ABSTRACT

Title of Dissertation:	AN APPROXIMATION FRAMEWORK FOR LARGE-SCALE SPATIAL GAMES
	Vincent Hsiao
	Doctor of Philosophy, 2023
Dissertation Directed by:	Dana Nau
	Department of Computer Science

Game theoretic modeling paradigms such as Evolutionary Games and Mean Field Games (MFG) are used to model a variety of multi-agent systems in which the agents interact in a game theoretic fashion. These models seek to answer two questions: how to *predict* the forward dynamics of a population and how to *control* them. However, both modeling paradigms have unique issues that can make them difficult to analyze in closed form when applied to spatial domains. On one hand, spatial EGT models are difficult to evaluate mathematically and both simulations and approximations run into accuracy and tractability issues. On the other hand, MFG models are not typically formulated to handle domains where agents have strategies and physical locations. Furthermore, any MFG approach for controlling strategy evolution on spatial domains need also address the same accuracy and efficiency challenges in the evaluation of its forward dynamics as those faced by evolutionary game approaches.

This dissertation presents a new modeling paradigm and approximation technique termed

Bayesian-MFG for large-scale multi-agent games on spatial domains. The new framework lies at an intersection of techniques drawn from spatial evolutionary games, mean field games, and probabilistic reasoning. First, we describe our Bayesian network approximation technique for spatial evolutionary games to address the accuracy issues faced by lower order approximation methods. We introduce additional algorithms used to improve the computational efficiency of Bayesian network approximations. Alongside this, we describe our Pair-MFG model, a method for defining pair level approximate MFG for problems with distinct strategy and spatial components.

We combine the pair-MFG model and Bayesian network approximations into a unified Bayesian-MFG framework. Using this framework, we present a method for incorporating Bayesian network approximations into a control problem framework allowing for the derivation of more accurate control policies when compared to existing MFG approaches. We demonstrate the effectiveness of our framework through its application to a variety of domains such as evolutionary game theory, reaction-diffusion equations, and network security.

AN APPROXIMATION FRAMEWORK FOR LARGE-SCALE SPATIAL GAMES

by

Vincent Hsiao

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2023

Acknowledgments

This dissertation would not have been possible without the generous support and mentorship of my advisor Dana Nau, whose unwavering support and guidance I am very grateful for. I'd also like to acknowledge Rina Dechter who has served as basically a second advisor to me and I greatly appreciate her expertise and guidance over the course of working on many of the contributions in this work.

I am also grateful for the support and insights of Xinyue Pan, who I have had regular interaction with during weekly meetings over the past few years. I'd like to thank Bobak Pezeshki and Alexander Ihler for their comments and suggestions during weekly meetings with Rina. I'd also like to thank Aaravind Srinivasen, Nuno Martins and Tom Goldstein for agreeing to be on my dissertation commitee.

Finally, I am grateful to my family for supporting me through many years of study including during the unusual times of the covid pandemic.

This work was also financially supported in part by NSF grant IIS-2008516 and AFOSR grant 1010GWA357. The information in this paper does not necessarily reflect the position or policy of the funders, and no official endorsement should be inferred.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AS	Abstraction Sampling
BE	Bucket Elimination
BFS	Breadth First Search
BNA	Bayesian Network Approximation
BMFG	Bayesian Mean Field Game
CPT	Conditional Probability Table
DBN	Dynamic Bayesian Network
DLK	Deadlock (game)
EGT	Evolutionary Game Theory
ES	Evolution Strategies
ESS	Evolutionary Stable Strategies
GMFG	Graphon Mean Field Game
IS	Importance Sampling
KL-divergence	Kullback-Leibler divergence
LQG-MFG	Linear Quadratic Gaussian Mean Field Game
MC	Monte Carlo
MFG	Mean Field Game
MFTC	Mean Field Type Control
PD	Prisoner’s Dilemma (game)
PMFG	Pair approximation Mean Field Game
RPS	Rock-Paper-Scissors
SD	Snowdrift (game)
SIS	Susceptible Infected Susceptible model
WMBE	Weighted Mini-Bucket Elimination

Chapter 1: Introduction

In this thesis, techniques from evolutionary game theory, mean field game theory, and probabilistic reasoning are employed to model spatial multi-agent systems. The work in this thesis will be mainly focus on large multi-agent systems where we are not directly concerned the individual strategy of each agent. That is to say the granularity of the model is limited to aggregate quantities of interest. In Evolutionary Game Theory (EGT), an example of such a quantity would be the percentage of the population that follows a given behavior or norm over time. These population level quantities are also referred to as a *mean field* measure with respect to the entire system. Mean Field Games (MFG) describe the behavior of representative agents that interact with respect to these mean field measures. Incorporating these mean field measures is a core concept in the development of efficient approximations for the analysis of large multi-agent system dynamics.

This thesis advances the state of the art in evolutionary game theory by introducing a novel probabilistic method for modeling and approximating evolutionary games. Furthermore, we develop novel algorithms for increasing the effectiveness of our probabilistic method allowing for more accurate and efficient approximations. We also introduce a novel mean field model for spatial strategic games, first starting by modeling spatial evolutionary games. This model can be relaxed into a control problem framework that opens up additional avenues of research into controlling the outcomes of such games as well as having applications to other domains such as reaction-diffusion equations and network security.

1.1 Motivation

Game theoretic modeling paradigms such as Evolutionary Games and Mean Field Games are used to model a variety of multi-agent systems in which agents interact in a game theoretic fashion. In these systems, there are typically two questions we'd like to ask:

- *Prediction*: Given an initial condition over a multi-agent system and known system dynamics, can we predict how the population will change over time?
- *Control*: Given an initial condition over a multi-agent system and known system dynamics, can we control the strategies of (some or all) of the agents in the population (directly or indirectly) to reach some target distribution?

In terms of the prediction task, evolutionary game models have been widely used to model both biological and cultural evolution, e.g., [1, 2, 3, 4, 5] and a variety of multi-agent systems topics [6, 7, 8, 9, 10]. For the control task, mean field games have found widespread use in the modeling of various optimization problems such as wireless network control [11], crowd evacuation [12], vaccine distribution [13], and swarm robotics [14].

While both modeling paradigms have been applied to a variety of existing applications, they both have unique issues that can make it difficult to apply them to domains where agents have strategies and physical locations. Such domains can be described as the following two components:

- *strategy component*: a set of options that an agent that can select that determine the actions that the agent will take at a given time point/iteration;
- *spatial component*: a geometric or network location defined for each agent that determines the strength of interactions between different agents.

There are many systems where such a separation can naturally occur. Two classic examples of where this separation can be found is the Ising model from statistical physics and related voter models in which agents choose one of two strategies that are distinct from their particular location on a lattice network. Other examples of these interactions also include computer networks, social networks, and other behavioral models.

For these domains, Mean Field Game models are useful in that they are an efficient representation of large multi-agent systems and can be solved for optimal agent behavior at equilibrium. However, in general, MFG models are not formulated to deal with distinct strategy and spatial components. For example, in some MFGs, the strategy and spatial components are equivalent. The strategy in these MFGs simply describe the set of an agent's possible movements in the spatial domain. This disallows models of categorical behaviors in spatial games in domains such as viral spread, computational oncology, or social networks. Other MFGs assume the population is well-mixed, i.e. that each player effectively interacts with the average distribution of the population. In these well-mixed models, the spatial component is entirely missing which can make the solutions to these games inapplicable to systems defined on networks.

In contrast to mean field games, an evolutionary game can be easily extended to be a *spatial evolutionary game* in order to model domains with distinct strategy and spatial components. However, these spatial evolutionary games cannot be easily evaluated. Most work on evaluating spatial evolutionary games is performed with computationally expensive simulations, an approach that does not scale well when the spatial component of the game is defined over a large graph. Stochastic simulations also run into issues with validation [15] and variability [16] which make it difficult to apply them towards the analysis of many evolutionary games. For example, in spatial evolutionary games with multiple equilibria and/or cyclic equilibria, averaging multiple simulations may not produce meaningful representations of population behavior.

An ongoing area of research in spatial evolutionary games is then to devise a computationally efficient method for approximating the behavior of these games. A well known approach is to model the forward evolution of quantities in spatial games using a system of differential equations known as game dynamics. One such approach is pair approximation and its generalizations (to be discussed in Chapter 2). While widely used, these models suffer from two major issues:

- In lower order models such as pair approximation, a significant amount of information can be lost by performing moment closure at the pair level. While these models will better approximate population behavior on networks compared to mean field approximations, there can still be significant differences compared to ground truth simulation results. Since the information about higher order clusters is lost, pair approximation cannot model any impacts of longer range agent dependencies.

- Generalizations of pair approximation [Appendix C in [17]] have been developed to reduce the approximation error in pair approximation. However, these models quickly become computationally intractable for many applications as they scale at a complexity of $O(|S|^n)$ where n is the configuration size and $|S|$ is the number strategies in the model.

Better algorithms and/or models are necessary for an effective application to complex domains such as social modeling.

To summarize, current modeling paradigms either i) don't directly handle domains that have distinct strategy/spatial components directly, limiting their application to complex domains or ii) are too expensive to evaluate directly. This thesis presents a novel efficient framework that can handle both prediction and control tasks for these distinct strategy/spatial games.

1.2 Approach

In this thesis, we develop a unified framework for the approximation and control of large-scale spatial games. The framework consists of two parts:

- In section 3.2, we describe Truncated Dynamic Bayesian Network Approximations (TDBNA), a method for approximating the forward dynamics of spatial markov processes. In general, any forward equation arising from a Markov process defined on spatial models with distinct strategy spaces (beyond just spatial evolutionary games) can be approximated using Bayesian networks to desired accuracy by adjusting the size of the resulting TDBNA.
- In section 6.2, we describe a model that we have developed called Pair-MFG to address the lack of spatial models with distinct strategy and spatial components in Mean Field Games.
- In chapter 7, we combine both parts into a model we call *Bayesian-MFG*.

By themselves, TDBNAs and Pair-MFGs still lack a few features necessary to completely solve the problems discussed in the previous section. In particular, the larger Bayesian Networks in a TDBNA can end up being intractable to compute if the number of strategies in the evolutionary game increases. To address this, we devise a new method for computing more accurate approximations by using surrogate Bayesian Networks. Instead of computing inference on the larger network directly, we perform inference on a much smaller surrogate network extended with parameters that exploits the symmetry inherent to the domain. To learn the parameters on the surrogate network, we treat the problem as a KL-divergence minimization problem between the original and surrogate network. We also describe a novel algorithm for accelerating inference on graphical models with causal independence through the use of the FFT. This algorithm can in turn be applied to speeding up inference for TDBNAs for spatial evolutionary games, since there is causal independence within those networks.

The Pair-MFG model still retains some of the same inaccuracy and computational efficiency issues that are inherent to pair approximation/generalized n-point approximation models. To address this, we expand upon the Pair-MFG model and introduce a *Bayesian-MFG* framework to define control problems over dynamics specified by an arbitrary Truncated Dynamic Bayesian Network Approximation (TDBNA). This will allow for the exploration of control policies that better match behavior on spatial populations compared to Pair-MFG. With the additional accuracy afforded by higher order TDBNAs, we can model a larger variety of interesting control problems.

1.3 Thesis Outline

Chapter 2 provides background on relevant prior work on approximation spatial evolutionary games as well as an overview of Bayesian Networks and several probabilistic inference algorithms. Chapter 3 describes our Truncated Dynamic Bayesian Network Approximation (TDBNA) method for approximating the dynamics of spatial games. Chapter 4 describes an approximate inference algorithm using surrogate Bayesian networks for complex TDBNAs. Chapter 5 describes an algorithm we developed using the FFT for accelerating probabilistic inference on graphical models with causal independence. Chapter 6 describes existing work on mean field games and then our Pair-MFG model. We use spatial evolutionary games as a running example for the model. Chapter 7 details the combined Bayesian-MFG framework, its relaxation to a Bayesian-MFTC. Furthermore, Pair-MFTC and its extension Bayesian-MFTC are applied to domains of interest such as reaction-diffusion equations and network security.

Chapter 2: Background on Evolutionary Games and Bayesian Networks

In many situations, we want to predict the dynamics of a population. In these situations, we assume that each member of the population possess some state. In a field like epidemiology, this state could represent whether an individual is susceptible to infection, already infected, or was recently vaccinated and so on. The goal of research in these fields is to predict how the proportions of the populations with different states changes over time. Being able to predict these dynamics also unlocks the ability to control them, which we will discuss in later chapters.

In game theoretic models, the state of an individual (or *agent*) corresponds to the strategy that they are playing. Evolutionary games attempt to model the behavior of these agents subject to some predetermined update rule. In cultural modeling, for example, an agent might have two choices: *cooperate* or *defect*. In these models, the goal is to examine how the proportions of cooperators and/or defectors changes over time. Much research has been devoted in developing models that analyze how different factors promote or impede the evolution of cooperation in populations.

One common factor that is studied is the effect of spatial structure. To examine the effect of spatial structure, it is necessary to extend the basic evolutionary game model into a spatial evolutionary game. Spatial models have traditionally been very difficult to analyze. In fact, it has been shown in prior research that computing the exact answer to these questions about population dynamics on spatial evolutionary games is NP-hard [18] and is intractable for all but the smallest population sizes. A central question is then what approach should be used to evaluate the population dynamics of spatial evolutionary games over time. In practice, research involving spatial evolutionary games for applications such as social and cultural modeling typically run stochastic simulations of the spatial game in order to analyze them. However, there are still many issues with this approach and it can be difficult to extract causal information which is typically desired from these models.

An alternative to using simulations is to instead model the evolution of the population directly using systems of differential equations commonly denoted as game *dynamics*. Examples of different game dynamics include the Smith dynamics, Best Response dynamics, and Replicator dynamics [19]. Each of these dynamics is a system of differential equations where the variables are the proportion of the population playing a given strategy. These dynamics also assume that the population is *well-mixed*, that is, the population does not have a spatial structure. Due to this, these dynamics have a fundamental limitation when applied to spatial games. If applied directly, these dynamics will produce erroneous results due not taking into account the influence of spatial structure. This can be seen in many models in social modeling where the replicator equation will predict that the population will tend towards a population full of defectors, while in practice due to the phenomena of clustering it is possible under many circumstances for the population to instead tend towards cooperation [20].

