IJWR

State-Space Analysis and Complexity Assessment of Puzzle Games Using Colored Petri Nets

Ahmad Taghinezhad-Niar*, Saeid Pashazadeh
Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran; a.taghinezhad@tabrizu.ac.ir, pashazadeh@tabrizu.ac.ir

ABSTRACT

The verification of complex systems has traditionally relied on semi-automatic theorem-proving methods. However, model checking represents a paradigm shift by enabling automated, exhaustive verification of behavioral properties through systematic state exploration. Among advanced formal verification tools, Colored Petri Net (CPN) stands out for its integration of the ML programming language, facilitating robust model checking and system validation. Nevertheless, the application of CPN to complex systems is often constrained by the state-space explosion problem, which presents a significant challenge in contemporary research. While state-space analysis offers powerful capabilities for validation and scenario extraction, its potential remains largely untapped due to computational complexity constraints. This limitation is particularly pronounced in concurrent systems with multiple interacting variables, exemplified by game systems where intricate rule sets, deadlock conditions, and termination scenarios demand sophisticated modeling approaches. This paper presents a novel methodological framework for modeling and analyzing such game riddles, introducing methods to mitigate the state-space explosion problem. We demonstrate the efficacy of our approach through a comprehensive case study of the Merchant Ship puzzle game, though the methodology generalizes across various game typologies. By synthesizing model-checking techniques with ML-based algorithmic implementations, we develop an optimized search strategy for traversing the state space graph, enabling the derivation of quantitative complexity metrics. These metrics encompass critical indicators such as the success-to-total scenario ratio and the minimal trajectory length for both successful and unsuccessful game completions. Our research contributes to both the theoretical understanding of game complexity analysis and practical applications in game design through formal methods.

Keywords*— *State-Space Analysis, Model-Checking, Colored Petri Net, Riddle Game.

1. Introduction

Designing distributed and concurrent systems require higher complexity, which raises the possibility of technical errors. In concurrent and sophisticated systems, a chain of events may occur that leads to system failure, which may be overlooked during the design phase. A lot of events occur during play in computer games, making it difficult for a human creator to examine all scenarios and riddles. Formal modeling can automatically calculate all of a system's behaviors. Most formal models can produce a state-space graph of the modeled system, which provides valuable information for studying and demonstrating the system's behavior. Because the state-space network can have a large number of nodes, each of which represents a state of the system obtained from various places, extracting features' information from these states is difficult. To address these issues, formal approaches are extensively employed in the design process to describe a system's function and confirm its behavioral features.

CPN is one of the well-known formal methods that is used to model and verify the behavioral characteristics of distributed and concurrent systems. The state-space graph of the colored Petri net model of a system is generated automatically using CPN tools. Designing a model that leads to a rational state space is a challenging problem in CPN that could lead to state-space explosion [1, 2, 3]. As a result, a thorough analysis of the system cannot begin until all state space of a model has been determined. However, many works [4, 5, 6] do not consider

http://dx.doi.org/10.22133/ijwr.2024.482725.1241
Citation A. Taghinezhad-Niar, S. Pashazadeh,"State-Space Analysis and Complexity Assessment of Puzzle Games Using Colored Petri Nets", *International Journal of Web Research*, vol.7, no.4,pp.13-27, 2024, doi: http://dx.doi.org/ 10.22133/ijwr.2024.482725.1241.

*Coressponding Author
Article History: Received: 9 June 2024; Revised: 3 September 2024; Accepted: 15 September 2024.

Copyright © 2024 University of Science and Culture. Published by University of Science and Culture. This work is licensed under a Creative Commons Attribution-Noncommercial 4.0 International license(https://creativecommons.org/licenses/by-nc/4.0/). Noncommercial uses of the work are permitted, provided the original work is properly cited.

model checking, and their models suffer from state-space explosion [7].

Game modeling can offer different advantages. For example, a riddle of the game can be studied by changing the initial markings of the model. The complexity of a game riddle can be found by model-checking its state space. Therefore, a game developer company can utilize the model to sort its game riddles based on their complexity. Riddles can also be examined to avoid deadlocks.

Even studies that utilized state-space analysis of CPN only consider the basic features like having no deadlock, however, a state-space graph can give us a lot of information about a model. Hence, in this paper, we consider a game riddle as a case of study and formally model and obtain advanced information like the complexity of a game riddle (explicit scenarios of a game riddle), the shortest path to success and failure of a riddle proposing an advanced search algorithm inspired by breadth-first search (BFS) over the state-space graph of the game model. By this means, a new metric is defined for evaluating the complexity of a game riddle, and a search algorithm of the state-space graph is proposed for evaluation and proving the value of this metric. Moreover, we propose two strategies in this study to avoid a state-space explosion. The proposed methods can be extended to other domains, such as web behavior modeling, ensuring security, and improving the reliability of concurrent systems. This work contributes to advancing the analysis and verification of complex systems in various applications, including web and IoT environments.

To the best of our knowledge, this is the first study that covers issues like model checking for extracting distinct features of a riddle game while also discussing strategies to avoid state-space explosion. Contributions of this paper are as follows:

- Modeling and analysis of a puzzle game as a case study for verification and validation of game riddles by CPN.
- Two novel approaches to control the state-space explosion of the model. Two new metrics for representing the complexity of a game riddle and algorithms for evaluation of these metrics
- A function to find success and failure scenarios of the game riddle for directing and scoring gamer's action

The remainder of the paper is structured as follows: Section 2 includes a survey of the literature in which formal methods and CPN were used to model and analyze analogous systems. Section 3 contains the problem definition and presents the fundamental concepts of the proposed case study. Section 4 delves into the specifics of the proposed game model. Section 5 employs the model-checking technique to validate the anticipated attributes of the modeled game and analyze its riddle by exploring the state-space graph. Methods for reducing the state-space explosion are also introduced. Section 6 concludes with a brief conclusion to this study.

2. Related Work

Colored Petri Nets (CPN) are highly effective for modeling and evaluating systems. They have been crucial in developing discrete-system models over the past 40 years. Designing and verifying parallel systems is more complex than successive systems, as discussed in [8]. They can be modeling in complex, concurrent or distributed systems such as scheduling in algorithms in computing environments [9,10] and cloud computing [11, 12, 13] or designing fault-tolerant frameworks software defined networks [14, 15].

CPN-tools is a sophisticated tool for modeling with CPN. Initially released in 1989, it included network editing and simulation features. The simulator, launched in 2000, faced performance challenges with larger models. Recent updates have improved the interface and simulation engine, enhancing efficiency for large state-space models [16, 17].

Zhao et al. [18] investigated a practical scheduling problem arising from wire rod and bar rolling processes in steel production systems using CPN. Their challenge involved optimizing serial batch scheduling, considering sequence-dependent family setup time, and release time, and minimizing the total number of late jobs across all batches. To address this, the study employs a Petri net (PN) model and formulates a mixed-integer linear program (MILP). Four iterated greedy algorithm (IGA)-based heuristics are proposed. Consequently, these heuristics hold promise for practical scheduling problems beyond the specific context studied.

Lages et al. [19] employed a Colored Petri Net (CPN) framework to comprehensively analyze energy consumption in Low-Power Wide-Area Network (LPWAN) systems. These findings hold significant implications for the ongoing development of energy-efficient Internet of Things (IoT) solutions. The proposed model, which focuses on CPU and communication transceivers, provides valuable insights for both future research and practical applications in the field. The CPN-based framework enables detailed analysis, considering various operational parameters and their interactions. Validation using a real-world hardware platform yielded impressive accuracy, with errors below 1.4% for CPU energy consumption and 0.14% for network energy consumption, enhancing the model's credibility for real-world implementation.

Kalid et al. [20] emphasize the importance of techniques, tools, and procedures for automatic threat diagnosis and recovery in IoT-based cyber-physical systems (CPSs). Specifically, it proposes an IoT-based Colored Resource-Oriented Petri Net (CROPN) to self-detect and self-treat failures, measuring reliability metrics such as uptime, downtime, and availability. The CROPN approach simplifies configuration, overcomes deadlock issues, and enhances reliability. Simulation results validate its effectiveness.

Shahidinejad et al. [21] proposed an elastic cloud controller using CPN. They proposed this approach to predict the required resources in order to cope with workload changes, fulfill service level objectives requirements, and avoid over- or under-provisioning problems. However, they utilized neither validation nor verification methods to analyze their CPN model.

