Probability in the Engineering and Informational Sciences, **30**, 2016, 413–430.
doi:10.1017/S0269964816000085

CARTESIAN PRODUCT PARTITIONING OF MULTI-DIMENSIONAL REACHABLE STATE SPACES

TUĞRUL DAYAR and M. CAN ORHAN
Department of Computer Engineering, Bilkent University, TR-06800 Bilkent, Ankara, Turkey
E-mail: tugrul@cs.bilkent.edu.tr; morhan@cs.bilkent.edu.tr

Markov chains (MCs) are widely used to model systems which evolve by visiting the states in their state spaces following the available transitions. When such systems are composed of interacting subsystems, they can be mapped to a multi-dimensional MC in which each subsystem normally corresponds to a different dimension. Usually the reachable state space of the multi-dimensional MC is a proper subset of its product state space, that is, Cartesian product of its subsystem state spaces. Compact storage of the matrix underlying such a MC and efficient implementation of analysis methods using Kronecker operations require the set of reachable states to be represented as a union of Cartesian products of subsets of subsystem state spaces. The problem of partitioning the reachable state space of a three or higher dimensional system with a minimum number of partitions into Cartesian products of subsets of subsystem state spaces is shown to be NP-complete. Two algorithms, one merge based the other refinement based, that yield possibly non-optimal partitionings are presented. Results of experiments on a set of problems from the literature and those that are randomly generated indicate that, although it may be more time and memory consuming, the refinement based algorithm almost always computes partitionings with a smaller number of partitions than the merge-based algorithm. The refinement based algorithm is insensitive to the order in which the states in the reachable state space are processed, and in many cases it computes partitionings that are optimal.

1. INTRODUCTION

Markov chains (MCs) are widely used to model systems which evolve by visiting the states in their state spaces following the available transitions. When such systems are composed of interacting subsystems, they can be modeled with various high-level formalisms. For analysis purposes, the high-level formalism is mapped to a multi-dimensional MC in which each subsystem normally corresponds to a different dimension. Then the MC is analyzed probabilistically for its steady-state or transient behavior [19] to improve the existing system or to devise a new system that meets certain requirements. The problem considered in this paper arises in this context of state based modeling.

We consider multi-dimensional MCs that are used to model systems composed of a finite number of interacting subsystems. The Cartesian product of the subsystem state spaces is called the product state space of the system. Due to semantic constraints, usually the system can only be in a proper subset of its product state space. This set is called the reachable state space of the system since its difference from the product state space consists of those unreachable states which the system never occupies. Compact storage of the matrix underlying the multi-dimensional MC incident on the reachable state space and efficient implementation of relevant analysis methods using Kronecker operations require the set of reachable states to be represented as a union of Cartesian products of subsets of subsystem state spaces [8]. We call this the problem of Cartesian product partitioning of multi-dimensional reachable state spaces. Currently, there are only ad hoc methods that can be used to this end, and to the best of our knowledge the study undertaken here is the first for state-based models.

A D-dimensional orthogonal polytope can be represented by a set of D-dimensional vectors [3]. In such a representation, the Cartesian product of sets of consecutive integers represents a hyper-rectangle. Hence, Cartesian product partitioning of a D-dimensional reachable state space is equivalent to the hyper-rectangular partitioning of the D-dimensional polytope that is represented by the reachable state space. Partitioning a two-dimensional orthogonal polytope into minimum number of hyper-rectangles is well studied and there are polynomial time algorithms for this problem [12,15,18]. However, the three-dimensional version of this problem is shown to be NP-complete, where NP stands for nondeterministic polynomial time [11]. In [14], an algorithm to partition three-dimensional orthogonal polytopes into hyper-rectangles is proposed. To the best of our knowledge, there is no algorithm to partition a D-dimensional orthogonal polytope into hyper-rectangles for $D > 3$.

Our motivation therefore is to automate the partitioning of a given multi-dimensional reachable state space into Cartesian products of subsets of subsystem state spaces. For practical purposes, the number of partitions in the partitioning should be kept as small as possible. With this objective in mind, we first show that the problem of partitioning the reachable state space of a three or higher dimensional system with a minimum number of partitions into Cartesian products of subsets of subsystem state spaces is NP-complete [13]. Then we present two algorithms that can be used to compute possibly non-optimal partitionings of the reachable state space into Cartesian products of subsets of subsystem state spaces.

We assume without loss of generality that the subsystem state spaces are defined on consecutive nonnegative integers starting from 0. Otherwise, it is always possible to enumerate the subsystem state spaces so that they satisfy this assumption. The first algorithm starts with partitions as singletons, each representing a reachable state. Two partitions are merged if their union is also a Cartesian product of sets of consecutive integers. The partitions are merged until there are no partitions that can be merged with each other. We call this the merge-based algorithm. The second algorithm takes a different approach. First, the unit distance graph of the reachable state space is constructed. The vertex set of this graph is the reachable state space and there is an edge between two vertices if the distance between them is one. Then this graph is refined [17] by removing edges until no further refinement is necessary. We call this the refinement-based algorithm.

Through a set of problems from the literature [2,4,6] and those that are randomly generated, the performance of the two algorithms is investigated. Results indicate that although it may be more time and memory consuming, the refinement-based algorithm almost always computes partitionings with a smaller number of partitions than the merge-based algorithm. In many cases, the partitionings computed by the refinement- based algorithm are the optimal ones. Furthermore, the refinement-based algorithm is insensitive to the order in which the states in the reachable state space are processed.

The next section introduces the notation and preliminary definitions used. The third section defines the partitioning problem formally, shows that it is NP-complete, and presents two algorithms that provide, possibly non-optimal, solutions to the problem. The fourth section reports on experimental results with the two algorithms. The last section concludes the paper.

2. NOTATION AND DEFINITIONS

Throughout the paper, calligraphic uppercase letters are used for sets. $|\cdot|$ and $\times$ respectively stand for the number of elements in a set and the Cartesian product operator. Vectors are represented with boldface lowercase letters and used to denote multi-dimensional states. For a D-dimensional vector $\mathbf{x}$, x_d denotes the dth element of the vector for $d = 1, \ldots, D$. The vector $\mathbf{e}_d$ denotes the dth column of the identity matrix, that is, the dth principal axis vector, and its length is determined by the context in which it is used. $O(\cdot)$ stands for the big O notation. The complexities of the algorithms are worst case complexities unless otherwise stated. $\mathbb{R}$, $\mathbb{Z}_{\geq 0}$, and $\mathbb{Z}_{>0}$ denote the sets of reals, nonnegative integers, and positive integers, respectively.

A D-dimensional hyper-rectangle is defined to be a Cartesian product of D intervals as in $\times_{d=1}^{D}[a_d, b_d]$, where $[a_d, b_d] \subseteq \mathbb{R}$ is an interval for $a_d < b_d$, $a_d, b_d \in \mathbb{R}$, and $d = 1, \ldots, D$. A point set $\mathcal{X} \subseteq \mathbb{R}^D$ is convex if the line segment between $\mathbf{x}$ and $\mathbf{y}$ is in $\mathcal{X}$ for $\mathbf{x}, \mathbf{y} \in \mathcal{X}$. The convex hull of $\mathcal{X}$ is the smallest set containing $\mathcal{X}$. A D-dimensional convex polytope is the convex hull of a finite set $\mathcal{X} \subseteq \mathbb{R}^D$ [20]. A D-dimensional polytope is the union of a finite number of D-dimensional convex polytopes. A D-dimensional polytope is orthogonal if it is a union of D-dimensional hyper-rectangles [3].

Let $G = (\mathcal{V}, \mathcal{E})$ be an undirected graph with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$. A graph $G' = (\mathcal{V}', \mathcal{E}')$ is a subgraph of $G = (\mathcal{V}, \mathcal{E})$ if $\mathcal{V}' \subseteq \mathcal{V}$ and $\mathcal{E}' \subseteq \{(\mathbf{x}, \mathbf{y}) \in \mathcal{E} \mid \mathbf{x}, \mathbf{y} \in \mathcal{V}'\}$. A subgraph of $G = (\mathcal{V}, \mathcal{E})$ induced by $\mathcal{V}' \subseteq \mathcal{V}$ is a graph whose vertex set is $\mathcal{V}'$ and edge set is $\{(\mathbf{x}, \mathbf{y}) \in \mathcal{E} \mid \mathbf{x}, \mathbf{y} \in \mathcal{V}'\}$. Two vertices of a graph are said to be connected if there exists a sequence of edges that lead from one of the vertices to the other. A subgraph of $G = (\mathcal{V}, \mathcal{E})$ forms a connected component if each pair of vertices in the subgraph are connected. A unit distance graph is a graph having a drawing in which all edges are of unit length [16].