The error present in these dynamical models for spatial populations has led to research towards developing more accurate dynamics. A key technique that has emerged in this research is the technique of *pair approximation*. Pair approximation represents the time evolution of the population using a system of differential equations with variables that proportion of pairs of agents playing a given strategy in addition to the terms present in regular dynamical equations. The addition of these higher order terms allows for the modeling of spatial behaviors such as clustering that is not captured by the aforementioned dynamics. Compared to game dynamics such as replicator dynamics, which are theoretically an exact model of a well-mixed population as the number of agents in the population tends to infinity, pair approximation is only exact at infinity for a Bethe lattice and thus still has substantial error when modeling behavior on structures such as networks or grids.

The focus of this chapter will be an exposition of the existing methods for approximating spatial evolutionary games and a an overview on Bayesian Networks and methods for probabilistic inference. These methods will serve as building blocks and points of comparison for the algorithms we develop in Chapter 3 and Chapter 4.

2.1 Evolutionary Games

Evolutionary Game Theory (EGT) provides a framework for modeling the time evolution of a population of agents that interact through strategic games whose outcome determines each individual's evolutionary fitness. These models disregard any game-theoretic assumptions of rationality and instead let individuals reproduce or change strategies based on a population update rule. More concretely, consider a population $N = \{1, ..., n\}$ of agents that play an iterated stage game over a finite time horizon $t \in [0, ..., T]$. An evolutionary game is (S, U, F):

- $S = \{s_1, ..., s_M\}$: the set of strategies where s_i is the i-th strategy in the in set of possible strategies S and $\mathrm{s}(t = 0) = (s_1^0, ..., s_n^0)$ denotes the initial strategy profile which is an a strategy assignment to each agent in the population where s_i^j denotes the strategy that the

i-th agent in the population is playing at iteration j.

- $U : S^N \to \mathbb{R}^N$: the payoff (or utility) function that maps the strategy profile to corresponding payoff values for each agent
- $\Pr(\mathbf{s}^{t+1} = \mathbf{s}' | \mathbf{s}^t = \mathbf{s}) = F(\mathbf{s}', \mathbf{s}, U(\mathbf{s}))$: an update rule that specifies the transition probabilities given the current strategy assignment and payoff values.

In more complex evolutionary games, it is possible to have an action space that is separate from the strategy space. In those games, the strategy of an agent will determine the agent's action for a given iteration as a function of its past actions and/or actions of other agent's in the population. In this section, we will only consider evolutionary games where the strategy space consists of only strategies from the stage game. This means strategies are *memoryless* and it is similar to making a Markovian assumption on the strategies of the players. Given this convention, many evolutionary games formulate their payoffs using a two-player normal-form stage game which can be specified as a payoff matrix $\mathbf{P}$:

Table 2.1: Two-player normal-form stage game

$\mathbf{P}$	$\mathcal{S}_1$	...	$\mathcal{S}_m$
$\mathcal{S}_1$	u_{11}, v_{11}	...	u_{1m}, v_{1m}
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$\mathcal{S}_m$	u_{m1}, v_{m1}	...	u_{mm}, v_{mm}

In a *well-mixed* population, each agent is equally likely to interact with any other agent in the population. Some common payoff functions in the well-mixed regime can be expressed as:

$$\begin{aligned}
&\text{Stochastic Payoff}: U_i(\mathbf{s}) = \mathbf{P}[s_i, s_j], \ \ j \sim \text{uniform}\{1, n\} \\
&\text{Average Payoff}: U_i(\mathbf{s}) = \frac{1}{n}\sum_{j}^{n} \mathbf{P}[s_i, s_j]
\end{aligned} \tag{2.1}$$

where the subscript of $U_i(\mathbf{s})$ denotes the i-th entry of the n dimensional payoff vector $U(\mathbf{s})$. The evolutionary game then consists of two phases:

- Interaction phase: Each agent i chooses some strategy $\mathcal{S}_i \in S$ and receives a payoff $\pi_i = U_i(\mathbf{s})$.
- Update phase: A percentage of agents γ in the population use an update rule to decide whether to change strategies or how to reproduce. There are a number of different update rules used in EGT models. Some commonly studied update rules include the following:
  - *Death-Birth rule*: Each agent has a non-zero chance of dying after the interaction phase. After an agent has died, agents that have not died have a probability to reproduce that depends on their fitness. [for a more detailed account, see the methods section of [21]]
  - *Fermi rule*: Each agent i compares its payoff π_i from the interaction phase with the payoff π_j of a randomly chosen teacher agent and switches to the teacher's strategy with probability $p_f(\pi_i, \pi_j)$, where

$$p_f(\pi_i, \pi_j) = \frac{1}{(1 + e^{-s(\pi_j - \pi_i)})} \tag{2.2}$$

where $s > 0$ is a constant called the selection strength.

– *Best response*: Different from the two aforementioned rules, the agent does not use the payoff at the current time to inform its strategy updating. Instead an agent will evaluate:

$$\arg\max_{s_i} \pi(s_i, (s)(t)) \tag{2.3}$$

where $\pi(s_i)$ is the payoff obtained playing strategy s_i versus the strategy profile of the current population $\mathbf{s}(t)$.

In the evolutionary game model, the interaction and update phases are repeated iteratively until a steady state or some predetermined time horizon is reached.

2.1.1 Microscopic vs Macroscopic

In the evolutionary game literature, there are two approaches to a model's representation and subsequently its evaluation:

- The Microscopic view: The model is represented using individual agents $\{1, ..., n\}$ and a strategy profile $(s_1, ..., s_n)$ as described in the previous section. The model is then evaluated using stochastic agent-based simulations.

- The Macroscopic view (also known as population model): The model is represented using a population profile:

$$\mathbf{p} = (p_1, p_2, p_3, ..., p_m) \text{ with } 0 \leq p_i \leq 1 \text{ and } \sum_i^m p_i = 1 \tag{2.4}$$

where p_i represents the proportion of the population playing a given strategy $\mathcal{S}_i \in S$. The evaluation is carried using the forward integration of a system of differential equations that can be derived from a given payoff function and update rule. In the macroscopic view, these differential equations are commonly known as *game dynamics*.

In some literature [22], these two approaches are also known as a node-state approach and a global-state approach. While the macroscopic view has its own limitations that we will discuss later, the macroscopic view has many advantages over the microscopic view in that it is much simpler to analyze and avoids many of the problems encountered when running stochastic agent-based simulations. A frequently used *game dynamic* in the macroscopic regime is the replicator equation:

$$\dot{p_i} = p_i[f_i(\mathbf{p}) - \phi(x)], \ \ \phi(x) = \sum_j^n p_j f_j(x) \tag{2.5}$$

where $f_i(x)$ is the fitness of the strategy $\mathcal{S}_i$ in the population and denotes the average payoff an agent playing $\mathcal{S}_i$ in the population would receive.

Given the assumption that agents are indistinguishable (the strategy profile $\mathbf{s}$ is permutation invariant), it is possible to transform a microscopic model into an asymptotically equivalent macroscopic model through the use of the *Master equation* [23] (for example, the replicator equation can be derived from a microscopic model defined using the Moran process). By asymptotically equivalent, we mean that as the number of agents in the model $n \to \infty$, the behavior of the microscopic model approaches the dynamics of the macroscopic model. In this sense, the macroscopic model is said to be a ***mean-field approximation*** of the microscopic model, where the *mean-field* is the population profile vector $\mathbf{p}$.

2.2 Spatial Evolutionary Games

An extension of evolutionary games to model a situation where the agents interact in a structured population is a spatial evolutionary game. Agents may interact more frequently with agents in a neighborhood around them or the utility received from their interactions may be weighted accordingly to some network structure. This in contrast to the well-mixed regime where agents interact with all other agents equally. These models can be useful in examining how network structure influences the evolution of agent behavior. More concretely, a spatial evolutionary game is (S, U, F, G) where

- (S, U, F) are analogous components as those found in the well-mixed evolution game, but the components can now also depend on a spatial structure G
- $G \in \{0, 1\}^{N \times N}$ is a spatial or network structure that specifies the strength of interactions between agents. For example, consider the payoff functions for the well-mixed regime in Eq. 2.1. In the spatial evolutionary game, an example payoff function is:

$$U_i(\mathbf{s}) = \sum_j^n G_{ij} \mathbf{P}[s_i, s_j] \tag{2.6}$$

Spatial evolutionary game models have been widely used to model both biological and cultural evolution, e.g., [1, 2, 3, 4, 5] and a variety of multi-agent systems topics [6, 7, 8, 9, 10]. Most work on evaluating spatial evolutionary games is performed with computationally expensive simulations on a microscopic model, an approach that often does not scale well for spatial structures [24]. Other limitations on validation [15] and variability [16] make it difficult to apply simulations towards the analysis of many evolutionary games.

Due to the limitations inherent in evaluating simulations on a microscopic model, it might seem desirable to approximate the microscopic model using a mean-field approximation like in the well-mixed regime. Indeed such an approach has been tried in works such as [25] to compute fixation time in a spatial populations. However, this approach can give very inaccurate results because of two issues:

1. Using well-mixed dynamics such as the replicator dynamics or other mean-field approximations to approximate model behavior can be widely inaccurate. Spatial phenomena such as clustering is not captured by replicator equations or other dynamics that are only defined on the population profile $\mathbf{p} = (p_1, ..., p_m)$. Consequently, the final equilibrium proportions reached by these well-mixed dynamics can be significantly different from the true behavior of agents in the structured population.

2. Alternatively, it is possible to apply the Master equation directly to the microscopic model for spatial evolutionary games. However, doing so does not produce a closed system of equations. When applying the master equation to a spatial evolutionary game to find the time evolution of the proportion of a strategy in the population, we arrive at equations that depend on higher order quantities. The equations that specify the time evolution of the proportion of agents playing each strategy p_i in the population depend on the proportion of pairs of agents p_{ij} playing different strategy pairs in the population. In turn, the equations that specify the time of evolution of pairs p_{ij} will depend on the proportion of triples p_{ijk}. This leads to a hierarchy of equations defined up to proportions of groups of agents the size

of the entire population:

$$\begin{aligned}
&\dot{p}_i = F(p_i, p_{ij}) \\
&\dot{p}_{ij} = G(p_i, p_{ij}, p_{ijk}) \\
&\dot{p}_{ijl} = H(p_i, p_{ij}, p_{ijk}, p_{ijkl}) \\
&\vdots
\end{aligned} \tag{2.7}$$

These systems of equations are intractable to solve given a large enough population size. Consequently, there is much work in past literature [2, 17, 26, 27, 28, 29, 30] in which higher order proportions are approximated using lower order proportions to reduce the intractable system of equations into something that can be computed. In general, there are two major ideas present in approximations of spatial evolutionary games:

- **Neighborhood Configurations**: A key idea is to choose a selection of agents (or a selection of sites as in [17]), with these agent(s) being denoted as focal agent(s). The transition probabilities for the focal agent or the collection of focal agents are computed by marginalizing over all possible assignments of neighboring agents in a given time step. An example of how this approach works is described in a Neighborhood Configuration model [31] for the purpose of modeling evolutionary dynamics on heterogeneous networks. In general, the number agents selected for this focal group vary based on the approximation method being derived. Depending on the network structure (e.g. whether it is a grid), it is also common to see various approximations that use different local shapes of focal agent groups.

- **Moment Closure**: Moment closures are an approximation technique that is frequently used in modeling systems of interacting particles/agents. Essentially, the idea boils down to closing intractable systems of equations by approximating higher order moments with a function of lower order moments. In the domain of spatial evolutionary games, a probability distribution over k agents can be considered a k-th order moment. In mean field approximations for spatial evolutionary games, the system is closed with first order moments and in pair approximation, which we will discuss in the following section, is closed with second order moments. Higher order moment closures such as triplet approximations or n-point approximations have also been considered [17]. It is mentioned in [32] that it is non-trivial to choose what moments to keep. Existing mean-field/pair/n-point approximation techniques choose a static set of moments to compute over time and they are typically defined over the entire set of k-th order moments for the order k the system is closed at.

2.2.1 Pair Approximation

Pair approximation is a well known approximation technique in the literature on spatial evolutionary games. A type of moment closure approximation, pair approximation was first described by Matsuda et al. for a spatial Lotka-Volterra model in [33], and subsequently has been widely used [34] [29] [31] [2] [35] to study the structural effects of spatial networks on the evolution of population behavior.

In pair approximation, the time evolution of the population is modeled using a set of differential equations that use global (p_i) and local (p_{ij}) density terms:

$$
\begin{aligned}
\dot{p}_i &= F_1(p_i, p_{ij}) \\
\dot{p}_{ij} &= F_2(p_i, p_{ij})
\end{aligned}
\tag{2.8}
$$

The functions are best expressed as summations over configurations *cf* of neighborhood strategy assignments (*neighborhood configurations*). More concretely, for an arbitrary agent x and its neighborhood $\{1, \ldots, N\}$, a configuration $cf = \{s_1, \ldots, s_N\}$ denotes the assignment of strategies to all neighbors of the agent x. Using the master equation, we have:

$$
\begin{aligned}
\dot{p}_i &= \sum_{cf} \left[\sum_{j \in S} P_{cf}(j) P(j \to i, cf) - P_{cf}(i) P(i \to j \mid cf) \right] \\
P(i \to j \mid cf) &= P\big(x(t+1) = x_j \mid x(t) = x_i, cf\big)
\end{aligned}
\tag{2.9}
$$

where an update rule (see section 2.1) is used to calculate the probability of an arbitrary agent changing its strategy conditioned on its local neighborhood. The key idea in pair approximation is the method for evaluating P_{cf} in Eq. 2.9. The true value of P_{cf} is a joint distribution over N variables comprising the neighborhood of an arbitrary agent x. For a configuration $cf = \{s_1, \ldots, s_N\}$, this is the probability $P(s_1, \ldots, s_N)$. In pair approximation, we approximate this probability using first and second order terms. For example, one possible second order approximation is:

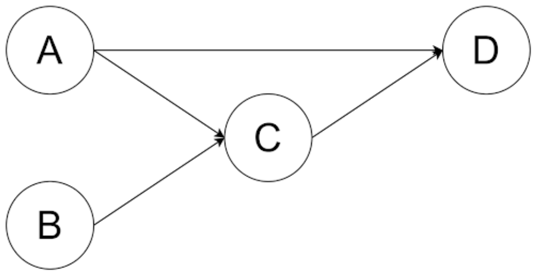
$$
P(cf) = P(s_1, \ldots, s_N) = p_{s_2|s_1} p_{s_3|s_1} \ldots, p_{s_N|s_1} p_{s_1} \tag{2.10}
$$

Outside of evolutionary dynamics, pair approximation can also be found in various applications such as voter models [36] and SIR/S dynamics [37] [38] [39]. As mentioned in the introduction, the naive application of pair approximation can encounter noticeable accuracy issues on several network structures. More concretely, pair approximation is formulated to be exact on Bethe lattices where the distribution of the neighboring nodes of a focal node can be assumed to be independent of each other. Approximation error is introduced when pair approximation is applied on graphs that possess cycles and the strategy distributions of neighboring nodes are no longer independent. The presence of cycles magnify the effect of longer range correlations which are not modeled in pair approximation. Extensions such as **generalized n-point** approximation (which is essentially a higher order moment closure) discussed in [17] have been created to address these issues but this extended approximation method can become quickly intractable as the number of strategies increases. It is important to note that all of these pair approximation methods are solely designed to approximate the forward time evolution of quantities of interest and there is lack of research on the control of these types of models. This will serve as a basic motivation for our work on Pair-MFG in later chapters.

