Apr 20th, 12:00 AM - 12:00 AM

An Algorithm for Finding Data Dependencies in an Event Graph

Erik J. Jensen
Old Dominion University

AN ALGORITHM FOR FINDING DATA DEPENDENCIES IN AN EVENT GRAPH

Erik J. Jensen

Department of Electrical and Computer Engineering
Modeling and Simulation Engineering
Old Dominion University
5115 Hampton Blvd
Norfolk, Va, USA
ejjensen@odu.edu

ABSTRACT

This work presents an algorithm for finding data dependencies in a discrete-event simulation system, from the event graph of the system. The algorithm can be used within a parallel discrete-event simulation. Also presented is an experimental system and event graph, which is used for testing the algorithm. Results indicate that the algorithm can provide information about which vertices in the experimental event graph can affect other vertices, and the minimum amount of time in which this interference can occur.

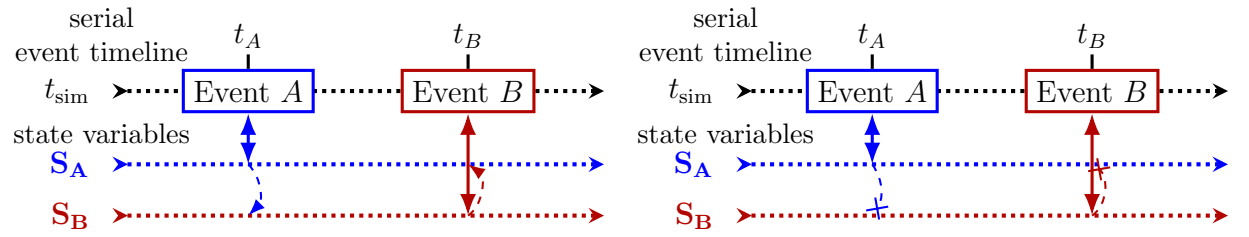
Keywords: parallel discrete-event simulation, event graphs, data dependencies, synchronization algorithm

1 MOTIVATION AND INTRODUCTION

This novel algorithm uses an event graph representation of a system to discover data dependencies between vertices, which may later be used in a parallel discrete event simulation (PDES) simulation executive. Given the event graph of the system, it is possible to determine the amount of time in which any one vertex can affect any other vertex, i.e. specifically the amount of time in which the execution of any one vertex can affect the state variables (SVs) that any other vertex (1) updates and/or (2) uses for decision-making. Figure 1 illustrates the execution of two events, *A* and *B*, in a serial simulation. N.B. in the context of this work, the terms "event" and "vertex" are used interchangeably, though for fig. 1, fig. 2, and fig. 3, event graphs and vertices corresponding to these events are not presented. In fig. 1, event *A* has a set of state variables $\mathbf{S_A}$ that it updates and/or uses for logical operations, and event *B* has a set of state variables $\mathbf{S_B}$ that it updates and/or uses for logical operations. In fig. 1(a), these sets $\mathbf{S_A}$ and $\mathbf{S_B}$ are not mutually exclusive for update operations, i.e. event A updates some SVs that event B also updates and/or uses for logical operations, and event B updates some SVs that event A also updates and/or uses for logical operations. In fig. 1(b), event *A* does not update any SVs shared by event *B*, and event *B* does not update any SVs shared by event *A*. For a serial simulation, either scenario is okay regarding the correctness of the simulation, but these serial examples are presented in order to better explain safe and unsafe parallel execution in fig. 2, and fig. 3. Figure 2 features the same events *A* and *B* as the previous example, but now they are selected to run in parallel with each other, as indicated by the ellipses. In fig. 2(a), events *A* and *B* are in conflict with each other because there is a data dependency, i.e. event *A depends* on some SVs that event *B* updates, and event *B depends* on some SVs that event *A* updates. In fig. 2(a), these events cannot execute in parallel safely

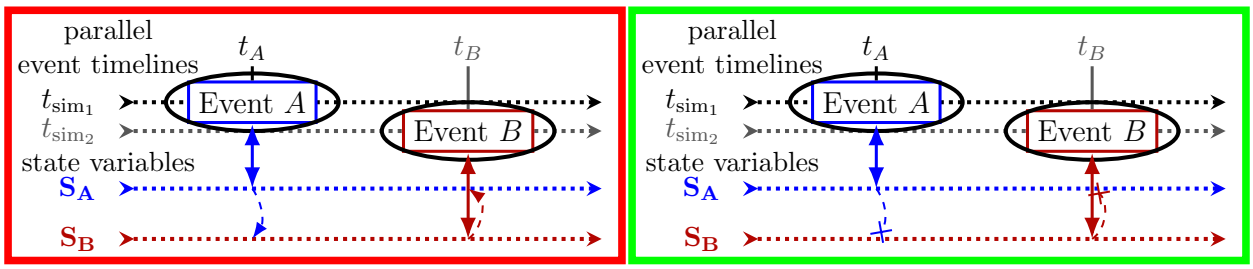
MSVSCC'23, April 20, 2023, Virginia Modeling, Analysis, and Simulation Center, VA, UNITED STATES;

because they will interfere with each other's updates and cause an incorrect simulation result. In fig. 2(b), there is no such data dependency and therefore they will run safely in parallel together, assuming that event A does not schedule any new events that update SVs in $\mathbf{S_B}$ before simulation time t_B.



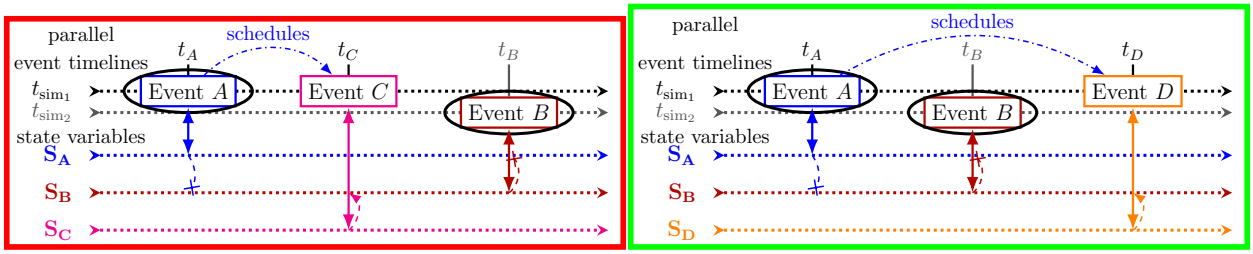
(a) Interaction. (b) No interaction.