3. CARTESIAN PRODUCT PARTITIONING ALGORITHMS

We consider multi-dimensional state based models with D interacting subsystems. The state space of the dth subsystem is denoted by $\mathcal{S}_d \subseteq \mathbb{Z}_{\geq 0}$ for $d = 1, \ldots, D$ and $\mathcal{S} = \times_{d=1}^{D} \mathcal{S}_d$ is said to be the product state space. In many cases, the reachable state space of the system, denoted by $\mathcal{R}$, is a proper subset of $\mathcal{S}$ [8] and can be partitioned in a myriad of ways. We are interested in partitionings, where the partitions themselves are Cartesian products of subsets of subsystem state spaces. Next we define the Cartesian product partitioning problem.

DEFINITION 3.1: The set $\{\mathcal{R}^{(1)}, \ldots, \mathcal{R}^{(K)}\}$ is said to be a Cartesian product partitioning of the multi-dimensional reachable state space $\mathcal{R}$ if $\mathcal{R}^{(k)} = \times_{d=1}^{D} \mathcal{R}_d^{(k)}$, the state space of the dth subsystem $\mathcal{R}_d^{(k)} \subseteq \mathcal{S}_d$ consists of consecutive integers, $\cup_{k=1}^{K} \mathcal{R}^{(k)} = \mathcal{R}$, and $\mathcal{R}^{(k)} \cap \mathcal{R}^{(l)} = \emptyset$ for $d = 1, \ldots, D$, $k \neq l$, $k, l = 1, \ldots, K$, and $K \in \mathbb{Z}_{>0}$.

For practical purposes that aid the use of Kronecker operations, the number of partitions in the Cartesian product partitionings of $\mathcal{R}$ should be as small as possible. Therefore, our interest lies in minimizing the number of partitions in the partitioning of $\mathcal{R}$. Unfortunately, the decision problem derived from the minimum Cartesian product partitioning problem is NP-complete [13] when the system has three or higher dimensions as we next show.

THEOREM 3.1: *It is NP-complete to decide whether there is a Cartesian product partitioning of the multi-dimensional reachable state space $\mathcal{R}$ with less than $K_{\mathcal{R}}$ partitions for given $D \in \mathbb{Z}_{>0}$, $K_{\mathcal{R}} \in \mathbb{Z}_{>0}$, and $\mathcal{R} \subseteq \mathbb{Z}_{\geq 0}^D$ when $D \geq 3$.*

PROOF: See [10]. ■

Now, we present two algorithms to compute Cartesian product partitionings of the multi-dimensional reachable state space $\mathcal{R}$. The first algorithm starts with partitions as singletons, each representing a reachable state. A partition is merged with another partition if their union is also a Cartesian product of sets of consecutive integers. This algorithm terminates when there are no partitions that can be merged with each other. The second algorithm takes a different approach. The unit distance graph of the reachable state space is constructed. Then this graph is refined by removing edges until the vertex set of each connected component can be expressed as a Cartesian product of sets of consecutive integers.

3.1. Merge-Based Partitioning

We start by defining set mergeability and then provide the condition for the mergeability of two partitions in a Cartesian product partitioning of $\mathcal{R}$.

DEFINITION 3.2: Let $\mathcal{X} = \times_{d=1}^D \mathcal{X}_d$ and $\mathcal{Y} = \times_{d=1}^D \mathcal{Y}_d$ be two partitions in a Cartesian product partitioning of $\mathcal{R}$. The partitions $\mathcal{X}$ and $\mathcal{Y}$ are said to be mergeable if $\mathcal{X} \cup \mathcal{Y} = \times_{d=1}^D (\mathcal{X}_d \cup \mathcal{Y}_d)$ and $\mathcal{X}_d \cup \mathcal{Y}_d$ consists of consecutive integers for $d = 1, \ldots, D$.

LEMMA 3.1: *Let $\mathcal{X} = \times_{d=1}^D \mathcal{X}_d$ and $\mathcal{Y} = \times_{d=1}^D \mathcal{Y}_d$ be two partitions in a Cartesian product partitioning of $\mathcal{R}$. The partitions $\mathcal{X}$ and $\mathcal{Y}$ are mergeable if and only if there exists some $i = 1, \ldots, D$ such that $\max(\mathcal{X}_i) + 1 = \min(\mathcal{Y}_i)$ or $\max(\mathcal{Y}_i) + 1 = \min(\mathcal{X}_i)$, and $\mathcal{X}_d = \mathcal{Y}_d$ for $d = 1, \ldots, i-1, i+1, \ldots, D$.*

PROOF: See [10]. ■

Now, we demonstrate the concept of mergeability on an example.

Example 3.1: Let $D = 3$, $\mathcal{S}_d = \{0, 1, 2\}$ for $d = 1, 2, 3$, and let $\mathcal{X} = \{0, 1\} \times \{1\} \times \{2\}$ be a partition in a Cartesian product partitioning of some $\mathcal{R} \subseteq \times_{d=1}^3 \mathcal{S}_d$. Then by Lemma 3.1, the partitions that can be merged with $\mathcal{X}$ are $\{2\} \times \{1\} \times \{2\}$, $\{0, 1\} \times \{0\} \times \{2\}$, $\{0, 1\} \times \{2\} \times \{2\}$, $\{0, 1\} \times \{1\} \times \{1\}$, and $\{0, 1\} \times \{1\} \times \{0, 1\}$.

Let $\mathcal{X} = \times_{d=1}^D \mathcal{X}_d$ be a partition in a Cartesian product partitioning of $\mathcal{R}$. The states $(\min(\mathcal{X}_1), \ldots, \min(\mathcal{X}_D)) \in \mathcal{X}$ and $(\max(\mathcal{X}_1), \ldots, \max(\mathcal{X}_D)) \in \mathcal{X}$ are said to be the end states of the partition $\mathcal{X}$. Due to Lemma 3.1, mergeability of two partitions can be determined by their end states. Besides, the states in a partition can be obtained from its end states. Therefore, it is sufficient to keep only the end states of the partitions instead of maintaining the partitions by using a disjoint-set data structure and a union-find algorithm [7] to merge two partitions.

In Algorithm 1, we provide the merge-based Cartesian product partitioning of $\mathcal{R}$. It starts by constructing a singleton for each state in $\mathcal{R}$. For each partition that is a singleton or is obtained by merging two partitions, a mergeable partition is sought. If such a partition is located, the two partitions are merged. There are $|\mathcal{R}|$ partitions; hence, the total time complexity of constructing a new partition is $O(D|\mathcal{R}|)$. A new partition is obtained when a

Algorithm 1 Merge-based algorithm to compute a Cartesian product partitioning of given multi-dimensional reachable state space

```
Input: $D$-dimensional reachable state space: $\mathcal{R}$
Output: Cartesian product partitioning of $\mathcal{R}$: $\mathcal{Q}$
 1: function MergeBasedPartitioning($\mathcal{R}$, $\mathcal{Q}$)
 2:     $\mathcal{Q} \leftarrow \emptyset$
 3:     for all $\mathbf{x} \in \mathcal{R}$ do
 4:         $\mathcal{X} \leftarrow \{\mathbf{x}\}$
 5:         while there exists some $\mathcal{Y} \in \mathcal{Q}$ mergeable with $\mathcal{X}$ do
 6:             $\mathcal{X} \leftarrow \mathcal{X} \cup \mathcal{Y}$; $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{\mathcal{Y}\}$
 7:         end while
 8:         $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\mathcal{X}\}$
 9:     end for
10: end function
```

singleton is constructed or two partitions are merged. Initially there are $|\mathcal{R}|$ partitions, and the partitions in the partitioning never get split, implying there can be at most $(|\mathcal{R}| - 1)$ merge operations. Then the total cost of merging two partitions is $O(D|\mathcal{R}|)$. Since each partition needs to check $O(D)$ end states for mergeability with another partition, a mergeable partition is sought $O(D|\mathcal{R}|)$ times.

The efficiency of the algorithm depends on the data structure used to keep the end states of the partitions. When a balanced tree such as an AVL tree [1] is used to keep the end states, the cost of searching for a partition becomes $O(\lg(L))$ time, where L is the maximum number of partitions during the execution of the algorithm. Therefore, the time complexity of the algorithm associated with seeking a mergeable partition is $O(D|\mathcal{R}|\lg(L))$ when the end states of the partitions are kept in a balanced tree. Another cost in the algorithm is inserting end states to the tree when no mergeable partition is found and removing the end states from the tree when two partitions are merged. The time complexities of these operations are also $O(|\mathcal{R}|\lg(L))$. Therefore, the time complexity of the algorithm is $O(D|\mathcal{R}|\lg(L))$. The space requirement for each partition is $O(D)$; hence, the space requirement of the algorithm is $O(DL)$. In the worst case, no partitions are merged and $L = |\mathcal{R}|$; hence, the time and space complexities of the algorithm are $O(D|\mathcal{R}|\lg(|\mathcal{R}|))$ and $O(D|\mathcal{R}|)$, respectively. If L is constant, the number of partitions is bounded by a constant during the execution of the algorithm. In that case, the time and space complexities of the algorithm become $O(D|\mathcal{R}|)$ and $O(D)$, respectively.