2.3 Bayesian Networks

A Bayesian Network is a graphical model (X, D, F), consisting of a discrete variable set $X = \{X_1, X_2, \ldots, X_N\}$, a set of corresponding domains: $D = \{D_{X_1}, D_{X_2}, \ldots, D_{X_N}\}$ with $x_i \in D_{X_i}, \forall i$, and a set of parent functions $F = \{F_1, F_2, \ldots, F_N\}$. Each X_i is associated with a parent function $F_i = P(X_i \mid pa_i)$ where pa_i is the set of parent variables of X_i. The conditional probability functions F_i's are typically specified in a tabular format (CPTs).

Bayesian Networks are useful for modeling probabilistic distributions through an efficient



A	B	C	$\mathbf{P(C \mid A, B)}$
F	F	F	0.5
F	F	T	0.5
F	T	F	0.2
F	T	T	0.8
⋮	⋮	⋮	⋮
T	T	T	0.5

Figure 2.1: Example Bayesian Network with 4 variables (left) and example CPT for variable C (right)

representation of conditional independence. There are many existing algorithms for inference such as determining $P(A|B)$ in Fig. 2.1. Exact inference on Bayesian Network is typically done using bucket elimination (aka variable elimination) [40, 41]. However, there are also many other alternative approaches such as clique/junction tree, model counting/search, and hybrid conditioning approaches [41, 42, 43, 44, 45] that can be used to perform exact inference .

Many approximate inference techniques also exist, of which some are deterministic such as Weighted Mini-Bucket Elimination (WMBE) [46] and some which are stochastic such as Abstraction Sampling [47, 48]. These networks and automatic inference algorithms form the basis for our approach for approximating the time evolution of spatial evolutionary games which we will discuss in Section 3.2.

2.3.1 Dynamic Bayesian Network

A Dynamic Bayesian Network (DBN) [42] is a time-dependent Bayesian network over variables with a Markov assumption. Because of the Markov assumption, variables in the DBN at time $t + 1$ from the same or previous time step. Thus, the network can be represented as a two time-slice network as seen in Fig. 2.2 with an initial distribution. DBNs are commonly used to

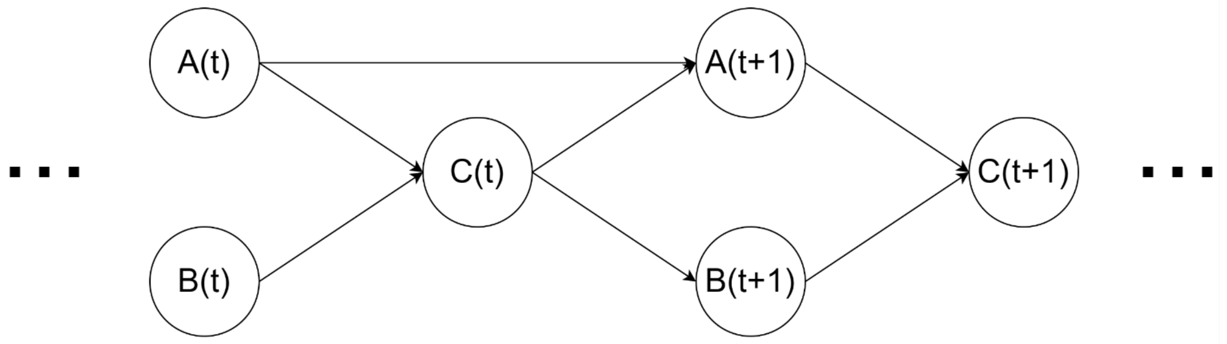


Figure 2.2: Example Dynamic Bayesian Network with 3 variables

model dynamical systems and we will apply this property later in our approximation approach.

2.3.2 Probabilistic Inference Algorithms

We will give a brief overview of different probabilistic inference algorithms that will be mentioned or compared against later. In the subsequent sections, one of the tasks that we are interested in is computing the marginal probability of some variable or the conditional probability distribution of a variable given an assignment to another variable. Given a Bayesian network with variables $\{X_1, ..., X_N\}$, the task is to compute $P(X_i = s), \forall s$ in its domain for some variables X_i.

One common method for computing this marginal distributions involves computing a normalization constant known as the partition function:

$$Z = \sum_x \prod_i F_i(x) \tag{2.11}$$

where the $\sum_x$ refers to a summation over all possible configurations of the variables in the model.

We can also define the partition function conditioned on certain variable assignments:

$$Z(X_1 = s) = \sum_{x|X_i=s} \prod_i F_i(x) \tag{2.12}$$

where the summation is over all configurations where $X_i = s$. In inference benchmarks, conditioned variables are termed *evidence*. To compute the marginal probability of a variable we can simply compute $Z(X_i = s)/Z$. One of the benefits of working with Bayesian networks is that we know that the partition function $Z = 1$ and so we can just compute $P(X_i = s) = Z(X_i = s)$.

Notice that to compute the partition function using the above equations, it would be necessary to enumerate through all possible variable configurations. This is clearly exponential in the number of variables in our network. For the purpose of this discussion, we will treat inference methods as algorithms for computing the partition function.

2.3.2.1 Bucket Elimination

Bucket Elimination (BE) [49], also known as variable elimination, is an exact algorithm for probabilistic inference. The key idea behind bucket elimination is to compute quantities such as the partition function by eliminating one variable at a time. To perform BE, it is first necessary to define an ordering o over the variables in the graphical model. Each variable is then associated with a set of functions or a *bucket* B_i. We place F_j into the bucket B_i corresponding to the variable X_i in the scope of F_j, S_j with the highest ordering.

After we have filled the buckets with the functions of the graphical model, we process each bucket one by one along the variable ordering o we chose earlier from last to first. To process a bucket, we compute a new function with a scope that is the union of the scopes of all functions in the bucket $S_{B_i} = \bigcup_{F_j \in B_i} S_j$ is:

$$\lambda_{B_i} : S_{B_i} \setminus \{X_i\} \to \mathbb{R}, \quad \lambda_{B_i} = \sum_{X_i} \prod_{F_j \in B_i} F_j \tag{2.13}$$

This new function is also commonly called a *message*. It is usually necessary to enumerate all configurations of the variables in S_{B_i} to compute the values of this new function.

After the computation, we then place the function λ_{B_i} into the bucket corresponding to the variable in the scope $S_{B_i} \setminus \{X_i\}$ with the highest ordering. Placing the function λ_{B_i} into the later bucket is also called a *message passing* operation. If the function's scope is empty, the λ_{B_i} function is actually a constant value and we place the value into the last bucket. If the variable X_i for B_i is actually an evidence variable with an assignment s, we can skip computing λ_{B_i} and instead just reduce each function: $F_j : S_j \to \mathbb{R}$ becomes $F'_j : S_j\ X_i \to \mathbb{R}$ with $F'_j = F_j(X_i = s)$. We then pass these reduced functions into later buckets following the same conventions as before.

The final computation performed in the last bucket will return a constant value and this will be the value of the partition function of the graphical model. Notice that the time and space complexity of this algorithm is dependent on the size of the largest scope S_{B_i} we need to compute a new function over. This size corresponds to the *induced-width* of the graphical model with the ordering o we chose.

2.3.2.2 Mini-Bucket Elimination

Instead of an exact answer, we might be wiling to settle for an upper or lower bound on the result of our Bucket Elimination algorithm. In Mini-Bucket Elimination (MBE) [50], we can do this by separating a bucket B_i into individual mini-buckets. Given a predetermined integer parameter *ibound*, we separate the functions $F_j \in B_i$ in to smaller mini-buckets such that the scope of each mini-bucket does not have more variables than the *ibound*. We perform the same message passing operation for each mini-bucket as in Bucket Elimination for all buckets except for one arbitrarily chosen bucket. In this arbitrarily chosen mini-bucket we instead compute:

$$\lambda_{MB_i} : S_{MB_i} \setminus \{X_i\} \to \mathbb{R}, \quad \lambda_{MB_i} = \max_{X_i} \prod_{F_j \in MB_i} F_j \tag{2.14}$$

instead of the summation operation from earlier. This will ensure that we get an upper bound as the final result from the algorithm. If we want a lower bound, we can simply take the min instead of the max. Compared to Bucket Elimination, the computations performed in computing the message functions are limited in complexity by the *ibound*. MBE is the exponential in the size of the $ibound$ and we can manually adjust the parameter to control a tradeoff between accuracy and time/space complexity.

2.3.2.3 Weighted Mini-Bucket Elimination

In Weighted Mini-Bucket Elimination (WMBE) [46], we replace the summation and max computations in MBE with a power sum. Given a bucket B_i separated into M mini-buckets MB_j, we compute the message for each mini-bucket as:

$$\lambda_{MB_i} = \left(\sum_{X_i} \prod_{F_j \in MB_i} F_j^{\frac{1}{p_j}} \right)^{p_j} \tag{2.15}$$

where p_j are real weights such that $p_j > 0$ and $\sum_j p_j = 1$. This computation uses Holder's inequality to ensure that the computation results in an upper bound. It is possible to improve the bounds using a technique called *cost shifting* and running weight optimization on the p_j's (see [46]). For the purpose of this work, all WMBE computations will just use equal weights, so p_j is $1/$(number of mini-buckets).

2.3.2.4 Monte Carlo Methods

Monte Carlo (MC) methods form another paradigm for probabilistic inference. In general, MC methods are used to estimate the expected value of a function $u(x)$ over a distribution $p(x)$:

$$E_p[u(x)] \approx \frac{1}{L}\sum_{i=1}^{L} u(x^i), \qquad x^i \sim p(x) \tag{2.16}$$

where $\{x^i\}_i^L$ are L independent samples drawn from the target distribution $p(x)$. Depending on the $u(x)$ and $p(x)$ chosen, MC methods can be used to perform various probabilistic inference tasks such as estimating the partition function [51].

For example, we can estimate marginal (and joint) probabilities such as $P(X_i = s)$ on a Bayesian network using MC directly through *forward sampling* [42]. On a Bayesian network, full samples of the network (a configuration of all the variables) can be generated through an iterative process by individually sampling variable assignments $x_i \sim P(X_i|pa_i)$ along the network from parents to children variables. To compute $P(X_i = s) = E_p[u(x)]$, we can use the following indicator function:

$$u(x^i) = \begin{cases} 1, & x^i \text{ consistent with } (X_i = s) \\ 0, & else \end{cases} \tag{2.17}$$

2.3.2.5 Importance Sampling

In many cases, it may be difficult to generate samples directly from the target distribution $p(x)$ or the samples generated may have high variance. For example, there can be configurations in $p(x)$ with a high value of $u(x)$ but a very low probability, which is often true in multi-dimensional distributions. One way to reduce variance in MC samples is to use Importance Sampling (IS) [42]. Instead of sampling directly from $p(x)$, we sample from a proposal distribution $q(x)$ and compute:

$$E_p[u(x)] \approx \frac{1}{L}\sum_{i=1}^{L} u(x^i)w(x^i), \qquad w(x^i) = \frac{p(x^i)}{q(x^i)}, \qquad x^i \sim q(x) \tag{2.18}$$

where $w(x^i)$ is called the importance weight. Depending on the proposal distribution $q(x)$ chosen, it can be possible to greatly reduce the variance of final estimate. There is a great deal of research that investigate extensions to IS for the task of estimating the partition function. We will focus mainly on Abstraction Sampling, an extension of IS that has been shown to be competitive with many state of the art methods for probabilistic inference on various inference benchmarks.

2.3.2.6 Abstraction Sampling

Abstraction Sampling (AS) is an approximate method for probabilistic inference that generates samples or *probes* which can be used to compute estimates of the partition function of a graphical model. The method can be seen as an extension of importance sampling inspired by the concept of abstractions in automated planning and is also similar to stratified importance sampling, a variance reduction technique in which the sample space is split into distinct groups.

In order to perform Abstraction Sampling, it is first necessary to define an AND/OR search space that corresponds the graphical model we want to perform inference on.

A graphical model can be transformed into alternative representations known as OR and AND/OR search spaces [45]. For the purpose of this discussion, we will focus on simple OR-trees. Given a variable ordering $o = (X_1, X_2, ..., X_N)$, let the root of the search tree be a dummy node. Each level i in the search tree corresponds to variable $X_i \in o$. Each node n_{X_i} in level i corresponds to a configuration to all variables up to X_i, $X_{\rightarrow i}$. The edges of the search tree are weighted such that each edge into n_{X_i} has a weight $c(n_{X_i}) = \prod_{F_i \in F | X_{F_i} \subseteq X_{\rightarrow i}} F(n_{x_i})$. We define the value of a node n_{X_i}, $Z(n_{X_i})$ to be the partition function of the rooted search tree where $ch(n_{X_i})$ is the set of children of the node n_{X_i} in the rooted search tree. It is easy to see that Z obeys that following recursive relation:

$$Z(n_{X_i}) = \sum_{n_{X_j} \in ch(n_{X_i})} c(n_{X_i}) Z(n_{X_j}) \tag{2.19}$$

It is known that different nodes in the search tree may root the same subtree and thus have the same value. Identifying all such nodes is hard but the configuration of a node's context, a subset of the preceding variables in the ordering, can help identify such equivalent subtrees. Given an ordering o, the context of variable X_i, $C(X_i)$ are the variables in o that precede it and whose assignments cause the conditional independence where $\{X_i, X_{i+1}, ..., X_N\}$ is independent from the set $o \setminus C(X_i)$ conditioned on $C(X_i)$. It can be shown that the configuration over $C(X_i)$ uniquely determines the value of the subtree rooted at n_{X_i} [45]. The partition function of the graphical model M can be computed as $Z = Z(root)$ of the corresponding OR-tree search space [45].