Figure 1: Serial simulation. A dashed arrow ending in a pointed tip on the SV timeline of another event indicates the executing event updates some SVs that the other event also updates and/or uses for logical operations. A dashed arrow ending in an ex indicates there is no such interaction.



(a) Unsafe parallel execution. (b) Safe parallel execution.

Figure 2: Parallel simulation. Each thread or process i has its own event execution timeline and corresponding simulation time t_{sim_i}. A green or red border indicates that parallel execution is safe or unsafe.



(a) Unsafe parallel execution with a new event scheduled. (b) Safe parallel execution with a new event scheduled.

Figure 3: Parallel simulation with new events scheduled, depicting consequences of *interference time*.

Figure 3 depicts two possible outcomes of event A scheduling a new event. In fig. 3(a), event A schedules a new event, event C, that executes before simulation time t_B, and event C updates some SVs in $\mathbf{S_B}$. Because of this data dependency, if A and B execute in parallel together, depending on the simulation implementation, event C either can or will interfere with event B and cause an incorrect simulation result. Event A interferes with event B, meaning A causes some updates to $\mathbf{S_B}$ before event B executes, in simulation time. Contrast this with fig. 3(b), in which event A schedules a new event, event D. Similar to fig. 3(a), event D updates $\mathbf{S_B}$, but this is inconsequential for event B at simulation time t_B because this update occurs at simulation time

t_D, which is after t_B. Event A does not interfere with event B because it does not cause updates to $\mathbf{S_B}$ before event B executes. The interference time interval from the execution of event A to the execution of event D is greater than the time interval of the execution of event A to event B. In this case there is no data dependency, and events A and B will run safely together in parallel.

These simple examples illustrate the concepts of *data dependency* and *interference time interval* in the context of PDES and motivate this work. More generally, if two scheduled events A and B in the event set, $t_A < t_B$, (1) have a scheduled timestamp difference d that is less than the interference time δ_{AB} from event A to event B, *and* (2) event B cannot immediately update $\mathbf{S_A}$, events A and B are data-independent and safe to run together in parallel. Now that that is explained, further discussion of parallel event execution is beyond the scope of this work, but it motivates the discussion of the interference time interval table construction algorithm. For an entire simulated system, each possible interference time interval δ is computed from the event graph G and organized into an interference time interval table (ITIT), which defines δ between each pair of vertices. The primary contribution of this work is the algorithm that constructs the ITIT. The remainder of this paper is organized as follows: section 2 briefly summarizes foundational PDES and event graph knowledge, section 3 defines the ITIT construction algorithm, section 4 introduces the example system used in this work to demonstrate the algorithm in section 5, and section 6 concludes.

2 BACKGROUND

Schruben, Sargent, and Buss provide foundations for representing discrete-event simulation models with event graphs (Schruben 1983), (Freimer and Schruben 2001), (Sargent 1988), (Buss 1996), (Buss 2001). Fujimoto presents conservative and optimistic synchronization algorithms that are commonly used in PDES (Fujimoto 1989), (Das, Fujimoto, Panesar, Allison, and Hybinette 1994), (Fujimoto 2000). For large-scale applications, the optimistic Time-Warp algorithm is generally preferred over the conservative "lookahead" algorithm for performance advantages, but for Time-Warp, the number of rollbacks can increase super-linearly compared with the number of processes (Plagge, Carothers, Gonsiorowski, and Mcglohon 2018), (Hou, Yao, Wang, and Liao 2013), (Carothers and Perumalla 2010), (Carothers, Bauer, and Pearce 2002). Xia et al. present an extension to the event graph modeling formalism called *extended event graph* (EEG) and explore developing domain-specific modeling languages for PDES, specifically a language based on event graphs (Xia, Yao, and Mu 2012). Other works have also discussed how to use event graphs for PDES application development (Wang, Deng, Xing, Wang, and Yao 2015), (Yao, Yao, Bao, Bao, and Zhang 2018). Some works have explored how to use logical processes to model physical processes and how to use event scheduling for modeling interactions between physical processes (Zhu, Yao, Tang, and Tang 2017), (Poshtkohi, Ghaznavi-Ghoushchi, and Saghafi 2019). Pelligrini and Quaglia introduce *cross-state events*, which can reduce application development complexity and improve performance in PDES (Pellegrini and Quaglia 2019). Jefferson and Barnes present a hybrid conservative-optimistic synchronization algorithm framework called unified virtual time UVT (Jefferson and Barnes Jr 2023), (Jefferson and Barnes 2023).

3 ITIT CONSTRUCTION ALGORITHM

Algorithm 1 defines the construction of the ITIT. Given an event graph G with n number of vertices, the algorithm compares each vertex v_j in G with each vertex v_k in G, to determine how quickly secondary vertex v_k can affect the SVs that primary vertex v_j uses for decision-making and/or the SVs that v_j updates. In section 5, use of the algorithm is demonstrated with an example system and event graph, after the example system is introduced in section 4.

Algorithm 1: ITIT Construction Algorithm

Input: Event Graph G with n number of vertices

Result: $n \times n$ Interference Time Interval Table $\Delta(v_1, v_2)$

for *each vertex* v_j *in* G, $j \leftarrow 1..n$ **do**
  Identify the SVs $\mathbf{P_j}$ that primary vertex v_j updates and uses for decision-making.
  Identify the vertices $\mathbf{U_{P_j}}$ from all vertices in G that can update SVs in $\mathbf{P_j}$.
  for *each vertex* v_k *in* G, $k \leftarrow 1..n$ **do**
    Identify the vertices $\mathbf{S_k}$ from all vertices in G that secondary vertex v_k can reach.
    Define $\mathbf{I_{jk}}$ as $\mathbf{U_{P_j}} \cap \mathbf{S_k}$.
    if $\mathbf{I_{jk}} \neq \emptyset$ **then**
      Identify the time intervals $\mathbf{T_{jk}}$ of the shortest paths from v_k to each vertex in $\mathbf{I_{jk}}$.
      Define the minimum interference time interval $\Delta(v_j, v_k) \leftarrow \delta_{v_j,v_k} \leftarrow \min(\mathbf{T_{jk}})$.
    else
      Define the minimum interference time interval $\Delta(v_j, v_k) \leftarrow \delta_{v_j,v_k} \leftarrow \infty$.
    end
  end
end

4 EXAMPLE SYSTEM

For testing the ITIT construction algorithm, an event graph is borrowed from Sargent's paper, "Event Graph Modeling for Simulation with an Application to Flexible Manufacturing Systems" (Sargent 1988). Sargent models a simple flexible manufacturing system. For this work, Sargent's system has been extended to create a new system that offers greater parallelism and rigorous testing of the algorithm. This heavily modified system is called the Extended Flexible Manufacturing System (EFMS). Briefly, the most significant modifications are that now: (1) there are multiple (N) manufacturing stages, through which each part traverses sequentially for processing, (2) a global transporter ferries parts between the manufacturing stages, (3) there can be more than two processing stations in each manufacturing stage, (4) parts can be rejected and sent to the previous stage for re-processing, (5) each machine station has a rejected parts buffer, and (6) each station zero has a loading buffer, an unloading buffer for processed parts, and an unloading buffer for rejected parts. Figure 4 depicts the physical system modeled by EFMS.

