The computational complexity of the high level architecture data distribution management matching and connecting processes

Mikel D. Petty a,*, Katherine L. Morse b,1

a Virginia Modeling, Analysis and Simulation Center, Old Dominion University, Norfolk, VA 23529, USA
b Science Applications International Corporation, 10260 Campus Point Drive MS C-2, San Diego, CA 92121, USA

Received 15 April 2003; accepted 15 October 2003
Available online 5 June 2004

Abstract

The High Level Architecture (HLA) is an architecture standard for constructing federations of distributed simulations that exchange data at run-time. HLA includes interest management capabilities (known as “Data Distribution Management”) that reduce the data sent during a federation execution using the simulations’ run-time declarations describing the data they plan to send and wish to receive. The total computation associated with Data Distribution Management during the execution of a federation can be separated into four processes: declaring, matching, connecting, and routing. These processes are defined and the computational complexities of the matching and connecting processes during a federation execution are determined. The matching process requires total time with a lower bound in $\Omega(n \log n)$ and an upper bound in $O(n^2)$, where $n$ is the number of run-time data distribution actions performed by the simulations. The commonly used approach to implementing the connecting process, multicast grouping, contains a problem that is NP-complete.

© 2004 Elsevier B.V. All rights reserved.

Keywords: High level architecture; Data distribution management; Multicast; Computational complexity; NP-completeness; Interest management

* Corresponding author. Tel.: +1-757-686-6210.
E-mail addresses: mpetty@vmasc.odu.edu (M.D. Petty), katherine.l.morse@saic.com (K.L. Morse).
1 Tel.: +1-858-826-6728.

1569-190X/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2003.10.004
1. Introduction

The High Level Architecture (HLA) is an architecture standard for distributed simulations [1]. HLA supports the development of federations of network-connected simulations, called federates, that exchange data at run-time. HLA includes interest management services, called Data Distribution Management (DDM), that use the federates’ declarations of data they will send and desire to receive to reduce the data sent and received during a federation execution. The DDM-associated computation to be performed by the federates and the supporting HLA Run-Time Infrastructure during the execution of a federation can be separated into four processes: declaring, matching, connecting, and routing. During a federation execution, federates declare the data they will send and the data they wish to receive (declaring); those declarations are matched to determine which sending and receiving federates should be connected for data transfer (matching); those connections are made using the available network capabilities (connecting); and data is routed between the federations along those connections (routing). In this paper the computational complexities of the matching and connecting DDM processes during a federation execution are examined.

First, background information is provided. Then formal problems equivalent to the matching process are defined and shown to have time complexity with a lower bound in $\Omega(n \log n)$ and an upper bound in $O(n^2)$. Next, a formal problem inherent in the common multicast implementation of the connecting process, that of assigning the federates to multicast groups, is defined and shown to be NP-complete. The distinction between the NP-completeness of multicast grouping and the computational complexity of the connecting process is discussed. Finally, some related work is described and compared to the current results and possible future work is identified.

2. Background

Background information on HLA and DDM is given in this section. In particular, the separation of the overall computation performed by the federates and the Run-Time Infrastructure to support DDM into four distinct processes, and the implementation of DDM using multicast, are explained. Brief reviews of computational complexity analysis and NP-completeness theory are also included in this section.  

2.1. High level architecture

HLA is an architecture standard for constructing distributed simulations. It facilitates interoperability among different simulation systems and types and promotes reuse of simulation software modules [2]. HLA is defined by three documents:  

---

2 The background explanations of computational complexity and NP-completeness are included here for the convenience of readers who may be unfamiliar with those topics.

3 Throughout this paper, we present HLA in terms of the IEEE 1516 Standard, which is somewhat different from the earlier US Department of Defense HLA 1.3 Standard [39].
1. The HLA Rules [3], which define interoperability and what capabilities a simulation must have to achieve it within HLA.

2. The Object Model Template [4], which is a format and method for specifying simulation data in terms of a hierarchy of object classes, and their attributes, and interactions between objects of those classes, and their parameters.

3. The Interface Specification [5], which is a precise specification of the interoperability-related services that a simulation may invoke, or be asked to provide, during a simulation execution.

Within HLA, a set of collaborating simulations is called a federation, each of the collaborating simulations is a federate, and a run of the distributed simulation is a federation execution. Federates that adhere to the Rules can exchange data defined according to the Object Model Template by invoking the services defined in the Interface Specification. An HLA Run-Time Infrastructure (RTI) is a software implementation of the Interface Specification. The RTI actually provides the services defined in the Interface Specification, including services to start and stop a federation execution, to send data between interoperating federates, to control the amount and routing of data that is passed, and to coordinate the passage of simulated time among the federates. The federates perform those functions by invoking the appropriate RTI services. The RTI may also invoke services that the federates must provide to it, such as receiving data; those services are likewise defined in the Interface Specification.