2.3.2.7 Abstraction Sampling Algorithm

Let $S_o = (X, E)$ be the full search tree generated from variable ordering o on a graphical model M. A probe $S_o^{'}$ is a subtree of S_o such that it includes the root of S_o. Let d be a constant denoting the number of abstract states, $A = \{1, 2, \dots, d\}$. Let $ch(S_o^{'i})$ be the not yet expanded to children of each node $n_{X_i} \in S_o^{'i}$. We define an abstraction function $a : (M, S_o^{'i}) \rightarrow A^{|ch(S_o^{'i})|}$ as a mapping of each child node $n \in ch(S_o^{'i})$ to their respective abstract state.

RandCB is an abstraction function introduced by Broka et al. [47]. With a random integer $k_{X_i} \in \{1, ..., K\}$ associated with each X_i (K is a provided parameter), we map each node $n \in ch(S_o^{'i})$ to the state $a(n) = [\sum_{X_j \in C(X_{(i+1)})} k_{X_j} x_j] \bmod d$ where x_j is the value for X_j in the configuration corresponding to node n.

In Abstraction Sampling, to generate a probe $S_o^{'}$, we perform the following steps iteratively starting from $S_o^{'0}$ which only includes a dummy root node.

1. **Partition** $ch(S_o^{'i})$ into abstract states using an abstraction function a.

2. **Sample** a representative node from each non-empty abstract state $A^i \in A$ with probability:

$$p(n) = \frac{w(n)g(n)h(n)}{\sum_{m \in A^i} w(m)g(m)h(m)} \tag{2.20}$$

   for each node $n \in A^i$, where $g(n)$ is the product of all $c(n)$ in the path from n to the root, $h(n)$ is a heuristic estimate of $Z(n)$ normally computed using weighted mini-bucket elimination [46], and $w(n)$ is the importance weight associated with node n (see [48] for more details).

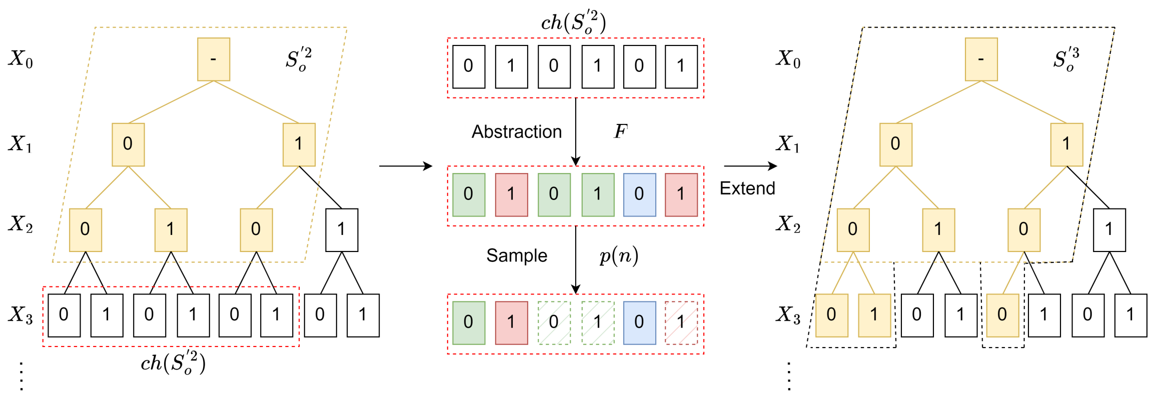


Figure 2.3: Example single step in the abstraction sampling process from a partial probe $S_o^{'2}$ at $t = 2$ to a partial probe $S_o^{'3}$ at $t = 3$. The next set of frontier nodes $ch(S_o^{'2})$ is partitioned into three abstract states (represented with different colors) using F, then a single representative is sampled from each state and used to extend the probe to the next level.

3. **Extend** $S_o^{'i}$ to $S_o^{'(i+1)}$ by adding the selected representative nodes.

It is possible to terminate this process early if we know heuristic $h(n_{X_i})$ is exact and set the value $Z(n_{X_i}) = h(n_{X_i})$. An example of this step by step process is shown in Fig. 2.3.

Given the fully expanded $S_o^{'N}$, we prune the probe recursively such that all leaves ***not*** located at the lowest level N are removed, creating the final pruned probe $S_o^{'}$. The Z-estimate of $S_o^{'}$ denoted $Z'(S_o^{'})$ is defined by:

$$Z'(S_o^{'}) = Z'(n_{X_0}), \quad Z'(n_{X_i}) = \sum_{n_{A_i} \in ch'(n_{V_i})} c(n_{A_i}) \cdot Z'(n_{A_i}) \frac{w_{n_{A_i}}}{w_{n_{X_i}}}, \quad Z'(n_{X_N}) = 1 \qquad (2.21)$$

where $ch'(n_{X_i})$ denotes the children of n_{X_i} in the pruned tree $S_o^{'}$. The output of Abstraction Sampling is the Monte Carlo estimate obtained from the average of the individual Z-estimates from sampled probes generated by the above process.

Chapter 3: Approximation Methods for Spatial Evolutionary Games

In this chapter, we will describe our novel method for approximating spatial games using Dynamic Bayesian Network Approximations. As mentioned in Chapter 2, existing approximation techniques such as pair approximation can lack accuracy when applied to games on networks. This leads us to the new method that we have developed for approximating the dynamics of spatial evolutionary games. By employing dynamic Bayesian networks as a building block, we design an iterative method for approximating spatial games that generalizes pair approximation and can be easily adjusted to produce more accurate approximations of spatial games.

We will start off with a description of our exact our Dynamic Bayesian Network (DBN) model that captures a given spatial evolutionary game, followed by our Truncated Dynamic Bayesian Network Approximation (TDBNA) algorithm for query processing. In Section 3.2.5, we will show how a special case of our approximation algorithm coincides with the pair approximation technique from evolutionary game theory [2] and in Section 3.3. we will show some empirical results demonstrating the effectiveness of our method in approximating the behavior of spatial evolutionary games.

3.1 Dynamic Bayesian Networks for Spatial Evolutionary Games

In this section, we define a Dynamic Bayesian Network model that fully captures a spatial evolutionary game. For our example, we will model the stochastic spatial evolutionary game using the parameters listed in Table 3.1. The chosen game is a two strategy game with a payoff matrix that is based on the prisoner's dilemma using the synchronous ($\gamma = 1.0$) Fermi-rule.

Given an evolutionary game, we define a Dynamic Bayesian Network M = ($X(t)$, $D(t)$, $F(t)$), where the variable set X is split into two sets of variables $X = A \cup Pay$ at each iteration.

- $A_{i,j}(t) \in A$: Each of these variables represents the strategy of the agent placed at location (i, j) on the grid at the start of each iteration t and its values are the strategy set S.
- $Pay_{i,j}(t) \in Pay$: Each of these variables represent the payoff received by the agent at (i, j) during the interaction phase. The domain of these variables consists of all possible payoff values.

Each X_i is also associated with a parent function $F(t)_i = \Pr(X_i \mid Pa_i)$ where Pa_i is the set of parent variables of x_i. Next we will define these functions $F(t)_i$ as Conditional Probability Tables

Table 3.1: Spatial Evolutionary Game Parameters

Parameter	Value
Graph Type	Grid
Graph Degree	$d = 4$ (von Neumann neighborhood)
Update Rule	Fermi Rule
Update Percentage	$\gamma = 1.0$
Selection Strength	$s = 5/3$
Mutation Rate	$\mu = 0.05$
Strategies	$S = \{C, D\}$
Payoff Matrix	$\mathbf{Pay} = \begin{pmatrix} 2 & -1 \\ 3 & 0 \end{pmatrix}$

(CPTs) for the payoff nodes and the strategy nodes at time $t+1$. In the following discussion we will interchange the terms ”nodes” and ”variables”.

The network can be seen in Figure 3.1, which displays a subset of the full network that contains all of the immediate parents of $A_{1,1}(t+1)$. Several of the parent nodes $A_{k,l}(t)$ of the payoff nodes have been omitted in this cross-sectional view.

3.1.1 CPT for Payoff Nodes

Each node $Pay_{i,j}(t)$ has $d+1$ parents: $A_{i,j}(t)$ and its d neighbors. The conditional probability function $P(Pay_{i,j}(t) \mid parents)$ is constructed as a logical function using the payoff matrix $\mathbf{P}$. We define $N(A_{i,j}(t))$ to be the neighborhood of $A_{i,j}(t)$ in the spatial evolutionary game. Then we have:

$$\Pr(Pay_{i,j}(t) \mid A_{i,j}(t), N(A_{i,j}(t))) = \begin{cases} 1 & \text{if } Pay_{i,j}(t) = \sum_{A_{k,l}(t) \in N(A_{i,j}(t))} \mathbf{U}[A_{i,j}(t), A_{k,l}(t)] \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

where $\mathbf{U}$ is the utility/payoff function from the evolutionary game.

3.1.2 CPT for t+1 Strategy Variables

As can be seen in Fig. 3.1, each $A_{i,j}(t+1)$ has $2(d+1)$ parents: $A_{i,j}(t)$, $Pay_{i,j}(t)$ and the $A(t)$ and $Pay(t)$ nodes for each of the d neighbors of $A_{i,j}(t)$. We will now define $\Pr(A_{i,j}(t+1) \mid parents)$. Recall that during the update phase of the evolutionary game, a percentage γ of agents are chosen for updating. Each such agent chooses a random neighbor from its d neighbors to compare its

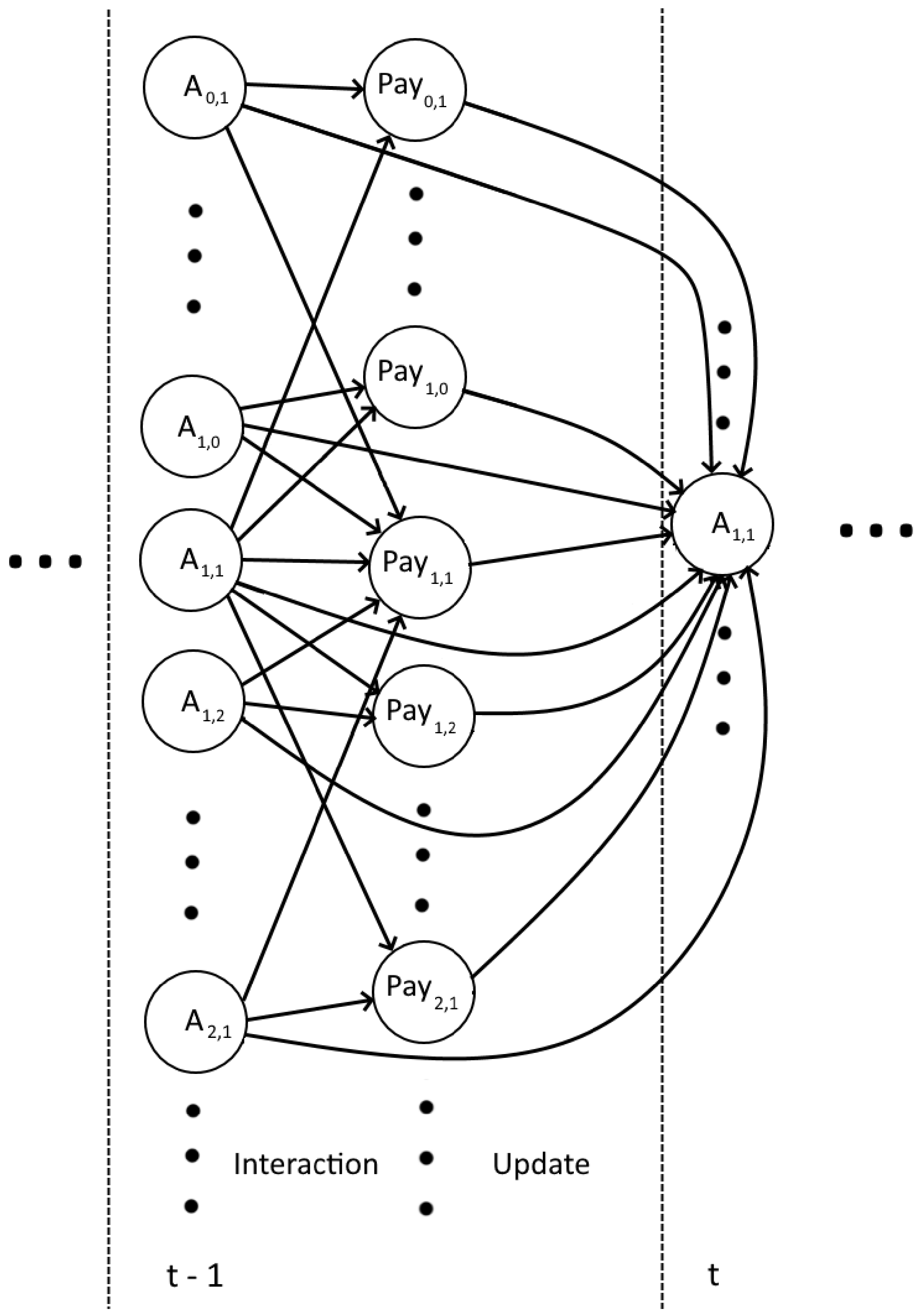


Figure 3.1: Slice of Dynamic Bayesian Network for the Fermi update rule centered at the agent located at position (1,1)

