




State space scalability to enable smart ships with statistical physics and multi-agent-based reinforcement learning

Alexander Manohar * 

University of Michigan, Department of Naval Architecture and Marine Engineering, 2600 Draper Dr, Ann Arbor, United States, alexmano@umich.edu

David J. Singer 

University of Michigan, Department of Naval Architecture and Marine Engineering, 2600 Draper Dr, Ann Arbor, United States, djsinger@umich.edu

Submitted: 26.03.2023

Accepted: 03.07.2023

Published: 31.12.2023



* Corresponding Author

Abstract:

The global marine industry is transitioning to smart ships where navigation, maintenance, and operations are done autonomously. Integrating autonomy into already complex ships presents many challenges, including identifying faults and taking corrective actions. These actions are key components in a Self-Adaptive Health Monitoring (SAHM) system which aims to maintain ship operations. The challenge lies in the failure state space's extraordinary size which current methods aren't capable of dealing with. Diagnosis has been achieved for smaller scale systems such as NASA deep space probes, but the complexity of a probe is equivalent to a ship's single small sub-system. The authors combine recent advancements in statistical physics and multi-agent-based reinforcement learning to address the scale issue and enable crewless vessels. Statistical physics works to extract information about objects through tensor networks, combining physical and logical representations of objects. By combining a sequence of contractions, an ensemble of data about the physical system can be constructed quickly. To demonstrate the proposed method, the algorithm is applied to a modified version of the N-Queens problem which contains operational decision making, geometrical constraints, and a scalable problem. The authors then apply an already proven method to the modified version of the N-Queens problem and compare the results. The tensor network enables agents to handle state space explosion by decoupling the system's complexity from decision making.

Keywords: *Diagnosis, Fault detection, Self-adaptive health monitoring, Tensor networks*

© 2023 Published by peer-reviewed open access scientific journal, C&I at DergiPark (dergipark.org.tr/ci)

Cite this paper as: Manohar, A., & Singer, D.J., State space scalability to enable smart ships with statistical physics and multi-agent-based reinforcement learning, *Computers and Informatics*, 2023, 3(2), 67-80

1. INTRODUCTION

Within recent years, the global marine industry has begun transitioning to smart ships [1], [2]. Smart ships refer to ships where systems and functionalities, which were previously done manually, are replaced with autonomous counterparts [3]. The ultimate goal is to incorporate autonomy in all aspects of the functionality of a ship, including navigation, maintenance, and operations, so that there is a transition toward a crewless vessel [4]. However, a variety of challenges arise resulting from the interdependencies between existing ship systems and their integration with state-of-the-art autonomous systems [5]. For these systems to function without direct human intervention, these autonomous systems must be capable of identifying when faults occur and then taking corresponding corrective actions to remedy those faults [6]. These are key components in a Self-Adaptive Health Monitoring (SAHM) system which aims to maintain ship operation and functionality autonomously [7], [8]. SAHM systems consist of two major components: prognosis and diagnosis. In this context, prognosis can be defined as determining if a fault in a system has occurred [9]. Then, given a fault, perform diagnosis to determine the cause of the fault and take a corrective action [10]. Diagnosis can be generalized by a three step-process [11]. First, examine the system to develop hypotheses for the fault's existence. Second, determine if the hypothesis is correct. Finally, take a nondestructive corrective action to either remedy the fault, repair the system, or determine a strategy for graceful degradation. On crewed vessels, this process is done by a human, but on crewless vessels, there is no one on board to perform these measures. Therefore, some other agent needs to take control of the diagnostic process. On a crewless ship, this agent can be a computer agent that interacts with and can modify the various interconnected systems. SAHM systems work autonomously to extend the effective operation lifespan of a system in a process called graceful degradation [12]. The DARPA No Manning Required Ship (NOMARS) project presents a concept of a crewless vessel that utilizes a SAHM system to maintain ship operations while away from port for extended periods of time [13]. Regardless of the SAHM strategy used, the major challenge in designing SAHM systems for smart ships lies in the size of the state space of possible failures [14]. With each additional system, the number of potential faults scales exponentially, with the total number of possible failures being on the scale of 2500, which is more than the number of atoms in the universe [14]. Current methods are not capable of handling a state space of this scale [15], [16]. Therefore, an obvious research opportunity lies in figuring out how to deal with an extraordinarily large state space. Diagnosis has been achieved in the past in NASA's deep space probes [17]. They created a SAHM system that was capable of a graceful degradation of the space probe's operations. However, a space probe is equivalent to a single, small sub-system in a modern ship in terms of complexity. By combining many sub-systems into systems and then further into a full ship, the complexity of the ship increases exponentially [18]. The increased complexity of ships and the subsequent integration of autonomous systems results in unknown and complex faults, also known as emergent faults. A SAHM system must be capable of handling emergent faults. However, the designer will have difficulty using current and prior methods since those methods reconcile with the state space's size. Therefore, the authors are looking into a new method to aid the designer in handling the state space to enable crewless vessels.