Observe that the partition computed by Algorithm 1 depends on the order in which the states of $\mathcal{R}$ are processed. Now, let us consider the following three-dimensional example.

Example 3.2: Let $\mathcal{S}_d = \{0, 1, 2, 3\}$ for $d = 1, 2, 3$,

$$\mathcal{R} = \{(0,0,1),(0,1,0),(0,1,1),(1,0,0),(1,0,1),(1,1,0),(1,1,1),(2,0,0),\\ (2,0,1),(2,1,0),(2,1,1),(3,0,1),(3,1,0)\},$$

and assume that the states in $\mathcal{R}$ are processed in lexicographical order.

When the first eight states are processed, the partitions are $\{(0,0,1),(0,1,1)\}$, $\{(1,1,1)\}$, $\{(0,1,0),(1,1,0)\}$, $\{(1,0,0),(1,0,1)\}$, and $\{(2,0,0)\}$. Then the singleton $\{(2,0,1)\}$ is merged with the singleton $\{(2,0,0)\}$, and their union is merged with $\{(1,0,0),(1,0,1)\}$. Finally, the remaining states are processed.

The number of partitions in the Cartesian product partitioning of $\mathcal{R}$ turns out to be 5, and the partitions are given by $\{(0,1,0),(1,1,0),(2,1,0),(3,1,0)\}$, $\{(1,1,1),(2,1,1)\}$, $\{(0,0,1),(0,1,1)\}$, $\{(1,0,0),(1,0,1),(2,0,0),(2,0,1)\}$, and $\{(3,0,1)\}$.

3.2. Refinement-Based Partitioning

The refinement based partitioning algorithm constructs the unit distance graph of the multi-dimensional reachable state space $\mathcal{R}$. The vertex set of this graph is $\mathcal{R}$ and there is an edge between two vertices $\mathbf{x},\mathbf{y} \in \mathcal{R}$ if $\mathbf{x}-\mathbf{y} \in \cup_{d=1}^{D}\{-\mathbf{e}_d,\mathbf{e}_d\}$. In other words, two vertices are adjacent in the unit distance graph if there is consecutiveness of the vertices along a particular dimension while the values of states variables in other dimensions remain constant. We next define conflicting edges in a subgraph of the unit distance graph.

Definition 3.3: Let $G=(\mathcal{R},\mathcal{E})$ be a subgraph of the unit distance graph of $\mathcal{R}$ and let $\mathbf{x},\mathbf{x}+\delta_i,\mathbf{x}+\delta_j \in \mathcal{R}$ for some $i,j=1,\ldots,D,\ i\neq j$, $\delta_i \in \{-\mathbf{e}_i,\mathbf{e}_i\}$, and $\delta_j \in \{-\mathbf{e}_j,\mathbf{e}_j\}$. Two edges $(\mathbf{x},\mathbf{x}+\delta_i)$, $(\mathbf{x},\mathbf{x}+\delta_j) \in \mathcal{E}$ are said to be conflicting if $\mathbf{x}+\delta_i+\delta_j \notin \mathcal{R}$ or $\{(\mathbf{x}+\delta_i,\mathbf{x}+\delta_i+\delta_j),(\mathbf{x}+\delta_j,\mathbf{x}+\delta_i+\delta_j)\} \nsubseteq \mathcal{E}$.

The next lemma and corollary show that in a subgraph of the unit distance graph of $\mathcal{R}$ with no conflicting edges, the vertices in each connected component can be written as a Cartesian product of sets of consecutive integers. Therefore, eliminating conflicting edges in the unit distance graph of $\mathcal{R}$ leads to a Cartesian product partitioning of $\mathcal{R}$.

Lemma 3.2: *Let $G=(\mathcal{R},\mathcal{E})$ be a subgraph of the unit distance graph of $\mathcal{R}$ with no conflicting edges, $M \in \mathbb{Z}_{>0}$, $\mathbf{x}^{(m)} \in \mathcal{R}$ for $m=0,\ldots,M$, and*

$$\mathcal{X}(M) = \times_{d=1}^{D}\left\{\min_{m=0}^{M}(x_d^{(m)}),\ldots,\max_{m=0}^{M}(x_d^{(m)})\right\}.$$

If $(\mathbf{x}^{(m-1)},\mathbf{x}^{(m)}) \in \mathcal{E}$ for $m=1,\ldots,M$, then $\mathcal{X}(M) \subseteq \mathcal{R}$ and $(\mathbf{u},\mathbf{v}) \in \mathcal{E}$ for $\mathbf{u}-\mathbf{v} \in \cup_{d=1}^{D}\{-\mathbf{e}_d,\mathbf{e}_d\}$ and $\mathbf{u},\mathbf{v} \in \mathcal{X}(M)$.

Proof: See [10]. ■

Corollary 3.1: *Let $G=(\mathcal{R},\mathcal{E})$ be a subgraph of the unit distance graph of $\mathcal{R}$ with no conflicting edges and $G'=(\mathcal{R}',\mathcal{E}')$ be a connected component in G. Then $\mathcal{R}'$ can be written as a Cartesian product of consecutive integers, that is, $\mathcal{R}' = \times_{d=1}^{D}\{u_d,\ldots,v_d\}$, where $\mathbf{u},\mathbf{v} \in \mathcal{R}$ and $u_d \le v_d$ for $d=1,\ldots,D$.*

The next definition introduces the concept of a separator, which is used in refining the unit distance graph of $\mathcal{R}$.

Definition 3.4: Let $\mathcal{U},\mathcal{V} \subseteq \mathcal{R}$ and $G=(\mathcal{R},\mathcal{E})$ be a subgraph of the unit distance graph of $\mathcal{R}$. The set of edges $\mathcal{Z} \subseteq \mathcal{E}$ is said to be a separator if

1. each edge in $\mathcal{Z}$ is incident to a vertex in $\mathcal{U}$ and a vertex in $\mathcal{V}$;
2. subgraphs of G induced by the sets $\mathcal{U}$ and $\mathcal{V}$ are each connected;
3. there does not exist two vertices $\mathbf{x},\mathbf{y} \in \mathcal{R}$ such that $(\mathbf{x},\mathbf{x}+h\mathbf{e}_d) \in \mathcal{Z}$, $(\mathbf{y},\mathbf{y}+h\mathbf{e}_d) \in \mathcal{E}\setminus\mathcal{Z}$, $\{(\mathbf{x},\mathbf{y}),(\mathbf{x}+h\mathbf{e}_d,\mathbf{y}+h\mathbf{e}_d)\} \subseteq \mathcal{E}$; and
4. there exists at least one edge in $\mathcal{Z}$ that conflicts with some edge in $\mathcal{E}$;

where $\mathcal{U} \subseteq \{\mathbf{x} \in \mathcal{R} \mid x_d = k\}$ and $\mathcal{V} \subseteq \{\mathbf{x} \in \mathcal{R} \mid x_d = k + h\}$ for some $k \in \mathcal{S}_d$, $h \in \{-1, 1\}$, and $d = 1, \ldots, D$.

The next lemma shows that removing the edges in a separator decreases the number of conflicting edges in a subgraph of the unit distance graph of $\mathcal{R}$.

LEMMA 3.3: *Let $G = (\mathcal{R}, \mathcal{E})$ be a subgraph of the unit distance graph of $\mathcal{R}$ and $\mathcal{Z}$ be a separator in G. Two edges in $G' = (\mathcal{R}, \mathcal{E} \setminus \mathcal{Z})$ conflict only if they also conflict in G.*

PROOF: See [10]. ■

Now, we are in a position to provide the refinement based algorithm for Cartesian product partitioning of $\mathcal{R}$ (see Algorithm 2). At the outset, the unit distance graph of $\mathcal{R}$ is constructed. Then the separators in this graph are constructed and inserted to a priority queue [7], where the priority of a separator is the total number of edges which conflict with an edge in the separator. The graph is refined by removing the separator with maximum priority until no conflicting edges remain. The edge set of the graph changes when a separator is removed; hence, the separators need to be reconstructed. By Lemma 3.3, an edge is conflicting in the refined graph only if it is also conflicting before refinement. Hence, each separator in the refined graph is a subset of a separator in the graph before refinement. Therefore, in order to reconstruct the separators, it is necessary and sufficient to visit the vertices incident to the edges in the separators intersecting the removed separator, where two separators are said to be intersecting if they include edges incident to the same vertex. If all conflicting edges are eliminated, then the vertices in each connected component can be written as a union of Cartesian products of sets of consecutive integers by Corollary 3.1.