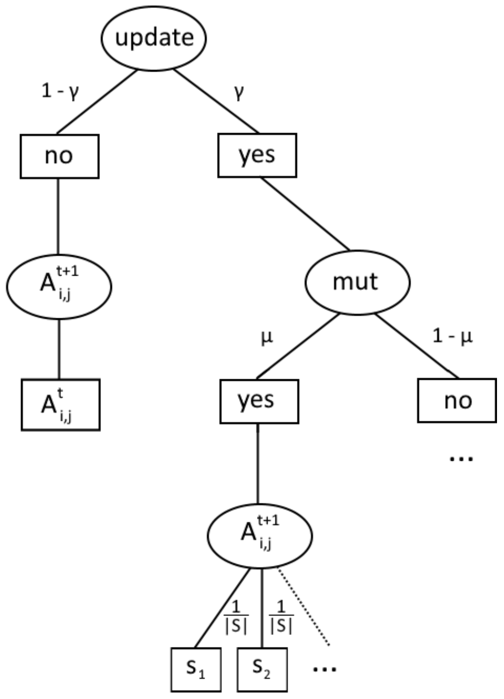


Figure 3.2: Upper part of the decision tree for t+1 variables

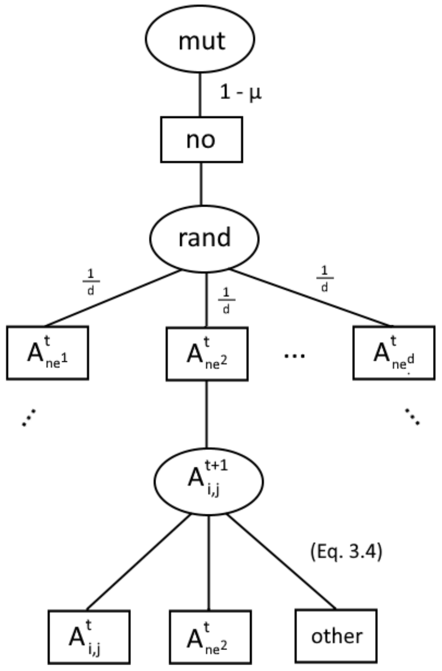


Figure 3.3: Cross section of (mut = no) branch

payoffs with. Given a random neighbor $A_{k,l}(t)$, $A_{i,j}(t)$ has a $p_f(Pay_{i,j}(t), Pay_{k,l}(t))$ (from Eq. 2.2) chance of copying the strategy of $A(t)_{k,l}$. Additionally, independently from the Fermi rule, each agent has μ probability of mutating to a random strategy. Conditioning on the value of each of these independent events, we can describe our transition probability on a case by case basis using a decision tree, as described in [52]. We define the following three additional variables to represent events in the decision tree:

- update: is the node within the fraction γ of the population chosen for updating?
- mut: did a mutation event happen?
- rand: which neighbor did the node choose to compare its payoff with?

The decision tree can be seen in Fig. 3.2 and Fig. 3.3 where circular nodes denote variables and square nodes denote the value of the variable. The decision tree specifies the context independent paths that make up the CPT for $A_{i,j}(t+1)$.

In the case where a node is chosen for updating and mutation does not happen, the final probability must be conditioned on the path chosen. In Fig. 3.3 we have branches of the variable rand where $A_{i,j}$ chooses different neighbors A_{ne^i} to compare its payoff to. For each neighbor A_{ne^i}, the probability of the bottom edges of Fig. 3.3 is:

$$\begin{aligned} P(A_{ij}(t+1) = A_{ne^i}(t)) &= p_f(Pay_{i,j}(t), Pay_{ne^i}(t)) \\ P(A_{ij}(t+1) = A_{i,j}(t)) &= 1 - p_f(Pay_{i,j}(t), Pay_{ne^i}(t)) \\ P(A_{ij}(t+1) = \text{other}) &= 0 \end{aligned} \tag{3.2}$$

An explicit representation for $P(A_{i,j}(t+1) = s_{t+1} \mid A_{i,j}(t) = s_t, \text{other parents})$ can also be obtained[1]. For notation purposes, we use indicator functions to define the following quantities:

$$p_\delta = \mathbb{1}_{A_{i,j}(t+1)=A_{k,l}(t)}$$

$$p_\varnothing = \mathbb{1}_{A_{i,j}(t+1)=A_{i,j}(t)}$$

Following a case by case breakdown, we can then write:

$$\begin{aligned} &\Pr(A_{i,j}(t+1) = s_{t+1} \mid A_{i,j}(t) = s_t, \text{other parents}) \\ &= (1-\gamma)p_\varnothing + \gamma\Big[\frac{\mu}{|S|} + (1-\mu)\sum_{A_{k,l}(t)\in N(A_{i,j}(t))} \frac{1}{d} p_f(k,l)p_\delta(1-p_\varnothing) + p_\varnothing\Big] \end{aligned} \tag{3.3}$$

where $p_f(k,l) = p_f(Pay(t)_{i,j}, Pay(t)_{k,l})$ and μ, γ are parameters taken from the evolutionary game.

Algorithm 1: Dynamic Bayesian Network for Spatial Evolutionary Game
Input: Spatial evolutionary game (S, U, F, G), agents $[X_1, ..., X_N]$, inital distribution $p_{s_i}(0)$
Output: Dynamic Bayesian Network $M = (X, D, F')$

```
1 $A \leftarrow \{X_1, ..., X_N\}$ ;
2 $Pay \leftarrow \{Pay_1, ..., Pay_N\}$ ;
3 $X \leftarrow A \cup Pay$ ;
4 $D_A \leftarrow S$ ;                                   // Domain for agent variables
5 $D_P \leftarrow$ all possible payoff values in $U$ ;   // Domain for payoff variables
6 $D \leftarrow D_A \cup D_P$ ;
7 $F' \leftarrow$
    $\begin{cases} P(A_i(0) = s_i) = p_{s_i}(0) \\ P(Pay_i(t)|A_i(t), N(A_i(t))) = \text{(see Eq. 3.1)} ; \quad // \ N(A_i)(t) \text{ defined using } G \\ P(A_i(t+1)|A_i(t), \text{other parents}) = \text{(see Eq. 3.3)} \end{cases}$
8 Return $(X, D, F')$;
```

[1]This CPT can be very high dimension $O(|S|^{(d+1)}|U|^{(d+1)})$ if the number of strategies are large where $|S|$ is the number of strategies and $|U|$ is the number of unique payoff values (approximately $O(|S|^d)$). For this chapter, we will explicitly compute its entries since we will only work with evolutionary games with 2 strategies. However, a sparse representation will be used in Chapter 4 for games with 4 or more strategies.

A summary of the construction process for the DBN can be seen in Algorithm 1. The resulting DBN formulation fully encodes the stochastic process of the spatial evolutionary game. For a given stochastic spatial evolutionary game, an agent-based simulation is equivalent to running a Monte-Carlo simulation on the corresponding DBN.

3.2 Truncated Dynamic Bayesian Network Approximations

In the following sections, we describe our novel iterative method, Truncated Dynamic Bayesian Network Approximations (TDBNA) [53], for approximating the time evolution of a multi-agent populations with a spatial structure. The primary advantage of a TDBNA is that it allows for the exploration of higher order approximations beyond pair approximation which allow for better accuracy with respect to the underlying stochastic model. Given a dynamic Bayesian network (see definition in chapter 2) that exactly models the stochastic processes present in the multi-agent structued population, we construct a sequence of smaller approximate Bayesian networks each evaluating the time evolution of our population for a single timestep (from t to $t+1$). The method consists of three components: 1. an *input neighborhood* and a *target neighborhood*, 2. *output query*, and 3. *input definition* which involves constructing a lower-order approximation of the input neighborhood using distributions obtain from the output query.

3.2.1 Dynamic Bayesian Network for Spatial Markov Processes

Consider a spatial multivariable Markov process defined over a set of agent variables $[X_1, ..., X_N]$ that take a strategy (or state) in the set S with some defined transition probability $P(X_i(t+1) = s|\mathbf{X}(t))$ for each X_i. By spatial Markov process, we assume that the transition probability of any arbitrary agent X_i depends locally on a neighborhood $N(X_i)$ specified by a graph G for each iteration. Therefore, the transition probability of the Markov process obeys the following relation:

$$P(X_i(t+1) = s|\mathbf{X}(t)) = P(X_i(t+1) = s|X_i(t), X_j(t), j \in N(X_i)) \tag{3.4}$$

We can construct a Dynamic Bayesian Network (DBN) that fully captures the above Markov process, where each variable in the DBN corresponds to an agent in the spatial process and its values are the strategies $S = \{s_1, ...s_m\}$. Formally the DBN is $M = (X(t), D(t), F(t))$ where

- $X(t)$: Each variable $X_i(t) \in X(t)$ corresponds to an agent in the original spatial process.
- $D(t)$: The domain of each variable is S
- $F(t)$: The CPT for $X_i(t+1)$ has a dimension of $|N(X_i)| + 1$ and has entries defined from the Markov process defined above (Eq. 3.4).

We will primarily consider DBNs that model spatial evolutionary games for this chapter as described in Section 3.1, but this model (and its approximation in 3.2.2) can also be applied to other domains such as the reaction-diffusion and network security applications in Chapter 7. When defining a DBN for a spatial Markov process, we assume that the initial distribution of any two agent variables at time $t = 0$ is identical: $P(X_i(t) = s) = P(X_j(t) = s)$ for any two agents X_i and X_j. Because of this assumption and because the DBN is highly symmetric, the marginal distributions $P(X_i(t) = s) = P(X_j(t) = s)$ are identical at any time t. This property also applies to any pairs of adjacent agents: $P(Nb(X_i)(t), X_i(t)) = P(Nb(X_j(t), X_j(t))$ where $Nb(X_i)$, $Nb(X_j)$ are neighboring adjacent agents of X_i and X_j respectively.

We denote by p_{s_i} the proportion of the population in state i and denote $\mathbf{p} = \{p_{s_i}, ..., p_{s_m}\}$ to be the population profile. Furthermore, denote $p_{s_i s_j}$ to be the proportion of adjacent agents in the population playing the strategy pair (s_i, s_j). Higher order terms are defined analogously with $p_{s_i s_j s_k ...}$. To evaluate the time evolution of our evolutionary game, we can simply query:

$$\begin{aligned} p_{s_i}(t) &= P(X(t) = s_i) \\ p_{s_i s_j}(t) &= P_t(X(t) = s_i, Nb(X)(t) = s_j) \end{aligned} \tag{3.5}$$

for any arbitrary variable/node X and an adjacent neighboring node $Nb(X)$. However, computing a query like that over the DBN using an inference algorithm such as bucket elimination is intractable once the number of agents in the population becomes large. Additionally, as seen in Section 3.1, when using a DBN to model spatial evolutionary games with complex update rules such as the Fermi rule, it is necessary to add additional nodes to the DBN such as payoff nodes which drastically increases the complexity of network. Therefore, in Section 3.2.2 we will address how to approximate this query computation over our DBN that models a spatial Markov process.

3.2.2 Truncated Dynamic Bayesian Network Approximation for Spatial Markov Processes

Given a DBN that models a spatial Markov process, at time $t = 0$, we assume that the prior and initial distribution for all agents $P(X(t = 0) = s_j)$ are identical for all agents. In the evolutionary game notation, this is denoted as $p_{s_i}(0)$. We assume the same for the second order terms $p_{s_i s_j}(0)$. A Truncated Dynamic Bayesian Network Approximation is the iterative method shown in Algorithm 2 for approximating the time evolution of p_{s_i} and $p_{s_i s_j}$.

Algorithm 2: Truncated Dynamic Bayesian Network Approximation for Spatial Markov Processes

Input: Dynamic Bayesian Network $M = (X, D, F)$ for a spatial Markov process, Input neighborhood set $I \subset X$, Output neighborhood set $O \subset I$, chosen focal node $X_F \in I \cap O$, Input definition function $\Psi : S^{|I|} \to [0, 1]$

Parameters: number of time steps T,

Output: $p_{s_i}(T), p_{s_i s_j}(T)$

```
1 $p_{s_i}(0) \leftarrow P_M(X_F(0) = s_i), \ \forall s_i \in S$ ;
2 $p_{s_i s_j}(0) \leftarrow P_M(X_F(0) = s_i, Nb(X_F)(0)), \ \forall s_i, s_j \in S$;
3 for $t = 0 \to T - 1$ do
4     Define a truncated two time step Bayesian network $B(t) = (X_{tr}(t), D, F_{tr}(t))$:
          • $X_{tr} \leftarrow I(t) \cup O(t + 1)$
          • $F_{tr} \leftarrow P(I(t)) = \Psi(I(t); p_{s_i s_j}(t), p_{s_i}(t))$, (see Section 3.2.3)
                  $P(O(t + 1)|I(t)) = P_M(O(t + 1)|I(t))$
5     $p_{s_i}(t + 1) \leftarrow P_B(X_F(t + 1) = s_i), \ \forall s_i \in S$ ;
6     $p_{s_i s_j}(t + 1) \leftarrow P_B(X_F(t + 1) = s_i, Nb(X_F)(t + 1)), \ \forall s_i, s_j \in S$
      where $P_B$ is computed using a probabilistic inference algorithm on $B(t)$ ;
7 end
8 Return $p_{s_i}(T), p_{s_i s_j}(T)$;
```

Before starting the algorithm, we must first choose a representative subset of agents to be the *input neighborhood* $I \subset [X_1, ..., X_N]$ and a subset $O \subset I$ to be the *output neighborhood.* It is necessary for every node in O to have a fully defined transition probability conditioned on the nodes in I as in Eq. 3.4. For example, in the simplified diagram of Fig. 3.4, $I(T - 1)$ consists of the three nodes at $T - 1$ within the highlighted area and $O(T)$ consists of the the single node at time T within the highlighted area. Denote a single node that is in both the input and output neighborhoods to be the *focal* node X_F.

To begin, on line 1 and 2 of Algorithm 2, we set the quantities $p_{s_i}(0), p_{s_i s_j}(0)$ to be the marginal and pairwise probabilities of nodes of our input DBN. From time $t = 0$ to $T - 1$, at each iteration we construct a truncated two step Bayesian network $B(t) = (X_{tr}(t), D, F_{tr}(t))$ and perform a query computation as such:

- The new two step BN has variables $X_{tr}(t) = I(t) \cup O(t+1)$ (line 4, first bullet)

- We define the distribution of $I(t)$ using a probability function $\Psi : S^{|I|} \to [0,1]$ parameterized with $p_{s_i}(t)$ and $p_{s_i s_j}(t)$ (line 4, second bullet, first part) (see Section 3.2.3 for more details).