Mishra et al. [22] proposed a model for a profile-based system using Petri nets which will operate in real-time to assess suspicious actions and automatically detect intrusion. The proposed approach will also examine the alarms to find intrusions and provide a real-time instant reaction to protect. They only utilized the traditional Petri nets without model-checking.

In research [1] modeling and analysis of agent-based human behavior are proposed. They tried to present validation and verification of their model by formalizing it, using CPN and state-space analysis. Their case study was a park environment with different playgrounds each one has its roles and different type of agents. Agents as a people of society had to follow roles of their own and playground and make decisions. Their modeling in CPN was so simple. nevertheless, they did not consider the state space of the given scenario automatically; they manually analyzed state space by observing the status of each state. In this paper, we provide methods for automatically checking the state space as well as a novel way for extracting all goal scenarios from the CPN-Tools state space reachability tree.

Kuchař and Vondrák [23] presented a method for simulating and analysis of business process management models. They tried to extract information using the simulation model to enhance strategies and methods in resource allocation. They modeled the human-based process for the activity of allocating workers based on their competencies to perform tasks. The productivity of their model using each resource capability has the formalism of Petri nets as they mentioned. However, they do not illustrate the model.

Aguiar et al. [24] addressed the monitoring system in productive cells with multiple robots using CPN and a graphical simulator. They modeled and evaluated cell behavior at a high level of abstraction using the model of CPN. The approach functionality is evaluated by modeling the activity of each of the cell components and coordinating them with a monitoring system.

Rehman et al. [5] simulate the agent-based warehouse control system with a Petri net. Identifying system features and evaluating their performance, they utilized the results of the state space. Hsiung et al. [25] addressed automated and correct designing of complex, real-time, and embedded software. A time-memory scheduler (TMS) method has been proposed for combining and generating automated code, using a timed CPN in real-time embedded systems.

In [26], the Petri net has been mapped by parallel programming in the C++ programming language, and an approach to extract the parallel framework for Petri nets has been provided, which established a link between Petri net and Parallel Object-Oriented Programming.

In our previous paper [27], we used CPN for proposing a model for the implementation of monotonic read consistency for distributed systems. In that model, clients were sending their transactions including reading and writing operations concurrently to distributed data servers, the model proved to be successful to guarantee monotonic read consistency.

In [2], a flood monitoring system based on CPN is studied. They proposed a CPN model of their system and analyzed it using model-checking. In their model, properties like liveness (no deadlock) and reliability are considered due to proposing a system without flow.

In [3], a model of hierarchical CPN for a network flow control system is investigated. They did not consider verification and state-space analysis to investigate the features and prove the flowless function of their model.

The authors in [28] used Petri net as a designing language for characteristics of complex workflows to model and analyze the business process flow management model, which allowed the business process logic to provide computer support.

Fedorova et al. [4] proposed a technique to simulate and assess such UAV systems using CPN. It is based on UAV application standards and enables a modular view of common setup components and various UAV kinds. Their method does not contain state-space analysis to validate their methodology.

In [29], Yu et al. investigate logical vulnerabilities in e-business. They propose dynamic slicing techniques, leveraging Colored Petri Nets (CPN), to identify these vulnerabilities during system design. Their streamlined approach enhances e-business

security. The developed model, known as the Interactive Business Process Fusion (IBPF) net, excels at pinpointing vulnerabilities during the design phase. However, the current analysis methods for IBPF require urgent innovation. By employing targeted dynamic slicing, they simplify the analysis process, prevent state space explosion, and gain a distinct advantage. The research outcomes contribute to system reliability enhancement, reduced maintenance costs, and improved e-business security analysis techniques.

Kaid et al. [6] proposed a two-step deadlock prevention strategy based on a CPN and a structurally simple mechanism that drastically lowers the number of monitors in flexible manufacturing systems. They used a vector covering the approach to produce a minimal covered set of first-met bad markings and legal markings. They merged all produced monitors into a global control area in their model. Their method just used the traditional Petri net capabilities, therefore CPN extension that enables ML programming and model-checking by code is not utilized in their study.

Drakaki et al. [30] proposed a simulation modeling approach based on CPN to assess how inventory management choices affect supply chain efficiency. The approach described models of inventory management in a multi-stage serial supply chain under normal operating conditions and in the case of interruptions, for both traditional and information-sharing setups, using hierarchical timed CPN.

Kristensen's Ph.D. dissertation [31] was the first to introduce state-space analysis methods for CPN. He proposed various strategies for generating the state space in CPN tools. They investigated data structures and methods of storing state space, as well as proposing a rigorous evaluation of the software. Every node in the state space denotes the values (markings) of places. However, problems like the different ordering of a record and list may lead to state-space explosion in concurrent systems and game riddles. In this paper, we will address techniques to circumvent this issue.

To this end, numerous articles underestimated the power of CPN state-space analysis. Furthermore, modeling concurrent systems typically result in state-space explosion, which is one of the reasons why state-space analysis is underutilized. Using state-space analysis, a few articles were successful in demonstrating properties such as no deadlock and reliability [2, 32]. However, this paper first formally models a game riddle using CPN, then proposes methods to prevent state-space explosion, and finally presents advanced model-checking methods utilizing the ML programming language to extract complex features from the model, which in this case is a game riddle. As a result, a graph search algorithm inspired by the BFS algorithm is suggested to compute various game characteristics such as the shortest answer, the complexity of a riddle, and the shortest failure scenario.

3. Modeling and Problem Formulation

3.1. Colored Petri Net (CPN) Basics

This section introduces CPN basic formulas at first, followed by the game's fundamental principles.

Petri net was first designed in the Ph.D. dissertation by Adam Petri and introduced as a formal method to model concurrency and synchronization in the concurrent system [33]. The traditional Petri net is a bi-directional graph consisting of two types of nodes: 1) places and 2) transitions. The nodes are linked by directional arrows. Places and transitions are represented by ovals and rectangles, respectively. Colored (high-level) Petri net is an extension of traditional Petri nets which is integrated with standard ML (Standard Meta Language). ML is A high-level, modular, functional programming language with compile-time type checking and type inference is called Standard Meta Language (ML). It is often used for creating compilers, researching programming languages, and creating theorem provers. Colored Petri Nets (CPNs) rely heavily on standard ML, which provides data manipulation primitives and allows for compact, parameterizable models. It gives Petri Nets the expressiveness required for modeling large industrial systems, as well as supporting the formal semantics and implementation of CPN computer tools. This work investigates the use of functional programming and Standard ML in the CPN modeling language and its associated tools to simulate, verify, and analyze concurrent systems.

In addition, appropriate required data types and functions are added to this language for being consistent with formal concepts of colored Petri net. By this means, the modeling capabilities of the concurrent systems using CPN are extended [34]. Colored Petri net is defined as follows:

($\sum$, P, T, A, V, C, G, E, I)

Σ: A finite set of defined colors.

P: A finite set of places.

T: A finite set of transitions.

A: A finite set of directional arcs: $A \subset P \times T \cup T \times P$

V: A limited set of variables including types, such as $\forall v \in V, Type[v] \in \sum$

C: The color function is defined as a mapping from P to Σ and specifies the color of places.

G: is a guard function and is defined for T and determines its activation condition.

E: The expression is the arc function. $E: A \rightarrow EXPR_V$

I: The initialization function for places.

3.2. Modeling the Game

The case study of this paper is a puzzle game named Merchant ship that is modeled using CPN with different configurations and riddles. The game has a two-dimensional environment with static and dynamic barriers that we call "Balk" in the modeling. The goal of the game is for a merchant ship to go from its starting point to a target destination. The ship can move in four directions (north, south, west, and east) and should not collide with immovable rocks; ship mobility is limited by walls. Pirate ships patrol on a predefined route with predefined timing and destruction radius. If the merchant ship crosses the hampering radius, it will be taken by pirates, and the game will fail.

The game environment has been studied in different configurations, including Riddle (α) and Riddle (β), as illustrated in Figure 1. The problem is first investigated in a simple 4x4 context (Riddle (α)) to provide a basic description of how the proposed solution works. Pirates can have a hindering radius, which implies they can destroy or hijack merchant ships within a certain radius.