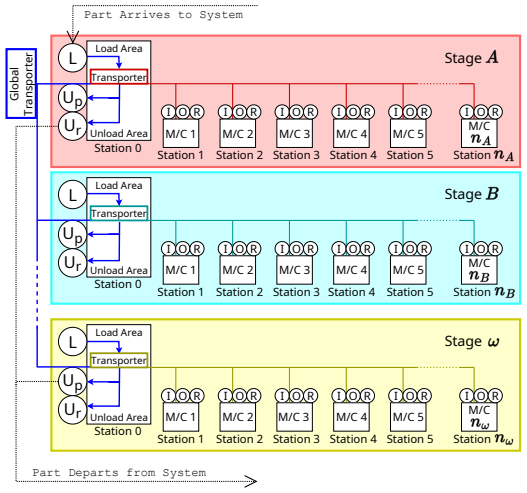


Figure 4: EFMS, the physical system, with ω stages and n_β processing stations per stage, where β identifies the stage number.

To quickly summarize the meanings of the state variables, G_S, G_b, and G_P define states for the global transporter; L_β, $U_{\beta p}$, and $U_{\beta r}$ define states for the station zero loading and unloading buffers; $T_{\beta s}$ and $T_{\beta b}$ define states for the local transporters; and $M_{\beta j}$, $I_{\beta j}$, $O_{\beta j}$, and $R_{\beta j}$ define states for each processing station. Complete definitions of state variables, delays, conditions, updates, and scheduling are omitted due to consideration for space and for the focus of this work, which is the introduction of the algorithm, which does not require a fine-grained understanding of EFMS. Figure 6 depicts the EFMS event graph,

which briefly describes the functionality of each vertex. The simulation application is object-oriented and structured in such a way that there are simulated system components containing pieces of the complete event graph. These simulated system component objects communicate with each other to get SV values that are needed for vertex updates. For this EFMS event graph, each of the N manufacturing stages has one station zero system component and n_β machine station system components. In fig. 6, vertices are colored according to functionality. A station zero system component contains the base station zero functionality (blue vertices) captured from Sargent's orginal system and the additional functionality needed for the global transporter (green vertices) in EFMS. A machine station system component contains Sargent's machine station functionality (red vertices) and the added functionality needed for rejected parts (yellow vertices) in EFMS. There is only one arrive vertex system component, and it functions to generate new parts and send newly-arrived parts to the first-stage loading buffer L_A. In fig. 6, dashed magenta lines group the vertices owned by the type of simulated system component in the simulation application.

5 ITIT GENERATION FOR EXAMPLE SYSTEM

Figure 5, fig. 8, and fig. 10 show ITIT output for a N=2,n_A=2,n_B=2 EFMS system at different scales. Figure 5 shows the highest-level interactions, where manufacturing stages are delineated by dark blue lines and system components within a manufacturing stage are delineated by bright blue lines. Darker heatmap colors indicate more immediate interference between vertex pairs, compared with lighter colors. The large diagonal squares showing each manufacturing stage interacting with itself show darker colors and quicker interference than the squares showing a stage interacting with the other stage. Within these large dark-blue diagonal squares, the darkest colors and most immediate interference times are found within the smaller bright-blue diagonal squares. These diagonal bright-blue squares show a system component's internal vertex interactions, e.g. machine station 1 in stage B interacting with itself. This behavior is expected because the vertices within a system component update and use a common set of SVs belonging to that system component. Rows or columns that are blank (white) in the ITIT heatmap figures correspond to vertices that only do scheduling and do not update SVs or use SVs for decision-making.

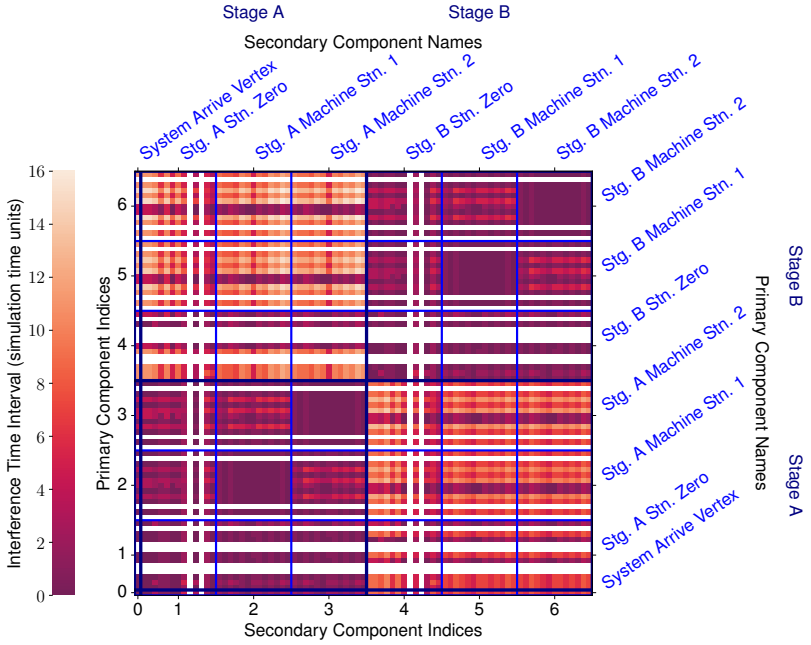


Figure 5: Interference time interval heatmap for a full system, N=2,n_A=2,n_B=2.