2.2. Data distribution management

One category of services defined in the Interface Specification, Data Distribution Management (DDM), is intended to reduce the amount of data delivered to a federate during a federation execution. To accomplish this, a multidimensional coordinate space with dimensions corresponding to user-selected simulation data items (simulation object attributes or other data items available in the simulation system) is provided by the RTI. During the federation execution federates specify what data values they expect to send (publish) and/or with to receive (subscribe). They do so by creating rectangular regions, called update and subscription regions, within the coordinate space. Geometrically, a region is a single rectangular or hyper-rectangular subspace within the coordinate space. Regions may be defined on any (non-empty) subset of the available dimensions of the coordinate space; we sometimes say that a region “has” the dimensions it is defined on. For each of a region’s dimensions, a range on that dimension is specified by its minimum and maximum coordinate values. Because regions’ ranges are defined as coordinates on axes rather than arbitrary points in the coordinate space, the regions are axis-parallel; that is, all region edges are parallel to an axis of the coordinate space and all region faces are parallel to the planes determined by pairs of axes of the coordinate space. Regions may be grouped into region sets, which consist of one or more regions. The regions in a region set need not all have the same subset of the dimensions of the coordinate space.
Regions can be used to specify the data a federate wants to send or receive by establishing a relationship between the dimensions of the coordinate space and data items that may be used to control the exchange of data between federates. Each dimension corresponds to such a variable. Update regions correspond to the limits of the values for the variable the publishing federate will produce, while subscription regions correspond to the limits of the values for the variable the subscribing federate wishes to receive.

Update region sets and subscription region sets may overlap within the coordinate space. Two region sets overlap if and only if they are opposite types (update and subscription) and any pair of their regions overlap. Regions overlap if and only if they have at least one common dimension and they have overlapping (i.e., intersecting) ranges on all common dimensions. It is possible for regions with different dimensions to overlap, as long as they have at least one dimension in common. If an update region set and a subscription region set overlap in the coordinate space then the RTI should deliver data from the publishing federate to the subscribing federate. The data to be delivered are data items that were explicitly associated with each update region set by the publishing federate; the data items associated with a region set are not necessarily those corresponding to the dimensions of the regions in the region set. The RTI should deliver updated values for the data items associated with the update region set from the publishing federate to the subscribing federate whenever the publishing federate provides them. The data flow relationship from the publishing federate to the subscribing federate is called a connection. Fig. 1 suggests the basic
ideas of DDM. In the figure, a notional federation has three federates; each has declared one region. The update region declared by federate A overlaps the subscription region declared by federate B, so updates to the data items associated with the update region are delivered by the HLA RTI from federate A to federate B. No data is delivered to federate C.

Using the DDM services, federates may create, modify, or delete update and subscription regions and region sets dynamically during a federation execution. Each time a region set is created, modified, or deleted, the intersections among region sets may change, and if so, the data delivery configuration used by the RTI must change to handle any changed connections. The overall DDM computation during a federation execution can be decomposed into four distinct processes [6]:

1. **Declaring.** The federates express the data they intend to send and/or desire to receive in terms of update and subscription region sets. They may create, change, or delete their region sets throughout a federation execution, as their data requirements change.

2. **Matching.** In response to each region set creation, change, or deletion in the declaring process, the RTI identifies the current connections by finding the overlapping pairs of update regions and subscription regions.

3. **Connecting.** The RTI establishes network data flow connectivity for the connections found during the matching process. The specific method for doing so depends on the network infrastructure.

4. **Routing.** Using the connectivity established during the connecting process, the RTI and the network infrastructure transport data from the publishing federate to the subscribing federate when the publishing federate sends it.

These “processes” are logical categories of processing and do not necessarily correspond to operating system processes. Over the course of a federation execution, individual declaring, matching, and connecting actions occur in that sequence in repeated cycles, with information being passed from declaring to matching to connecting. The routing process occurs asynchronously from the others, with each routing action using the network connectivity established by the most recent cycle of the other processes. Fig. 2 illustrates this.

### 2.3. DDM, interest management, and multicast

Internet Protocol multicast is a point-to-multipoint network routing protocol that allows a network node to send network packets to a group of receiving nodes identified by a group address [7,8]. Hosts may join or leave multicast groups dynamically. Multicast is commonly used to implement DDM in HLA and to route data in

---

4 Early analyses of DDM separated it into two processes or into five processes [34]. The decomposition into four processes used here was introduced later [6].
distributed simulation in general. We briefly review several applications of multicast for these purposes.

Multicast was used to support interest management in distributed simulation systems that utilized the Distributed Interactive Simulation (DIS) protocol. One of the predecessors to HLA, DIS had predefined entity types and message formats [9]. DIS interest management schemes were typically based on static partitions of the data to be sent. The simulated battlefield was partitioned into geographic grid cells and/or the range of data messages was partitioned into message types. Multicast groups were statically assigned to the grid cells or types. Data relevant to certain geographic grid cells or of certain message types was sent to the assigned multicast groups; simulations interested in receiving data for a certain grid or of a certain type joined the assigned groups. Variations of this approach have been used by a number of simulation systems, including Modular Semi-Automated Forces (ModSAF) [10,11], Close Combat Tactical Trainer (CCTT) [12], Synthetic Theater of War Engineering (STOW) Engineering Demonstration [13,14], and Naval Postgraduate School NET (NPSNET) [15–18].