The research presented attempts to resolve the state space scaling issue by combining recent advancements in statistical physics with multi-agent-based reinforcement learning in order to enable crewless vessels and smart ships. Statistical physics uses tensor networks [19] to extract information about objects from systems [20]. Tensor networks form one of the essential building blocks of statistical physics and drive much of its research. In recent work by Klishin et al., tensor networks are used to bridge the gap between the physical and logical constraints on a system [21]. By contracting a constructed tensor network, information about the system is extracted that respects both the geometric constraints and the logical architecture of a system. As a result, by performing a series of contractions with slight modifications to the environment before each contraction, an ensemble of data about the system can be rapidly created independent of the scale of the system. Autonomous agents can use the tensor

network to both query actions to see their effect before they are taken and to build an ensemble of experiences from which they can learn. Multiagent based reinforcement learning consists of several agents working cooperatively or competitively within an environment with a task in mind [22]. For each action the agent takes, they are given a reward based on how well the action taken moves the agent towards its goal. As simulations take place, the agents gain knowledge and improve their performance. The proposed method combines both statistical physics and multiagent reinforcement learning to learn the ship's environment. The data gained and the relationships learned can aid the designer by discovering emergent relationships and faults between various systems and components.

This paper is presented in the following sections. In section 2, background information and related work is presented and discussed. Section 3 discusses the proposed method and section 4 presents the case study. Section 5 discusses the results and finally, section 6 discusses future work.

2. BACKGROUND

The following subsections provide the necessary foundation required to describe the proposed method. Specifically, this section covers diagnosis, state spaces, statistical physics, and reinforcement learning.

2.1. Fault Diagnosis

Faults can be viewed as states of a system. The state of a system consists of the values of variables that persist temporally from time step to time step [23]. The corresponding state space of a system contains all possible states of the system [24]. It follows that a failure state is a state in which unique values of components indicate the failure of a part of the system. When a system can be modeled by states, failure states represent possible system faults. Faults are broadly classified into 4 main categories: known knowns, known unknowns, unknown knowns, and unknown unknowns [25]. Known knowns are faults that are known, understood, and can be planned around. Typically, these faults are classified using methods such as Failure Mode and Effects Analysis (FMEA), where a system is thoroughly examined and known faults with components and their interdependencies are discretely expressed [26]. Unknown knowns are faults that are known, but the consequence and occurrence are not understood. Known unknowns are faults that haven't been discovered, but the risk of them occurring is known. Information about known unknown and unknown known faults is typically found through modeling and machine learning techniques, thus allowing the designer to plan around them. Some examples of this include the use of neural networks in power systems [27] and wind turbines [28]. However, unknown unknowns, also referred to as emergent faults, are the most difficult type of fault to work with since the designer is neither aware, nor understands the reason for these faults. Their respective failure states haven't been observed. Emergent faults are difficult to find but their states can be found through enough simulation [29], [30]. Faults can be further broken down into simple and complex faults, where simple faults have a single hypothesis of the fault's cause and complex faults have multiple hypotheses of the fault's cause. Complex faults are typically harder to identify as the fault's failure state has multiple connected states that each could be the state of the system prior to the fault. If the fault has been observed (known), then diagnosis becomes a case of evaluating the hypotheses. However, if the complex fault has not been observed (unknown), then it becomes much more difficult to develop hypotheses for an unobserved failure state. It is also not understood how to take a non-destructive corrective action. Compounding upon this difficulty is the fact that modern ships' failure state space is on the scale of 2500 [14] and the time required to identify both simple and complex emergent fault failure states is prohibitively long.

Given that a fault has occurred, fault diagnosis is a 3 step process [11]. First, hypotheses about the cause of the fault are developed. Then, the hypotheses are verified until the correct reason is found. Finally, take a corrective action to remedy the fault. Typically, a worker performs all 3 steps manually, but

advances in technology have allowed for parts of the process to be automated [31]–[33]. Modern fault diagnosis has two main approaches. The first is to use prior knowledge to identify known failure states of the system. This typically takes the form of Failure Mode and Effects Analysis (FMEA) [26] and Fault Tree Analysis (FTA) [34]. The second approach is to use simulation [35] and basic machine learning methods such as neural networks and Support Vector Machines (SVMs) to determine the causes of known failure states [33], [36]. In both approaches, the first two steps of the diagnostic process are replaced, but a worker is still required to perform the corrective action. On a crewless vessel, since no one is on board, the corrective action must be taken by a computer.

NASA encountered a similar problem in the 1990s on its deep space probes. They designed a SAHM method to maintain the operation of the probe and to allow the systems to gracefully degrade [17]. However, since the scale of products like the NASA probes are equivalent to a single small ship subsystem, their techniques fail to deal with the large scale of ships.

2.2. Statistical Physics

Statistical physics uses statistical methods to describe physical problems in areas such as thermodynamics, superfluidity, and quantum statistics [37]. Two of the main information structures used are partition functions and tensor networks [38].

A partition function is encoded with information about a system. Given an objective function $O(\alpha)$, where α represents the state of the system, and coefficients λ_i representing design pressure, the partition function Z is written as in (1). While knowing the partition function is useful, finding its value can be difficult and hard to work with. However, by taking its derivative, statistical relationships and information about the system it represents can be extracted. An example of this is that statistical averages are encoded in the derivatives of the partition function when taken with respect to λ_i . As a result, partition functions can provide context and information about a system in a relatively straightforward way.

$$Z = \sum_{\alpha} e^{-\sum_i \lambda_i O_i(\alpha)} \quad (1)$$

Tensor networks are another key information structure in statistical physics. Tensor networks are made up of a network of tensors with contractions between them. A tensor is an algebraic object that is used to represent the relationships between other objects. In physics, these relationships are related to physical relationships, such as stress forces on an object. In a mathematical sense, a tensor is a “series of numbers labeled by N indexes, with N called the order of the tensor” [39]. For example, a scalar is a 0th order tensor, a vector is a 1st order tensor, and a matrix is a 2nd order tensor. A contraction between two tensors is the summation over a shared index. For example, given two second-order tensors A_{ij} and B_{ki} , contraction between them results in a new tensor:

$$A_{ij}B_{ki} = A_{1j}B_{k1} + A_{2j}B_{k2} + \dots + A_{1n}B_{kn} = C_{jk} \quad (2)$$

Note that in (2), there are left over indices that have not been contracted over. These are known as free indices. From these operations, tensor networks are a representation of a series of contractions between a number of tensors. In a tensor network, the order of contractions does not affect the output value of the contraction. However, choosing an efficient contraction ordering is crucial to reducing the computation time and space required to contract the network, both of which can increase exponentially depending on the chosen contraction order [40], [41]. Figure 1 shows an example of a simple tensor network consisting of 5 tensors (nodes) and 5 contractions (edges) between the tensors.

