Constraint Optimization Problems with Logic Programming

(Extended Abstract)

Tiep Le, Tran Cao Son, Enrico Pontelli, William Yeoh
Department of Computer Science
New Mexico State University
Las Cruces, NM 88003, USA
{tile,tson,epontell,wyeoh}@cs.nmsu.edu

ABSTRACT

Researchers have used *Distributed Constraint Optimization Problems* (DCOPs) to model various multi-agent coordination and resource allocation problems. However, existing DCOP algorithms have focused almost exclusively on imperative programming techniques. This paper explores a new direction, which is to develop algorithms that use declarative programming, specifically logic programming, techniques.

Categories and Subject Descriptors

I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving—*Logic Programming*; I.2.11 [**Artificial Intelligence**]: Distributed AI—*Multiagent Systems*

Keywords

DCOP; ASP

1. INTRODUCTION

Distributed Constraint Optimization Problems (DCOPs) are problems where agents need to coordinate their value assignments to maximize the sum of resulting constraint utilities [6, 8, 9]. While the field has matured considerably over the past decade, existing DCOP algorithms have focused almost exclusively on *imperative programming* techniques, where the algorithms define a control flow, that is, a sequence of commands to be executed. In this paper, we are interested in exploring a new direction, which is to develop algorithms that use *declarative programming* techniques.

Declarative programs differ from imperative programs in that declarative programs only specify the problem as a set of logical rules to be solved without defining a specific control flow. The declarative programming paradigm offers several advantages, including a more compact representation of the problem and a high level of elaboration tolerance. Using a rule representation as an input allows a logical solver to exploit the interactions between rules to prune the search space, thus reducing the memory requirement and improving the scalability of the system. Finally, a declarative program is a more natural representation for solving problems that are originally defined by a set of rules (e.g., constraints in functional form).

In addition to showing how one can formulate a DCOP as a logic program, specifically an *answer set program*, we also introduce an algorithm, called ASP-DPOP, which emulates the computation and communication operations of DPOP [8] via the use of *Answer Set Programming* (ASP). Our experimental results show that, not only is ASP-DPOP competitive compared to DPOP in terms of performance, but it can also solve problems that DPOP fails to solve due to memory limitations.

Appears in: *Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns (eds.), Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), May 5-9, 2014, Paris, France.*

2. BACKGROUND

DCOP: A *Distributed Constraint Optimization Problem* (DCOP) [6, 8, 9] is defined by $\langle \mathcal{X}, \mathcal{D}, \mathcal{F}, \mathcal{A}, \alpha \rangle$, where $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a set of *variables*; $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of finite *domains*, where D_i is the domain of variable x_i; $\mathcal{F} = \{f_1, \ldots, f_m\}$ is a set of *utility functions* (also called *constraints*), where each k-ary utility function $f_i : D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \mapsto \mathbb{N} \cup \{-\infty, 0\}$ specifies the utility of each combination of values of variables in its *scope* (i.e., $x_{i_1}, \ldots, x_{i_k}$); $\mathcal{A} = \{a_1, \ldots, a_p\}$ is a set of *agents*; and $\alpha : \mathcal{X} \rightarrow \mathcal{A}$ maps each variable to one agent. A *solution* is a value assignment for all variables. Its utility is the evaluation of all utility functions on that solution. The goal is to find a utility-maximal solution.

DPOP: The *Distributed Pseudo-tree Optimization Procedure* (DPOP) [8] is a complete DCOP algorithm and has the following three phases:

- **Pseudo-tree Generation Phase:** DPOP calls existing distributed pseudo-tree construction algorithms to construct its pseudo-tree.
- **UTIL Propagation Phase:** Each agent, starting from the leafs of the pseudo-tree, computes the optimal sum of utilities in its subtree for each value combination of its ancestor agents and propagates them to its parent. The agent does so by summing the utilities in the UTIL messages received from its child agents and then projecting out its own variables by optimizing over them.
- **VALUE Propagation Phase:** Each agent, starting from the root of the pseudo-tree, determines the optimal value for each of its variables and propagates them to its children. The root agent does so using the UTIL messages received in the second phase.

ASP: Let us now provide some general background on logic

programming under the answer set semantics and *Answer Set Programming* (ASP). Consider a logic language $\mathcal{L} = \langle \mathcal{C}, \mathcal{P}, \mathcal{V} \rangle$, where $\mathcal{C}$ is a set of constants, $\mathcal{P}$ is a set of predicate symbols, and $\mathcal{V}$ is a set of variables; the notions of terms, atoms, and literals are the traditional ones.