In order to construct the unit distance graph, first all adjacent vertices need to be determined. In order to facilitate this, each vertex keeps an adjacency list of length $2D$ to identify the adjacent vertices. The vertices are first inserted to an AVL tree [1]. Then for each vertex, vertices that might be adjacent in the product state space are sought in the tree. There are $2D$ such vertices; hence, $2D$ vertices are sought for each vertex. Therefore, the space complexity of maintaining the graph becomes $O(D|\mathcal{R}|)$ and the time complexity of constructing the unit distance graph becomes $O(D|\mathcal{R}| \lg(|\mathcal{R}|))$. After the graph is constructed, the AVL tree is destroyed.

After the adjacency list of each vertex is set, separators need to be constructed. Each separator includes at least one conflicting edge, that is, its priority is positive. Algorithm 2 visits each edge and checks whether it conflicts with some other edge. A separator is formed by the two vertex sets ($\mathcal{U}$ and $\mathcal{V}$ in Definition 3.4) and the subgraphs induced by these two sets are connected. Besides, the separator is maximal; hence, the separator including a particular edge is unique and only one separator can include the edge (see Item 3 of Definition 3.4). Therefore, the separator including an edge is constructed only once in the unit distance graph. While constructing a separator $\mathcal{Z}$ including the edge $(\mathbf{x}, \mathbf{y})$, a breadth-first search starting at $\mathbf{x}$ is used to visit the vertices connected to $\mathbf{x}$ [7]. The time complexity of constructing the separator $\mathcal{Z}$ is $O(D|\mathcal{Z}|)$. Since each edge is added to at most one separator, the number of edges in the union of separators is $O(|\mathcal{E}|)$. Therefore, the total time complexity of constructing all separators in the unit distance graph becomes $O(D|\mathcal{E}|)$. At each vertex, we use an array of size $2D$ to keep the separators including the edge incident to that vertex. The space complexity of the algorithm remains as $O(D|\mathcal{R}|)$, but it takes constant time to retrieve the separator including a given edge.

Algorithm 2 Refinement based algorithm to compute a Cartesian product partitioning of given multi-dimensional reachable state space

Input: D-dimensional reachable state space: $\mathcal{R}$

Output: Cartesian product partitioning of $\mathcal{R}$: $\mathcal{Q}$

```
1: function RefinementBasedPartitioning($\mathcal{R}$, $\mathcal{Q}$)
2:     Construct unit distance graph of $\mathcal{R}$, $G = (\mathcal{R}, \mathcal{E})$
3:     Construct an empty priority queue $\mathcal{P}Q$
4:     for all $\mathbf{x} \in \mathcal{R}$ do
5:         for all $d = 1, \dots, D$ do
6:             if $\mathbf{x} + \mathbf{e}_d \in \mathcal{R}$ then
7:                 if $(\mathbf{x}, \mathbf{x} + \mathbf{e}_d)$ conflicts with some edge in $\mathcal{E}$ AND $(\mathbf{x}, \mathbf{x} + \mathbf{e}_d)$ is not in a
   separator then
8:                     Construct separator $\mathcal{Z}$ including $(\mathbf{x}, \mathbf{x} + \mathbf{e}_d)$; Insert $\mathcal{Z}$ to $\mathcal{P}Q$
9:                 end if
10:            end if
11:        end for
12:    end for
13:    while $\mathcal{P}Q$ is not empty do
14:        $\mathcal{Z}_{\max} \leftarrow$ separator $\mathcal{Z}$ in $\mathcal{P}Q$ whose priority is maximum
15:        Remove $\mathcal{Z}_{\max}$ from $\mathcal{P}Q$
16:        for all $(\mathbf{x}, \mathbf{y}) \in \mathcal{Z}_{\max}$ do
17:            $\mathcal{L} \leftarrow \emptyset$
18:            for all separators $\mathcal{Z}$ in $\mathcal{P}Q$ including an edge incident to $\mathbf{x}$ or $\mathbf{y}$ do
19:                $\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathcal{Z}\}$; Remove $\mathcal{Z}$ from $\mathcal{P}Q$
20:            end for
21:        end for
22:        $\mathcal{E} \leftarrow \mathcal{E} \setminus \mathcal{Z}_{\max}$
23:        for all $\mathcal{Z} \in \mathcal{L}$ do
24:            while $\mathcal{Z} \neq \emptyset$ do
25:                Construct the separator $\mathcal{Z}'$ including some edge $(\mathbf{x}, \mathbf{y}) \in \mathcal{Z}$; $\mathcal{Z} \leftarrow \mathcal{Z} \setminus \mathcal{Z}'$
26:                if priority of $\mathcal{Z}'$ is positive then
27:                    Insert $\mathcal{Z}'$ to $\mathcal{P}Q$
28:                end if
29:            end while
30:        end for
31:    end while
32:    $\mathcal{Q} \leftarrow \emptyset$
33:    for all connected components $G' = (\mathcal{R}', \mathcal{E}')$ of $G = (\mathcal{R}, \mathcal{E})$ do
34:        $\mathcal{Q} \leftarrow Q \cup \{\mathcal{R}'\}$
35:    end for
36: end function
```

When a separator is constructed, it is added to the priority queue. We use a binary heap as the priority queue. Observe that the number of separators does not exceed $|\mathcal{E}|$ since each separator includes at least one edge. Therefore, the cost of inserting separators to the priority queue is $O(|\mathcal{E}| \lg(|\mathcal{E}|))$. We consider an array implementation of the priority queue and the maximum number of separators is not known in advance; therefore, we allocate $O(|\mathcal{E}|) = O(D|\mathcal{R}|)$ space for the priority queue.

After the separators in the unit distance graph are inserted to the priority queue, the graph is refined by removing the edges in the separator with maximum priority until no separators remain in the priority queue. The graph is refined at most $|\mathcal{E}|$ times since at least one edge is removed from the graph at each refinement step. At each refinement step, the separator with maximum priority, $\mathcal{Z}_{\max}$, is chosen. When $\mathcal{Z}_{\max}$ is removed, the intersecting separators need to be reconstructed. Hence, at each refinement step, all edges of $\mathcal{Z}_{\max}$ are visited to obtain the intersecting separators. These separators are inserted to a list $\mathcal{L}$ and removed from the heap. Then these separators are reconstructed for the edge set $\mathcal{E} \setminus \mathcal{Z}_{\max}$ and inserted to the heap. Reconstruction of a separator $\mathcal{Z}$ requires visiting all the vertices incident to the edges in it. The number of edges in the union of separators intersecting with $\mathcal{Z}_{\max}$ is $O(D|\mathcal{E}|)$; hence, reconstruction of separators at each step has time complexity $O(D|\mathcal{E}|)$. Therefore, the total time complexity of reconstructing separators becomes $O(D|\mathcal{E}|^2)$ since reconstruction is required in all refinement steps. Now, consider the costs related to the heap operations. At each refinement step, the separators intersecting with $\mathcal{Z}_{\max}$ are removed from the heap. The separator $\mathcal{Z}_{\max}$ intersects with $O(D|\mathcal{Z}_{\max}|)$ separators. Therefore, during the execution of the algorithm, the total number of removals from the heap is $O(D|\mathcal{E}|)$ since the total size of the separators removed from the graph is $O(|\mathcal{E}|)$. The algorithm starts and ends with an empty heap, so the number of insertions to the heap is also $O(D|\mathcal{E}|)$. Hence, the total complexity of heap operations is $O(D|\mathcal{E}| \lg(D|\mathcal{E}|))$. When all costs are considered, the time and space complexities of Algorithm 2 are $O(D|\mathcal{E}|^2) = O(D^3|\mathcal{R}|^2)$ and $O(D|\mathcal{R}|)$, respectively.

Example 3.2: (cont'd) Algorithm 2 first constructs the unit distance graph of $\mathcal{R}$ and then refines the graph by removing edges. Let $G^{(k)} = (\mathcal{R}, \mathcal{E}^{(k)})$ denote the subgraph of the unit distance graph of $\mathcal{R}$, where $\mathcal{E}^{(k)}$ is the edge set after k separators are removed. In the unit distance graph of $\mathcal{R}$, $G^{(0)}$, there are seven pairs of conflicting edges (see Figure 1).