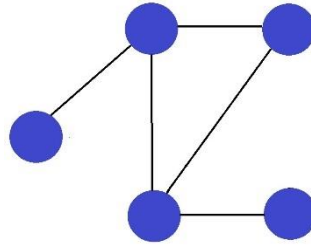


Figure 1: Tensor Network of 5 tensors and 5 contractions. Created by the authors.

In Klishin et al., they integrate the logical architecture of a system with the corresponding physical architecture into a single tensor network that, when contracted, evaluates the partition function of the system [21]. This partition function is constructed as in (1) with design objective $O(\alpha)$. The effective design objective function \mathcal{O}_{eff} is written in (3). The physical architecture dictates where every unit $\{\vec{x}_i\}$ can be placed and the corresponding function $f(\vec{x}_i, \vec{x}_j; \theta)$ determines the entries of the coupling tensors within the tensor network. Additional parameters θ can be passed into the function depending on the situation to affect the weight of any aspect of the effective design objective function.

$$\lambda \mathcal{O}_{eff}(\vec{x}) = \sum_{ij} A_{ij} f(\vec{x}_i, \vec{x}_j; \theta) \quad (3)$$

The partition function can then be factorized into (4). From this form, one can construct a tensor network that represents the combined logical and physical system.

$$\mathcal{Z}(\theta) = \sum_{\{\vec{x}\}} \prod_{i < j} e^{-f(\vec{x}_i, \vec{x}_j; \theta)} \quad (4)$$

The tensor network can then be further modified with 3 types of actions: attaching an external leg, attaching an anchor, and modifying a tensor coupling. Appending an external leg onto a tensor causes the corresponding object not to disappear when the network is contracted. The resulting contraction when an external leg is attached onto the tensor corresponding to the object \vec{x}_i results in an unnormalized probability distribution over all available physical locations for that object. Attaching an anchor represents decisions that have already been made about the location of a specific object, thus fixing the location of \vec{x}_i . Other objects cannot be in the location where another object has been anchored. The resulting contraction of the tensor network is then described as a conditional contraction on the fixed location of \vec{x}_i . The final type of action, the modified coupling, modifies the coupling between two objects and then traces its effect back into the partition function. An example of a modified coupling is the suppression of weights corresponding to a specific region to create a zone where no objects can be located. One can create an ensemble of different scenarios about the combined logical and physical system by adding and removing external legs, modifying anchors, and modifying tensor couplings. The resulting contractions of the network then provide an ensemble of usable data that can provide context and information to the overall system. Contraction time and space requirements are dictated by the size of the number of edges that need to be contracted over and their density [40]. The difficulty of contraction stems from the selection of an edge contraction order. Hyper optimized contraction methods result in an exponential run time of $O(2^E)$ where E is the number of edges [42]. However, by limiting the logical network density, the optimal contraction path becomes obvious to greedy contraction algorithms [39]. Therefore, the time and space requirements of tensor network contraction can be decoupled from the complexity of the environment.

2.3. Reinforcement Learning

Reinforcement learning involves an agent taking actions in an environment to maximize the cumulative rewards it receives [43]. One of the main algorithms utilized in reinforcement learning is Q-learning. Q-

learning is a model-free form of learning in which the agent does not need to build a map of the environment [44]. The goal of Q-learning is to find an optimal policy $\pi: S \rightarrow A$ that maps the state space S to the action space A . The algorithm finds the optimal policy by building a function $Q^*: S \times A \rightarrow \mathcal{R}$ that maps the state space S and the action space A to a reward. The policy function π^* maximizes the reward given as shown in (5).

$$\pi^*(s) = \arg \max_a Q^*(s, a) \quad (5)$$

Instead of building a Q table directly, due to the large state space, a neural network is used instead. This variation is called deep Q-learning (DQN) [45]. Neural networks are used to replace the Q function since they are universal function approximators [46]. The Q function is updated and an optimal policy is found by running simulations in which the agent takes actions and receives rewards. At each time step, the agent takes the following steps: observes its current state s_t , selects and performs action a_t , observes the following state s_{t+1} , receives a reward for its action r_t , and updates its Q function.

Standard Q-learning and DQN assume that only a single agent is present in the environment, but what if there were more than one agent? The presence of multiple agents is referred to as reinforcement learning with self-play or multiagent reinforcement learning [47], [48]. Within this setting, there are two classifications of games: competitive and cooperative. For this research, the authors look at cooperative games where agents share a common reward function [22]. Using this framework, the Q function and the reward function are identical for all agents. As a result, one can use the DQN and Q-learning algorithms described above. The degree of centralization of information varies depending on the design of the system. In a centralized setting, there is a single controller for all agents and all agents share observations with one another via this central controller. In a partially decentralized environment, there is no central controller, and agents make decisions based on the information they have. However, some agents may be connected to other agents through a network structure and share information. This differs from a fully decentralized structure where all agents act independently of each other and cannot share information. The varying levels of centralization present flexibility in agent design by allowing the agents to replicate real-world information conditions and constraints.