Multicast has also been used to support interest management, and in particular DDM, in HLA implementations. In the STOW Advanced Concept Technology Demonstration, DDM was implemented in a prototype HLA RTI with update regions restricted to points and subscription regions mapped onto a static geographic grid partition of the battlefield [19,20]. Multicast groups were statically assigned to the grid cells before the federation execution. During a federation execution, the publishing federates sent the data associated with each update region to the multicast group assigned to the grid cell containing the update region. The subscribing federates had joined the multicast groups for the grid cells their subscription regions overlapped and so received the data. A variable density grid was used; smaller grid cells were defined in areas of the battlefield that were expected to experience a high volume of simulated entity interactions. This was possible because the demonstration
scenario was planned in advance. The number of multicast groups used was close to the capacity limits of the network equipment employed for the demonstration. A more developed version of the same RTI was used successfully in the Joint Semi-Automated Forces (JSAF) federation to support two large simulation experiments involving 20,000–100,000 battlefield entities, simulation by over 100 federates [21]. In that version of the RTI the DDM implementation was still based on the gridded approach. The grid layout and the assignment of grid cells to multicast groups were manually reconfigured and tuned for each exercise [22].

The gridded approach to implementing DDM is a heuristic that has the advantage of avoiding the computation associated with the matching and connecting processes as defined here. Because update and subscription regions are tested for intersection with a fixed grid, there is no need to find region intersections in the matching process. Because federates are assigned to multicast groups based on the grid, there is no need to group the connections in the connecting process. However, the gridded approach has disadvantages as well. It can result in unwanted data being delivered to a federate if update and subscription regions intersect the same grid cell but not each other. Also, the number of multicast groups needed for the grid cells and the number of group join and leave operations triggered as the regions are updated during a federation execution can approach or exceed the limits of current multicast hardware [23].

DDM implementations that rely on a priori knowledge of the communication patterns between simulations and static pre-assignment of multicast groups based on those patterns have achieved some good results, but they are ultimately limited in scale and flexibility because they do not account for changing connection patterns between senders and receivers. As larger, more complex, and less predictable simulation systems are implemented, there is a growing need for dynamic matching of update and subscription regions and more efficient use of multicast groups. RTI implementations that provide some of these capabilities are now available [24,25]. For this reason we are interested in the computational complexity of exact, non-heuristic implementations of the DDM matching and connecting processes as implied by the HLA definition [5]. That complexity is examined in Section 3.

2.4. Computational complexity

The time and space (memory) required by algorithms to solve computational problems are matters of interest. Those requirements, known as computational complexity, are often expressed as a function \( f(n) \) of the size of the input instance, which is conventionally denoted \( n \) (or sometimes \( N \), e.g., [26]). Exactly what is counted by \( n \) depends on the problem; e.g., for a sort algorithm, \( n \) is the number of items to be sorted. In the remainder of this background explanation and in the results to follow we focus on time requirements.

Upper bounds for algorithms and problems are stated using the notation \( O(f(n)) \), defined as the set of all functions \( g(n) \) bounded above by a positive constant multiple
C of \( f(n) \), provided that \( n \) is greater than some threshold.  

The upper bound for an algorithm is the largest amount of time that the algorithm could require on any instance of the problem. Usually the function \( f(n) \) counts, as a function of the input size \( n \), some primitive constant-time operation that forms the basis of the algorithm, with the unstated multiplicative constant \( C \) representing the time required for each of those operations. For example, in an algorithm that given a set of \( n \) line segments in the plane finds all intersecting pairs of segments by testing every possible pair for intersection, the basic operation is the segment-segment intersection test. The actual time required for each segment-segment intersection test depends on a number of things, including programming language and processor speed. However, it is fixed for any specific execution of the algorithm and does not depend on the input size, so it is abstracted away by the unstated multiplicative constant \( C \) in the order notation. Given \( n \) segments, the number of segment pairs to test is \( n(n-1)/2 \in O(n^2) \), so the algorithm’s upper bound time requirement is in \( O(n^2) \).

Note that an upper bound for an algorithm \( X \) is also an upper bound for the computational complexity of the problem \( U \) solved by that algorithm [27]. If a given algorithm \( X \) for problem \( U \) can solve \( U \) in time with an upper bound in \( O(f(n)) \), then any efficient algorithm \( Y \) to solve \( U \) should require at worst \( O(f(n)) \) time; if not, it would be faster to use \( X \) and \( Y \) is not efficient.  

Lower bounds for algorithms and problems are given using the notation \( \Omega(f(n)) \), defined as the set of all functions \( g(n) \) bounded below by a positive multiple \( C \) of \( f(n) \), provided that \( n \) is greater than some threshold. The lower bound for a problem \( U \) is related to, but not necessarily the same as, the lower bound for an algorithm for problem \( U \). The lower bound for a problem can be understood as the smallest of the upper bounds of all possible algorithms for the problem. If problem \( U \) has a lower bound in \( \Omega(f(n)) \), then for every possible algorithm \( X \) to solve \( U \) there is some instance of \( U \) for which \( X \) will require time in \( \Omega(f(n)) \). The lower bound for a problem is a measure of the inherent difficulty of the problem. For example, it is well known that the sorting problem has a lower bound in \( \Omega(n \log n) \) [26,28], in spite of the fact that for some sorting algorithms and some instances less time may be required, because for every sorting algorithm there is at least one instance that will require time in \( \Omega(n \log n) \).

In general, establishing the lower bound for a problem can be more difficult than finding an upper bound because a lower bound for a problem must be proven for all possible algorithms for the problem.

---

5 Note that as defined here \( O(f(n)) \) is a set (of functions), not a quantity, so it is technically imprecise to say “…the time required = \( O(n^2) \)”; better is “…the time required is in \( O(n^2) \)”. We will use the latter formulation. However, we will follow the widespread convention of using shorthand phrases like “an \( O(f(n)) \) algorithm” to mean “an algorithm that requires time in \( O(f(n)) \).”

