Recent Advances in Graph-Based Abstractions for Modeling and Simulating Complex Systems

Jordan Jalving\textsuperscript{a,b} and Victor Zavala\textsuperscript{a,*}

\textsuperscript{a}Department of Chemical and Biological Engineering, University of Wisconsin-Madison, 1415 Engineering Dr, Madison, WI 53706, USA
\textsuperscript{b}Decision and Infrastructure Sciences Division, Argonne National Laboratory, 9700 South Cass Ave, Lemont, IL 60439, USA
victorzavala@wisc.edu

Abstract

Current graph-based approaches for modeling, simulation and optimization of complex cyber-physical systems have motivated the development of new graph-based abstractions. We propose an algebraic graph to represent physical connectivity in complex optimization models and a computing graph to capture computational aspects of cyber/control architectures. The algebraic graph facilitates the analysis and decomposition of optimization problems and the computing graph enables the simulation of optimization and control algorithms in virtual distributed environments. The proposed abstractions are implemented in a Julia software package called \textsc{Plasmo.jl}.

Keywords: cyber-physical, structured modeling, complex systems, decomposition

1. Introduction

Modeling and simulating cyber-physical systems is becoming increasingly important, but capturing interdependencies between cyber and physical systems in a coherent manner is technically challenging. Physical systems (such as chemical processes) can be driven by control systems, which in turn are cyber systems comprised of computing devices (e.g. sensors, controllers, actuators) that execute tasks (e.g., data processing, control action computation) and that exchange information (e.g., measurements and control actions) through a communication network. Physical systems are expressed mathematically in the form of \textit{algebraic models} while cyber systems are expressed in the form of \textit{algorithms} which are often executed under heterogeneous computing architectures that exhibit complex communication protocols.

This work discusses newly proposed \textit{graph-based} abstractions that facilitate modeling and simulation of cyber-physical systems. Graph-based modeling approaches have appeared in various engineering applications. Preisig (2007) used graph-theoretic approaches to perform on-line mass and energy balancing in large-scale plants given incomplete sensor data. Couenne et al. (2007) presented bond-graph modeling for chemical engineering applications and highlighted the advantages of model encapsulation and re-use. Elve and Preisig (2018) recently applied graph-based concepts to develop a process simulator with automatic code generation capabilities.

In this work we analyze the concept of an \textit{algebraic model graph} that facilitates the im-
plementation and solution of physical systems and the concept of a computing graph that facilitates simulation of cyber systems. The graph abstractions exploit physical and communication topology to facilitate model construction, data management, and analysis. Both abstractions are implemented in the Julia package Plasmo.jl (https://github.com/zavalab/Plasmo.jl).

2. Algebraic Graphs

Here we present a model graph abstraction to represent complex optimization models found in physical applications. A model graph $\mathcal{M}(\mathcal{N}, \mathcal{E})$ is a hypergraph wherein each node $n \in \mathcal{N}(\mathcal{M})$ has an associated component optimization model of the form:

$$\min_{x_n \in \mathcal{X}_n} f_n(x_n).\quad (1)$$

where $x_n$ are decision variables, $\mathcal{X}_n := \{x | c_n(x) \geq 0\}$ is the feasible set with associated constraint vector mapping $c_n(\cdot)$, and the mapping $f_n(\cdot)$ is a scalar objective function.

Coupling is captured between component models using link constraints. In particular, we consider linear linking constraints of the form:

$$\sum_{n \in \mathcal{N}(e)} \Pi_{e,n} x_n = 0 \quad e \in \mathcal{E}(\mathcal{M}).\quad (2)$$

where $e \in \mathcal{E}(\mathcal{M})$ are the hyperedges of the model graph and $n \in \mathcal{N}(e)$ is the set of nodes that support hyperedge $e$. The matrix $\Pi_{e,n}$ corresponds to the coefficients of the linking constraints between edge $e$ and node $n$.

We can represent hierarchical optimization structures using subgraphs as shown in Figure 1. A model graph can contain an arbitrary number of subgraphs by defining $\mathcal{G}(\mathcal{M}_0)$ (with elements $\mathcal{G}$) as the set of subgraphs defined on the parent model graph $\mathcal{M}_0$. This allows us to express the model graph in the form of Equation 3.

$$\min_{x,\mathcal{H}_0} \sum_{n \in \mathcal{N}(\mathcal{M}_0)} f_n(x_n) \quad \text{(Graph objective function)}\quad (3a)$$

subject to:

$$x_n \in \mathcal{X}_n, \quad n \in \mathcal{N}(\mathcal{M}_0) \quad \text{(Local node constraints)}\quad (3b)$$

$$\Pi_{\mathcal{H}_0} x_{\mathcal{H}_0} = 0 \quad \text{(Graph link constraints)}\quad (3c)$$

$$\Pi_{\mathcal{G}_0} x_{\mathcal{G}_0} = 0 \quad \mathcal{G}_0 \in \mathcal{G}(\mathcal{M}_0) \quad \text{(Subgraph link constraints)}\quad (3d)$$

Graph models provide structural information which facilitates decomposition and model management. For example, Formulation (3) has a partially separable structure because eliminating the linking constraints (3c and 3d) results in a fully separable problem. These problems can be solved using Schur decomposition schemes such as done in Chiang et al. (2014). It is also possible to aggregate the nodes in a model graph and into a traditional optimization form which can be solved with an off-the-shelf solver such as Ipopt (Wächter and Biegler (2006)). The graph structure also allows one to produce decompositions using graph analysis techniques such as partitioning and community detection (Tang et al. (2017)).
Figure 1: A model graph with two subgraphs. $\mathcal{M}_1$ contains three nodes ($n_1, n_2, n_3$) and one linking constraint ($e_1$). $\mathcal{M}_2$ contains three nodes ($n_4, n_5, n_6$) and two linking constraints ($e_2$ and $e_3$). $\mathcal{M}_0$ is the top layer of the graph containing all six nodes and connects the two subgraphs with a single linking constraint between nodes $n_3$ and $n_4$ ($e_4$).