- We define the transition probabilities that transform the strategies of agents in $I(t)$ to the agents at $O(t+1)$. Namely, for each variable in $o \in O(t+1)$, define the CPT $P(o|Pa_o)$ where Pa_o are the variables in $I(t)$ that are the parents of o. Note that these are the same CPTs that are found in our input network M, so we can write (abusing notation) that $P(O(t+1)|I(T)) = P_M(O(t+1)|I(T))$ (line 4, second bullet, second part).

- Use a probabilistic inference algorithm [40, 41] such as Bucket Elimination [49] to evaluate $p_{s_i}(t+1), p_{s_i s_j}(t+1)$. These distributions are defined over the agents in the output $O(t+1)$ (lines 5 and 6).

To summarize, a 2 timestep Bayesian network is constructed for each iteration that takes the strategies of each agent from t to $t+1$. Using a probabilistic inference algorithm, we evaluate the probabilities $P(X(t+1))$ and $P(X(t+1), N_X(t+1))$ and we repeat the process at the next timestep by defining the distribution of the next input set $I(t+1)$ using the quantities we queried.

For more complex spatial systems such as spatial evolutionary games using the Fermi rule, it is more efficient to have additional Pay nodes like in Fig 3.1 that represent payoff nodes instead of compiling the probabilities of the spatial evolutionary game into a Markov process in the form of Eq. 3.4. In these cases, when defining the transition probabilities from $I(t)$ to $O(t+1)$ in Algorithm 2 (third bullet above), we retain all the necessary intermediate nodes (such as the Pay nodes) and CPTs from the DBN M when constructing $B(t)$.

Depending on the size of the output neighborhood, it may be necessary to modify line 6 in Algorithm 2. If the input neighborhood is very small (i.e. only 8 nodes), such as in the TDBNA for pair approximation (discussed in 3.2.5), the output neighborhood can only be a single node. As a result in line 6 of Algorithm 2 instead of using $p_{s_i s_j}(t+1) \leftarrow P_B(X_F(t+1) = s_i, Nb(X_F)(t+1))$, we approximate this quantity using $p_{s_i s_j}(t+1) \leftarrow P_B(X_F(t+1) = s_i, Nb(X_F)(t))$ with another node from the input neighborhood (at time t) instead of the output neighborhood (at time $t+1$).

3.2.3 Defining I(t) using Tree Approximations

The TDBNA is a truncation approximation of the exact Dynamic Bayesian Network (DBN) for Eq. 3.4. Intuitively, in Algorithm 2, we truncate the DBN temporally and spatially (visually this would be blue and red respectively in Fig. 3.4). As a result, the number of input nodes and output nodes in a given two timestep BN are not the same. When we construct the next two step BN, we need to approximate the distribution of the new input nodes using lower order distributions queried from the output nodes in the previous network.

To better understand this, consider the TDBNA example in Fig. 3.5. In this example, each $B(t)$ has 4 input nodes and 2 output nodes. Therefore, when we move from $B(t)$ to $B(t+1)$, we cannot fully define the 4th order distribution of the input nodes of $B(t+1)$. This is because when we finish the computation on $B(t)$, the highest order distribution we can evaluate is only a

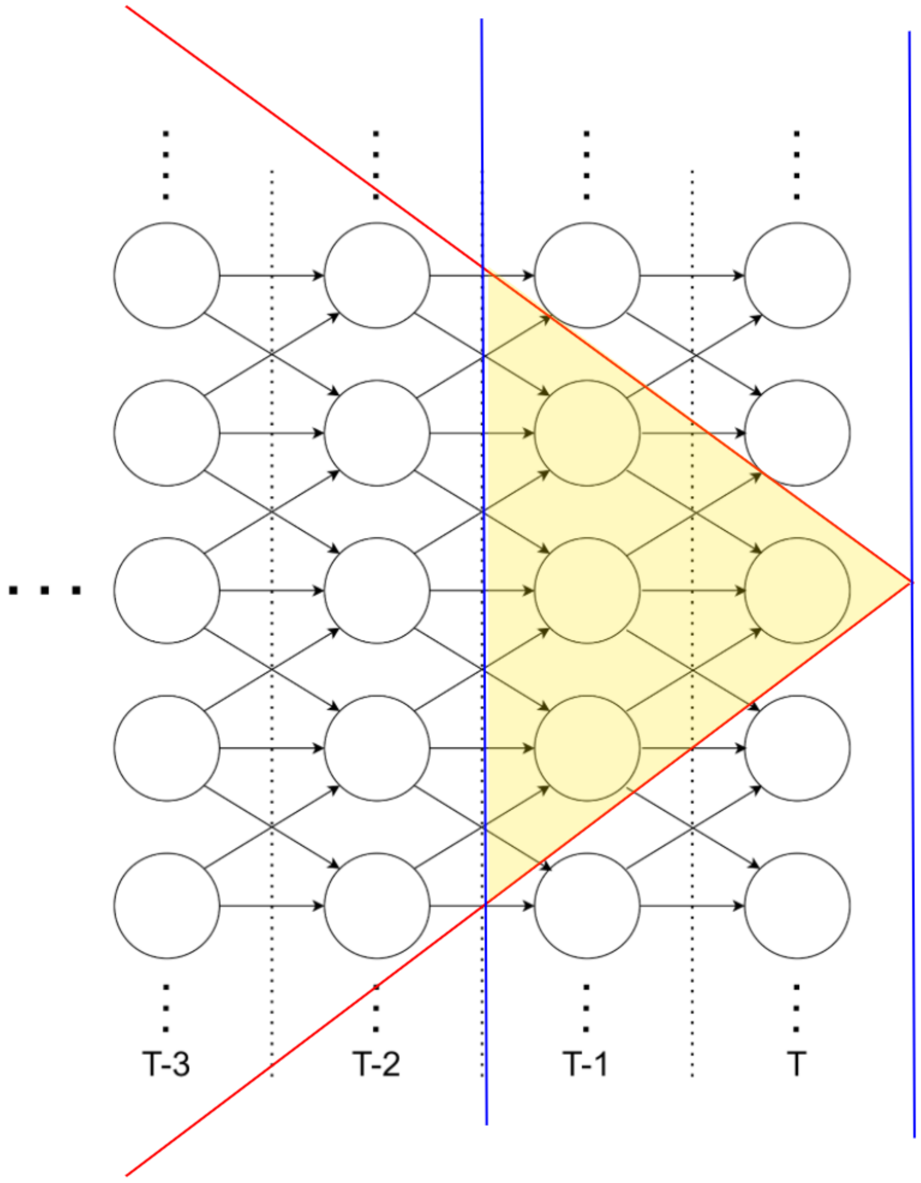


Figure 3.4: Spatial (red) and Temporal (blue) Truncation for DBN approximation

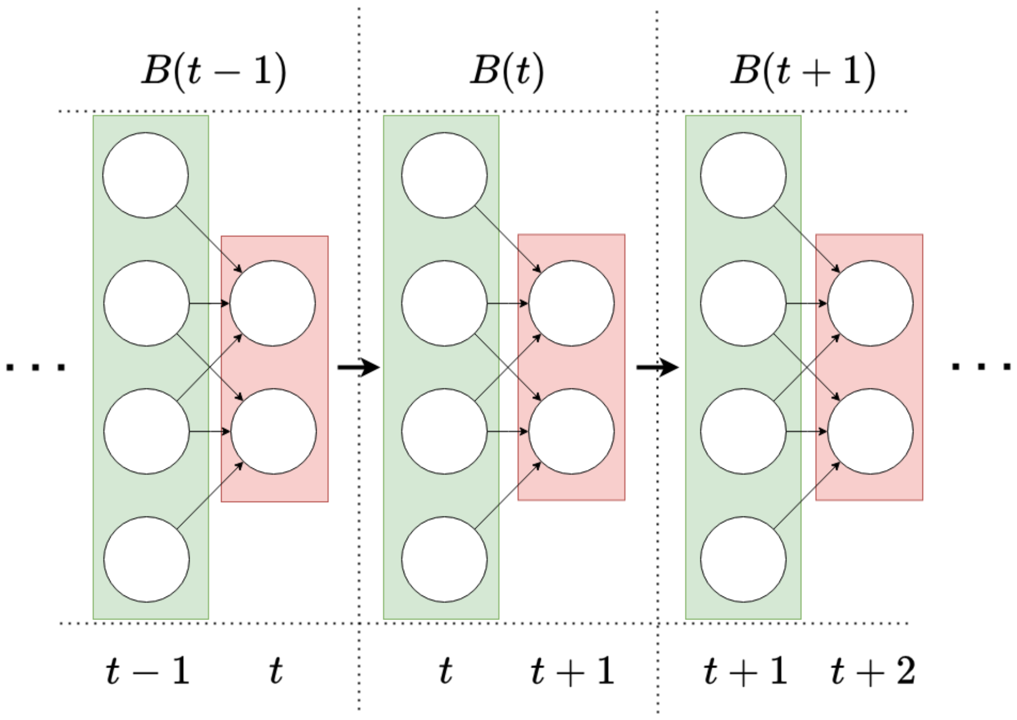


Figure 3.5: Two timestep BNs for TDBNA. Green nodes correspond to input neighborhoods, red nodes correspond to output neighborhoods.

2nd order pairwise distribution over the two output nodes.

This is why in Algorithm 2, it is necessary to have a function Ψ that defines the distribution of the input neighborhood. This function Ψ uses the first and second order distributions $p_{s_i}(t+1)$ and $p_{s_i s_j}(t+1)$ from the previous network (from t to $t+1$), to approximate the distribution of $I(t+1)$ for the next network (from $t+1$ to $t+2$).

The function Ψ can be thought of as a type of moment closure approximation [32]. In this work, in the language of the moment closure literature, we close our distributions at the pair level. For our particular Ψ function, we use a tree approximation as in Fig. 3.6 to model the distribution of input nodes.

For example, if the input set consists of 8 adjacent nodes labeled $I = (A_1, ..., A_8)$, we can

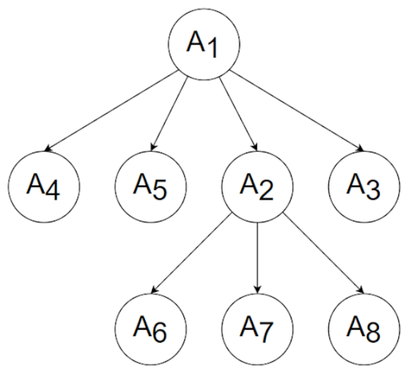


Figure 3.6: Tree approximation for input neighborhood definition

specify a tree approximation as in Fig. 3.6 where:

$$P(A_1(t) = s_i) = p_{s_i}(t)$$
$$P(A_2(t) = s_i, A_1(t) = s_j) = p_{s_i s_j}(t) \tag{3.6}$$

and so forth for each node in the tree. To select the tree for our tree approximation, we will use Breadth-First Search (BFS). We use BFS to search the input neighborhood nodes along the graph G from the original spatial Markov process starting from the focal node in the input neighborhood I. This will result in a tree as in Fig. 3.6. In this example, the function Ψ would be defined as:

$$\Psi(I(t); p_{s_i s_j}(t), p_{s_i}(t)) = P(A_1(t)) \prod_{i=2}^{8} P(A_i(t)|Anc(A_i)(t))$$
$$= P(A_1(t))P(A_2(t)|A_1(t))...P(A_6(t)|A_2(t))...P(A_8(t)|A_2(t)) \tag{3.7}$$

where $Anc(A_i)$ is the parent of A_i in the BFS tree. The conditional probabilities in the equation are defined according to Eq. 3.6 with $P(A_i(t) = s_i|A_j(t) = s_j) = \frac{P(A_i(t)=s_i, A_j(t)=s_j)}{P(A_j(t)=s_j)} = \frac{p_{s_i s_j}(t)}{p_{s_j}(t)}$.

3.2.4 Fermi Rule and Rotational Symmetry

In Section 3.1, when modeling the spatial evolutionary game with the Fermi rule, we assume that an agent looks at a randomly chosen neighboring agent to compare its payoff.

During the computation of the Fermi update rule in the spatial evolutionary game, an agent randomly selects a neighbor to update their strategy. If we assume that the spatial evolutionary game takes place on a regular structure such as a grid, the population also possesses rotational symmetry beyond spatial symmmetry. Therefore, beyond the temporal and spatial truncations shown in Fig. 3.4, we can also truncate rotationally.

When choosing $O \subset I$, instead of requiring a fully defined Fermi transition probability for each node in O, we can relax this requirement and only require a *sided* Fermi transition probability for a node in O. Recall that in Eq. 3.3, we sum over the neighborhood $N(A_{i,j})$ to compute the probability of $Pr(A_{i,j}(t+1)|A_{i,j}(t), \text{other parents})$. This corresponds to a fully defined Fermi transition probability. Instead of the summation in Eq. 3.3, in the sided Fermi rule, we can instead compute:

$$\begin{aligned} &\Pr(A_{i,j}(t+1) = s_{t+1} \mid A_{i,j}(t) = s_t, \text{other parents}) \\ &= (1-\gamma)p_\varnothing + \gamma\Big[\frac{\mu}{|S|} + (1-\mu)(p_f(k,l)p_\delta(1-p_\varnothing) + p_\varnothing)\Big] \end{aligned} \tag{3.8}$$

where $A_{k,l}$ is just one predetermined neighbor of $A_{i,j}$.

This will let us define TDBNA's with much smaller input and output neighborhoods such as BN-MF, BN-PA, and BN-Medium in Section 3.3.1 which have input neighborhoods too small to include multiple nodes with the full Fermi transition probability.