6 Clearly, if efficiency is not required there is no upper bound for any problem.
Nevertheless, there is a method for proving lower bounds or problems known as reduction [27,29] or transformation [26]. Suppose problems $U$ and $V$ are related so that every instance of problem $U$ can be solved as follows:

1. Transform the instance of $U$ into an instance of $V$.
2. Solve the instance of $V$.
3. Transform the solution for the instance of $V$ into a solution to the original instance of $U$.

Once the transformations of steps 1 and 3 have been defined then problem $U$ is said to have been reduced to problem $V$. Let $\tau(n)$ denote an upper bound on the time required for the transformation steps. Then the reduction of $U$ into $V$ is written $U \preceq_{\tau(n)} V$.

Now suppose problem $U$ is known to have a lower bound in $\Omega(f(n))$. If the transformation time $\tau(n)$ grows strictly slower than $f(n)$, then $\Omega(f(n))$ is also a lower bound for problem $V$. The reduction transfers the lower bound from $U$ to $V$ because if an algorithm existed to solve an instance of $V$ faster than $\Omega(f(n))$, then instances of $U$ could be solved by transforming them into instances of $V$ and solving them using the faster algorithm for $V$, contradicting the known lower bound for $U$.

Reduction also transfers upper bounds by similar reasoning. Fig. 3 summarizes the reduction method.

2.5. NP-completeness

NP-completeness theory is concerned with the computational complexity of decision problems [30]. A decision problem is defined in two parts. The first part a formal

---

7 The term “transformation” is more descriptive of the method [26] but “reduction” continues to be widely used [27,29]. We will call the proof method “reduction” and use “transformation” to refer to the actual transformations of instances used in the lower bound and NP-completeness proofs.

8 Sources differ regarding the transformation time function $\tau(n)$; some require that it grow strictly slower than $f(n)$ [27], whereas others allow it to grow at the same rate as $f(n)$ [26]. Here we follow the more restrictive requirement.

9 This does not, however, prove that an algorithm that executes that quickly necessarily exists for problem $V$; it simply proves that no algorithm to solve $V$ can be faster than $\Omega(f(n))$. 
specification of the information (such as sets, graphs, matrices, or numbers) that is the subject of the problem. The specification is given in precise yet general terms, for example, calling for “a graph of \( n \) vertices” rather than some specific graph. An instance of a decision problem is a specific set of information that complies with the information specification. The second part of a decision problem is a question, which can be answered “yes” or “no” (hence “decision problem”), about the properties of an instance. A solution to a decision problem is with respect to a specific instance; the solution is “yes” if the instance satisfies the question, “no” if it does not. For decision problem \( U \), the set \( Y_U \) is the set of all instances of \( U \) for which the solution is “yes”. For an instance \( I \) of problem \( U \), \( I \in Y_U \) if and only if the solution to instance \( I \) is “yes”. 10

In computational complexity theory, problems are categorized based on their upper bound on time \( O(f(n)) \), where \( n \) is the size of the instance. Those problems where \( f(n) \) is a polynomial function on \( n \) (e.g., \( f(n) = n^2 \)) on a deterministic computer belong to set \( P \). Problems for which the time function \( f(n) \) of the best known algorithm is an exponential function on \( n \) (e.g., \( f(n) = 2^n \)) belong to set \( NP \). (Problems in \( NP \) can be solved in polynomial time on a non-deterministic computer.) Though it remains unproven, it is widely assumed that \( P \neq NP \). \( NP \)-complete problems are those problems in \( NP \) such that every instance of any \( NP \)-complete problem can be transformed into an equivalent instance of any other \( NP \)-complete problem by some process that requires \( O(f(n)) \) time, where \( f(n) \) is polynomial on \( n \).

Given a decision problem \( V \), \( V \) can be shown to be \( NP \)-complete using a two-step procedure. The first shows that the problem is in \( NP \). The second shows that it is at least as difficult as other \( NP \)-complete problems. The steps together imply that a problem is \( NP \)-complete.

1. Show that \( V \) is in \( NP \), by giving a polynomial time algorithm to check a solution for \( V \).
2. Transform a known \( NP \)-complete problem \( U \) to \( V \), as follows:
   (a) Define a transformation function \( h \) from an instance \( I \) of \( U \) to an instance \( h(I) \) of \( V \). 11
   (b) Show that \( h \) requires polynomial time in \( n \), the size of \( I \).
   (c) Show that \( I \in Y_U \) if and only if \( h(I) \in Y_V \).

Showing a problem to be \( NP \)-complete has practical value. Once a problem is proven to be \( NP \)-complete it is known to be as hard as all other \( NP \)-complete problems.

---

10 Problems of other types, such as subset selection, search, or optimization, can generally be shown to be no easier than their corresponding decision problems, so results about the difficulty of the decision problems apply to the other types as a lower bound. Surprisingly, problems of the seemingly more difficult types can also be shown in many cases to be no harder than their corresponding decision problems [26].

11 The transformation is more commonly known as \( f \), rather than \( h \) [30]. We use the latter to avoid name conflict with the time function in the conventional order notation \( O(f(n)) \).
The failure, to date, to find a polynomial algorithm for any NP-complete problem suggests that finding one for the given problem may be problematic.