4. Proposed Modelling of a Merchant Ship Puzzle Game as a Case of Study

In this section, we will illustrate the proposed game model, including the colors and functions that it employs. The proposed game model's color sets are explained first, followed by each function based on ML code and the CPN model.

4.1. Color Sets of the Game Model

Color sets are usually known as data types in programming languages and are used to declare variables. Declarations for color sets, functions, variables, and constants are provided by CPN tools. A specific color set should be assigned to each place in the colored Petri net, and only tokens from that color set can be present there. The transitions and arcs are inscribed using variables and functions. The colored sets of the proposed model are shown in Code 1.

The SHIP color set indicates the ship's features, including its id and the ship's present latitudinal and longitudinal position, which are shown by row and col, respectively. trow and tcol show the latitudinal and longitudinal positions of their destination target, respectively. The SHIPS color represents a set of SHIP color sets that can be defined in the game.

The ROCK color is defined to express the profile of a cliff, which includes a latitudinal and longitudinal position of it. The ROCKS color is used to define a list of ROCK colors (i.e., cliff properties). The PSTATUS color is a numerical type and is defined to represent the patrolling direction of the pirates (north, south, east, and west).

The PIRATE color indicates the characteristics of a pirate ship including the characteristics of the current latitudinal and longitudinal position, its movement direction, the next position, the next movement direction (after a movement), the start of the movement, the end of the movement (patrolling between these two positions), and in the end the radius of damage influence ('sight') of the pirate. PIRATES color is defined to display a list of PIRATE color types. BALK color shows the hazards and obstacles in the game, including pirates and rocks, which is a record of PIRATES and ROCKS colors. These color sets are shown in Code 1.

4.2. Initial Markings and the Model of the Game

The constants values of intialWidth and intialHeight are defined to indicate the length and width of the game environment, and their values are equal to 4 for the Riddle (α) shown in Figure 1. Code 2 shows these initial variables. Two more constant variables, intialShip, and intialBalk, are used to initialize the game's ship attributes (identity, ship location, and destination), as well as the game's barrier properties (the position of pirate and rocks). The initial markings of the game are shown in Figure 1.

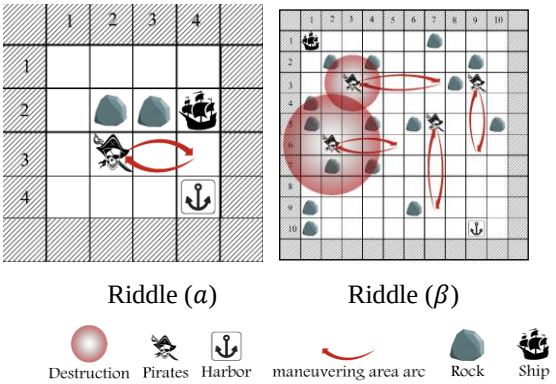

Figure. 1. Two configurations of the game riddle

```
colset SHIP=record no:NO* row:INT* col:INT* trow:INT
*tcol:INT
closet SHIPS=list SHIP;
colset ROCK=record row:INT*col:INT;
colset ROCKS=list ROCK;
colset PSTATUS=with E|W|N|S;
colset PIRATE=record row:INT*col:INT*
pstatus:PSTATUS*nrow:INT*ncol:INT* nstatus: PSTATUS
* starte:INT* ende:INT* sight:INT;
colset PIRATES=list PIRATE;
colset BALK=record pirates:PIRATES*rocks: ROCKS;
```

Code 1. color sets of the proposed game model

4.3. Functions of the Proposed Game Model

In this part, several functions of the game model are described. The canGoUp guard function receives a merchant ship, a barrier list, and a list of merchant ships shown in Figure 1. It checks when the merchant ship moves north, whether it hits other ships, rocks, and pirates or not. If the ship is free to go return true to enable the transition otherwise return false to disable the transition. Other guard functions such as canGoLeft, canGoRight, and canGoDown are like canGoUp to examine the possibility of merchant ship movement to the west, east, and south, respectively. The function canGoUp calls the checkup function defined in Code which is used to check if the next location is free from pirates, rocks, and other merchant ships. The function members defined in Code 5 and membership defined in Code 6 are also called hierarchal by checkup function.

The isDead guard function shown in Code 7, checks if the ship is captured by pirates. It is given a merchant ship, a list of obstacles (including rocks and pirates), and a list of ships. If the merchant ship was not at its target and could not move north, south, west, or east, the return value is true (the ship is being blocked by pirates) to enable the transition of the ship from 'harbor' place to 'dead' place of the model.

```
val intialWidth=4;
val intialHeight=4;
val intialSHIP=[{no=1,row=2,col=4,trow=4,tcol=4 }];
val intialBalk={pirates=[{row=3,col=2,pstatus=E
,nrow=3,ncol=3,nstatus=E,starte=2,ende=4,sight=0}],Rock
s=[{row=2,col=2},{row=2,col=3]};
```

Code 2. initial markings of the game model for the riddle (***a***)

```
fun checkup(y,x,rock,pirate,ships):BOOL =
   if y> intialHeight  orelse y< 1 orelse x>intialWidth orelse
x<1 then false
   else if memberp(y,x,pirate) andalso members(y,x,rock)
andalso membership(y,x,ships)
       then true
   else false
```

Code 3. canGoUp function of the proposed game model

```
fun canGoUp(ship:SHIP,balk:BALK,ships)=
let
            val sprow= #row ship
            val spcol= #col(ship);
            val sno= #no ship;
            val rocks= #rocks balk;
            val pirate= #pirates balk;
            val stcol= #tcol ship;
            val strow= #trow ship;
in
  if sprow=strow andalso spcol=stcol then  false
            else (if checkup (sprow-1,
spcol,rocks,pirate,ships) then
            true
                        else false )
end
```

Code 4. checkup function of the proposed game model

For updating the locations of pirates and merchant ships, other methods like updateBalk and direct are employed, respectively. If the merchant ship is at its intended destination, the guard function isGoal will return true.

The proposed model of the game is illustrated in Figure 2. This model is captured from CPN-Tools, where green boxes at the top of places show the initial marking (values) of the place's colors. A transition box with a green border shows that it is ready to fire. The proposed model includes four places: harbor, balk, dead, and destination. The harbor is where merchant ships are serviced. The 'Balk' place stores the pirate locations and the rocks detailed. The merchant ships captivated by pirates are stored in the place 'dead'. The place 'destination' stores the merchant ships that reach their destination.

5. Proposed State Space Analysis Methods of the Game Model

The model's simulation can be executed in two forms: 1) interactive, and 2) automated using CPN-tools software. In interactive form, the user selects the desired transition and fires it in each step to change the system state. In the automatic simulation form, the CPN-tools fire enabled transitions non-deterministically to obtain all potential model states.

This form generates a state-space graph for all sequences of states which could represent a specific execution scenario. The complete state-space graph contains all possible execution scenarios of the system. In the state space graph, each node represents a state of the system and the nodes are connected by arcs. Arcs represent the firing of an enabled transition

```
fun members(y,x,st:ROCKS):BOOL =
   if st= [] then true
   else
     let
                        val head = List.hd st
                        val tail = List.tl st
                        val col= #col head;
                        val row= #row head;
     in
        if y=row andalso x=col then false
        else members (y,x,tail)
          end;
```

Code 5. members function of the proposed game model

```
fun membership(y,x,st:SHIPS):BOOL =
if st= []  then true
else
            let
                        val head = List.hd st
                        val tail = List.tl st
                        val col= #col head
                        val row= #row head
     in
        if y=row andalso x=col then false
        else membership (y,x,tail)
    end;
```

Code 6. membership function of the proposed game model

of the model in the current state of the system and the next system state.