3. METHODOLOGY

The authors propose a new method to represent, model, and explore the ship system state space by combining multi-agent reinforcement with statistical physics. When designing a crewless vessel, integrating an already complex platform with further complex autonomous systems leads to additional difficulties in design. Since the size of the state space of possible failures is extremely large, it is difficult to explore the state space and find emergent faults resulting from the interdependencies between the systems. Therefore, the proposed method aims to explore the state space and find complex non-obvious interdependencies that exist between systems to aid the designer in a reasonable amount of time.

The method first consists of constructing the agents that will explore the environment. The agents have limited knowledge about the environment since if the agents are omniscient, then there is no problem as the environment is fully known. However, since this is a cooperative task between agents, they have a centralized information structure where the observations of each agent are shared with all other agents. If one agent learns something about the environment, all others will become aware of it as well. Each agent is assigned a specific part of the ship's environment to explore such as a component, a sub-system, or an entire system. At each time step, the agent takes the following steps: it observes the environment, queries a number of possible actions, selects the best action, and observes the new state. Note that this is similar to the Q-learning algorithm outlined in section II-C with the only difference being the process of selection of an action. In the original Q-learning algorithm, the agent selects an

action given its current state. However, in the proposed method, the agent first queries a number of actions, sees their potential results, and then selects the best action among those queried. In addition, if none of the actions are beneficial actions, the agent may decide that the best course of action is to take another action that has not been queried. The agent has a limited number of queries using the tensor network. If the agent does not have this limit, then the agent can exhaustively search the environment. This is equivalent to performing an exhaustive search of the state space and as noted in section II-A, is not feasible due to time constraints. Algorithm 1 details a high-level version of the proposed method. Algorithm 1 was created by the authors.

To query actions into the environment, the agent utilizes tensor network contractions. The agent anchors its position in the environment, the positions of other agents that it has knowledge of, and then contracts the network. The contraction evaluates the partition function encoded into the network. The partition function includes information that the agents learn through exploration. As time progresses in the simulation, the knowledge gained from previous actions helps to inform the agents about the consequences of future actions.

Algorithm 2. Agent Environment Exploration

```
Require: Environment Require:  $k, T \in \mathbb{Z}^+$   
Require:  $n$  Agents  
  Initialize Environment  
  Initialize Agents in Environment  
   $t \leftarrow 0$   
  while  $t < T$  do  
    for agent in Agents do  
      agent observes its current state  
      Query  $k$  actions by agent into Environment  
      Evaluate the ensemble of action  
      Select best action from ensemble of  $k$  actions  
    end for  
     $t \leftarrow t + 1$   
  end while
```

The authors use the framework as outlined in section 2.2 to construct the tensor network. The tensor network requires that certain components be set up prior to simulation to save computational time later. The first component to consider is the physical geometry of the environment. The environment must be organized into a grid-like object consisting of 1's and 0's where 1's indicate valid agent positions and 0's indicate invalid agent positions. The valid agent positions in the environment represent the possible locations of the agents within their respective environments. For example, if the agent represents a valve, the positions might represent the degree of openness of the valve. The second component of the tensor network is the partition function. The partition function is evaluated by contracting the tensor network. The function is defined in the network using the relationship between individual spaces within the physical geometry of the environment. Let \vec{x}_i and \vec{x}_j be the positions of two agents i, j within the environment. Then $f(\vec{x}_i, \vec{x}_j; \theta)$, as used in (3), gives the relationship between the two spaces, with θ representing any additional parameters needed. At this point, the tensor network has enough information so that during the simulation, the computational time to construct the network is limited. However, for each query, the network must be reconstructed. For each query, the agent constructs the logical architecture using the available information. Call the agent who makes the query agent i . The authors construct a logical network in which agent i and another agent j have an edge between them if agent i has encountered agent j . Then, the known positions of the encountered agents are anchored. Now, all the information needed to construct the network has been acquired: the geometry, the partition function, the logical network, and the anchors. The agent creates the network and then contracts it. For each subsequent action query at that time step, the agent updates its own position anchor and repeats the process to construct the logical network and anchors. In a real environment, all systems operate simultaneously and that is reflected in the simulation. Each agent queries and takes an action at the same time step. Therefore, agents, in addition to considering the other encountered agents' current

positions, need to consider the other agents' next moves as well. This adds an additional layer of depth to the simulation as the agents make decisions and explore the environment.

The agent improves its decision-making capabilities by learning from its past experiences. Specifically, if the agent has a limited number of queries, then the agent must learn the best way to use them to maximize its information. Each type of agent, depending on the system the agent corresponds to, has a neural network from which it draws actions based on its current state. To learn the network, the agents employ DQN learning as described in section 2.3. By using this approach, the agent learns how use their limited number of queries. Initially, the DQN learning algorithm focuses on exploration by selecting random actions. Over time, as the network sees more data and updates, less actions are randomly sampled, and more are drawn directly from the network. At the end of the simulation, nearly all samples are sampled directly from the network.