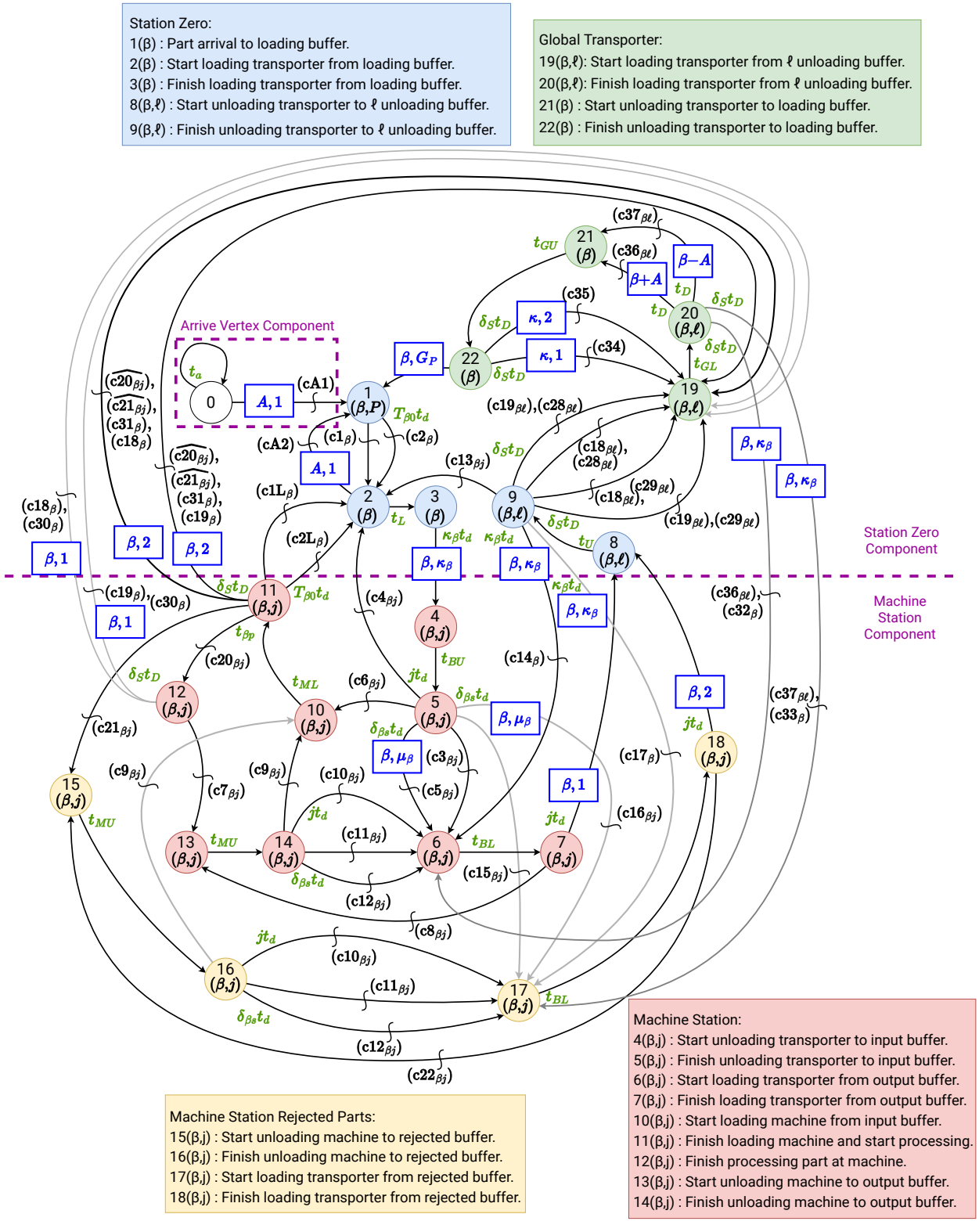


Figure 6: EFMS event graph. Dashed pink lines separate simulated system components, and vertex color refers to functionality within system component type. Edges have different colors for visualization purposes.

The non-diagonal large dark-blue squares show generally slower interference times. Darker colors in the large lower-right square, compared with the large upper-left square, indicate that manufacturing stage B generally affects manufacturing stage A (lower-right square) somewhat differently from how manufacturing stage A affects manufacturing stage B (upper-left square). Figure 8 and fig. 10 and accompanying text explain the causes of some of these differences. The following subsections take some vertices shown in fig. 8 and fig. 10 and demonstrate how algorithm 1 generates the stated interference time intervals, interfering vertices, and interfering SVs for those vertices.

5.1 ITIT Construction Algorithm Demonstrations

The following subsections present demonstrations explicitly showing how the ITIT construction algorithm generates some of the data shown in fig. 5, fig. 8, and fig. 10. For specific pairs of *primary vertex* $= v_j$, *secondary vertex* $= v_k$ vertices, demonstration 1 explains why for $v_j = 18(\beta{=}A, j{=}1)$ and $v_k = 18(\beta{=}B, j{=}1)$ the interference time interval in fig. 8 is 7 time units, and demonstration 2 explains why for $v_j = 18(\beta{=}B, j{=}1)$ and $v_k = 18(\beta{=}A, j{=}1)$ the interference time interval in fig. 10 is 11 time units.

5.1.1 Algorithm Demonstration 1

For this demonstration of the ITIT construction algorithm, the primary vertex is $18(\beta{=}A, j{=}1)$ and the secondary vertex is $18(\beta{=}B, j{=}1)$. The stated interference time is 7, as emphasized by the blue circle in fig. 8. Table 1 shows that primary vertex $v_j = 18(\beta{=}A, j{=}1)$ has three SVs that it updates and/or uses for decision-making in the set $\mathbf{P_j}$, and there are 21 vertices in $\mathbf{U_{P_j}}$ that can update those SVs. The secondary vertex $v_k = 18(\beta{=}B, j{=}1)$ can reach all vertices in EFMS event graph G, except for vertex 0: those vertices make the set $\mathbf{S_k}$. $\mathbf{I_{jk}}$, which is the intersection of $\mathbf{U_{P_j}}$ and $\mathbf{S_k}$, contains exactly the same vertices as $\mathbf{U_{P_j}}$ because $\mathbf{S_k}$ contains all vertices in G except vertex 0, and vertex 0 is not in $\mathbf{U_{P_j}}$. $\mathbf{T_{jk}}$ contains the shortest-path times from secondary vertex $v_k = 18(\beta{=}B, j{=}1)$ to each vertex in $\mathbf{I_{jk}}$, ordered corresponding to the ordering of vertices in $\mathbf{I_{jk}}$. Therefore, $\delta(v_j, v_k)$ is 7, meaning that secondary vertex $v_k = 18(\beta{=}B, j{=}1)$ can reach interfering vertex $1(\beta{=}A,P)$ in 7 simulation time units, and that is the shortest path to interfere with primary vertex $v_j = 18(\beta{=}A, j{=}1)$.

Table 1: ITIT construction algorithm intermediate and final values for primary vertex $v_j = 18(\beta{=}A, j{=}1)$ and secondary vertex $v_k = 18(\beta{=}B, j{=}1)$.