In our model, for example, when a merchant ship or a pirate advance, a new state is generated. Three CPN state-space nodes from a 2-dimensional merchant ship game are shown in Figure 3, where the place 'harbor' has a ship with no='1' in row '1' and column '1'. Node 3 in the state-space graph shows that the merchant advanced south, which is represented by row '2' and column '3'. Following that, node '5' displays the ship at the initial node's location. As a result, a circle of game agents could represent various nodes in the state space. However, a state-space explosion may occur, if a game is not carefully modeled in CPN. Hence, we present methods in the next part to circumvent the issue.

```
fun IsDead(ship:SHIP,balk:BALK,ships)=
let
        val r= #row ship
        val c= #col(ship);
        val sno= #no ship;
        val rocks= #rocks balk;
        val pirate= #pirates balk;
        val tr= #trow ship;
        val tc= #tcol ship;
in
  if r=tr andalso c=tc then  false
  else (if  checkup(r,c+1,rocks,pirate,ships)=false andalso
checkup(r,c-1,rocks,pirate,ships) =false andalso
checkup(r+1,c,rocks, pirate,ships ) =false andalso
checkup(r-1,c,rocks,pirate,ships)=false
        then true  else  false )
end
```

Code 7, isDead function of the proposed game model

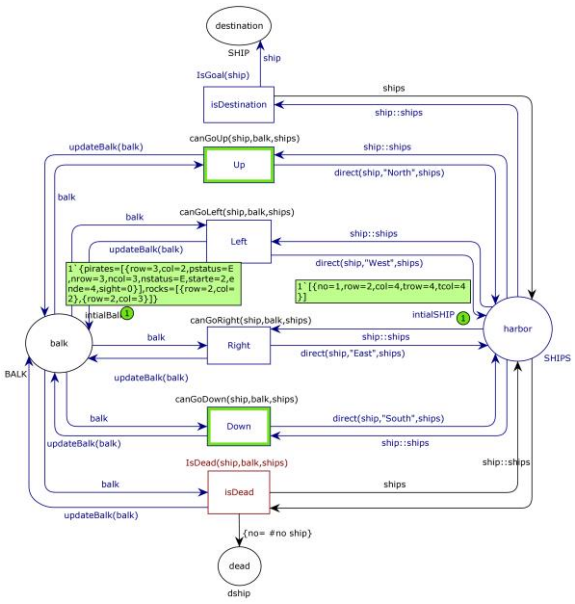

Figure. 2. modeling of a merchant ship game on the CPN-tools

5.1. Proposed methods for avoiding the state space explosion

One of the most problematic issues in CPN models is the state space explosion. This problem leads to a block on model-checking the model. This paper proposes two methods for controlling the state-space explosion: 1) Sorting the color sets; 2) Using complex functions on transitions; in other words, creating transitions that perform extensive processing on the system's state.

When the placements of items in a list change, CPN tools assume a new state of the system; nevertheless, in most models, the list is used to represent a set, and the order of its components is irrelevant. Sorted tokens of a place, which is a list type in the model of this study, result in a huge reduction in the size of the state-space model. The presented model sorts the list of pirates' color sets following each transition. This method avoids the needless generation of redundant states. Our study leads to the conclusion that sorting tokens of a list in the places of a model in CPN can reduce over 20 percent of redundant state spaces generated by CPN tools.

The second strategy advocated in this paper for modeling with CPN is to employ sophisticated CPN features such as defining list and record data types and using programming functions. Complex color sets (data types) serve as the foundation for the complex processing of system states via transition firing. Every fire of transitions in the CPN generates a state in the state-space graph; therefore, if we can build a transition that performs more operations per fire than a standard Petri net transition, we can avoid the production of redundant states in the model. For example, in the proposed model, we built advanced functions such as updateBalk and direct, which can update the direction of a merchant ship and a pirate at the same time, eliminating the need to update each actor in a game separately. This method alone can reduce redundant states by 50%. In addition, we also built advanced guard functions such as canGoUp, canGoLeft, canGoRight, canGoDown, IsGoal, and IsDead to determine whether a transaction can fire.

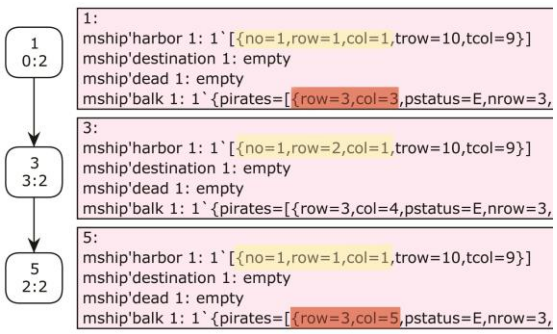

Figure. 3. state-space of a 2-dimensional riddle game

Figure 4 shows the time of generating state space of the modeled game in various environment sizes. This result shows that all states of the modeled game can be calculated at a very convenient time. In our initial models, we were facing state space explosion and the CPN tools could not calculate all the states of the model in a convenient time. However, applying the proposed methods dramatically reduced the state space size and calculation time which is shown in Figure 4.

5.2. Proposed Model-Checking for the Model of a Game Riddle

In this part, the state space graph of the proposed model will be analyzed. In addition, we also propose a method to recognize sequences of state space nodes leading to a termination of the game riddle. All the modeling of this paper has experimented with an Intel Core i7 3630QM processor and 16 GB of memory.

Figure 5 illustrates the automatically generated complete state-space graph of the modeled game with the initial marking of the riddle (a) using CPN tools. In another word, this graph display all possible events that could occur in the simulated game with initial markings of riddle (a). Nodes with a red border line and font color (i.e. 9, 27) in Figure 5 are the dead markings which are also shown in the state-space report of the game model in Table 1. The first state of the state space graph is where that model can be started which is node 1. In our example of riddle (a), the ship can only go up and down as shown in nodes 2 and 3 in Figure 5, respectively. At the same time that the ship moves, pirates are also moving by each arc and node. Each node of the generated state space of CPN tools shows the id of a node at the top and two numbers separated by ':'; where the first number indicates the number of incoming nodes and the latter denotes the number of outgoing nodes according to Figure 5.

The Riddle (β) of this paper is a more complex environment of the game to examine the speed of the proposed validation and verification functions. Riddle (β) in Figure 1 shows the abstract picture of the configuration of the second example. The graph of state space of the second example, shown in Table 1, has 609 nodes and 1326 edges. The presented methods to reduce the state-space volume and avoid its explosion enabled us to run our validation and verification tests in the model. The experimental implementation illustrates that the presented approach in this article is suitable for the validation and verification of complex models of Petri Net and the extraction of the ideal scenarios from its state space. Figure 6 shows the state space graph nodes and arc size for various dimensions of the game puzzle. We altered the initial markings of the game riddle between 10x10 (10 rows and 10 columns) and 50x50, and then we demonstrated in Figure 6 the number of nodes and arcs that CPN tools generates automatically from the model. The number of nodes is the all states that can be generated out of the CPN tools, and arcs show the transition between them.

5.3. Proposed Functions for Model-Checking of the Modeled Game Riddle

In this part, we propose a methodology to discover the expected scenarios from state space, by another meaning, methods to identify different sets of

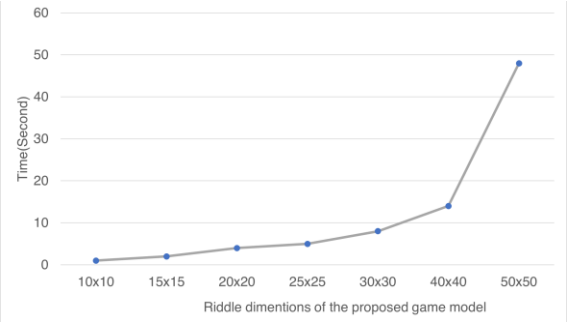

Figure. 4. time to calculate state space for various dimensions of the game riddle

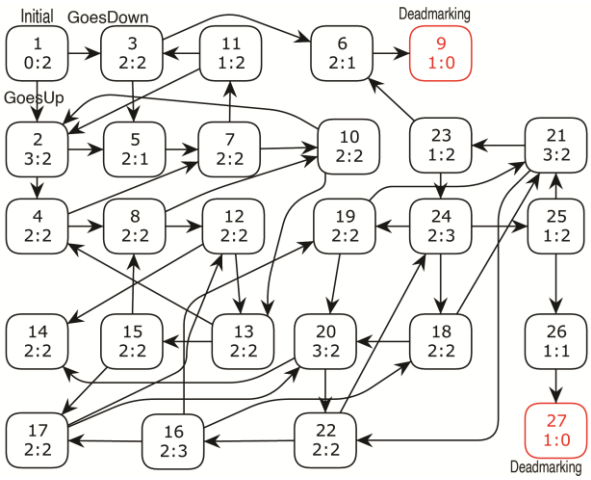

Figure. 5. A complete state-space graph of riddle (a) generated by CPN-Tools

Table 1. Report of generated state space of the riddle (a) by CPN-Tools

Properties of the proposed model	***Riddle (a)***	***Riddle (β)***
Number of Nodes	27	609
Number of Arcs	49	1326
Time to generate state-space	0 Sec	1 Sec
Status of state-space	Full	Full
Number of Dead marks	2	17
Id of Dead markings	27, 9	594,580,58,561, 537, …

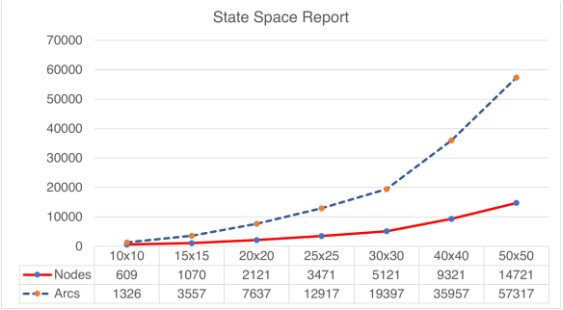

Figure. 6. state space graph nodes and arc size for various dimensions of the game puzzle.

sequential state space nodes that follow a scenario from the start to the termination of the model will be proposed.