3.2.5 Pair Approximation as a Special Case

In this section, we demonstrate that there exists a set of parameters to construct a TDBNA that coincides with in pair approximation. We will show this with an example by modeling the general spatial evolutionary game in the appendix of [2]. The strategy space consists of two strategies $S = \{C, D\}$ and the evolutionary game uses asynchronous updating with the Fermi rule. We set our TDBNA parameters to be:

- Input Neighborhood:

  $\{A_{2,2}, A_{2,1}, A_{3,2}, A_{2,3}, A_{1,2}, A_{1,1}, A_{3,1}, A_{2,0}\}$

- Output Neighborhood:

  $\{A_{2,2}, A_{2,1}, A_{3,2}, A_{2,3}, A_{1,2}\}$

- Tree Approximation of $P(A_{2,2}, A_{2,1}, A_{3,2}, A_{2,3}, A_{1,2}, A_{1,1}, A_{3,1}, A_{2,0})$ uses the tree:

$$\begin{array}{lr} \text{Root:} & A_{2,2} \\ \text{Edges:} & (A_{2,2}, A_{2,1}), (A_{2,2}, A_{3,2}), (A_{2,2}, A_{2,3}), (A_{2,2}, A_{1,2}) \\ & (A_{2,1}, A_{1,1}), (A_{2,1}, A_{3,1}), (A_{2,1}, A_{2,0}) \end{array}$$

The structure of the input neighborhood can be seen in Fig. 3.7. We choose $A_{2,2}$ (the red highlighted node in the figure) to be the focal node X_F of our TDBNA. To compute p_{s_i} at the next timestep, we query $P(A_{2,2}(t+1) = s_i)$. For the Fermi rule on this small input neighborhood, we use the sided Fermi rule and assume that the focal node $A_{2,2}$ always learns from its left neighbor $A_{2,1}$. The resulting Bayesian network can be seen in Fig. 3.8.

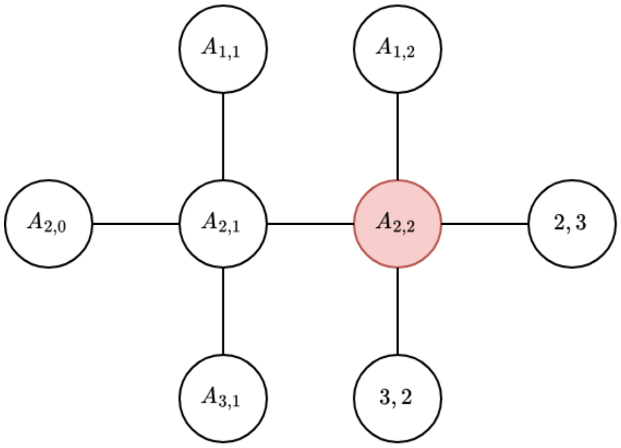


Figure 3.7: Input neighborhood for Pair Approximation

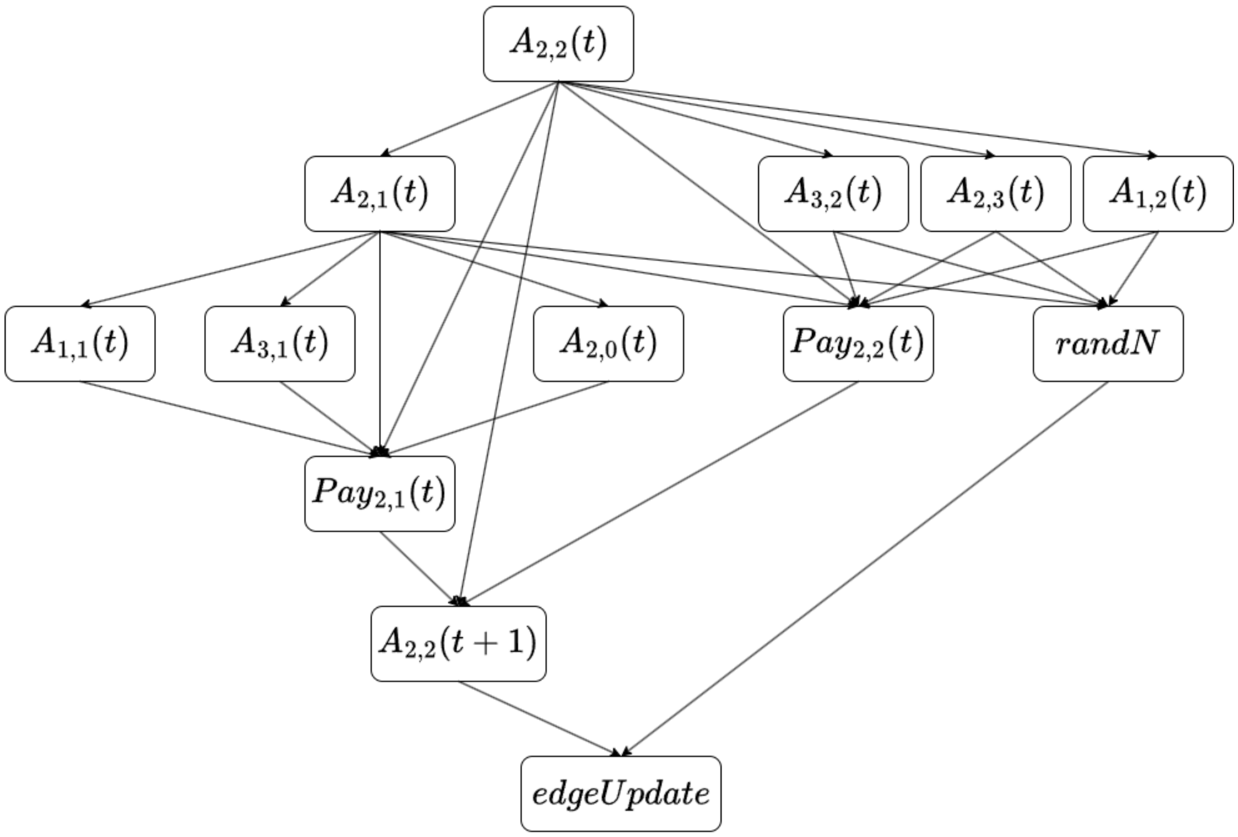


Figure 3.8: Truncated Dynamic Bayesian Network Approximation for Pair Approximation

There are additional nodes such as $randN$, $edgeUpdate$ for implementation convenience. In this network, $randN$ is a random selection over the strategies of the neighbors of $A_{2,2}$. For example, if we have the partial configuration $(A_{2,1}(t) = C, A_{3,2}(t) = C, A_{2,3}(t) = D, A_{1,2}(t) = D)$, then $P(randN = C) = 0.5$ and so forth. The $edgeUpdate$ node is a dummy node that represents the joint probability of $A_{2,2}$ and $randN$. To compute $p_{s_i s_j}$, we can query $P(edgeUpdate = s_i, s_j)$ or query $P(A_{2,2}(t+1) = s_i, randN = s_j)$, depending on which is easier to implement with the chosen inference algorithm.

To show that the computation in the TDBNA and pair approximation is the same, we will show that the computation of $P(A_{2,2}(t+1) = C)$ over the TDBNA results in a pair approximation equation. We start by considering the probability that $A_{2,2}(t+1)$ takes the value of C. We condition this on the value of $A_{2,2}(t)$:

$$P(A_{2,2}(t+1) = C) = \sum_{s \in S} P(A_{2,2}(t+1) = C \mid A_{2,2}(t) = s) P(A_{2,2}(t) = s) \tag{3.9}$$

Recall that $A_{2,2}(t)$ is distributed according to the current marginal distribution p_i:

$$P(A_{2,2}(t+1) = C) = \sum_{s \in S} p_s \cdot P(A_{2,2}(t+1) = C \mid A_{2,2}(t) = s) \tag{3.10}$$

Consider the probability that the focal player playing the strategy D switches to the strategy C. Let k_C be the number of neighbors of the focal node playing C. Since all nodes are independently defined, the nodes are C with probability $p_{C|D}$. The probability that k_C of the nodes

$A_{2,1}, A_{3,2}, A_{2,3}, A_{1,2}$ are C is:

$$p_{k_C} = \frac{d!}{k_C!(d-k_C)!} p_{C|D}^{k_C} p_{D|D}^{d-k_C} \tag{3.11}$$

Without loss of generality, we assign the neighboring C-player to be $A_{2,1}$. We can denote k'_C as the number of nodes among the neighbors of $A_{2,1}$ that are playing C and calculate the probability of a given configuration of k'_C.

$$p_{k'_C} = \frac{d-1!}{k'_C!(d-k'_C-1)!} p_{C|D}^{k'_C} p_{D|D}^{d-k'_C-1} \tag{3.12}$$

We can now condition on the nodes $A_{2,1}, A_{3,2}, A_{2,3}, A_{1,2}$ and the nodes $A_{1,1}, A_{3,1}, A_{2,0}$ using k_C and k'_C:

$$\sum_{k_C}^{d} \sum_{k'_C}^{d-1} \frac{k}{k_C} p_{k_C} p_{k'_C} P(A_{2,2}(t+1) = C \mid A_{2,2}(t) = D, k_C, k'_C) \tag{3.13}$$

The value of both payoff nodes can be uniquely obtained from the amount of cooperators k_C, k'_C among the corresponding neighbors. Given that we condition on $pay_{2,1}$, $pay_{2,2}$, $A_{2,2}$, and $A_{2,1}$, the distribution of $A_{2,2}(t+1)$ can be obtained from the CPT of $A_{2,2}(t+1)$. In this case for the asynchronous Fermi rule, we have that:

$$P(A_{2,2}(t+1) = C \mid A_{2,2}(t) = s, k_C, k'_C)$$
$$= \begin{cases} \frac{1}{M} f(Pay_C(k'_C) - Pay_D(k_C)) & \text{if } A_{2,2}(t) = D \\ \frac{M-1}{M} + \frac{1}{M}(1 - f(Pay_D(k'_C) - Pay_C(k_C))) & \text{if } A_{2,2}(t) = C \end{cases} \tag{3.14}$$

Then the final probability has the form:

$$
\begin{aligned}
P(A_{2,2}(t+1) = C) = &\sum_{k_C}^{d}\sum_{k'_C}^{d-1}\frac{k}{k_C}p_{k_C}p_{k'_C}\big[\frac{p_D}{M}f(Pay_C(k'_C) - Pay_D(k_C)) + \\
&p_C(\frac{M-1}{M} + \frac{1}{M}(1 - f(Pay_D(k'_C) - Pay_C(k_C))))\big] \qquad (3.15)
\end{aligned}
$$

We can move the probability p_C out from this expression and write an equation of the form $p_C(t+1) = p_C(t) + \Delta p_C$:

$$
\begin{aligned}
p_C(t+1) = &p_C(t) + \sum_{k_C}^{d}\sum_{k'_C}^{d-1}\frac{k}{k_C}p_{k_C}p_{k'_C}\big[\frac{p_D}{M}f(Pay_C(k'_C) - Pay_D(k_C)) + \\
&- \frac{p_C}{M}f(Pay_D(k'_C) - Pay_C(k_C))))\big] \qquad (3.16)
\end{aligned}
$$

Taking

$$\dot{p}_C = \lim_{M\to\infty}\frac{\Delta p_C}{1/M} \qquad (3.17)$$

recovers exactly the differential equation for the time evolution of proportion of C agents in the population as in [2]. We can follow this step by step expansion of the probabilities in the Bayesian Network for the pair level probabilities and arrive at the corresponding equations for the time evolution of the pair level quantity p_{CC}.

3.2.6 Features

Here we summarize the main features of the TDBNA. The two main features roughly correspond to the two major ideas present in higher order approximations of spatial evolutionary games:

- **Neighborhood Configurations**: TDBNAs generalizes the idea of neighborhood configurations [31]. The *input definition* step allows for the choice of arbitrary configurations to compute transition probabilities. For any choice of sites or focal agents in existing approximations, there exists an equivalent input neighborhood that captures the same set of configurations. We showed this earlier using pair approximation, but it is also possible to find input neighborhoods that correspond to more complex approximations such as triplet or n-point approximations [17].

- **Moment Closure**: In a TDBNA, moment closure is handled through the query step (which quantities to evaluate p_{s_i}, $p_{s_is_j}$, etc.) and input definition step Ψ. The query step defines which sets of moments we will extract at each time step. The input definition step defines how we want to use these moments. In contrast to existing methods, TDBNAs allow for the query of arbitrary subsets of k-th order moments.

  For example, in pair approximation on a two strategy game, all four of the 2nd order moments: $(p_{s_1s_1}, p_{s_1,s_2}, p_{s_2s_1}, p_{s_2s_2})$ are needed when computing the pair approximation equations. In contrast, in a TDBNA, when choosing a Ψ, we are not required to query all $p_{s_is_j}$ from the previous network. Instead we could just query two moments $(p_{s_1s_1}, p_{s_2s_2})$ and approximate $p_{s_1s_2} = p_{s_1}p_{s_2}$, $p_{s_2s_1} = p_{s_1}p_{s_2}$ in the function Ψ. In a game with a large amount of strategies, we could intelligently choose select a subset of all of the 2nd order moments instead of all of the moments. Additionally, we could even query higher order moments (e.g. $p_{s_1s_1s_1}$) if our output neighborhood is large enough without evaluating all of the higher order moments of a given level. This is contrast to pair/triplet/n-point approximations in literature where it is always necessary to keep track of all moments on a given order.

This can be advantageous from a efficiency/accuracy perspective as some moments will impact the behavior of the system more than others. Furthermore, since TDBNAs are an iterative algorithm, it is possible to dynamically choose which moments to query over time. There are many possibilities in how to handle this generalization of moment closure, but we will leave this research for future work.

One notable advantage we get from generalizing the idea of neighborhood configurations is that the level of accuracy can be adjusted by adjusting the size of the input neighborhood. If we consider the TDBNA a truncation of the full DBN, a larger neighborhood size will reduce the difference between our approximation and the full model. To show this effect in practice, we run a several empirical tests on different evolutionary games.

3.3 Empirical Evaluation

We compare the results of the Truncated DBN Approximation on selection of commonly encountered games in evolutionary game theory literature (see Table 3.3). Formally, we define our empirical task to be: given a spatial evolutionary game (S, U, F) with initial condition $p_{s_i}(0)$, we want to compute an approximation of $p_{s_i}(t)$. For our spatial evolutionary game, we will choose our spatial structure G to be a grid domain (where $d = 4$). We will use the Fermi rule as the update rule for our game. The payoff matrices we will use in our experiments are the Prisoner's Dilemma, a symmetric version of the Battle of Sexes, and the Snowdrift game (see Table 3.3 for the corresponding payoff matrices).