Set/Var.	Alg. Line	Value(s)
v_j	line 1	$18(\beta{=}A, j{=}1)$
$\mathbf{P_j}$	line 2	M_{A1}, R_{A1}, T_{As}
$\mathbf{U_{P_j}}$	line 3	$1(\beta{=}A,P)$, $3(\beta{=}A)$, $9(\beta{=}A,\ell{=}1)$, $9(\beta{=}A,\ell{=}2)$, $20(\beta{=}A,\ell{=}1)$, $5(\beta{=}A,j{=}1)$, $7(\beta{=}A,j{=}1)$, $10(\beta{=}A,j{=}1)$, $11(\beta{=}A,j{=}1)$, $12(\beta{=}A,j{=}1)$, $13(\beta{=}A,j{=}1)$, $14(\beta{=}A,j{=}1)$, $15(\beta{=}A,j{=}1)$, $16(\beta{=}A,j{=}1)$, $18(\beta{=}A,j{=}1)$, $5(\beta{=}A,j{=}2)$, $7(\beta{=}A,j{=}2)$, $11(\beta{=}A,j{=}2)$, $14(\beta{=}A,j{=}2)$, $16(\beta{=}A,j{=}2)$, $18(\beta{=}A,j{=}2)$
v_k	line 4	$18(\beta{=}B, j{=}1)$
$\mathbf{S_k}$	line 5	all vertices in G minus vertex 0
$\mathbf{I_{jk}}$	line 6	exactly the same vertices as $\mathbf{U_{P_j}}$
$\mathbf{T_{jk}}$	line 8	7, 8, 13, 13, 10, 10, 11, 10, 11, 15, 11, 12, 11, 12, 11, 11, 11, 12, 12, 13, 12
$\delta(v_j, v_k)$	line 9	7

This shortest path from secondary vertex $v_k = 18(\beta{=}B, j{=}1)$ to interfering vertex $1(\beta{=}A,P)$ is:

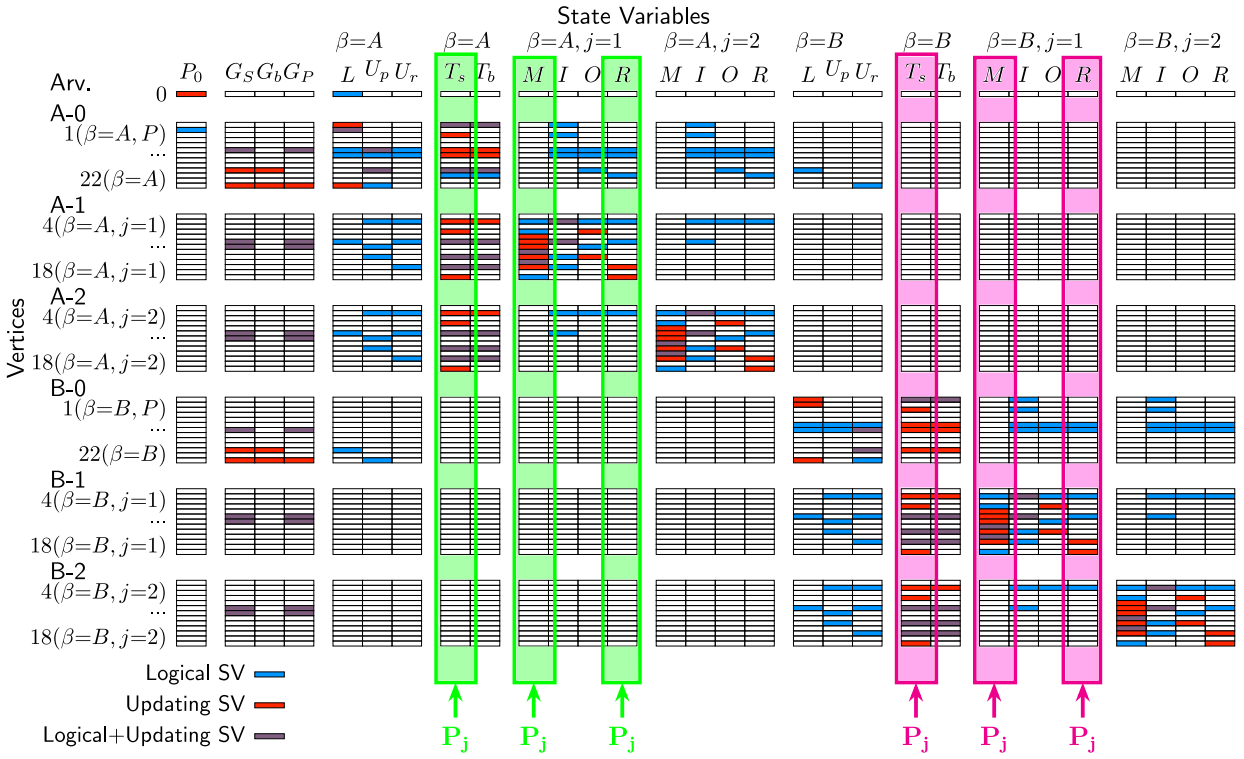
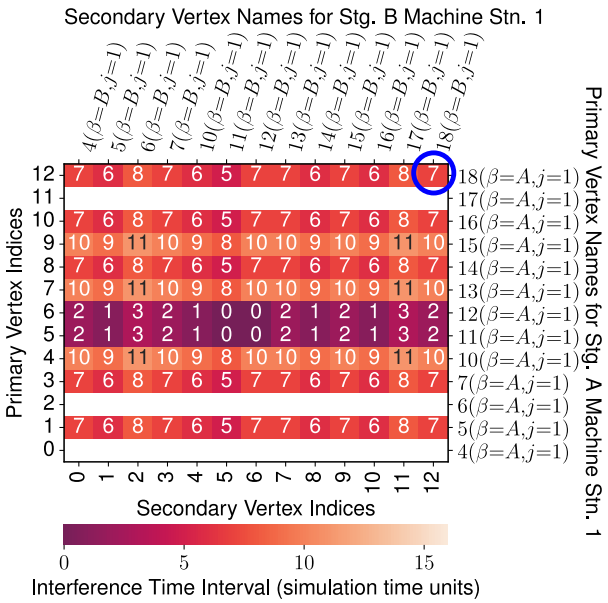


Figure 7: SVs that each vertex updates, uses for logical operations, or both. The set of SVs $\mathbf{P_j}$ is highlighted for algorithm demonstrations 1 (green) and 2 (pink). Anywhere the highlighted SV in $\mathbf{P_j}$ intersects a row with a red or purple cell, the vertex corresponding to that row is in $\mathbf{U_{P_j}}$. Note for each system component, only the first and last vertices are labeled, and the values of $\mathbf{P_j}$ and $\mathbf{U_{P_j}}$ depicted here match those SVs and vertices listed in table 1 (green) and table 2 (pink).