Dead markings of the state-space graph are divided into two different categories 1) solution of the riddle and 2) gamer failure. For example, in our model, when a merchant ship successfully reaches its destination, we call that the game successfully terminated. However, if a pirate captures or destroys all the merchant ships then it will be terminated by failure.

We can discover state space nodes from the starting state to the dead marking states by knowing the success and failure dead markings. Hence, we propose methods for discovering success and failure dead markings, as well as a set of paths from the start state to these dead markings and explicit paths (paths without an unnecessary circle) from the start to a dead marking. We also find the complexity of a game by model-checking its state space. The complexity of the game is defined as the ratio of explicit successful solution paths to the total number of solution paths (both success and failure solution paths).

Let us call the set of connected nodes from the start to a dead marking node as a solution path. Therefore, based on the complexity of the modeled game, it can have many solution paths, however, these paths may contain unnecessary circles in the state space. Therefore, in the following, we will illustrate advanced ML functions for model-checking to discover different game features.

Numerous functions are written to model checking of the proposed game model. This part only illustrates important model-checking functions and briefly describes the remaining. Table 2 provides an overview of the functions used that are not covered in an ML code format.

5.4. Model Checking of the Successful Termination Scenarios for the Game Riddle

Successful termination is dependent on our concept in the game riddle. In the modeled game riddle, when a merchant ship reaches its destination in a two-dimensional environment, CPN considers it as the termination of the game riddle since no transition in the model will be enabled.

We proposed the successDeadmarkings function of the state space to find a list of dead markings known as a successful termination of the game riddle. It traverses the dead marking list and returns a list of dead marking state nodes where the merchant ship has arrived at its destination; these dead markings are known as the game riddle's success dead markings. Code 8 displays the function's ML code. We can use the successDeadmarkings function to find the shortest scenario from the state space of the modeled game riddle. Hence, the shortest scenario to game termination is another property of a game riddle that can be discovered by model-checking. This can be regarded as a criterion for assessing the complexity of the modeled game riddle. Therefore, we proposed the function ShortestSenario in Code 9, which receives a list of dead markings and returns the shortest dead marking node.

When the shortest dead marking node of the state space is found, a list of state nodes from the beginning node to the dead marking node (solution path) can be found by calling the following function: NodesInPath(1, ShortestSenario()).

Using the NodesInPath and ShortestSenario functions, the CPN-tools software reaches four sequential state space nodes (solution path) from state 1 to a dead marking state 9, as illustrated in Figure 7. In this figure, node 1 shows the place harbor containing the merchant ship with id 1 in row 2 and column 4. In node 3, the location of the merchant ship is updated to row 3 and column 4. In node 6, the merchant ship has moved to row 4 and column 4. Node 9, shows the state that the merchant ship marker transitioned from place harbor to place destination, where no transition is enabled anymore which indicates a dead marking.

5.5. Model Checking for the Presence of a Deadlock in the Modeled Game Riddle

Formal methods are used to evaluate design defects. If a gaming riddle reaches a point when no action can be taken and the riddle can not be solved (in this case, all ships have not arrived at their destination), it signifies that there is a deadline in the riddle. To avoid deadlines, we created the 'DeadlockExistance' function that analyzes the modeled game state space demonstrated in Code 10. Testing the code for both presented game riddles (α) and (β) in CPN tools will return 'False' which indicates the riddles do not encounter deadlock.

5.6. Model Checking for Solution Paths in the Modeled Game Riddle

In this part, we will illustrate how model- checking

Table 2. Notations of functions used in the proposed model

Description of the proposed model-checking functions	
RemoveFromList1(list1, list2)	Returns list1 - list2
List.length list	Returns count of a list of members
extractColAgent	Extracts the longitude (column) location of a ship
extractRowAgent	Extracts the latitude (row) location of a ship
List.hd	Returns the first content of the list
List.tail	Returns all content of the list except the first one
hasCommonNodes	This function takes two solution paths and returns true if they have at least one common node, and false otherwise.
isMemberOfPath	Receive a state node and a solution path and determine whether or not a merchant ship has passed through the locations of the solution path.
nodeRepetition	Receive a state node and solution path and based on the merchant ships' location of the state node, calculate the number of times, they were standing in the same location on the solution path states.
failDeadmarkings	Returns a list of dead markings that the merchant ship is destroyed by pirates.
NodesInPath(s,e)	Find the shortest state nodes connecting the nodes s and e.
ContainCommonNodes (n, list)	Receive a state space node and a list then returns true if the node is in the list.
solutionPathsDetail	Receives a solution path (list of state space node) and describes the merchant ship locations in a list of lists (each list is solution path) from start to a dead marking. Only works for the riddles that one merchant ship is defined.
RemoveFromList	Receives two lists and subtract them (i.e., list1- list2)
ListDeadmarkings	This is an inbuilt CPN tools function to obtain a list of dead markings from the state space
NodesInPath	This is an inbuilt CPN tools function that receives two node states of the state space graph and returns a list of shortest sequential nodes states between them.
OutNodes	This is an inbuilt CPN tools function that receives an id of a state space node and returns a list of its output nodes in the state space.
ShortestDeadmarking	Receives a list of dead markings and returns an integer as an id of the shortest dead marking (steps from start to termination)

can help discover desired scenarios of the modeled case. Therefore, we investigate the complexity of a riddle utilizing the model's automatically created state space for the modeled game riddle by determining the number of solution paths to success and failure. Success solution path means the process of a game from start to successful finish of the game riddle. Here it means paths that a merchant ship reaches its destination at the end of the game. Three straightforward explicit success solution paths can be deducted for the riddle (a) of the modeled game which is illustrated in Figure 8.

We will illustrate functions to acquire these solution paths from the state space of the modeled game riddle. Therefore, only by changing the initial markings of the modeled game, we can study different riddles by their state space. Different features of the game riddle such as solution paths, deadlines, and the complexity of the game riddle can be revealed using the proposed functions at the following. Figure 9 demonstrates the proposed function that we used for model-checking and analyses of the state space graph.

PathExtractor is the first function to extract simple movements of a merchant ship from the modeled riddle. It is a recursive function that can be recalled at most n times, where n is the number of nodes in the state space. It accepts as a first argument the node index at which the state space search begins (i.e., index 1 at the first call). In each recursion, it finds a list of reachable nodes in the first argument and adds to the second argument. In every recursion, the first parameter (state node) is tested to determine if it is eligible; if it is, the function will invoke the state node's output nodes. Code 11 illustrates the ML code of the PathExtractor.