The separators in the unit distance graph of $\mathcal{R}$ (see Figure 2(a)) are

$$\begin{aligned}
\mathcal{Z}^{(0,1)} &= \{((0,0,1),(0,1,1)),((1,0,0),(1,1,0)),((1,0,1),(1,1,1)),\\
&\qquad ((2,0,0),(2,1,0)),((2,0,1),(2,1,1))\}\\
\mathcal{Z}^{(0,2)} &= \{((0,1,0),(0,1,1)),((1,0,0),(1,0,1)),((1,1,0),(1,1,1)),\\
&\qquad ((2,0,0),(2,0,1)),((2,1,0),(2,1,1))\}\\
\mathcal{Z}^{(0,3)} &= \{((0,0,1),(1,0,1)),((0,1,0),(1,1,0)),((0,1,1),(1,1,1))\}\\
\mathcal{Z}^{(0,4)} &= \{((2,0,1),(3,0,1))\}, \quad \mathcal{Z}^{(0,5)} = \{((2,1,0),(3,1,0))\}.
\end{aligned}$$

The priorities of the separators $\mathcal{Z}^{(0,1)}$ and $\mathcal{Z}^{(0,2)}$ are 4, and the priorities of the other separators are 2. There are two separators with maximum priority and one of them needs to be chosen. Let the edges in $\mathcal{Z}^{(0,1)}$ be chosen for removal. Then all other separators are reconstructed and three conflicting edges remain in $G^{(1)}$ (see Figure 2(b)). The separators in this graph are

$$\begin{aligned}
\mathcal{Z}^{(1,1)} &= \{((1,0,0),(1,0,1)),((2,0,0),(2,0,1))\},\\
\mathcal{Z}^{(1,2)} &= \{((0,1,0),(0,1,1)),((1,1,0),(1,1,1)),((2,1,0),(2,1,1))\},\\
\mathcal{Z}^{(1,3)} &= \{((0,0,1),(1,0,1))\}, \quad \mathcal{Z}^{(1,4)} = \mathcal{Z}^{(0,4)}, \quad \mathcal{Z}^{(1,5)} = \mathcal{Z}^{(0,5)}.
\end{aligned}$$

The priority of separator $\mathcal{Z}^{(1,1)}$ is 2, and the priorities of the other separators are 1. Then the edges in $\mathcal{Z}^{(1,1)}$ are removed from the edge set $\mathcal{E}^{(1)}$. After these edges are removed,

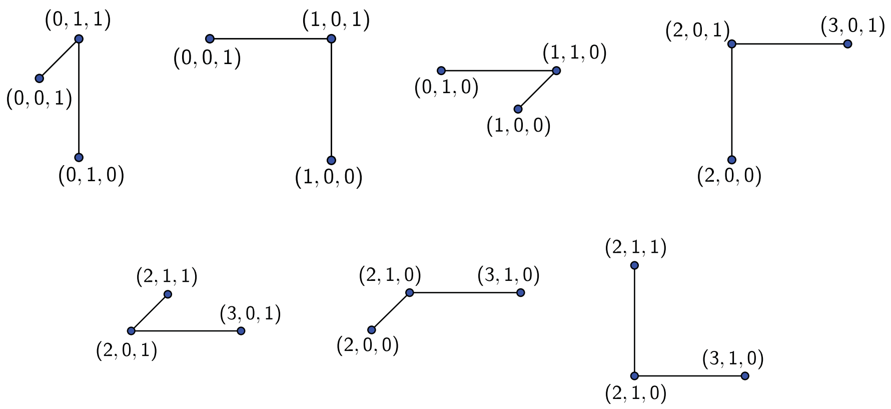

FIGURE 1. Conflicting edges in the unit distance graph of $\mathcal{R}$ in Example 3.2.

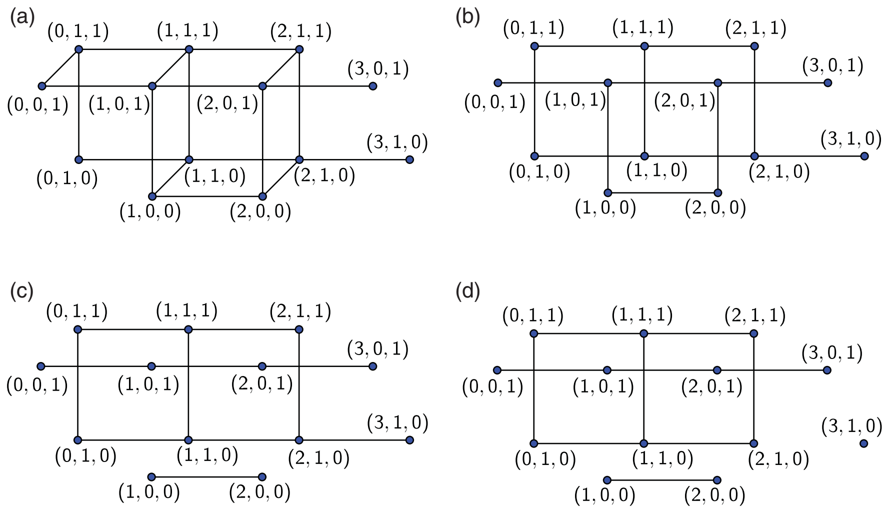

FIGURE 2. Subgraphs of unit distance graph of $\mathcal{R}$ in Example 3.2 for Algorithm 2 (a) $G^{(0)} = (\mathcal{R}, \mathcal{E}^{(0)})$, (b) $G^{(1)} = (\mathcal{R}, \mathcal{E}^{(1)})$, (c) $G^{(2)} = (\mathcal{R}, \mathcal{E}^{(2)})$ and (d) $G^{(3)} = (\mathcal{R}, \mathcal{E}^{(3)})$.

$\mathcal{Z}^{(1,3)}$ and $\mathcal{Z}^{(1,4)}$ are reconstructed and one conflicting edge remains in $G^{(2)}$ (see Figure 2(c)). The separators in this graph are $\mathcal{Z}^{(2,1)} = \mathcal{Z}^{(1,2)}$ and $\mathcal{Z}^{(2,2)} = \mathcal{Z}^{(1,5)}$. The priorities of both separators are 1, and either of them can be chosen for refinement. If $\mathcal{Z}^{(2,2)}$ is chosen, $\mathcal{Z}^{(2,1)}$ is reconstructed and no conflicting edges remain in $G^{(3)}$ (see Figure 2(d)). By Corollary 3.1, the vertices in each connected component of $G^{(3)}$ are the elements of the same partition of the Cartesian product partitioning. Therefore, the number of partitions in the Cartesian product partitioning of $\mathcal{R}$ is 4 and the partitions are given by $\{(0,0,1),(1,0,1),(2,0,1),(3,0,1)\}$, $\{(1,0,0),(2,0,0)\}$, $\{(3,1,0)\}$, and $\{(0,1,0),(0,1,1),(1,1,0),(1,1,1),(2,1,0),(2,1,1)\}$.

4. EXPERIMENTAL RESULTS

We implemented the proposed partitioning algorithms in C; the code can be obtained from [9]. All experiments are performed on a PC with an Intel Core2 Duo 2.4 GHz processor and

TABLE 1. Properties of models from literature

Model	D	$\|\mathcal{S}\|$	$\|\mathcal{R}\|$	*MinPrt*	*Sep*	*SepInt*
N–model	5	2,857,680	50,982	35	47	168
V–model (2)	3	2,695,784	103,709	2	3	3
V–model (3)	4	634,114	24,414	2	4	6
V–model (4)	5	260,000	10,025	2	5	10
W–model	5	1,377,810	20,142	4	5	10
courier_large	4	56,265,300	1,632,600	9	8	4
courier_med	4	8,593,200	419,400	7	6	3
kanban_fail	4	19,131,876	2,302,911	8	0	0
kanban_large	4	1,742,400	1,742,400	1	0	0
msmq_large	7	1,280,000,000	1,311,744	28	0	0

4 Gigabytes (GB) of main memory. All times are reported as seconds of CPU time. We monitor the memory allocation of the algorithms with the `pidstat` command of the `sysstat` package under Linux and all memory allocation results are reported as Megabytes (MB). We consider two groups of test problems. The first group consists of multi-dimensional reachable state spaces of Markovian models from the literature. The second group consists of a class of randomly generated multi-dimensional reachable state spaces with known minimum Cartesian product partitioning. We use the second group to find out how good the partitioning algorithms perform in terms of number of partitions with respect to the optimal solution.

4.1. Test Problems from Literature

We consider state spaces of call center models with different control policies (*N–model*, *V–model*, *W–model*) [2], a parallel communication software model (*courier_large*, *courier_med*) [5], a manufacturing systems model with Kanban control (*kanban_fail*, *kanban_large*) [4], and a queueing model with multiple servers and queues (*msmq_large*) [6]. The *V–model* has different variations with 2, 3, and 4 types of customers that we consider. "*V–model (2)*", "*V–model (3)*" and "*V–model (4)*" in the results refer to the *V–model* with 2, 3, and 4 customer types [2].