1. vertex 18(β=B, j=1) to
2. vertex 8(β=B, ℓ=2) in $1t_d =$ 1 time unit, to
3. vertex 9(β=B, ℓ=2) in $t_U =$ 1 time unit, to
4. vertex 19(β=B, ℓ=2) in 0 time units, to
5. vertex 20(β=B, ℓ=2) in $t_{GL} =$ 2 time units, to
6. vertex 21(β=A) in $t_D =$ 1 time unit, to
7. vertex 22(β=A) in $t_{GU} =$ 2 time units, to
8. vertex 1(β=A, P) in 0 time units.

Summing each edge on the path, $1+1+0+2+1+2+0=7$, which is the value for the interference time interval presented in fig. 8. Figure 10 indicates in the table entry for "12,12", where 12 and 12 are the corresponding system component vertex indices of the primary and secondary vertices in this demonstration, that the interfering vertex is 1(β=A, P) and the interfering SV in that vertex is T_{As}. Vertex 1(β=A, P) as the shortest-distance interfering vertex is confirmed from the output of the ITIT construction algorithm in table 1. Defining the updates in fig. 9 for the primary vertex v_j = 18(β=A, j=1) and interfering vertex 1(β=A, P) illustrates how vertex 1(β=A, P) interferes with primary vertex 18(β=A, j=1). The interfering vertex updates SV T_{As}, and the primary vertex also updates T_{As}, as indicated by the red boxes around those SV updates. It is problematic for both vertices to update T_{As} in parallel because it cannot be guaranteed which vertex will update T_{As} first and at which simulation time, according to the updating vertices, that update will occur.



	Interfering Vertices	Interfering SVs
0,0 - 0,12		
1,0 - 1,12	$1(\beta{=}A,P)$	T_{As},T_{Ab}
2,0 - 2,12		
3,0 - 3,12	$1(\beta{=}A,P)$	T_{As}
4,0 - 4,12	$10(\beta{=}A,j{=}1)$	M_{A1}
5,0 - 5,5	$11(\beta{=}B,j{=}1)$	G_b,G_S
5,6 - 5,6	$12(\beta{=}B,j{=}1)$	G_b,G_S
5,7 - 5,10	$11(\beta{=}B,j{=}1)$	G_b,G_S
5,11 - 5,12	$9(\beta{=}B,\ell{=}2)$,$11(\beta{=}B,j{=}1)$	G_b,G_S
6,0 - 6,5	$11(\beta{=}B,j{=}1)$	G_b,G_S
6,6 - 6,6	$12(\beta{=}B,j{=}1)$	G_b,G_S
6,7 - 6,10	$11(\beta{=}B,j{=}1)$	G_b,G_S
6,11 - 6,12	$9(\beta{=}B,\ell{=}2)$,$11(\beta{=}B,j{=}1)$	G_b,G_S
7,0 - 7,12	$10(\beta{=}A,j{=}1)$	M_{A1}
8,0 - 8,12	$1(\beta{=}A,P)$	T_{As},T_{Ab}
9,0 - 9,12	$10(\beta{=}A,j{=}1)$	M_{A1}
10,0 - 10,12	$1(\beta{=}A,P)$	T_{As},T_{Ab}
11,0 - 11,12		
12,0 - 12,12	$1(\beta{=}A,P)$	T_{As}

	t_a	t_D	t_{GL}	t_{GU}	t_d	t_p	t_L	t_U	t_{BL}	t_{BU}	t_{ML}	t_{MU}
Min. Delay	2	1	2	2	1	4	1	1	1	1	1	1

Figure 8: Interference time interval heatmap at the system component scale, in a $N{=}2, n_A{=}2, n_B{=}2$ system, for stage A, machine station 1 primary vertices, and stage B, machine station 1 secondary vertices. Shortest-distance interfering vertices and SVs are defined for each *primary vertex*,*secondary vertex* vertex pair, in the range of vertices containing that pair. Ranges are specified in row labels. The bottom table defines the minimum delay time for each scheduling delay.

5.1.2 Algorithm Demonstration 2

For this second demonstration, the vertices are switched so that now vertex $18(\beta{=}B, j{=}1)$ is the primary vertex and vertex $18(\beta{=}A, j{=}1)$ is the secondary vertex. The stated interference time is 11, as emphasized by the blue circle in fig. 10. Table 2 shows the values in the ITIT construction algorithm for this case.

The shortest path from secondary vertex $v_k = 18(\beta{=}A, j{=}1)$ to interfering vertex $1(\beta{=}B,P)$ is:

1. vertex $18(\beta{=}A, j{=}1)$ to
2. vertex $8(\beta{=}A, \ell{=}2)$ in $1t_d =$ 1 time unit, to
3. vertex $9(\beta{=}A, \ell{=}2)$ in $t_U =$ 1 time unit, to
4. vertex $2(\beta{=}A)$ in 0 time units, to
5. vertex $11(\beta{=}A, j{=}1)$ in 0 time units, to
6. vertex $12(\beta{=}A, j{=}1)$ in t_{Ap} 4 time units, to
7. vertex $19(\beta{=}A, \ell{=}1)$ in 0 time units, to
8. vertex $20(\beta{=}A, \ell{=}1)$ in $t_{GL} =$ 2 time units, to
9. vertex $21(\beta{=}B)$ in $t_D =$ 1 time unit, to
10. vertex $22(\beta{=}B)$ in $t_{GU} =$ 2 time units, to
11. vertex $1(\beta{=}B,P)$ in 0 time units.

Summing each edge on the path, $1+1+0+0+4+0+2+1+2+0 = 11$, which is the value for the interference time interval presented in fig. 10. Note this path differs significantly from the path in section 5.1.1. Figure 10 indicates in the table entry for "12,12", where 12 and 12 are the corresponding system component vertex indices of the primary and secondary vertices in this demonstration, that the interfering vertex is

Figure 9: Vertex update definitions for primary vertex $18(\beta{=}A, j{=}1)$ and interfering vertex $1(\beta{=}A, P)$.