Two constraints apply to an eligible node: 1) its index is not repeated in any of its ancestor nodes in the list of the second argument (i.e. a path from the initial state of the system to that node) 2) the position of the merchant ship in that node is not repeated in any of the predecessor nodes; however, an exception occurs when all of the index node children are repeated in the path, in which case it is eligible. This function is an application-specific extension of a BFS algorithm. To illustrate, consider the function called

```
fun successDeadmarkings() =
    let
        val harbor=List.hd (Mark.PAgent 'harbor 1 1)
        val harborlen=List.length harbor
        val L= ListDeadMarkings()
        val len = List.length( L )
        val n = ref (List.nth( L, 0))
        val dead=ref (Mark.PAgent'destination 1 (!n))
        val lendead= ref 0
        val j= ref 0
        val ls= ref []
    in
    while !j<len do(
        n := List. Hd;
        dead:= Mark.PAgent'destination 1 (!n);
        lendead:= List.length(!dead);
        if !lendead<>0 then ls:= !ls ^^[!n]
        else ls:= !ls;
        j:= !j+1);
    !ls
end
```

Code 8. successDeadmarkings function of MLcode

```
fun ShortestSenario(deadstates):INT =
let
        val index= ref (List.hd deadstates)
        val i= ref 0
        val min= ref (!index)
in
        while !i<(List.length deadstates) do(
                let
                        val item= ref (List.nth
                (deadstates,!i))
                in
                        index:= !item;
                        if !index< !min then min:=
                        !index
                        else min:= !min;
                        i:= !i+1
                end
        );
        !min
end;
```

Code 9. ShortestSenario to find the id of the shortest dead marking state

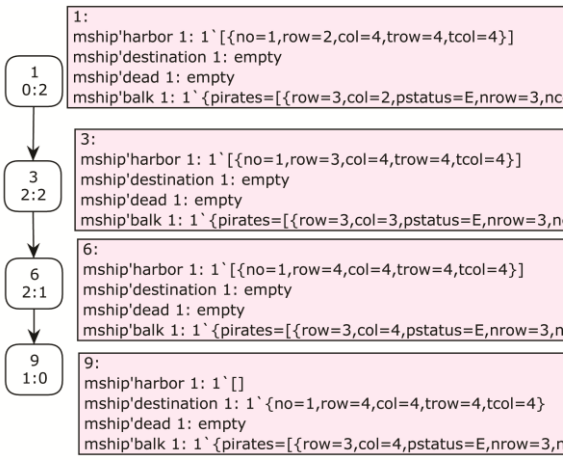

Figure. 7. State space of the CPN tools from the modeled game

```
fun DeadlockExistance()=
let
        val k=List.hd (Mark.mship'harbor 1 1)
        val lengh=List.length k
        val lst=UpperMultiSet
        (Mark.mship'destination 1)
        val lenlst=List.length lst;
in
        lenlst=lengh
end
```

Code 10, DeadlockExistance function determines whether or not a deadlock is possible in the modeled game puzzle

```
fun pathExtractor(s:INT, path:PATH):PATHS=
    let
        val outs= OutNodes(s)
        val souts=RemoveFromList (outs, path)
        val len= List.length souts
        val i = ref (len-1)
        val ls= ref []
        val q= ref []
        val q2= ref[]
        val n=ref 0
    in
    while !i>=0 do(
        n:= List.nth(souts,!i);
        if (isMemberOfPath(!n, path) andalso len>=
        nodeRepetition (!n, path)) then
                i:= !i -1
        else
            (q:= path^^ [!n];
            ls := !ls^^[!q]^^ pathExtractor(!n, !q);
            i := !i - 1)
        );
    !ls
    end
```

Code 11, pathExtractor to find eligible nodes from the parameter node

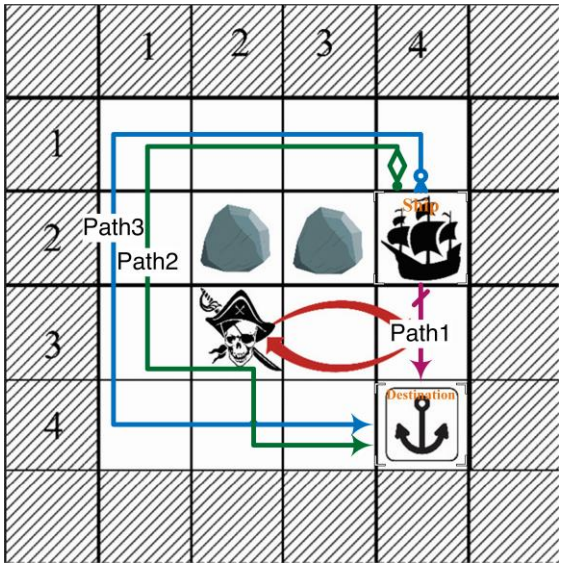

Figure. 8. Schematic description of extracted paths by function "PathExtractor" from the graph of Figure 2 and riddle of Figure 1.

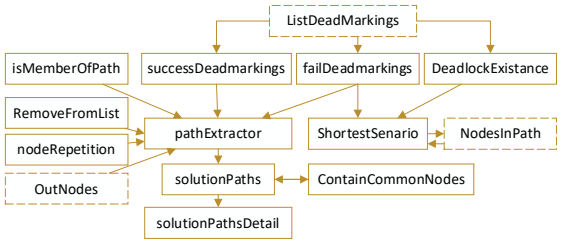

Figure. 9. function structure diagram of the state space analysis functions, rectangles are the proposed functions, rectangles with a dashed border are CPN tools inbuilt functions.

PathExtractor (4, [1, 2]) of the graph of state-space in the riddle (a). It results in a recursive call to PathExtractor (7, [1, 2, 4]) and PathExtractor (8, [1, 2, 4]). Some recursive calls to the PathExtractor function may result in the development of the cycle. To eliminate these loops, we implemented the RemoveFromList method. If any of the nodes available with one step from the first input parameter node of function calls present in the second parameter path, the recursive call to that node will be skipped by using the function RemoveFromList, preventing the formation of a cycle in the extracted paths.

PathExtractor can be used to extract all of the ship's simple movement paths (1, []). However, we require solution paths (game scenarios) from the start node of the state space to a dead marking which is the 'solution paths' function defined in Code 12. This function receives a list of all cycle-free paths utilizing calling PathExtractor and a list of dead markings of the state space, respectively. Then, it selects from the final list only solution paths, based on the dead markings it received, and eliminates those paths that do not terminate in any of the received deadmarkings in the second argument.

We have executed the model checking proposed functions for the riddle (a) using CPN tools which are demonstrated in Figure 10. As can be seen in the figure, the result of executing solution paths returns three solution paths as a list of lists like the schematic paths of Figure 8. The function 'solutionPathsDetail' in Figure 11 shows the location of a merchant ship step by step in the acquired solution paths.

We can alter the second argument of the solution path function to failDeadmarkings to identify failed solution paths (failure scenarios of the game's puzzle).

Figure 11 also shows these solution paths based on the state space graph nodes in the CPN tools. This figure was generated using CPN tools; however, the arc's text and solution path numbers have been added to make it more understandable. Arcs depict the firing of a transition in the CPN-Tools model. The phrase on the arc's edge denotes a transition and type of change in the system's states. The word preceding ":" in the statement denotes the ship ID. The word

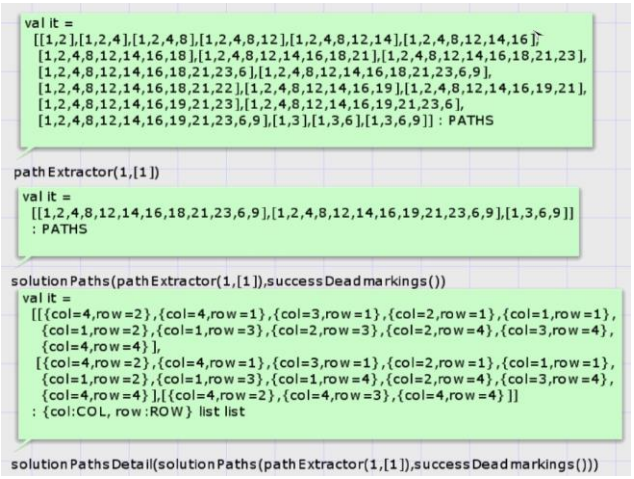

Figure. 10. executed proposed functions of the state space for the riddle (***a***) in the CPN tools