Table 1 gives the properties of the first group of test problems. The first four columns give the model name, the dimension of the state space, the size of the product state space, and the size of the reachable state space. The fifth column gives the minimum number of partitions in Cartesian product partitioning. The states of each connected component in the graph, that is obtained by removing the separators from the unit distance graph of $\mathcal{R}$, are in same partition in the minimum Cartesian product partitioning of $\mathcal{R}$ as shown in the next lemma.

LEMMA 4.1: *Let $G = (\mathcal{R}, \mathcal{E})$ be the unit distance graph of $\mathcal{R}$, $\mathcal{Z}$ be the union of separators in G, $N \in \mathbb{Z}_{>0}$ be the size of the minimum Cartesian product partitioning of $\mathcal{R}$, $K \in \mathbb{Z}_{>0}$ be the number of connected components in $G' = (\mathcal{R}, \mathcal{E} \setminus \mathcal{Z})$, and $G^{(k)} = (\mathcal{R}^{(k)}, \mathcal{E}^{(k)})$ be a connected component in G' for $k = 1, \ldots, K$. Then $\{\mathcal{P}^{(1)}, \ldots, \mathcal{P}^{(N)}\}$ is the minimum Cartesian product partitioning of $\mathcal{R}$, where $\mathcal{P}^{(n)} = \cup_{k \in \mathcal{K}(n)} \mathcal{R}^{(k)}$ for some $\mathcal{K}(n) \subseteq \{1, \ldots, K\}$ and $n = 1, \ldots, N$.*

PROOF: See [10]. ■

TABLE 2. Experimental results for models from literature when states are processed in lexicographical order

	Merge			Refinement		
Model	*Prt*	Time	Memory	*Prt*	Time	Memory
N–model	35	0.3	0.5	43	0.3	0.7
V–model (2)	3	0.2	0.5	2	0.4	15.5
V–model (3)	4	0.1	0.5	2	0.1	0.7
V–model (4)	5	0.0	0.5	2	0.0	0.7
W–model	5	0.1	0.5	4	0.1	0.7
courier_large	9	6.0	0.5	9	10.2	301.0
courier_med	7	1.5	0.5	7	2.4	77.9
kanban_fail	8	8.2	0.5	8	14.4	424.3
kanban_large	1	5.7	0.5	1	10.9	321.2
msmq_large	28	9.0	0.5	28	14.0	336.4

Hence, each partition in the minimum Cartesian product partitioning of $\mathcal{R}$ is a union of partitions obtained by removing the separators from the unit distance graph of $\mathcal{R}$. We compute all possible Cartesian product partitionings satisfying this condition in order to obtain the number of partitions in the minimum Cartesian product partitioning. This is relatively simple when there are small number of separators and intersecting separator pairs. However, the number of Cartesian product partitionings to compute increase exponentially with the number of separators and intersecting separator pairs. We also report two important characteristics of the models. The sixth and seventh columns give the number of separators and the number of separator pairs that have at least one edge incident to the same vertex in the unit distance graph, respectively. The reachability of a multi-dimensional state depends on the interaction of the subsystems. Hence, the number of separators and intersecting separator pairs depends on the model and the reachability conditions. There are many different reachability conditions, such as the sum of different subsystem state values might be bounded or a subsystem cannot be in a state depending on other subsystem states. In the *N–model*, the sum of different subsystem state values is bounded implying more separators and intersecting separator pairs. There is no such interaction in the other models we consider. *V–model*, *W–model*, *courier_large*, and *courier_med* are simpler than the *N–model*, and have relatively small numbers of separators. There are no separators in *kanban_fail*, *kanban_large*, and *msmq_large*, that is, the unit distance graphs of these models do not include any conflicting edges. Therefore, these models can be considered as relatively simple.

In Table 2, we present the results of experiments for the models from literature when the states are processed in lexicographical order. If the states are not given in this order, then it may be easily obtained by sorting. The second and fifth columns give the number of partitions in the Cartesian product partitionings computed by merge and refinement based partitioning algorithms, respectively. When the states are processed in lexicographical order, both algorithms compute the optimal solution for the *courier_large*, *courier_med*, *kanban_fail*, *kanban_large*, and *msmq_large* models. The merge-based algorithm computes the optimal solution of only the *N–model* among the call center models. The refinement based algorithm computes the optimal solution for all call center models except the relatively complicated *N–model*. It is reasonable to use merge-based algorithm for relatively complicated problems. For all the models, the merge-based algorithm requires less time and memory than the refinement based algorithm.

TABLE 3. Experimental results for models from literature when states are processed in random order

	Merge			Refinement		
Model	*Prt*	Time	Memory	*Prt*	Time	Memory
N–model	(6,098; 46)	(0.4; 0.0)	(1.8; 0.0)	(43; 0)	(0.4; 0.0)	(0.7; 0.0)
V–model (2)	(12,500; 78)	(0.6; 0.0)	(2.9; 0.0)	(2; 0)	(0.5; 0.0)	(15.5; 0.0)
V–model (3)	(5,792; 33)	(0.1; 0.0)	(1.3; 0.0)	(2; 0)	(0.1; 0.0)	(0.7; 0.0)
V–model (4)	(2,825; 20)	(0.1; 0.0)	(0.8; 0.0)	(2; 0)	(0.1; 0.0)	(0.7; 0.0)
W–model	(4,715; 35)	(0.1; 0.0)	(1.0; 0.0)	(4; 0)	(0.1; 0.0)	(0.7; 0.0)
courier_large	(444,818; 254)	(23.4; 0.1)	(50.8; 0.0)	(9; 0)	(20.8; 0.0)	(301.0; 0.0)
courier_med	(114,648; 124)	(4.5; 0.0)	(13.5; 0.0)	(7; 0)	(4.3; 0.0)	(77.8; 0.0)
kanban_fail	(626,586; 284)	(33.3; 0.1)	(71.3; 0.0)	(8; 0)	(28.6; 0.0)	(424.3; 0.0)
kanban_large	(483,245; 238)	(25.1; 0.1)	(54.6; 0.1)	(1; 0)	(21.9; 0.0)	(321.2; 0.0)
msmq_large	(432,018; 154)	(25.1; 0.0)	(55.8; 0.1)	(28; 0)	(25.8; 0.0)	(336.4; 0.0)

In Table 3, we present the results of experiments for the models from literature when the states are processed in random order. For both algorithms, we compute the Cartesian product partitioning by processing the states in 51 random orderings so as to provide mean values. The second and fifth columns of Table 3 give the mean number of partitions together with the confidence interval both rounded to the nearest integer for a confidence probability of 95% of the partitionings obtained by merge and refinement based algorithms, respectively. In "Time" and "Memory" columns, the mean values and the confidence intervals for a confidence probability of 95% of the experiments are reported. In all columns, the first value in parentheses is the mean and the other value is the confidence interval.

The partitioning size and the memory requirement remain the same for the refinement based algorithm, but the time requirement increases when the states are processed in random order. In all of the models, the number of separators is small, so the increase in time is not due to the refinement of the graph and updating of separators, but due to the graph construction. These results suggest that cache is more efficiently used while accessing the states when the states are ordered lexicographically since each state becomes a child of the last accessed state in the AVL tree. Hence, insertion to the tree becomes much more efficient for the lexicographic ordering of the states. The order of states is much more important for the merge-based algorithm. In this algorithm, two partitions are merged along dimension d if the subsets of their subsystems are the same in all dimensions except d. When the states are ordered lexicographically, a partition is merged with all possible partitions along a dimension before it is merged with a partition in another dimension. When the states are ordered randomly, partitions are merged along random dimensions depending on the order of the states. Hence, all possible partitions along a dimension are not considered. Therefore, the number of partitions for random ordering of the states increases by a factor between 174 (*N–model*) and 483,000 (*kanban_large*) on average with respect to the number of partitions for the lexicographical ordering. The time and memory requirements also increase substantially which agrees with the complexity analysis since the time and memory complexities of the merge-based algorithm depend on the number of partitions (see Section 3.1). The time taken by the merge-based algorithm becomes close to the time taken by the refinement based algorithm. However, the memory requirement of the merge-based algorithm is still better than the memory requirement of the refinement based algorithm.