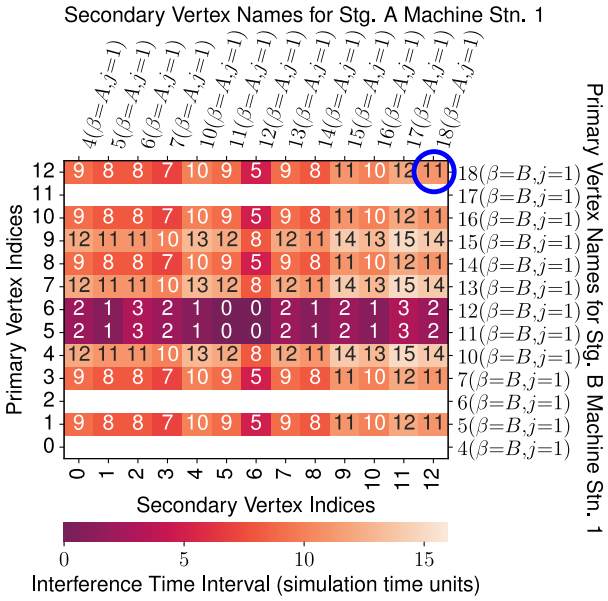
Vertex $18(\beta{=}A, j{=}1)$: (Finish loading the A-stage local transporter from rejected part output buffer R_{A1})
Required condition definitions:
c22$_{\mathbf{A1}}$ $M_{A1} = -2$.
1. **Update** $R_{A1} \leftarrow R_{A1} - 1$,
2. **update** $T_{As} \leftarrow 0$,
3. ***schedule*** Vertex $8(\beta, \ell)$ with attributes $(A, 2)$ after delay $1t_d$.
4. **If** (c22$_{A1}$), then: (a) ***Schedule*** Vertex $15(\beta{=}A, j{=}1)$.

Vertex $1(\beta{=}A, P)$: (Part arrival to manufacturing stage $\beta{=}A$ loading buffer L_A)
Required condition definitions:
c1$_{\mathbf{A}}$ $L_A > 0 \quad \wedge \quad \exists k \in \{1 \ldots n_A\} \ni I_{Ak} < C_{\text{mach}} \quad \wedge \quad T_{Ab} = 0 \quad \wedge \quad T_{As} = 0.$
c2$_{\mathbf{A}}$ $L_A > 0 \quad \wedge \quad \exists k \in \{1 \ldots n_A\} \ni I_{Ak} < C_{\text{mach}} \quad \wedge \quad T_{Ab} = 0 \quad \wedge \quad T_{As} > 0.$
1. **Update** $L_A \leftarrow L_A + P$.
2. **If** (c1$_A$), then: (a) **Update** $T_{Ab} \leftarrow 1$, (b) ***schedule*** Vertex $2(\beta{=}A)$.
3. **If** (c2$_A$), then: (a) **Update** $T_{A0} \leftarrow T_{As}$, (b) **update** $T_{Ab} \leftarrow 1$, (c) **update** $T_{As} \leftarrow 0$, (d) ***schedule*** Vertex $2(\beta{=}A)$ after delay $T_{\beta 0} t_d$.

$1(\beta{=}B, P)$ and the interfering SV in that vertex is T_{Bs}. Vertex $1(\beta{=}B, P)$ as the shortest-distance interfering vertex is confirmed from the output of the ITIT construction algorithm in table 2. The updates for the primary vertex $v_j = 18(\beta{=}B, j{=}1)$ and interfering vertex $1(\beta{=}B, P)$ are the same as the updates defined for the vertices $18(\beta{=}A, j{=}1)$ and $1(\beta{=}A, P)$ in the previous demonstration in section 5.1.1, except that for any SV or vertex where $\beta{=}A$ in those updates, $\beta{=}B$ for these updates. The result is that the interfering vertex $1(\beta{=}B, P)$ updates the SV T_{Bs}, and the primary vertex $v_j = 18(\beta{=}B, j{=}1)$ also updates T_{Bs}. This is problematic for the same reason as in the previous demonstration in section 5.1.1.

Table 2: ITIT construction algorithm intermediate and final values for primary vertex $v_j = 18(\beta{=}B, j{=}1)$ and secondary vertex $v_k = 18(\beta{=}A, j{=}1)$.

Set/Var.	Alg. Line	Value(s)
v_j	line 1	$18(\beta{=}B, j{=}1)$
$\mathbf{P_j}$	line 2	M_{B1}, R_{B1}, T_{Bs}
$\mathbf{U_{P_j}}$	line 3	$1(\beta{=}B,P)$, $3(\beta{=}B)$, $9(\beta{=}B,\ell{=}1)$, $9(\beta{=}B,\ell{=}2)$, $20(\beta{=}B,\ell{=}2)$, $5(\beta{=}B,j{=}1)$, $7(\beta{=}B,j{=}1)$, $10(\beta{=}B,j{=}1)$, $11(\beta{=}B,j{=}1)$, $12(\beta{=}B,j{=}1)$, $13(\beta{=}B,j{=}1)$, $14(\beta{=}B,j{=}1)$, $15(\beta{=}B,j{=}1)$, $16(\beta{=}B,j{=}1)$, $18(\beta{=}B,j{=}1)$, $5(\beta{=}B,j{=}2)$, $7(\beta{=}B,j{=}2)$, $11(\beta{=}B,j{=}2)$, $14(\beta{=}B,j{=}2)$, $16(\beta{=}B,j{=}2)$, $18(\beta{=}B,j{=}2)$
v_k	line 4	$18(\beta{=}A, j{=}1)$
$\mathbf{S_k}$	line 5	all vertices in G minus vertex 0
$\mathbf{I_{jk}}$	line 6	exactly the same vertices as $\mathbf{U_{P_j}}$
$\mathbf{T_{jk}}$	line 8	11, 12, 17, 17, 14, 14, 15, 14, 15, 19, 15, 16, 15, 16, 15, 15, 16, 16, 17, 16, 15
$\delta(v_j, v_k)$	line 9	11



	Interfering Vertices	Interfering SVs
0,0 - 0,12		
1,0 - 1,12	$1(\beta{=}B,P)$	T_{Bs},T_{Bb}
2,0 - 2,12		
3,0 - 3,12	$1(\beta{=}B,P)$	T_{Bs}
4,0 - 4,12	$10(\beta{=}B,j{=}1)$	M_{B1}
5,0 - 5,1	$11(\beta{=}A,j{=}1)$	G_b,G_S
5,2 - 5,3	$9(\beta{=}A,\ell{=}1)$,$11(\beta{=}A,j{=}1)$	G_b,G_S
5,4 - 5,5	$11(\beta{=}A,j{=}1)$	G_b,G_S
5,6 - 5,6	$12(\beta{=}A,j{=}1)$	G_b,G_S
5,7 - 5,12	$11(\beta{=}A,j{=}1)$	G_b,G_S
6,0 - 6,1	$11(\beta{=}A,j{=}1)$	G_b,G_S
6,2 - 6,3	$9(\beta{=}A,\ell{=}1)$,$11(\beta{=}A,j{=}1)$	G_b,G_S
6,4 - 6,5	$11(\beta{=}A,j{=}1)$	G_b,G_S
6,6 - 6,6	$12(\beta{=}A,j{=}1)$	G_b,G_S
6,7 - 6,12	$11(\beta{=}A,j{=}1)$	G_b,G_S
7,0 - 7,12	$10(\beta{=}B,j{=}1)$	M_{B1}
8,0 - 8,12	$1(\beta{=}B,P)$	T_{Bs},T_{Bb}
9,0 - 9,12	$10(\beta{=}B,j{=}1)$	M_{B1}
10,0 - 10,12	$1(\beta{=}B,P)$	T_{Bs},T_{Bb}
11,0 - 11,12		
12,0 - 12,12	$1(\beta{=}B,P)$	T_{Bs}