As the simulation progresses and the environment is explored, additional information about what the agents encounter is gathered and the parameters θ of the partition function are updated. These data help inform agents of the benefits and drawbacks of their queried actions through the tensor network. In addition, the agent learns from its experiences and updates its neural network through DQN. As a result, when the agent selects what it believes to be the best action, this action will be better informed than in the previous time steps. This leads to better choices, resulting in a better understood environment. How the agent selects the best action from the ensemble of queried actions depends on what system or component the agent is representing. However, each agent uses a combination of the result of the tensor network contraction, what the agent observes during its queries, and its own internal goals. By the end of the simulation, the relationships between various systems and agents are found through exploration of the environment, including non-obvious complex relationships between systems. Information about system interdependencies can be used by the designer to have a better understanding of the relationships that exist within the ship and enable crewless vessels.

4. CASE STUDY

In the following section, a modified version of the N-Queens problem is used to demonstrate the method proposed in section III. The standard N-Queens problem is a classical constraint satisfaction problem where given an $N \times N$ board, place up to N queens such that no queens conflict with each other [49]. The N-Queens problem is modified for this demonstration where queens will only have vision one square out. An example of this vision is in Figure 2. While it is clear that the yellow queen conflicts with both the blue and red queens, the red and yellow queens are not aware of each other since their vision is limited to one space away. This limitation of the queen's knowledge reflects the limited knowledge of agents as stipulated in the proposed method above since agents are not omniscient.

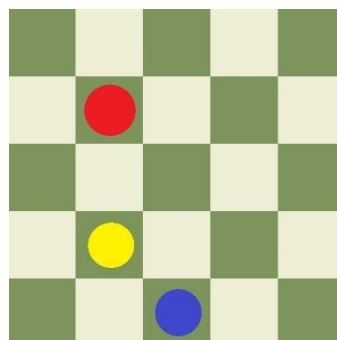


Figure 2: Example of the modifications to the queens' vision. Created by the authors.

For the demonstration, a single $n \times n$ board with n queens was used. The board was converted into the physical geometric object required for the tensor network by creating an $n \times n$ grid of 1's. Next, the partition function to be encoded into the tensor network was set up. The effective design objective function, which is used in the partition function, was defined as (3). The effective design objective function utilized the function $f(\vec{x}_i, \vec{x}_j; T)$ which dictates the relationship between two spaces and is defined in (6). The function $check_conflict(\vec{x}_i, \vec{x}_j)$ returns true if a conflict has been registered between the spaces occupied by \vec{x}_i and \vec{x}_j and false otherwise. The parameter T denotes the degree of impact conflicts have. When T is larger, conflicts have greater weight. Initially, the partition function starts empty without any information about the board. Conflicts are registered in the partition function when agents encounter one another during action queries. The simulation algorithm is found below in Algorithm 2. Algorithm 2 was created by the authors.

$$f(\vec{x}_i, \vec{x}_j; T) = T \times check_conflict(\vec{x}_i, \vec{x}_j) \quad (6)$$

Algorithm 2. Modified N-Queens Problem

Require: $n, k, T \geq 0$
Initialize Board(n, n)
 n queens $\leftarrow n$
Place n queens on Board
 $t \leftarrow 0$ **while** $t < T$ **do**
 for queen in n queens **do**
 queen observes its current state Query k
 moves by queen on Board
 Evaluate the ensemble of moves
 Move queen to best of k moves
 end for $t \leftarrow t + 1$ **end while**

Queen agents have several characteristics. There are 17 possible actions they can take: move one space in each direction, move more than one space in each direction, and remain stationary. How far a queen moves when it elects to move more than one space is determined by an internal parameter called $query_range \in [0,1]$. The larger $query_range$ is, the farther the queen moves to perform the query. Once the queen has made its queries and has an ensemble of possible actions, it uses a priority list to determine the best action to take. The best is a move that is one space away and results in no conflicts with other queens. The second best is a move more than one space away with no known conflicts. Third is a move more than a space away that results in a conflict. Finally, the worst is a move one space away that results in a conflict. However, if the queen finds that all of its moves cause visible conflicts, it may instead choose to take another action that has not been queried. The chance that the queen chooses to take this alternative action is based on a parameter called $move_risk \in [0,1]$ where the closer the parameter is to 1, the more likely it is to pick another action. The reward that the queen receives is given by the number of known conflicts of the queen in its new location. If there are no conflicts, the reward is 0. With each additional conflict, the reward decreases by 1. After the queen has moved, its encounters and conflicts with other queens are logged into memory to update the partition function. The metric for success in a game is by how much the number of conflicts between queens is minimized. At the end of a game, the ideal result is that there are no conflicts.

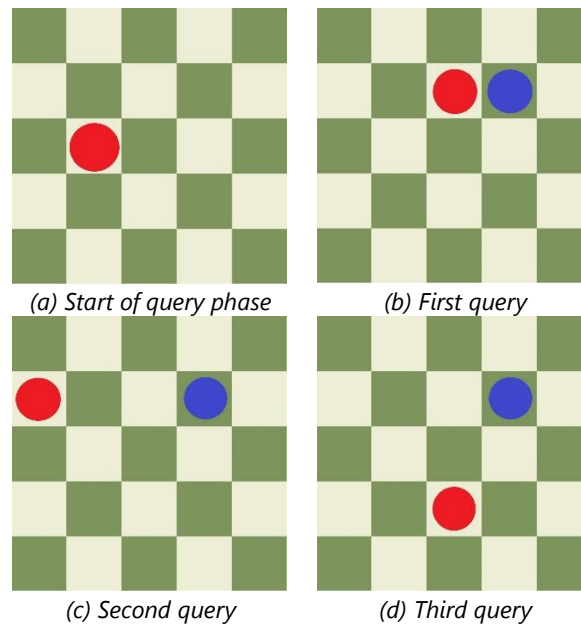


Fig. 3: An example of a queen's query phase. Created by the authors.