Algorithm 3 Algorithm that generates a random multi-dimensional state space with known minimum Cartesian product partitioning

```
Input: Dimension of reachable state space: $D$
       State space size of each subsystem: $M$
       Probability of a state to be reachable in initial two-dimensional state space: $p$
Output: $D$-dimensional reachable state space: $\mathcal{R}$
        Size of minimum Cartesian product partitioning of $\mathcal{R}$: $K$
1: function GenerateRandomTestProblem($D, M, p, \mathcal{R}, K$)
2:     $\mathcal{R}' \leftarrow \emptyset$
3:     for all $(i, j) \in \{0, \ldots, M-1\} \times \{0, \ldots, M-1\}$ do
4:         $\mathcal{R}' \leftarrow \mathcal{R}' \cup \{(i, j)\}$ with probability $p$
5:     end for
6:     Compute minimum Cartesian product partitioning of $\mathcal{R}'$, $\{\mathcal{R}^{(1)}, \ldots, \mathcal{R}^{(K)}\}$ by
       using a modified version of minimum rectangular partitioning algorithm [12]
7:     for all $d = 3, \ldots, D$ do
8:         $m \leftarrow$ Unif(0, $M-1$)
9:         for all $k = 1, \ldots, K$ do
10:            $a \leftarrow$ Unif(0, $m$); $b \leftarrow$ Unif($m$, $M-1$)
11:            $\mathcal{R}^{(k)} \leftarrow \mathcal{R}^{(k)} \times \{a, \ldots, b\}$
12:        end for
13:    end for
14:    for all $k = 1, \ldots, K$ do
15:        $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}^{(k)}$
16:    end for
17:    Rearrange the dimensions of the states in $\mathcal{R}$ with a new random ordering
18: end function
```

4.2. A Class of Random Test Problems

Cartesian product partitioning of a two-dimensional reachable state space is equivalent to the hyper-rectangular partitioning of the two-dimensional polytope that is represented by the reachable state space. Hence, minimum Cartesian product partitioning of a two-dimensional reachable state space can be obtained by using a modified version of the minimum rectangular partitioning algorithm in [12]. Consider the next lemma that specifies a condition for increasing the dimension of the state space in Cartesian product partitioning without changing the minimum number of partitions.

Lemma 4.2: *Let $K \in \mathbb{Z}_{>0}$ and let $\{\mathcal{R}^{(1)}, \ldots, \mathcal{R}^{(K)}\}$ be the minimum Cartesian product partitioning of $\cup_{k=1}^{K} \mathcal{R}^{(k)} \subseteq \mathbb{Z}_{\geq 0}^{D}$. Then the minimum Cartesian product partitioning of $\cup_{k=1}^{K} (\mathcal{R}^{(k)} \times \mathcal{Q}^{(k)}) \subseteq \mathbb{Z}_{\geq 0}^{D+1}$ is $\{\mathcal{R}^{(1)} \times \mathcal{Q}^{(1)}, \ldots, \mathcal{R}^{(K)} \times \mathcal{Q}^{(K)}\}$ if $\cap_{k=1}^{K} \mathcal{Q}^{(k)} \neq \emptyset$ and $\mathcal{Q}^{(k)} \subseteq \mathbb{Z}_{\geq 0}$ consists of consecutive integers for $k = 1, \ldots, K$.*

Proof: See [10]. ■

Lemma 4.2 states that when the minimum Cartesian product partitioning of a multi-dimensional state space is known, it is possible to add states and increase the dimension without changing the size of the minimum Cartesian product partitioning.

Using Lemma 4.2, Algorithm 3 presented next generates a random multi-dimensional state space whose minimum Cartesian product partitioning is known. The algorithm starts

Table 4. Properties of random test problems

p	D	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{R}\rvert$	$MinPrt$	Sep	$SepInt$
0.25	3	10^3	(134; 14)	(17; 1)	(10; 3)	(11; 4)
		20^3	(1,059; 62)	(66; 3)	(46; 8)	(60; 14)
		30^3	(3,525; 123)	(148; 4)	(110; 12)	(149; 27)
	4	10^4	(719; 109)	(17; 2)	(15; 5)	(27; 13)
		20^4	(10,719; 809)	(66; 3)	(64; 12)	(129; 33)
		30^4	(54,075; 2,069)	(148; 4)	(156; 19)	(327; 60)
	5	10^5	(4,247; 682)	(16; 1)	(18; 5)	(38; 15)
		20^5	(117,901; 10,960)	(67; 3)	(85; 16)	(212; 56)
		30^5	(856,358; 49,238)	(148; 5)	(204; 23)	(589; 128)
0.50	3	10^3	(272; 20)	(23; 1)	(44; 5)	(88; 22)
		20^3	(2,090; 100)	(87; 2)	(204; 16)	(566; 97)
		30^3	(7,016; 212)	(195; 3)	(484; 25)	(1,517; 188)
	4	10^4	(1,484; 169)	(23; 1)	(61; 8)	(197; 49)
		20^4	(22,146; 1,379)	(88; 3)	(284; 18)	(1,325; 207)
		30^4	(108,987; 3,902)	(196; 4)	(677; 35)	(4,046; 540)
	5	10^5	(8,629; 1,341)	(23; 1)	(75; 8)	(351; 74)
		20^5	(232,892; 15,774)	(88; 3)	(352; 25)	(2,572; 482)
		30^5	(1,626,489; 82,115)	(196; 4)	(870; 47)	(7,664; 921)
0.75	3	10^3	(392; 32)	(21; 1)	(65; 4)	(227; 34)
		20^3	(3,249; 144)	(72; 3)	(261; 9)	(1,720; 90)
		30^3	(10,587; 208)	(161; 3)	(590; 17)	(5,418; 270)
	4	10^4	(2,285; 264)	(21; 1)	(83; 6)	(518; 56)
		20^4	(33,827; 2,004)	(75; 3)	(339; 15)	(4,201; 286)
		30^4	(162,781; 7,263)	(162; 4)	(765; 21)	(13,607; 628)
	5	10^5	(12,101; 1,693)	(20; 1)	(99; 7)	(863; 93)
		20^5	(354,698; 26,549)	(75; 2)	(412; 17)	(7,590; 480)
		30^5	(2,560,044; 163,475)	(160; 3)	(915; 28)	(24,969; 1,364)

with a random two-dimensional state space over $\{0,\ldots,M-1\}\times\{0,\ldots,M-1\}$ that is generated by assigning each state a reachability probability p (lines 3–4). Then the minimum Cartesian product partitioning of this random set is computed (lines 6–6). Next, $(D-2)$ dimensions are added to the state space one at a time (lines 7–13). For each additional dimension, a uniformly distributed random integer that determines the range of consecutive integers is generated (line 8). Then the Cartesian product of each partition with the set of consecutive integers is computed and the corresponding integer is updated to continue with the next dimension (lines 10–11). Finally, dimensions of the states in $\mathcal{R}$ are rearranged so that the initial two dimensions are not always the first two dimensions (line 17).

Table 4 reports the properties of the randomly generated state spaces. The first three columns provide the parameters used in generating the test problems. The first two columns give the inputs of Algorithm 3, p, and D, respectively. The third column gives the size of the product state space, K^D, when each subsystem state space size is K. The remaining columns have the same meaning as in Table 4, where mean values of 51 random test problems are reported for each set of parameters. Observe that the properties of the reachable state space depend on the initial two-dimensional state space. As p and K increase, the number of separators and the number of intersecting pairs increase. As these numbers increase, the