3. Computational complexity of DDM matching

We first consider the computational complexity of the matching process. To do so, the matching process must be formally defined. Recall that during a federation execution federates can dynamically create, modify, and delete regions. These actions each invoke operations in the matching process. Each region creation action requires a determination if the new region overlaps any previously created regions of the opposite type, thereby requiring new data delivery connections, and the insertion of the new region into the region list data structure. Each region deletion action requires a retrieval of a list of the existing regions of opposite type that the deleted region overlaps, so that the connections produced by those overlaps can be broken, and a deletion of the deleted region from the region list data structure. Region modification actions are logically equivalent to a deletion followed by a creation. With this in mind we restate the matching process formally as follows.

**DDM Rectangle Matching (DRM)**

*Instance.* Dimension \(d \geq 1\); set \(R = \{r_1, r_2, \ldots, r_m\}\) of \(d\)-rectangles, where initially \(R = \emptyset\); and sequence \((a_1, a_2, \ldots, a_n)\) of operations. Each operation is a 2-tuple \(a_i = (o_i, q_i)\), where \(o_i \in \{\text{insert}, \text{delete}, \text{query}\}\) is the operation type and \(q_i\) is a \(d\)-rectangle, for \(1 \leq i \leq n\).

*Problem.* Process each operation \(a_i = (o_i, q_i)\) in sequence from \(a_1\) to \(a_n\), as follows. If \(o_i = \text{insert}\), set \(R = R \cup \{q_i\}\). If \(o_i = \text{delete}\), set \(R = R - \{q_i\}\). If \(o_i = \text{query}\), determine and report all \(d\)-rectangles \(r \in R\) that intersect the query \(d\)-rectangle \(q_i\), i.e., all \(r \in R \ni r \cap q_i \neq \emptyset\).

To establish computational complexity of this problem, and thus the matching process, we focus on the computational complexity of the query operation, which must be performed for each new region as it is created, and is sufficient to establish a lower bound for the matching process. The query operation checks for overlaps (geometric intersections) between the new region and the regions of opposite type previously created by the federates. Geometrically, those regions are axis parallel \(d\)-dimensional hyper-rectangles (\(d\)-rectangles, for short). The query operation is stated formally as follows.

**Rectangle Intersection Query (RIQ)**

*Instance.* Dimension \(d \geq 1\); set \(R = \{r_1, r_2, \ldots, r_m\}\) of \(d\)-rectangles; and query \(d\)-rectangle \(q\).

*Problem.* Determine and report all \(d\)-rectangles \(r \in R\) that intersect the query \(d\)-rectangle \(q_i\), i.e., all \(r \in R \ni r \cap q_i \neq \emptyset\).
A familiar problem, binary search, will be needed in the proof to follow and is stated here formally; binary search is known to have a lower bound $\Omega(\log m)$ [26].

Binary Search (BS)

*Instance.* Sequence $X = (x_1, x_2, \ldots, x_m)$ of $m$ distinct integers, given ordered $\exists x_i \leq x_{i+1}$ for $1 \leq i < m$; query integer $q$.

*Problem.* Determine the value of $i \ni 0 \leq i \leq m$ and $x_i < q < x_{i+1}$, assuming $x_0 = -\infty$ and $x_m+1 = \infty$.

With the definitions of BS and RIQ in hand, we can establish upper and lower bounds for DRM.

**Theorem 1.** The DDM matching process requires time with a lower bound in $\Omega(n \log n)$ and an upper bound in $O(n^2)$, where $n$ is the number of run-time data distribution actions performed by the federates.

**Proof.** Upper bound. DRM requires processing a sequence of $n$ insert, delete, or query operations. The query operation problem RIQ can be solved by the following naive algorithm: test the query $d$-rectangle $q$ for intersection with each of the $d$-rectangles $r \in R = \{ r_1, r_2, \ldots, r_m \}$. Each intersection test requires time in $O(d)$, by a simple $O(1)$ comparison of each of the respective $d$-rectangle sides for overlap, so the total time required for this naive algorithm is in $O(dm)$. However, because $d$ is a constant for any specific instance of RIQ, $O(dm) \subseteq O(m)$, i.e., an upper bound for RIQ \(\in O(m)\). If $R$ is to be kept in order, the *insert* and *delete* operations can be processed in $O(\log m)$ time using AVL trees [31], or $O(1)$ if not. Therefore an upper bound for DRM \(\in O(nm)\). But each region creation action by a federate can create at most a constant number of regions, so $O(m) \subseteq O(n)$, and therefore an upper bound for DRM in $O(n^2)$.

Lower bound. A lower bound for the query operation problem RIQ can be established by reduction from BS. An instance of BS can be solved using an algorithm for RIQ in this manner. The sequence $X = (x_1, x_2, \ldots, x_m)$ of $m$ integers to be searched in BS are treated in RIQ as $m$ 1-rectangles (line segments), where for $1 \leq i < m$, 1-rectangle $r_i = [x_i, x_{i+1}]$ and 1-rectangle $r_m = [x_m, \infty]$. The query integer $q$ is treated as a degenerate 1-rectangle $[q, q]$. Note that $X$ is not actually converted in its entirety to 1-rectangles, which would require $O(m)$ time and invalidate the transformation.