The modified version of the N-Queens problem has the same characteristics as the diagnostic problem in ship design. First, there is the operational decision making. Components within a ship continuously update and make decisions based on their known information and current objective. These decisions will impact other components throughout the ship directly and indirectly. Similarly, since each queen is moving at each time step, each of their actions may impact all other queens. There are also geometric constraints on a ship. For example, certain systems cannot be located next to other systems due to concerns such as excess heat from one system impacting another. By the nature of the modified N-Queens problem, there are also geometric constraints. Queens cannot conflict with each other and must maintain adequate spacing. When the queens move, these constraints must be taken into account. Next, designing a ship is a scalable problem [14]. As more components and systems are added to the ship design, so does the challenge of integrating them into a single platform. Similarly, the modified N-Queens problem can be scaled in difficulty by increasing the size of the board and increasing the number of queens. The state space of the game for an $N \times N$ board is given by $\binom{N \times N}{N}$. As N increases, the state space grows exponentially large, similar to ships. Finally, the modified N-Queens problem replicates the limited information that is present inside a ship. Since each component does not know the state of every other component, then each component needs to use its best judgement in decisions in a decentralized approach. The queens also have limited knowledge since they can only see one space away from themselves. They have to rely on their limited information to make a decision.

Figure 3 is an example of a query phase for the red queen. The red queen finds the blue queen on its first query in (b), but sees that remaining next to it will lead to a conflict. The next query (c) sees the red queen find a space where no other queens are, but still conflicts with the blue queen. Finally, in (d), the queen finds a location where there are no other queens and does not conflict with the blue queen. According to the priority list, the queen will select the final query as its action.

5. RESULTS AND DISCUSSION

Next, the current testing results are presented. The system was tested on 4x4, 6x6, 8x8, and 10x10 boards. The agents share a single neural network since they all occupy the same board and have the same shared goal. The neural network consists of the following layers, where the final layer is a mapping to the actions available to queen agents.

- *Convolution_2D*(in_channels=1, out_channels=3, kernel=(3,3), stride=1, padding=1)
- *Linear_Layer*(in_dim= $n^2 \times 3$, out_dim=128)
- *Linear_Layer*(in_dim=128, out_dim=64)
- *Linear_Layer*(in_dim=64, out_dim=32)
- *Linear_Layer*(in_dim=32, out_dim=17)

Each agent is allowed 5 movement queries using the tensor network and each simulation is stopped after 50 time steps. The best parameters for *query range* and *move risk* varied based on board size, which are located in Table 1. $T = 5$ was found to be the best choice as smaller choices did not penalize conflicts as much as required.

A total of 200 games were played for training on each board with discount $\gamma = 0.999$ and learning rate $\alpha = 10^{-3}$. Afterwards, 100 games were played using the trained network.

Table 1: Queen agent parameters.

Board Size	query range	move risk
4x4	0.5	0.8
6x6	0.5	0.5
8x8	0.7	0.5
10x10	0.9	0.3

The results were then compared with a modification of the algorithm created by Susic et al. [50]. Their work was chosen for the following reasons. First, other N-Queens algorithms rely on placing queens on the board one at a time and then using a method, such as backtracking, to explore the state space until a valid solution is found [51]. However, similar to the proposed method, Susic et al. start with all queens initially placed on the board and then move them around. Second, similar limitations can be placed upon their algorithm that mirror the authors' modified N-Queens environment, where agents act under limited information. This limitation is accomplished by only allowing the queens to have knowledge of other queens a single column away. Their algorithm was ran on each board for 1,000 runs each. The percentage of successful games and the average number of conflicts present on the board were recorded for both methods in Table 2. While the modified version of Susic et al. has a perfect success rate on the 4×4 board, it fails to handle the larger scales of 6×6 boards and above with almost a complete failure rate on the 10×10 board where it only succeeds once. This is compared to the authors' method which is capable of solving each successive version of the board with slightly less ability than the prior board. The proposed method also results in larger numbers of conflicts between queens compared to the modified Susic et al. method. From these results, the authors' method is capable of solving smaller boards with its current setup, it has difficulty solving larger boards. This is most likely due to the fact that since queens can only see one space away, it is hard to know if a queen conflicts with another queen across the board. Increasing the amount of the board that the queens can see may benefit this problem. However, the tensor network contraction time and memory requirement will increase as a result of the increase in the amount of data that it needs to consider during each query. With these results, the authors' method has been shown to be capable of solving the N-Queens problem and that the queen agents can use tensor networks to build ensembles of data to inform their decisions.

Table 2: Success rate and average number of conflicts present on the board.

Board Size	Success Rate (%)		Avg # of Conflicts	
	Proposed Method	Susic	Proposed Method	Susic
4×4	30	100	1.33	0
6×6	24	3.4	3.25	1.67
8×8	17	1.3	5.60	3.03
10×10	9	0.1	7.40	4.30

The other key aspect of the results is that the tensor network approach decouples diagnostic decision making from the complexity of the environment. As the board increases in size, the size of the state space increases exponentially. In Sosic et al.'s modified method, their success rate decreases exponentially to the point where it only succeeded once out of 1,000 runs on a 10x10 board. However, the proposed method succeeded 9% of the time. It can be noted that as the board size increases, the proposed method's success rate decreases linearly in comparison. Therefore, even as the state space grows exponentially, the success rate decreases linearly in comparison, thus effectively demonstrating that agent decision making has been decoupled from the scale of the state space.