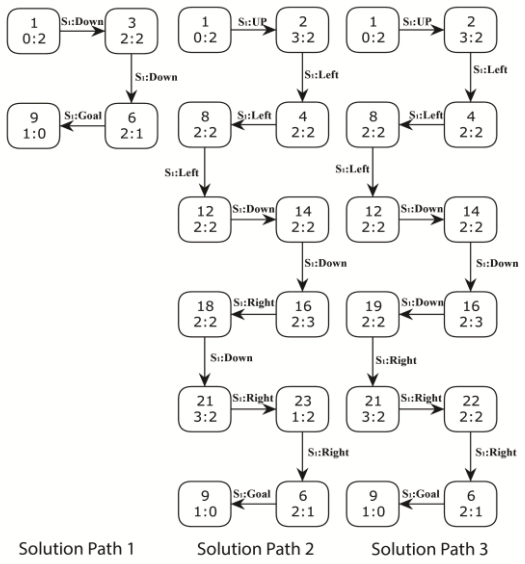

Figure. 11. extracted paths of top-level call of function PathExtractor from riddle (***a***)

```
fun SolutionPaths(ls:PATHS,dk:PATH):PATHS=
let
    val len= List.length ls
    val i = ref 0
    val n= ref []
    val st= ref []
    val bol= ref true
in
    while !i<len do(
        n := List.nth(ls, !i) ;
        if ContainCommonNodes(!n,dk)= true then
            st:= !st ^^ [!n]
        else
            st:= !st;
        i := !i + 1);
    !st
end
```

Code 12. Solution paths function to discover explicit paths to termination of the riddle

following the ":" is a pass marker consisting of Up, Down, Right, and Left, which implies traveling to the north, south, east, and west, respectively. The term Goal denotes the merchant ship's arrival at the target; for example, the S1: Up statement in Figure 11 denotes traveling north to the merchant ship with ID 1.

5.7. Complexity Metric of the Proposed Model

Another effective indicator for assessing the complexity of the puzzle game is the number of alternative solution paths that can lead to a successful game solution or a game over. The bigger the number of individual situations leading to a successful game solution, the easier the game.

This paper proposes a comparable metric, the ratio of successful scenarios to failed scenarios, as a measure of the puzzle game's complexity. Equ (1) calculates the total number of successful scenarios, while Equ (2) indicates failed scenarios that lead to the termination of a specific riddle. Equ (3) defines the riddle complexity of the game, denoted by ω.

$$SS = \sum_{i=1}^{n} [S_i = success\ solution\ paths] \quad (1)$$

$$FS = \sum_{i=1}^{n} [F_i = failed\ solution\ paths] \quad (2)$$

$$\omega = \frac{FS}{SS + FS} \quad (3)$$

Examination of the state-space graph for the riddle (β), extracts 542 possible scenarios for solving the modeled puzzle game. The number of solution paths leading to a pirate blockade of the ship before it arrives at its destination, as determined by an analysis of the state-space graph, is 380. The length of the shortest path from the graph of state space that leads to the successful end of the game riddle is the second metric of riddle complexity. A low number for this statistic indicates that the puzzle is simple.

Figure 12 depicts the average complexity of a game riddle determined from Equ (1) with various pirate numbers and their sight after 10 experiments in a 15x15 (15 rows and 15 columns) riddle with random pirate and ship locations. As shown, pirates' sight (destruction of power radios) and the number of pirates have a direct impact on the complexity of a game riddle. Another aspect that is clear from the results is that increasing pirate sight has a more complex effect on the game riddle than increasing pirate number.

Figure 13 shows the shortest steps (movement) of the merchant ship to succeed or fail in the modeled game. This result was acquired in a 10x10 area of the modeled game, using the average values of ten experiments with different pirate counts and pirate sight. The results reveal that raising the pirate sight and pirate number leads to fewer steps to failure and more steps to success in the game. For example, when the pirate sight and number are one, the shortest step to success is 14 and the shortest step to failure is 10.

6. Conclusion

The CPN using the ML programming language is one of the most powerful formal tools for system behavioral analysis. This paper introduced the modeling of the *Merchant Ship* puzzle game. It proposed two innovative techniques to address state-space explosion, a key limitation of CPNs that hinders their application to large and complex systems. By integrating advanced data structures, programming functions, and a model-checking technique implemented in ML, we developed a framework capable of effectively generating and analyzing state-space graphs. In addition, we propose two metrics for evaluating game complexity and player performance: the success-to-total scenario ratio and the minimal trajectory length. These metrics provide a quantitative evaluation framework for comparing riddles across similar puzzle games. We also detail the calculation techniques, their implementation, and the resulting efficiency

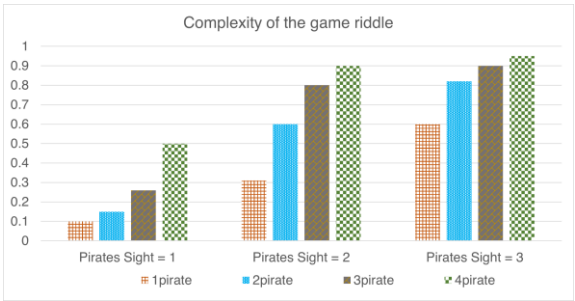

Figure 12: Complexity of the game riddle in a two-dimensional environment

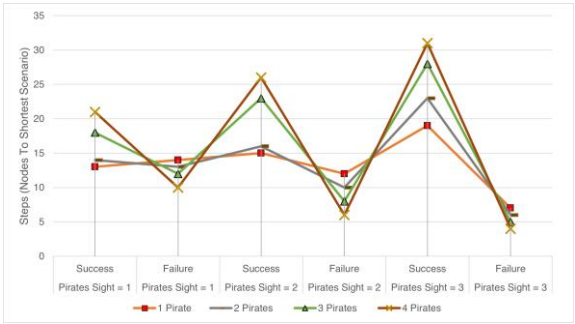

Figure 13: Shortest steps to success and failure termination of the game riddle

improvements, demonstrating that our methods significantly reduce the computational overhead to acceptable levels. In the future, timed and hierarchal colored Petri net and its validation methods through time and state space will be studied on time-restricted games.

Declarations

Funding

There has been no significant financial support for this work.

Authors' contributions

Ahmad Taghinezhad-Niar: Study design, acquisition of data, interpretation of results, drafting the result; Saeid Pashazadeh: Study design, revision of the manuscript.

Conflict of interest

There are no conflicts of interest associated with this publication.

References

[1] M. A. Piera, R. Buil, and E. Ginters, "State space analysis for model plausibility validation in multi-agent system simulation of urban policies," *J. Simul.*, vol. 10, no. 3, pp. 216–226, Aug. 2016, https://doi.org/10.1057/jos.2014.42

[2] A. Rehman, N. Akhtar, and O. H. Alhazmi, "Formal Modeling, Proving, and Model Checking of a Flood Warning, Monitoring, and Rescue System-of-Systems," *Sci. Program.*, vol. 2021, no. 1, p. 6685978, 2021, https://doi.org/10.1155/2021/6685978.

[3] J. Li, Z. Wang, L. Sun, and W. Wang, "Modeling and Analysis of Network Control System Based on Hierarchical Coloured Petri Net and Markov Chain," *Discret. Dyn. Nat. Soc.*, vol. 2021, no. 1, p. 9948855, 2021, https://doi.org/10.1155/2021/9948855.

[4] A. Fedorova, V. Beliautsou, and A. Zimmermann, "Colored Petri Net Modelling and Evaluation of Drone Inspection Methods for Distribution Networks," *Sensors*, vol. 22, no. 9, pp. 1–20, 2022, https://doi.org/10.3390/s22093418.

[5] M. Drakaki and P. Tzionas, "Modeling and performance evaluation of an agent-based warehouse dynamic resource allocation using Colored Petri Nets," *Int. J. Comput. Integr. Manuf.*, vol. 29, no. 7, pp. 736–753, Jul. 2016, https://doi.org/10.1080/0951192X.2015.1130239.

[6] H. Kaid, A. Al-Ahmari, Z. Li, and W. Ameen, "An Improved Synthesis Method Based on ILPP and Colored Petri Net for Liveness Enforcing Controller of Flexible Manufacturing Systems," *IEEE Access*, vol. 10, pp. 68570-68581, 2022, https://doi.org/10.1109/access.2022.3186287.

[7] S. Saeedvand, M. Abbaszadeh, and F. Ansaroudi, "Modelling causal consistency for distributed systems using hierarchical coloured petri net," *Indian J. Sci. Technol.*, vol. 8, no. 35, pp. 1-7, 2015, https://doi.org/10.17485/ijst/2015/v8i35/54980.

[8] A. Karatkevich, *Dynamic analysis of Petri net-based discrete systems*, vol. 404. Springer Science and Business Media, 2007.