Rather, each integer $x_i$ in $X$ is treated as a 1-rectangle as it is accessed by the RIQ algorithm. The binary search is performed on the sequence $X$ as if it were a single 1-rectangle, with the query $q$ treated as a point inside the rectangle. The lower bound for binary search is $\Omega(\log m)$, which translates to $\Omega(\log n)$ since $m = n$ for the RIQ problem. Therefore, the lower bound for DRM is $\Omega(n \log n)$.

\[12\] The Binary Search problem and its lower bound $\Omega(\log m)$ are conventionally written with $n$ as the number of items to search, rather than $m$. We use $m$ for consistency with the corresponding value $m$ (number of rectangles) in the RIQ problem definition and to avoid confusion with the value $n$ (number of DDM actions) in the DRM problem. The relationship of $m$ and $n$ will be established in the proof of Theorem 1.
algorithm by retrieving it and its successor $x_{i+1}$, thereby adding only $O(1)$ time to each of the accesses. This time is part of the RIQ execution, not the transformation, so the transformation from BS to RIQ is null, and thus time $\tau(m) \in O(1)$, which is strictly less than $\Omega(\log m)$.

The solution to RIQ, i.e., those 1-rectangles from the sequence that intersect the query 1-rectangle, will consist of exactly zero, one, or two 1-rectangles. If the query integer $q$ in the instance of BS is $< x_1$, then query 1-rectangle $[q, q]$ in the instance of RIQ will intersect zero 1-rectangles, and the solution to BS is 0. If $q$ is between any two integers $x_i$ and $x_{i+1}$ with $i < m$, then query 1-rectangle $[q, q]$ will intersect one 1-rectangle $r_i = [x_i, x_{i+1}]$ and the solution to BS is $i$. If $q > x_m$, then query 1-rectangle $[q, q]$ will intersect one 1-rectangle $r_m = [x_m, \infty]$ and the solution to BS is $m$. If $q$ is the same as some integer $x_i$, then query 1-rectangle $[q, q]$ will intersect two 1-rectangles $r_{i-1} = [x_{i-1}, x_i]$ and $r_i = [x_i, x_{i+1}]$ and the solution to BS is $i$.

Transforming the solution to RIQ to a solution to BS takes $O(1)$ time, so $\tau(m) \in O(1)$. Therefore, the lower bound for BS transfers to RIQ, i.e., BS $\asymp O(1)$ RIQ and BS $\in \Omega(\log m) \Rightarrow$ RIQ $\in \Omega(\log m)$. As noted earlier, DRM requires processing a sequence of $n$ actions, each of which may be an instance of RIQ. Therefore, the lower bound for DRM $\in \Omega(n \log m)$ and because $O(m) \subset O(n)$ as noted earlier, DRM $\in \Omega(n \log n)$. □

Fig. 4 illustrates the reduction. It shows an example of a sequence of integers $X$ from an instance of BS transformed into an instance of RIQ as well as the results from three different query points $q$. In the figure, the sequence of integers to be searched $X = (x_1, x_2, x_3, x_4)$ and three query points $q_1, q_2, q_3$ are shown as points on a number line. Note that $x_4 = q_3$. Below the number line, the 1-rectangles corresponding to the sequence $X$ are shown; they are spread out in two dimensions in the figure for clarity, but the 1-rectangles are, of course, actually 1-dimensional.

The dashed vertical lines below each query point show the 1-rectangles that intersect that query point, and thus would be found by RIQ. For example, $q_3$ intersects both $r_3$ and $r_4$. The number of 1-rectangles found by RIQ for query $q_1$ is 0, for $q_2$ is 1, and for $q_3$ is 2. The solution to BS for query $q_1$ is 0, for $q_2$ is 2, and for $q_3$ is 4.
4. Computational complexity of DDM connecting

We now consider the computational complexity of the multicast grouping method for the connecting process. Note that we are addressing the computational complexity of a particular solution to the connecting process, not that of the connecting process itself. Using the conventions of NP-completeness theory [30], we define the decision problem Multicast Grouping [32]. Note that \( Z^+ \) denotes the set of positive non-zero integers.

**Multicast Grouping (MG)**

*Instance.* Set of connections \( C = \{c_1, c_2, \ldots, c_n\} \), connection weights \( w(c) \in Z^+ \) for every \( c \in C \), positive integer \( m \), and positive integer \( t \).

*Question.* Is there a partition of \( C \) into \( m \) disjoint subsets \( G_1, G_2, \ldots, G_m \) such that

\[
\sum_{c \in G_i} w(c) \leq t \quad \text{for} \quad 1 \leq i \leq m?
\]

The MG decision problem represents, in an abstract way, a problem inherent in the multicast grouping approach to establishing data delivery connections in DDM. Each connection in MG corresponds to a needed data delivery connection, i.e., a publishing federate and a subscribing federate whose DDM update and subscription regions overlap. If the RTI is performing multicast grouping, in the connecting process it must determine which multicast group to assign each connection to; normally there may be more connections than available multicast groups, so the connections must be grouped. In the MG problem, the weights \( w(c) \) represent the amount of data sent on each connection, including both data length and send frequency. The sum of the weights for all the connections in a group represents the amount of data sent via that group and thus received by all the federates that have joined the group. Integer \( m \) is the number of multicast groups available. Integer \( t \) is the maximum amount of data that can be sent using a multicast group to each group member before the maximum data input capacity will be exceeded for some federate. The problem requires that the total data received by each federate in the group be kept below \( t \).