6. CONCLUSION AND FUTURE WORK

The design of SAHM systems for crewless vessels is a challenging problem due to the size of the vessel's state space. Through the method presented in this paper and its corresponding results on a modified version of the N-Queens problem, a tensor network-based approach has been shown to be capable of handling rapidly scaling environments. As the board size increases, the state space increases in size quickly and the modifications to the game introduce a limited information environment. Therefore, since the proposed method is capable of solving large 10x10 boards and Sosic et al.'s modified method failed to accomplish the same, the proposed method successfully decouples agent decision making from the scale of the environment.

Based on these promising results for the N-Queens problem with a simple approach and model, further work can be done to improve the method on a larger scale by increasing the vision range of the queen agents and by further increasing the size of the board. Both of these options increase the complexity of the problem and may introduce additional computational difficulties into the problem. In addition to further application to the N-Queens problem, additional investigation is required to translate the method onto the smart ship platform and other real-world systems.

Supporting Institution

Ms. Kelly Cooper of the Office of Naval Research

Grant Number

N00014-17-1-2491

REFERENCES

- [1] Register, L. Global marine technology trends 2030, 2014. <www.0427.co.uk>
- [2] O'Rourke, R. Navy large unmanned surface and undersea vehicles: Background and issues for congress. CRS Report No. R45757, 2021 [Online]. <<https://crsreports.congress.gov/product/pdf/R/R45757/44f>>
- [3] Kobylinski, L. Smart ships-autonomous or remote controlled? *Scientific Journals of the Maritime University of Szczecin*, 53:28–34, 2018. <<https://doi.org/10.17402/262>>
- [4] Evgeniy, Aleksandra, I., Vladimir, B. A. K., Ol'Khovik. Technology level and development trends of autonomous shipping means. pages 421–432. Springer International Publishing, 2021.
- [5] Ang, J. H., Goh, C., Li, Y. Smart design for ships in a smart product through-life and industry 4.0 environment, 2016.

- [6] Ellefsen, A. L., Æsøy, V., Ushakov, S., Zhang, H. A comprehensive survey of prognostics and health management based on deep learning for autonomous ships. *IEEE Transactions on Reliability*, 68:720–740, 2019. <<https://doi.org/10.1109/TR.2019.2907402>>
- [7] Dasgupta, A., Doraiswami, R., Azarian, M., Osterman, M., Mathew, S., Pecht, M. The use of canaries for adaptive health management of electronic systems. 2010.
- [8] Gao, Z., Liu, X. An overview on fault diagnosis, prognosis and resilient control for wind turbine systems, 2 2021. <<https://doi.org/10.3390/pr9020300>>
- [9] Chiang, L. H., Russell, E. L., Braatz, R. D. Fault detection and diagnosis in industrial systems. Springer Science & Business Media, 2000.
- [10] Scheidt, D., Mccubbin, C., Pekala, M., Vick, S., Alger, D. Intelligent control of auxiliary ship systems, 2002. <www.aaai.org>
- [11] Yajnik, S., Jha, N. K. Graceful degradation in algorithm-based fault tolerant multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 8:137–153, 1997. <<https://doi.org/10.1109/71.577256>>
- [12] DARPA. The no manning required ship (nomars) program kicks off, 10 2020.
- [13] Scheit, D. Email from david scheit's about research at weathergage, 2021.
- [14] Venkatasubramanian, V., Rengaswamy, R., Kavuri, S. N., Yin, K. A review of process fault detection and diagnosis: Part iii: Process history based methods. *Computers & Chemical Engineering*, 27:327–346, 3 2003. <[https://doi.org/10.1016/S0098-1354\(02\)00162-X](https://doi.org/10.1016/S0098-1354(02)00162-X)>
- [15] Xu, Y., Sun, Y., Wan, J., Liu, X., Song, Z. Industrial big data for fault diagnosis: Taxonomy, review, and applications. *IEEE Access*, 5:17368–17380, 7 2017. <<https://doi.org/10.1109/ACCESS.2017.273194>>
- [16] Williams, B. C., Nayak, P. P. A. A model-based approach to reactive self-connguring systems, 1996.
- [17] Gaspar, H. M., Rhodes, D. H., Ross, A. M., Erikstad, S. O. Addressing complexity aspects in conceptual ship design: A systems engineering approach. *Journal of Ship Production*, 28:145–159, 11 2012. <<https://doi.org/10.5957/JSPD.28.4.120015>>
- [18] Biamonte, J. Lectures on quantum tensor networks. 12 2019. <<http://arxiv.org/abs/1912.10049>>
- [19] Okunishi, K., Nishino, T., Ueda, H. Developments in the tensor network – from statistical mechanics to quantum entanglement. 11 2021. <<http://arxiv.org/abs/2111.12223>>
- [20] Klishin, A. A., Singer, D. J., Anders, G. V. Avoidance, adjacency, and association in distributed systems design. *Journal of Physics: Complexity*, 2, 4 2021. <[doi:10.1088/2632-072X/abe27f](https://doi.org/10.1088/2632-072X/abe27f)>
- [21] Zhang, K., Yang, Z., Başar, T. Multi-agent reinforcement learning: A selective overview of theories and algorithms. 11 2019. <<http://arxiv.org/abs/1911.10635>>
- [22] Friedland, B. Control system design : an introduction to state-space methods. Dover Publications, 2005.
- [23] Hamilton, J. D. State-space models*, 1994.
- [24] Kim, S. D. Characterizing unknown unknowns. 4 2012.
- [25] Lipol, L. S., Haq, J. Risk analysis method:Fmea/fmeca in the organizations. *International Journal of Basic & Applied Sciences*, 11:74–82, 2011.
- [26] Wang, Y., Liu, M., Bao, Z. Deep learning neural network for power system fault diagnosis. volume 2016-August, 2016. <<https://doi.org/10.1109/ChiCC.2016.7554408>>
- [27] Abishekraj, N., Prashanna, G. R. J., Suriyaa, M. S., Barathraj, T., Mohanraj, D. Condition based monitoring for fault detection in windmill gear box using artificial neural network. volume 912. IOP Publishing Ltd, 9 2020. <<https://doi.org/10.1088/1757-899X/912/3/032061>>
- [28] Chan, W. K. V. Interaction metric of emergent behaviors in agent-based simulation. pages 357–368, 2011. <<https://doi.org/10.1109/WSC.2011.6147763>>
- [29] Johnson, T. L., Genc, S., Bush, S. F. Active probing for diagnosis of emergent faults. *IFAC Proceedings Volumes*, 42:293–298, 6 2009. <<https://doi.org/10.3182/20090610-3-IT-4004.00055>>
- [30] Isermann, R., Freyermuth, B. Process fault diagnosis based on process model knowledge-part i: Principles for fault diagnosis with parameter estimation, 1991. <http://asmedigitalcollection.asme.org/dynamicsystems/article-pdf/113/4/620/5555274/620_1.pdf?casa_token=-uvY5IbINc8AAAAA:7oH6rA4tHm2rGdTS3WR-m45keYL9obFoSyEQg5EgY_N7Vu0SK0CL8ou-2YhSTyZAbHRLslk>
- [31] Freyermuth, B. R. isermann process fault diagnosis based on process model knowledge-part ii: Case study experiments, 1991. <http://asmedigitalcollection.asme.org/dynamicsystems/article-pdf/113/4/627/5555434/627_1.pdf?casa_token=0RwFHWxtWRQAAAAA:aftkGPr6TMTUI68kjzeWTOZs3ARbI4Jivq2O2r9nZtIQFbm3zy-2ufj-pdDkWCgubVc>