[9] A. Taghinezhad-Niar and J. Taheri, "Security , Reliability , Cost , and Energy-aware Scheduling of Real-Time Workflows in Compute-Continuum Environments," *IEEE Trans. Cloud Comput.*, vol. 12, no. 3, pp. 954–965, 2024, https://doi.org/10.1109/TCC.2024.3426282.

[10] A. Taghinezhad-Niar, S. Pashazadeh, and J. Taheri, "Workflow scheduling of scientific workflows under simultaneous deadline and budget constraints," *Cluster Comput.*, vol. 24, no. 4, pp. 3449–3467, Dec. 2021, https://doi.org/10.1007/s10586-021-03314-3.

[11] A. Taghinezhad-niar, J. Taheri, and S. Member, "Reliability, Rental-Cost and Energy-Aware Multi-Workflow Scheduling on Multi-Cloud Systems," *IEEE Trans. Cloud Comput.*, vol. 11, no. 3, pp. 2681–2692, 2023, https://doi.org/10.1109/TCC.2022.3223869.

[12] A. Taghinezhad-Niar, S. Pashazadeh, and J. Taheri, "QoS-aware online scheduling of multiple workflows under task execution time uncertainty in clouds," *Cluster Comput.*, vol. 25, pp. 3767–3784, 2022, https://doi.org/10.1007/s10586-022-03600-8.

[13] A. Taghinezhad-Niar, S. Pashazadeh, and J. Taheri, "Energy-efficient workflow scheduling with budget-deadline constraints for cloud," *Computing*, vol. 104, no. 3, pp. 601–625, Mar. 2022, https://doi.org/10.1007/s00607-021-01030-9.

[14] P. Valizadeh and A. Taghinezhad-niar, "A Fault Tolerant Multi-Controller Framework for SDN DDoS Attacks Detection," *Int. J. Web Res.*, vol. 5, no. 1, pp. 1–7, 2022, https://doi.org/10.22133/ijwr.2022.345927.1119.

[15] P. Valizadeh and A. Taghinezhad-Niar, "DDoS Attacks Detection in Multi-Controller Based Software Defined Network," in *2022 8th International Conference on Web Research (ICWR)*, Tehran: IEEE, May 2022, pp. 34–39. https://doi.org/10.1109/ICWR54782.2022.9786246.

[16] "CPN Tools Homepage," CPN Tools support. Accessed: Jul. 12, 2022. [Online]. Available: http://cpntools.org/start

[17] A. Taghinezhad-Niar, "A Client-Centric Consistency Model for Distributed Data Stores using Colored Petri Nets," in *2024 10th International Conference on Web Research (ICWR)*, 2024, pp. 309–314. https://doi.org/10.1109/ICWR61162.2024.10533365

[18] Z. Zhao, S. Liu, M. Zhou, D. You, and X. Guo, "Heuristic Scheduling of Batch Production Processes Based on Petri Nets and Iterated Greedy Algorithms," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 1, pp. 251–261, 2022, https://doi.org/10.1109/TASE.2020.3027532.

[19] D. Lages, E. Borba, E. Tavares, A. Balieiro, and E. Souza, "A CPN-based model for assessing energy consumption of IoT networks," *J. Supercomput.*, vol. 79, no. 12, pp. 12978–13000, 2023, https://doi.org/10.1007/s11227-023-05185-4.

[20] H. Kaid, A. Al-Ahmari, and K. N. Alqahtani, "Fault Detection, Diagnostics, and Treatment in Automated Manufacturing Systems Using Internet of Things and Colored Petri Nets," *Machines*, vol. 11, no. 2, p. 173, 2023, https://doi.org/10.3390/machines11020173.

[21] A. Shahidinejad, M. Ghobaei-Arani, and L. Esmaeili, "An elastic controller using Colored Petri Nets in cloud computing environment," *Cluster Comput.*, vol. 23, no. 2, pp. 1045–1071, 2020, https://doi.org/10.1007/s10586-019-02972-8

[22] V. P. Mishra, B. Shukla, and A. Bansal, "Analysis of alarms to prevent the organizations network in real-time using process mining approach," *Cluster Comput.*, vol. 22, no. s3, pp. 7023–7030, 2019, https://doi.org/10.1007/s10586-018-2064-8.

[23] Š. Kuchař and I. Vondrák, "Automatic allocation of resources in software process simulations using their capability and productivity," *J. Simul.*, vol. 10, no. 3, pp. 1–10, 2015, https://doi.org/10.1057/jos.2015.8.

[24] A. J. Cunha De Aguiar, E. Villani, and F. Junqueira, "Coloured Petri nets and graphical simulation for the validation of a robotic cell in aircraft industry," *Robot. Comput. Integr. Manuf.*, vol. 27, no. 5, pp. 929–941, 2011, https://doi.org/10.1016/j.rcim.2011.03.005.

[25] P. A. Hsiung and C.H . Gau, "Formal Synthesis of Real-

Time Embedded Software by Time-Memory Scheduling of Colored Time Petri Nets," *Electron. Notes Theor. Comput. Sci.*, vol. 65, no. 6, pp. 140–159, Jun. 2002, https://doi.org/10.1016/S1571-0661(04)80474-2.

[26] Weili Yao and Xudong He, "Mapping Petri nets to parallel programs in CC++," in *Proceedings of 20th International Computer Software and Applications Conference: COMPSAC '96*, Weili: IEEE Comput. Soc. Press, 1996, pp. 70–75. https://doi.org/10.1109/CMPSAC.1996.542428.

[27] A. Taghinezhad and S. Pashazadeh, "Modelling and analysis of the monotonic read consistent distributed system using coloured Petri net," *2016 8th Int. Conf. Inf. Knowl. Technol. IKT 2016*, Hamedan, Iran, 2016, pp. 85–90, https://doi.org/10.1109/IKT.2016.7777791.

[28] W. M. P. Van Der Aalst, "The Application Of Petri Nets To Workflow Management," *J. Circuits, Syst. Comput.*, vol. 08, no. 01, pp. 21–66, Feb. 1998, https://doi.org/10.1142/S0218126698000043.

[29] W. Yu, J. Feng, L. Liu, X. Zhai, and Y. Cheng, "Enhancing security in e-business processes: Utilizing dynamic slicing of Colored Petri Nets for logical vulnerability detection," *Futur. Gener. Comput. Syst.*, vol. 158, pp. 210–218, 2024, https://doi.org/10.1016/j.future.2024.04.035.

[30] M. Drakaki and P. Tzionas, "A colored petri net-based modeling method for supply chain inventory management," *Simulation*, vol. 98, no. 3, pp. 257–271, 2022, https://doi.org/10.1177/00375497211038755.

[31] L. M. Kristensen, "State space methods for coloured petri nets," *DAIMI Rep. Ser.*, vol. 29, no. 546, 2000, https://doi.org/10.7146/dpb.v29i546.7080.

[32] A. Taghinezhad-Niar, T. Javadzadeh, and L. Farzinvash, "Modeling of resource monitoring in federated cloud using Colored Petri Net," in *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*, Tehran, Iran, Dec. 2017, pp. 0577–0582. https://doi.org/10.1109/KBEI.2017.8324866.

[33] C. A. Petri, "Kommunikation mit Automaten," Technische Hochschule Darmstadt, Darmstadt, Germany, 1962. Accessed: Oct. 18, 2017. [Online]. http://edoc.sub.uni-hamburg.de/informatik/volltexte/2011/160/.

[34] K. Jensen, "Colored Petri Nets : A Graphical Language," *Commun. ACM*, vol. 58, no. 6, pp. 61–70, 2015, https://doi.org/10.1145/2663340.

Ahmad Taghinezhad-Niar received the MSc and PhD degrees in computer engineering from the University of Tabriz, in 2017 and 2021, respectively. He is an Assistant Professor with the Department of Computer Engineering, University of Tabriz, Iran. His research interests lie in distributed systems, cloud computing, scheduling algorithms, and formal methods. Additionally, he actively contributes to the academic community by serving as a reviewer for esteemed journals.

Saeid Pashazadeh received a B.Sc. degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 1995, and the M.Sc. and Ph.D. degrees in computer engineering from the Iran University of Science and Technology, Tehran, in 1998 and 2010, respectively. He is currently a Full Professor with the Department of Information Technology, Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran. His research interests include wireless sensor networks, target tracking, formal methods, distributed systems, and stochastic systems.