We will prove that Multicast Grouping (MG) is NP-complete by a straightforward transformation from the problem 3-Partition (3P), i.e., we will show \( 3P \propto MG \). The problem 3P, which is known to be NP-complete (by transformation from 3-Dimensional Matching) [33], is defined as follows [30]:

**3-Partition (3P)**

*Instance.* Set \( A \) of \( 3m \) elements, a bound \( B \in Z^+ \), and size \( s(a) \in Z^+ \) for each \( a \in A \) such that \( B/4 < s(a) < B/2 \) and such that

\[
\sum_{a \in A} s(a) = mB.
\]

*Question.* Can \( A \) be partitioned into \( m \) disjoint sets \( A_1, A_2, \ldots, A_m \) such that for \( 1 \leq i \leq m, \sum_{a \in A_i} s(a) = B? \) Note that each \( A_i \) must contain exactly three elements from \( A \).
**Theorem 2.** $MG$ is NP-complete.

**Proof.** By transformation from 3P, i.e., $3P \propto MG$. First it must be shown that $MG$ is in NP. Given a partition of $C$ into disjoint subsets $G_1, G_2, \ldots, G_m$ check each $G_i$ for $1 \leq i \leq m$ to confirm that

$$\sum_{c \in G_i} w(c) \leq t \quad \text{for} \quad 1 \leq i \leq m.$$

To do so requires accessing and summing each connection $c \in G_1 \cup G_2 \cup \ldots \cup G_m$ exactly once, along with $m$ multiplications and comparisons. This process is clearly in $O(n)$, i.e., polynomial in $n$, so $MG$ is in NP.

We now define the transformation function $h$ from any instance $I$ of 3P to an instance $h(I)$ of MG. Given any instance $I$ of 3P as defined, for $h(I)$, $C = A (\Rightarrow n = 3m), w(c) = s(a_i)$ for $1 \leq i \leq 3m = n, m$ (of $h(I)$) = $m$ (of $I$), and $t = B$.

The transformation clearly requires $O(n)$ time, i.e., is polynomial in $n$.

It must now be shown that $I \in Y_{3P}$ if and only if $h(I) \in Y_{MG}$, i.e., for a given instance $I$ of 3P, $A$ can be partitioned into $m$ disjoint sets $A_1, A_2, \ldots, A_m$ such that for $1 \leq i \leq m$,

$$\sum_{a \in A_i} s(a) = B,$$

with exactly three elements in each set if and only if for the transformed instance $h(I)$ of MG, $C$ can be partitioned into disjoint subsets $G_1, G_2, \ldots, G_m$ such that

$$\sum_{c \in G_i} w(c) \leq t \quad \text{for} \quad 1 \leq i \leq m.$$

(Only if) Assume $I \in Y_{3P}$. Then $A$ has been partitioned into $m$ disjoint subsets $A_1, A_2, \ldots, A_m$ such that

$$\sum_{a \in A_i} s(a) = B \quad \text{for} \quad 1 \leq i \leq m.$$

Now it must be shown that $h(I) \in Y_{MG}$. Partition $C$ into disjoint subsets $G_1, G_2, \ldots, G_m$ such that $G_1 = A_1, G_2 = A_2, \ldots, G_m = A_m$. Recall that $w(c_i) = s(a_i)$ for $1 \leq i \leq 3m = n$. Then

$$\sum_{c \in G_i} w(c) = B = t \quad \text{for} \quad 1 \leq i \leq m,$$

thus meeting the condition for a valid partitioning of $C$. Therefore $I \in Y_{3P} \Rightarrow h(I) \in Y_{MG}$.

(If) Assume $h(I) \in Y_{MG}$. Then $C$ has been partitioned into disjoint subsets $G_1, G_2, \ldots, G_m$ such that

$$\sum_{c \in G_i} w(c) \leq t \quad \text{for} \quad 1 \leq i \leq m.$$

Now it must be shown that $I \in Y_{3P}$. Partition $A$ into disjoint subsets $A_1, A_2, \ldots, A_m$ such that $A_1 = G_1, A_2 = G_2, \ldots, A_m = G_m$ and consider the sizes of the subsets
We will show that $|G_i| = 3$ for $1 \leq i \leq m$. Recall that by $h(I)w(c_i) = s(a_i)$ and because $h(I)$ was transformed from an instance of 3P $B/4 < s(a_i) < B/2$ for $1 \leq i \leq 3m = n$. Suppose by way of contradiction that one of the subsets, say $G_j$, has more than three elements, i.e., $|G_j| \geq 4$. Then

$$\sum_{c \in G_j} w(c) > B$$

because by assumption $|G_j| \geq 4$ and each weight $w(c) > B/4$. But $B = t$, so subset $G_j$ total weight $> B$ contradicts the assumption that $h(I) \in Y_{MG}$.

Similarly, suppose by way of contradiction that one of the subsets, say $G_j$, has less than three elements, i.e., $|G_j| \leq 2$. Then

$$\sum_{c \in G_j} w(c) < B$$

because $|G_j| \leq 2$ by assumption and each weight $w(c) < B/2$ because $h(I)$ was transformed from an instance of 3P. But $B = t$, so subset $G_j$ total weight $< B$ contradicts the assumption that $h(I) \in Y_{MG}$. Therefore $|G_i| = 3$, and thus by transformation $h$, $|A_i| = 3$ for $1 \leq i \leq m$.