3. Computing Graphs

Simulating a cyber system requires capturing real-time computing aspects such as latency, failures, and asynchronicity. To approach these challenges, we introduce the computing graph which is a directed multi-graph that we denote as $\mathcal{G}(\mathcal{N}, \mathcal{E})$ and that contains a set of nodes $\mathcal{N}(\mathcal{G})$ which execute tasks and edges $\mathcal{E}(\mathcal{G})$ which communicate attributes to other nodes.

In a computing graph, a node $n \in \mathcal{N}(\mathcal{G})$ contains a set of attributes $\mathcal{A}_n$ and tasks $\mathcal{T}_n$ and an edge $e \in \mathcal{E}(\mathcal{G})$ contains a pair of attributes $\mathcal{A}_e$ which it communicates between supporting nodes $\mathcal{N}(e)$. The attributes $\mathcal{A}_n$ represent data and tasks $\mathcal{T}_n$ are computations that operate with and/or change attributes. Tasks require execution time $\Delta \theta_t$ and edges involve communication delay $\Delta \theta_e$. Under the proposed abstraction, computing and communication can be synchronous (a task is not executed until all attributes are received) or asynchronous (a task is executed with current values). This enables capturing a wide range of behaviors seen in cyber-system applications.

A simple computing graph is depicted in Figure 2. Each node contains a single task which operates on its attributes $x$, $y$, and $z$ as input and updates one of their values. The nodes communicate attribute values with each other using six edges. For example, attribute $y$ is communicated to both nodes $n_2$ and $n_3$ which updates the value of $y$ on these respective nodes. The superscript $+$ means that attributes are updated after a given time which captures computing and communication delays.

The computing graph differs from the model graph abstraction in that it contains a dynamic component (a task) while a node in a model graph contains a static component (a model). Moreover, edges in a computing graph connect attributes while edges connect algebraic variables in a model graph. We highlight that the solution of a model graph is a computing task and we can thus use model graphs in individual nodes of the computing graph. This greatly facilitates simulating the behavior of cyber-physical systems.
Figure 2: Depiction of a computing graph with three nodes and six edges. Node $n_1$ computes $\text{task}_{n_1}$ using the data attributes ($x$, $y$, and $z$) and updates the value of attribute $y$. Similarly, node $n_2$ computes $\text{task}_{n_2}$ and updates attribute $x$, and node $n_3$ computes $\text{task}_{n_3}$ and updates attribute $z$. Attribute values are communicated between nodes using edges.

4. Case Study: Evaluating Control Architectures

This case study demonstrates how a model graph can be used to formulate and solve an optimal control problem for a dynamic physical system and how solving the model graph can be embedded in a computing graph to evaluate various distributed control architectures subject to real-time latency (i.e., computation and communication times). We consider a reactor-separator system (see Figure 3) from Stewart et al. (2010) which is a standard for evaluating distributed model predictive control.

We first create a model graph using Plasmo.jl for the process in Figure 3 using three nodes. Two nodes contain optimization models to track reactor setpoints, and the other node contains a model to track the separator setpoints. The three nodes are linked together using algebraic link constraints which represent the flows that couple the systems together.

The process is simulated using the computing graph for three different control architec-
tures. We simulate a **centralized** model predictive control (MPC) architecture wherein a central MPC controller calculates all inputs for the process. To do so, we solve the entire model graph as a single MPC node in the computing graph (Figure 4a) and use another computing node to simulate and advance the state of the plant.

We also decompose the process into three MPC controllers and simulate a **decentralized** control architecture wherein controllers do not communicate (Figure 4b) and a cooperative architecture (Figure 4c) such that controllers share state and intended control trajectories. For these cases, the nodes in the computing graph (MPC1, MPC2, and MPC3) compute tasks which solve and update the individual model nodes we defined in the model graph. We highlight that the computing graph captures the **asynchronous** behavior of the decentralized and cooperative schemes. Because we account for real computation times, the controllers inject their inputs when they finish their computation as opposed to all at the same time.

Figure 5 presents communication and computation patterns for each MPC algorithm. The centralized MPC pattern shows the communication delays between the plant and the controller (grey arrows), the time required to compute the control action (the purple bar), and highlights how the plant state advances continuously while computation and communication tasks execute. Despite the delays enforced for the controller, centralized MPC is able to drive the state to the setpoint (as shown in Figure 6). Decentralized MPC does
not require communication between controllers which decreases computing times, but we observe that the setpoint cannot be reached. Finally, cooperative MPC shows a more complex communication pattern but we observe that this mimics the performance of centralized MPC.

5. Conclusions

We have presented graph-based abstractions that facilitate modeling complex cyber-physical systems. We showed how a model graph abstraction facilitates modeling complex physical systems and how a computing graph enables simulating computational behaviors. We provided a case study using the package PLasmo.jl that demonstrates how these abstractions can be used to evaluate distributed computing architectures.

6. Acknowledgements

This material is based on work supported by the U.S. Department of Energy (DOE), Office of Science, under Contract No. DE-AC02-06CH11357. This work was also partially supported by the U.S. Department of Energy grant DE-SC0014114.

References