	t_a	t_D	t_{GL}	t_{GU}	t_d	t_p	t_L	t_U	t_{BL}	t_{BU}	t_{ML}	t_{MU}
Min. Delay	2	1	2	2	1	4	1	1	1	1	1	1

Figure 10: Interference time interval heatmap, similar to that in fig. 8, for stage B, machine station 1 primary vertices, and stage A, machine station 1 secondary vertices.

6 DISCUSSION AND CONCLUSIONS

After explaining the concepts of *data dependency* and *interference time interval* in the context of PDES in section 1 with simple illustrative examples, section 3 defines an algorithm that identifies the interference time for each vertex pair in an event graph. The example system introduced in section 4 is used in section 5 as a means to demonstrate the algorithm working to generate interfernce times. The demonstrations selected show that interference times are not necessarily symmetric for a given pair of vertices, i.e. $\delta_{AB} \neq \delta_{BA}$. In conclusion, the ITIT construction algorithm exhibits the ability to find data dependencies and the set of interference times for an event graph. This information may later be used to select safe parallel events in a PDES. Future work includes the development of PDES simulation executive algorithms that incorporate the ITIT for the selection of safe parallel events and execution time speedup.

REFERENCES

Buss, A. 2001. “Basic event graph modeling”. *Simulation News Europe* vol. 31 (1), pp. 1–6.

Buss, A. H. 1996. “Modeling with event graphs”. In *Proceedings of the 28th conference on winter simulation*, pp. 153–160.

Carothers, C. D., D. Bauer, and S. Pearce. 2002. “ROSS: A high-performance, low-memory, modular Time Warp system”. *Journal of Parallel and Distributed Computing* vol. 62 (11), pp. 1648–1669.

Carothers, C. D., and K. S. Perumalla. 2010. “On deciding between conservative and optimistic approaches on massively parallel platforms”. In *Proceedings of the 2010 Winter Simulation Conference*, pp. 678–687.

Das, S., R. Fujimoto, K. Panesar, D. Allison, and M. Hybinette. 1994. "GTW: a time warp system for shared memory multiprocessors". In *Proceedings of Winter Simulation Conference*, pp. 1332–1339. IEEE.

Freimer, M., and L. Schruben. 2001. "Stochastic Optimization Using Simulation: Graphical Representation of IPA Estimation". In *Proceedings of the 33nd Conference on Winter Simulation*, WSC '01, pp. 422–427. USA, IEEE Computer Society.

Fujimoto, R. M. 1989. "Time Warp on a shared memory multiprocessor". Technical report, UTAH UNIV SALT LAKE CITY SCHOOL OF COMPUTING.

Fujimoto, R. M. 2000. *Parallel and distributed simulation systems*, Volume 300. Citeseer.

Hou, B., Y. Yao, B. Wang, and D. Liao. 2013. "Modeling and simulation of large-scale social networks using parallel discrete event simulation". *Simulation* vol. 89 (10), pp. 1173–1183.

Jefferson, D. R., and P. D. Barnes. 2023. "Virtual Time III, Part 2: Combining conservative and optimistic synchronization". *ACM Transactions on Modeling and Computer Simulation* vol. 32 (4), pp. 1–21.

Jefferson, D. R., and P. Barnes Jr. 2023. "Virtual Time III, Part 1: Unified Virtual Time synchronization for parallel discrete event simulation". *ACM Transactions on Modeling and Computer Simulation* vol. 32 (4), pp. 1–29.

Pellegrini, A., and F. Quaglia. 2019. "Cross-state events: A new approach to parallel discrete event simulation and its speculative runtime support". *Journal of parallel and distributed computing* vol. 132, pp. 48–68.

Plagge, M., C. D. Carothers, E. Gonsiorowski, and N. Mcglohon. 2018. "Nemo: A massively parallel discrete-event simulation model for neuromorphic architectures". *ACM Transactions on Modeling and Computer Simulation (TOMACS)* vol. 28 (4), pp. 1–25.

Poshtkohi, A., M. Ghaznavi-Ghoushchi, and K. Saghafi. 2019. "PSML: parallel system modeling and simulation language for electronic system level". *The Journal of Supercomputing* vol. 75, pp. 2691–2724.

Sargent, R. G. 1988. "Event graph modelling for simulation with an application to flexible manufacturing systems". *Management science* vol. 34 (10), pp. 1231–1251.

Schruben, L. 1983. "Simulation modeling with event graphs". *Communications of the ACM* vol. 26 (11), pp. 957–963.

Wang, B., B. Deng, F. Xing, D. Wang, and Y. Yao. 2015. "Partitioned event graph: formalizing LP-based modelling of parallel discrete-event simulation". *Mathematical and Computer Modelling of Dynamical Systems* vol. 21 (2), pp. 153–179.

Xia, W., Y. Yao, and X. Mu. 2012. "An extended event graph-based modelling method for parallel and distributed discrete-event simulation". *Mathematical and Computer Modelling of Dynamical Systems* vol. 18 (3), pp. 287–306.

Yao, F., Y. Yao, S. Bao, Y. Bao, and X. Zhang. 2018. "An Integration Architecture Based on Event Graph for PDES Application Development". In *Proceedings of the 2018 2nd International Conference on Management Engineering, Software Engineering and Service Sciences*, pp. 250–256.

Zhu, F., Y. Yao, W. Tang, and J. Tang. 2017. "A hierarchical composite framework of parallel discrete event simulation for modelling complex adaptive systems". *Simulation Modelling Practice and Theory* vol. 77, pp. 141–156.

AUTHOR BIOGRAPHIES

ERIK J. JENSEN M.Sc. is an MSE Ph.D. student in ECE who also works on data analytics projects with VMASC. His email address is ejjensen@odu.edu.