It remains to be shown that

$$\sum_{a \in A_i} s(a) = B \quad \text{for } 1 \leq i \leq m.$$ 

Recall that that for any instance $I$ of 3P (not just those instances in $Y_{3P}$)

$$\sum_{a \in A} s(a) = mB.$$ 

Therefore, because in $h(I)w(c_i) = s(a_i)$ for $1 \leq i \leq 3m = n$,

$$\sum_{c \in C} w(c) = mB.$$ 

Because $h(I) \in Y_{MG}$ we know that

$$\sum_{c \in G_j} w(c) \leq t \quad \text{for } 1 \leq i \leq m.$$ 

Assume by way of contradiction that one of the subsets, say $G_j$, has

$$\sum_{c \in G_j} w(c) < t = B.$$ 

But there are $m$ subsets $G_1, G_2, \ldots, G_m$ and the total weight of all the sets’ elements is $mB$. If $G_j$ has weight sum $< B$ as assumed, then one of the other sets, say $G_k$, must have weight sum $> B$ so that the total will be $mB$. This contradicts the assumption that $h(I) \in Y_{MG}$. Therefore

$$\sum_{c \in G_i} w(c) = B \quad \text{for } 1 \leq i \leq m,$$
and thus by transformation $h$

$$\sum_{a \in A_i} s(a) = B \text{ for } 1 \leq i \leq m.$$ 

Together with the fact that $|A_i| = 3$ for $1 \leq i \leq m$, this meets the condition for a valid partitioning of $A$. Therefore $h(I) \in Y_{MG} \Rightarrow I \in Y_{3P}$.

Thus $I \in Y_{3P}$ if and only if $h(I) \in Y_{MG}$, and therefore MG is NP-complete. \qed

5. Conclusions

The matching process requires total time with a lower bound in $\Omega(n \log n)$ and an upper bound in $O(n^2)$, where $n$ is the number of run-time data declarations made by the federates. This was shown by reduction from Binary Search. The commonly used multicast grouping approach to the connecting process is NP-complete; this was shown by transformation from 3-Partition. Approaches to the connecting process other than multicast grouping are possible, and Theorem 2 does not necessarily apply to those approaches.

In comparison to a number of empirical studies of DDM (see Section 6), these results contribute to a theoretical understanding of its computational complexity. From a practical standpoint, results such as these do not provide specific solutions to simulation developers charged with implementing DDM matching or connecting. However, a developer aware of theoretical results is prepared to recognize an optimal algorithm and avoid a futile search for one better than the theoretical bound [30].

6. Related and future work

The potential for DDM optimization via multicast grouping was positively demonstrated in work that experimentally evaluated two new algorithms to group connections [23]. The algorithms, which are essentially greedy algorithms with optimizing heuristics based on connection weight, seek to minimize total message latency and extraneous messages delivered while staying within a multicast group limit. They were tested in both off-line (static) and on-line (dynamic) modes using synthetic connection sets, some generated randomly and others based on a large military simulation scenario.

A number of studies have examined a range of algorithms and heuristics for matching and assessed their performance empirically, as opposed to theoretically. The gridded approach has been compared to algorithms based on multidimensional binary trees (e.g., quadtrees) [34], interval trees [35], and dynamic grids [36]. Another such study assessed a meta-algorithm that selects different matching algorithms depending on the distribution of regions in the coordinate space [37].

The computational complexity of the channelization problem, a problem similar to the Multicast Grouping problem of Section 4, has also been studied [38]. The channelization problem differs from Multicast Grouping in that in the former the mapping from publisher to subscriber is split into two separate mappings, from
publisher to multicast groups and then from multicast groups to subscribers, whereas in the latter there is a single mapping, from connections that include publisher and subscriber to multicast groups. The channelization problem is found in general to be NP-complete. A constrained form of the problem that includes only the second mapping phase is shown to be linear. There is potential for RTI implementations of DDM to be based on the constrained form of the problem.

The specification of the DDM services changed substantially from the US Department of Defense 1.3 version of the HLA standard to the Institute for Electrical and Electronic Engineers 1516 version of the standard [39]. Despite those differences, the two versions of DDM have been proven using geometric arguments to be equivalent in data connection representational power [40].

An earlier version of the proof of the lower bound of the matching process [41] has been improved in rigor and clarity here. An earlier version of the Multicast Grouping problem treated the total data sent to all members of a multicast group, rather than to one member as in this paper, as the quantity to be kept below a limit [42]. The form of the problem given here is arguably more intuitive and the proof of NP-completeness is simpler. Beyond the separate improvements to the theorems, the present work places them in a unified context of DDM computational complexity.

The multicast routing problem, similarly named but actually rather different from the multicast grouping problem, is to find routes in a network of nodes and connections for multicast communication. The problem, can solved by finding a Steiner tree in the network that spans the sending and receiving nodes as well as other methods, is also NP-complete [43]. Multicast routing when a multicast group can be established with multiple communications paths, each with a limited number of receivers, has also been studied [44].

Regarding the matching process, closing the gap between the lower bound of $\Omega(n \log n)$ and the upper bound of $O(n^2)$ by finding a matching algorithm in $O(n \log n)$ would be desirable. We conjecture that a general $d$-dimensional rectangle intersection approach based on decomposing orthogonal object intersection queries into four types of subproblems, each supported by a different nested tree data structure [45] may be relevant, but further work is needed to clearly establish or refute its applicability to the DDM matching problem, especially for $d > 2$.

Acknowledgements

This work was supported in part by the Defense Modeling and Simulation Office. Eric W. Weisel reviewed a preliminary version of the paper and provided useful comments. The anonymous referees provided useful recommendations for improvements to the paper.

References