- [32] Xu, X., Yan, X., Yang, K., Zhao, J., Sheng, C., Yuan, C. Review of condition monitoring and fault diagnosis for marine power systems. *Transportation Safety and Environment*, 3, 2021. <<https://doi.org/10.1093/tse/tdab005>>
- [33] Lee, W. S., Grosh, D. L., Tillman, F. A., Lie, C. H. Fault tree analysis, methods, and applications: A review. *IEEE Transactions on Reliability*, R-34:194–203,1985. <<https://doi.org/10.1109/TR.1985.5222114>>
- [34] Toliyat, H. A., Nandi, S., Choi, S., Meshgin-Kelk, H. *Electric machines: modeling, condition monitoring, and fault diagnosis*. CRC press, 2012.
- [35] Saari, J., Strombergsson, D., Lundberg, J., Thomson, A. Detection and identification of windmill bearing faults using a one-class support vector machine (svm). *Measurement: Journal of the International Measurement Confederation*, 137:287–301, 4 2019. <<https://doi.org/10.1016/j.measurement.2019.01.020>>
- [36] Huang, K. *Huang - introduction to statistical physics*. 2009.
- [37] Klishin, A. A. *Statistical physics of design*.
- [38] Ran, S.-J., Tirrito, E., Peng, C., Xi, , Luca, C., Gang, T. , Lewenstein, S. M. *Lecture notes in physics 964 tensor network contractions methods and applications to quantum many-body systems*. <<http://www.springer.com/series/5304>>
- [39] Peng, C., Lewenstein, M., Ran, S.-J. Review of tensor network contraction approaches review of tensor network contraction approaches emanuele tirrito, 2017. <<https://www.researchgate.net/publication/319391456>>
- [40] Liang, L., Xu, J., Deng, L., Yan, M., Hu, X., Zhang, Z., Li, G., Xie, Y. Fast search of the optimal contraction sequence in tensor networks, 2021. <<https://github.com/liangling76/tensor-contraction-sequence-searching>>
- [41] Gray, J., Kourtis, S. Hyper-optimized tensor network contraction. 2 2020. <<http://arxiv.org/abs/2002.01935>>
- [42] Soong, W. *Reinforcement learning*, 2018.
- [43] Watkins, C. J. C. H., Dayan, P. *Q-learning*, 1992.
- [44] Fan, J., Wang, Z., Xie, Y., Yang, Z. A theoretical analysis of deep q-learning. 1 2019. <<http://arxiv.org/abs/1901.00137>>
- [45] Sonoda, S., Murata, N. Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*, 43:233–268, 9 2017. <<https://doi.org/10.1016/J.ACHA.2015.12.005>>
- [46] DiGiovanni, A., Zell, E. C. Survey of self-play in reinforcement learning. 7 2021. <<http://arxiv.org/abs/2107.02850>>
- [47] Liu, Q., Yu, T., Bai, Y., Jin, C. A sharp analysis of model-based reinforcement learning with self-play, 2021.
- [48] Rivin, I., Vardi, I., Zimmermann, P. The n -queens problem. *The American Mathematical Monthly*, 101:629–639, 8 1994. <<https://doi.org/10.1080/00029890.1994.11997004>>
- [49] Sasic, R., Gu, J. A polynomial time algorithm for the n-queens problem. *ACM SIGART Bulletin*, 1(3):7–11, 1990.
- [50] Bell, J., Stevens, B. A survey of known results and research areas for n-queens. *Discrete Mathematics*, 309:1– 31, 1 2009. <<https://doi.org/10.1016/J.DISC.2007.12.043>>