TABLE 5. Experimental results for random test problems

			Merge			Refinement		
p	D	$\lvert\mathcal{S}\rvert$	*Prt*	Time	Memory	*Prt*	Time	Memory
0.25	3	10^3	(17; 2)	(0.0; 0.0)	(0.5; 0.0)	(17; 1)	(0.0; 0.0)	(0.5; 0.0)
		20^3	(70; 4)	(0.0; 0.0)	(0.5; 0.0)	(67; 3)	(0.0; 0.0)	(0.7; 0.0)
		30^3	(154; 5)	(0.0; 0.0)	(0.5; 0.0)	(148; 4)	(0.0; 0.0)	(0.9; 0.1)
	4	10^4	(17; 2)	(0.0; 0.0)	(0.5; 0.0)	(17; 2)	(0.0; 0.0)	(0.6; 0.1)
		20^4	(69; 5)	(0.0; 0.0)	(0.5; 0.0)	(66; 3)	(0.0; 0.0)	(0.7; 0.0)
		30^4	(156; 8)	(0.2; 0.0)	(0.5; 0.0)	(148; 4)	(0.3; 0.0)	(0.7; 0.0)
	5	10^5	(18; 2)	(0.0; 0.0)	(0.5; 0.0)	(16; 1)	(0.0; 0.0)	(0.8; 0.1)
		20^5	(73; 6)	(0.6; 0.1)	(0.5; 0.0)	(67; 3)	(0.8; 0.1)	(24.2; 2.2)
		30^5	(155; 11)	(5.0; 0.3)	(0.5; 0.0)	(148; 5)	(6.5; 0.4)	(171.9; 9.8)
0.50	3	10^3	(26; 2)	(0.0; 0.0)	(0.5; 0.0)	(23; 1)	(0.0; 0.0)	(0.5; 0.0)
		20^3	(101; 7)	(0.0; 0.0)	(0.5; 0.0)	(87; 2)	(0.0; 0.0)	(0.7; 0.0)
		30^3	(224; 15)	(0.0; 0.0)	(0.5; 0.0)	(197; 4)	(0.0; 0.0)	(0.7; 0.1)
	4	10^4	(28; 4)	(0.0; 0.0)	(0.5; 0.0)	(23; 1)	(0.0; 0.0)	(0.7; 0.0)
		20^4	(106; 14)	(0.1; 0.0)	(0.5; 0.0)	(89; 3)	(0.1; 0.0)	(0.7; 0.0)
		30^4	(241; 25)	(0.5; 0.0)	(0.5; 0.0)	(197; 4)	(0.7; 0.0)	(20.7; 0.7)
	5	10^5	(27; 4)	(0.0; 0.0)	(0.5; 0.0)	(23; 1)	(0.0; 0.0)	(0.7; 0.0)
		20^5	(117; 20)	(1.3; 0.1)	(0.5; 0.0)	(89; 3)	(2.0; 0.2)	(47.2; 3.2)
		30^5	(254; 47)	(9.9; 0.6)	(0.5; 0.0)	(198; 4)	(15.5; 1.0)	(326.0; 16.4)
0.75	3	10^3	(29; 4)	(0.0; 0.0)	(0.5; 0.0)	(23; 3)	(0.0; 0.0)	(0.5; 0.0)
		20^3	(106; 13)	(0.0; 0.0)	(0.5; 0.0)	(122; 36)	(0.0; 0.0)	(0.8; 0.1)
		30^3	(228; 31)	(0.0; 0.0)	(0.5; 0.0)	(619; 183)	(0.1; 0.0)	(0.7; 0.0)
	4	10^4	(35; 7)	(0.0; 0.0)	(0.5; 0.0)	(21; 2)	(0.0; 0.0)	(0.8; 0.1)
		20^4	(129; 26)	(0.2; 0.0)	(0.5; 0.0)	(94; 28)	(0.3; 0.0)	(0.7; 0.0)
		30^4	(240; 65)	(0.8; 0.1)	(0.5; 0.0)	(421; 240)	(2.7; 0.3)	(30.6; 1.3)
	5	10^5	(37; 9)	(0.1; 0.0)	(0.5; 0.0)	(21; 2)	(0.1; 0.0)	(0.7; 0.0)
		20^5	(113; 41)	(2.0; 0.2)	(0.5; 0.0)	(82; 18)	(7.7; 1.0)	(71.7; 5.3)
		30^5	(307; 104)	(16.0; 1.2)	(0.5; 0.0)	(287; 112)	(108.5; 16.4)	(514.1; 32.8)

problems become more complicated and it tends to be more difficult to find separators leading to a better solution. Observe that the size of the partitioning, K, depends on $\mathcal{R}'$ (line 6 of Algorithm 3). When p is small, there are less reachable states to be merged. When p is large, there are less conflicting edges in the unit distance graph of $\mathcal{R}'$. Therefore, the size of the partitioning does not increase as p increases.

In Table 5, we present the results of experiments with the randomly generated state spaces of Table 4. The first three columns have the same meaning as in Table 4 and the remaining columns have the same meaning as in Table 3. When p is 0.25 and 0.50, the sizes of the partitionings obtained by the refinement based algorithm are close to their minimum values. However, when p is 0.75, the number of partitions become larger as the size of the product state space increases when number of separators and intersecting separator pairs increase. As for the refinement based algorithm, the number of partitions obtained by the merge-based algorithm is closer to the optimal value for smaller p. For all test problems, the merge-based algorithm is better than the refinement based algorithm in terms of time and memory requirements.

5. CONCLUSION

In this paper, we define Cartesian product partitioning of a D-dimensional reachable state space $\mathcal{R}$ and show that the problem of finding the partitioning with the minimum number of partitions is NP-complete when the dimension D is larger than 2. In order to obtain a Cartesian product partitioning, two algorithms are presented. The first algorithm starts with a partitioning, where each partition is a singleton. In this algorithm, each partition is merged with the first possible partition until it is not possible to merge any two partitions. This algorithm has $O(D|\mathcal{R}|lg(|\mathcal{R}|))$ time and $O(D|\mathcal{R}|)$ space complexities. The second algorithm starts with a partitioning that includes only one partition. Then the unit distance graph of the reachable state space is constructed. This is followed by the refinement of the graph by removing edges, until the vertex set of each connected component becomes a partition in the Cartesian product partitioning of the reachable state space. The time and space complexities of this algorithm is $O(D^3|\mathcal{R}|^2)$ and $O(D|\mathcal{R}|)$, respectively. We consider two groups of test problems, which are from the literature and are randomly generated. The merge-based algorithm fails to compute the optimal partitioning in almost all problems. Furthermore, the size of the partitioning increases substantially when the states in the reachable state space are processed in random order instead of lexicographical order. The time and memory requirements of the merge-based algorithm are larger when the states are processed in random order. However, even when the states are processed in random order, the merge-based algorithm is still faster and requires less space than the refinement based algorithm. In many problems, the refinement based algorithm computes a partitioning that is either optimal or close to the optimal solution. Although it may be more time and memory consuming, the refinement based algorithm almost always computes partitionings with a smaller number of partitions than the merge- based algorithm.

Acknowledgement

The work of M. Can Orhan is supported by The Scientific and Technological Research Council of Turkey.

References

1. Adelson-Velskii, G.M. & Landis, E.M. (1962). An algorithm for the organization of information. *Soviet Mathematics Doklady* **3**: 1259–1263.
2. Baumann, H., Dayar, T., Orhan M.C., & Sandmann, W. (2013). On the numerical solution of Kronecker-based infinite level-dependent QBD processes. *Performance Evaluation* **70**: 663–681.
3. Bournez, K., Maler, O., & Pnueli, A. (1999). *Orthogonal Polyhedra: Representation and Computation, in: Hybrid Systems: Computation and Control.* Lecture Notes in Computer Science, Heidelberg: Springer, vol. 1569, pp. 46–60.
4. Buchholz, P. (1999). Structured analysis approaches for large Markov chains. *Applied Numerical Mathematics* **31**: 375–404.
5. Buchholz, P. (1999). Hierarchical structuring of superposed GSPNs. *IEEE Transactions on Software Engineering* **25**: 166–181.
6. Buchholz, P. & Dayar, T. (2004). Comparison of multilevel methods for Kronecker-based Markovian representations. *Computing* **73**: 349–371.
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., & Stein, C. (2009). *Introduction to Algorithms*, 3rd ed. Cambridge: The MIT Press.
8. Dayar, T. (2012). *Analyzing Markov Chains using Kronecker Products: Theory and Applications.* New York: Springer.
9. Software for computing Cartesian product partitioning of multi-dimensional reachable state spaces. (2013). http://www.cs.bilkent.edu.tr/~tugrul/software.html Accessed 6 June 2015
10. Dayar, T. & Orhan, M.C. *Cartesian product partitioning of multi-dimensional reachable state spaces.* Technical Report, BU-CE-03013, December 2013, Department of Computer Engineering, Bilkent University, Ankara, Turkey.

11. Dielissen, V.J. & Kaldewaij, A. (1991). Rectangular partition is polynomial in two dimensions but NP-complete in three. *Information Processing Letters* **38**: 1–6.
12. Ferrari, L., Sankar, P.V. & Sklansky, J. (1984). Minimal rectangular partitions of digitized blobs. *Computer Vision, Graphics, and Image Processing* **28**, 58–71.
13. Garey, M.R. & Johnson, D.S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: Freeman.
14. Jain, A., Sahni, S., Palta, J., & Dempsey, J. (2003). Partitioning 3D phantoms into homogeneous cuboids. *International Journal of Foundations of Computer Science* **14**: 905–931.
15. Keil, J.M. (eds). (2000). *Polygon Decomposition.* In: Sack, J.R., Urrutia, J. Handbook of Computational Geometry, Amsterdam: Elsevier, pp. 491–518.
16. Maehara, H. (1989). Note on induced subgraphs of the unit distance graph E^n. *Discrete and Computational Geometry* **4**: 15–18.
17. Paige, R. & Tarjan, R.E. (1987). Three partition refinement algorithms. *SIAM J. Comput.* **16**: 973–989.
18. Soltan, V. & Gorpinevich, A. (1993). Minimum dissection of a rectilinear polygon with arbitrary holes into rectangles. *Discrete and Computational Geometry* **9**: 57–79.
19. Stewart, W.J. (1994). *Introduction to the Numerical Solution of Markov Chains.* Princeton: Princeton University Press.
20. Ziegler, G.M. (1995). *Lectures on Polytopes.* New York: Springer.