An *ASP program* Π is a set of *rules* of the form

$$c \leftarrow a_1, \ldots, a_m, not\ b_1, \ldots, not\ b_n \quad (1)$$

where $m \geq 0$ and $n \geq 0$. Each a_i and b_i is a literal from $\mathcal{L}$, and each $not\ b_i$ is called a negation-as-failure literal (or naf-literal). c can be a literal or omitted. If a rule contains some variables, then it is called a *non-ground rule*. Otherwise, it is called a *ground rule*. If $n = 0$ and $m = 0$, then the rule is called a *fact*. If c is omitted, then the rule is called an *ASP constraint*. The solution of a program, called an *answer set* [3], is a set of consistent ground literals, representing a stable model of the program. ASP solves a problem by encoding it as an ASP program whose answer sets correspond one-to-one to the problem's solutions [5, 7]. These answer sets can be computed using answer set solvers [1, 2].

3. ASP-DPOP ALGORITHM

The overall idea underlying our approach is to capture the structure of a DCOP in ASP—by mapping each individual agent of a DCOP to an ASP program. The process of resolving the DCOP problem is fully distributed, and realized by using a logic programming infrastructure (based on the Linda model) for communication among agents.

In this paper, we introduce *ASP-DPOP*—a complete ASP-based DCOP algorithm that emulates the computation and communication operations of DPOP. Like DPOP, there are also three phases in the operation of ASP-DPOP: the pseudo-tree generation phase, the UTIL propagation phase, and the VALUE propagation phase. At a high-level, each agent a_i in the system is composed of two main components: the set of rules representing the DCOP agent, and a fixed Prolog module that implements the actual extraction of solutions and communication among agents during the propagation phases. The two steps of propagation—generation of a table to be sent from an agent to its parent in the pseudo-tree during the UTIL propagation, and propagation of variables assignments from an agent to its children in the pseudo-tree during the VALUE propagation—are achieved by computing solutions of two ASP programs.

4. EXPERIMENTAL RESULTS

We implemented two versions of the ASP-DPOP algorithm. The first version, which we call "ASP-DPOP (facts)," uses ground programs. The second version, which we call "ASP-DPOP (rules)," uses non-ground programs. We compare them against DPOP [8], their imperative-programming-based counterpart on a 13-Bus power optimization problem [4]. Figure 1 shows the runtime results. ASP-DPOP is slower than DPOP when the domain size is small and faster when the domain size is large. The runtime of the ASP solver called by each ASP-DPOP agent is roughly equivalent to the sum of the initialization time (used to initialize the data structures) and the computation time (used to solve the problem). The ratio of initialization time to computation time is larger for simpler problems (i.e., smaller domain sizes). As such, the speedups gained by the ASP solver in the computation time is lost in the initialization time for the simpler problems. Additionally, ASP-DPOP can solve problems that DPOP failed to solve due to memory limitations.

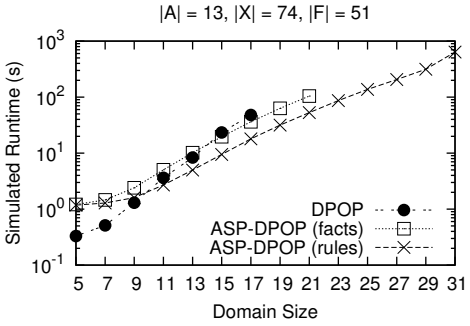

Figure 1: Experimental Results

5. CONCLUSIONS

In this paper, we explore the new direction of developing DCOP algorithms that use logic programming techniques. Our proposed logic-programming-based algorithm, ASP-DPOP, is shown to be experimentally faster DPOP, its imperative programming counterpart, on CDMG optimization problems with large domain sizes and is able to solve some problems that DPOP failed to solve due to memory limitations. In conclusion, we believe that these results show that the declarative programming paradigm is a promising new direction of research for DCOP researchers.

6. REFERENCES

[1] S. Citrigno, T. Eiter, W. Faber, G. Gottlob, C. Koch, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The dlv system: Model generator and application frontends. In *Proc. of WLP*, pages 128–137, 1997.

[2] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. clasp: A conflict-driven answer set solver. In *Proc. of LPNMR*, pages 260–265, 2007.

[3] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *Proc. of ICLP*, pages 579–597, 1990.

[4] S. Gupta, P. Jain, W. Yeoh, S. Ranade, and E. Pontelli. Solving customer-driven microgrid optimization problems as DCOPs. In *Proc. of the Distributed Constraint Reasoning Workshop*, pages 45–59, 2013.

[5] V. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-year Perspective*, pages 375–398, 1999.

[6] P. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1–2):149–180, 2005.

[7] I. Niemelä. Logic programming with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3–4):241–273, 1999.

[8] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *Proc. of IJCAI*, pages 1413–1420, 2005.

[9] W. Yeoh and M. Yokoo. Distributed problem solving. *AI Magazine*, 33(3):53–65, 2012.