We computed the ground truth values by stochastic simulations of the spatial evolutionary

Table 3.2: Approximation Framework Parameters

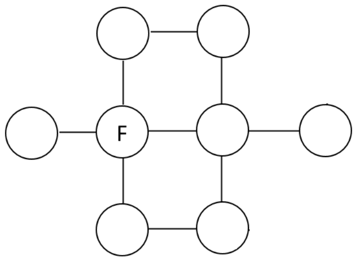
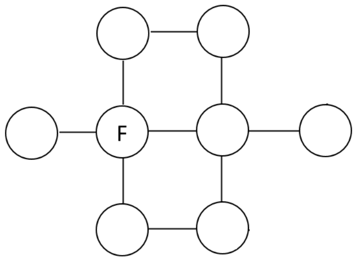
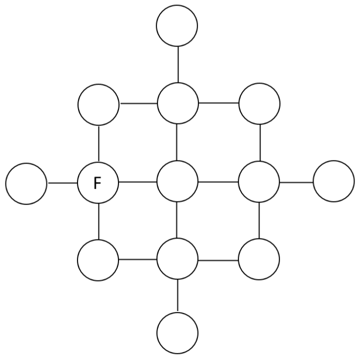
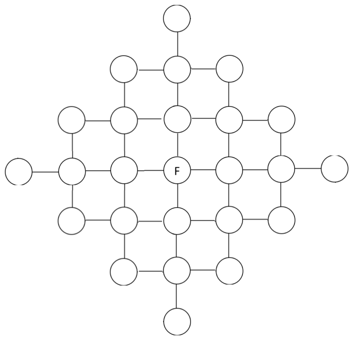
Name	Input Neighborhood	Query/Input Definition
BN-MF		p_{s_i}, i.i.d
BN-PA		p_{s_i}, $p_{s_i\|s_j}$, BFS
BN-Medium		p_{s_i}, $p_{s_i\|s_j}$, BFS
BN-Large		p_{s_i}, $p_{s_i\|s_j}$, BFS

Table 3.3: Two Strategy Evolutionary Game Payoff Matrices

Game Name	Payoff Matrix
Prisoner's Dilemma (pd)	$\begin{pmatrix} 2 & -1 \\ 3 & 0 \end{pmatrix}$
Snowdrift (sd)	$\begin{pmatrix} 2 & 1 \\ 3 & 0 \end{pmatrix}$
Battle of the Sexes (bos), symmetric version [54]	$\begin{pmatrix} 0 & 1 \\ 2 & 0 \end{pmatrix}$

games on a 50 x 50 grid. The ground truth values for $p_{s_i}(t)$ are obtained by averaging the results of 20 separate simulations on the 50 x 50 grid.

3.3.1 Methods

We consider four different TDBNAs each with different parameter settings. We want to show that as expected, larger input neighborhoods improve the resulting approximation. The various scenarios for each of the approximations tested are shown in Table 3.2. Each of the models smaller than BN-Large need to be augmented with additional assumptions resulting in adjustments to Algorithm 2 because several components of the TDBNA algorithm are not well-defined if the output neighborhood is smaller than 5 nodes. A full description of each setting is as follows:

- **BN-MF**: This version of Algorithm 2 uses an input neighborhood I with 8 nodes and an output neighborhood O of a single node X_F. We will only query $p_{s_i}(t)$ at each iteration (lines 2 and 8 in Algorithm 2 are removed). To model the transition probability for $X_F(t+1)$, we use a sided Fermi rule (see Eq. 3.8) such that the focal node always learns from the agent to the right of itself on the evolutionary game grid.

For Ψ, instead of a tree approximation, we use a simple product approximation:

$$\Psi(I(t)) = \prod_{A_i \in I(t)} P(A_i) \tag{3.18}$$

This version of the algorithm corresponds to a mean field approximation in evolutionary game theory literature.

- **BN-PA**: This version of Algorithm 2 is described in Section 3.2.5. This scenario uses an input neighborhood I with 8 nodes and an output neighborhood O of a single node X_F. Like in BN-MF, a sided Fermi rule is used for the transition probabilities of $X_F(t+1)$. On line 6, instead of computing $p_{s_i s_j}(t+1) \leftarrow P_B(X_F(t+1) = s_i, Nb(X_F)(t+1))$, we compute $p_{s_i s_j}(t+1) \leftarrow P_B(X_F(t+1) = s_i, Nb(X_F)(t))$. For Ψ, we use a tree approximation constructed using BFS on the input neighborhood graph shown in Table 3.2 starting from the node labeled with an "F". This version of the TDBNA corresponds to pair approximation in evolutionary game theory literature.

- **BN-Medium**: BN-Medium uses an input neighborhood I with 13 nodes and an output neighborhood O of two nodes $\{X_F, Nb(X_F)\}$ (the node labeled "F" and the node directly to the right of it in Table 3.2). A sided Fermi rule is used for X_F, but a non-sided (regular) Fermi rule is used for $Nb(X_F)$. Ψ is a tree approximation like in BN-PA. On line 6 of

Algorithm 2, we compute the weighted average:

$$p_{s_i s_j}(t+1) \leftarrow \frac{1}{d} P_B(X_F(t+1) = s_i, Nb(X_F)(t)) \\ + \frac{d-1}{d} P_B(X_F(t) = s_i) P_B(X_F(t) = s_j) \tag{3.19}$$

The above weighted average is a explicit representation of the probability of $X_F(t+1)$ conditioned on a randomly selected neighbor (its neighbor $Nb(X_F)(t+1)$ in O or one of the three neighbors at t that are not in the output neighborhood).

- **BN-Large**: BN-Large uses an input neighborhood I of 25 nodes and an output neighborhood O of 5 nodes (the node labeled F and its four neighbors). The Fermi rule can be directly modeled on a neighborhood of this size. The function Ψ is a tree approximation like in BN-PA. No additional changes are made to the computation in Algorithm 2.

At the largest size, BN-Large captures all the nodes that can impact a focal node directly and all their direct neighbors for the course of 1 iterations, so it is actually an exact model for 1 iteration of the game. All methods use use Bucket Elimination as described in 2.3.2.1 for the probabilistic inference routine in Algorithm 2.

3.3.2 Results

Fig. 3.9 shows the time evolution of the first strategy p_{s_2} in a selection of three games. In the Prisoner's Dilemma, we plot $p_{s_2} = p_D$, the proportion of defectors (agents playing strategy D) over time. The ground truth trajectory is displayed as a blue line labeled "Simulation". Estimated trajectories for p_{s_2} are plotted for our various TDBNA scenarios.

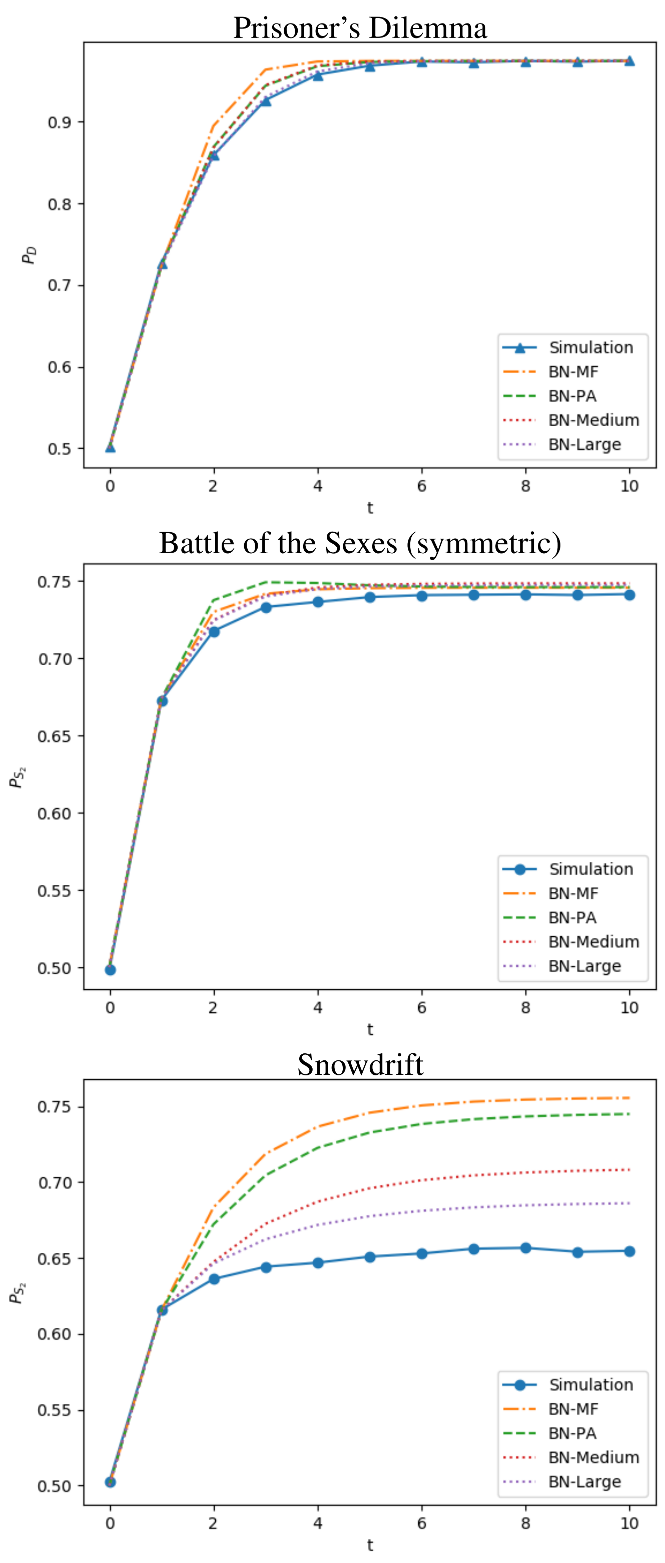


Figure 3.9: Proportion of agents playing the second strategy in a Prisoner's Dilemma game (top) and a symmetric Battle of the Sexes game (middle), and a Snowdrift game (bottom) for different approximations

Our results show that in games (Prisoner's Dilemma, symmetric Battle of the Sexes) where pair approximation (BN-PA) already obtains a good agreement with the simulation results (Fig. 3.9), the larger approximation neighborhoods reduce the quantitative error in the time evolution graphs. In these cases, all approximation methods converge to the same equilibrium as the simulation. However, larger neighborhoods yield more accurate values for the rate of change of each strategy over time before equilibrium.

Previous research in pair approximation on 2x2 games (see [55, Section 3.8]) has indicated that pair approximation does not have good quantitative agreement with simulation results in Stag Hunt and Snowdrift games. Fig. 3.9 demonstrates this phenomena for a Snowdrift game, one of the pathological cases where pair approximation does not converge to the same equilibrium as the simulation. In this case we see clearly that choosing a larger neighborhood decreases the error between the approximation and the simulation. This suggests that games where the difference between pair approximation and larger neighborhood approximation is large are also games where the difference between pair approximation and the simulation is large. Using this difference as a guideline, we can determine what games it is sufficient to use pair approximation and what games where it is necessary to rely on a larger neighborhood approximation.

Interestingly, on the tested games we observe that by increasing the neighborhood size we can obtain results approaching the simulation results even without using higher order probabilistic factors. This suggests that it is not necessary to use higher order moments to improve accuracy as discussed in Section 3.2.6. To improve our approximations, we can simply increase the input (and output) neighborhoods to a desired size.

The empirical results in this chapter are limited to a selection of two-player evolutionary games. As will be discussed in Chapter 4, this is due to limitations on computational complexity.

For games with 3 or more strategies, it becomes difficult to query a the BN-Large TDBNA using Bucket Elimination. To solve this, we will propose new approximate methods for accelerating this computation.

3.4 Summary

This chapter presents a dynamic Bayesian network formulation of spatial evolutionary games, thus making Bayesian network techniques applicable to such games. Since exact inference on large Bayesian networks is intractable, we accompany our formulation with a new flexible approximation scheme that is tailored to the inherent symmetry in spatial games by using a truncated neighborhood of agents that lead to truncated DBNs, which are then solved by exact inference algorithms. The truncated neighborhood can be arbitrarily defined up to the size of the population in the original stochastic simulations. By controlling the neighborhood size we can control the strength of the approximation and its complexity. We further show that certain approximated DBNs can be used to recover discrete analogs of existing pair approximations in literature. Our empirical results illustrate the potential of this approach.

Chapter 4: Surrogate Bayesian Networks for Approximating Spatial Games

In this chapter, we develop a novel method for accelerating inference on Bayesian Networks for evolutionary games. One important technique we analyzed in section 3.3 was increasing the size of the *input neighborhood*, the number of agents represented by the Bayesian Network at an initial time step t. Increasing the number of agents in the input improves the accuracy of the distributions that we query at the next iteration. However, increasing the size of the network is not without demerits. To query the distributions at the next iteration, it is necessary to perform probabilistic inference on the Bayesian Network that we generate. As mentioned in Chapter 2, one way to do this is using bucket elimination.

Recall that the time complexity of variable elimination algorithms and exact probabilistic inference algorithms in general is dependent on a graph property called the *induced width*. For example, in bucket elimination, the number of operations that need to be performed in a single bucket is $O(|D|^{|B|})$, where $|D|$ is the size of variable domains (assuming each variable has the same domain size) and where $|B|$ is the maximum number of variables in a bucket. Observe in table 4.1 how the induced width of the networks grows as we increase the input neighborhood size. In this comparison we are only increasing $|B|$, the number of variables in each bucket. This behavior is even more extreme when we increase the number of strategies and thus increase $|D|$. Even though the largest approximation we formulated in Chapter 3 (BN-Large) is within

Table 4.1: Induced widths of different evolutionary game Bayesian Networks

Network Name	Input Neighborhood Size	Number of Variables	Induced Width
BN-MF	8	11	6
BN-PA	8	13	7
BN-Medium	13	24	11
BN-Large	25	60	20

reasonable computation bounds for 2 strategy evolutionary games, this becomes intractable for 3 strategy evolutionary games and above.

The goal of this chapter is to develop an approximate inference algorithm for cases such as the BN-Large network. Our goals are twofold:

1. The first goal is that of accuracy. Namely, to produce results that are close to the ground truth simulation results. Specifically, the pairwise distribution p_{s_1,s_2} is the quantity of interest when we use the Kullback-Leibler divergence $D_{KL}(P||Q)$ as the measure of statistical distance between distributions. As a general baseline, we expect the new algorithm to be able to be more accurate than sampling methods such as forward Monte Carlo sampling on BN-Large, Pair Approximation, and state of the art sampling approaches such as Abstraction Sampling [47] when given the same amount of computational resources [48].

2. The second goal is to significantly decrease the amount of time needed for inference. In particular, we will derive a new algorithm capable of performing inference on high strategy evolutionary games, where existing inference approaches are intractable.

This brings us to the focus of this chapter: surrogate Bayesian Networks, a novel method for computing approximate inference on large evolutionary game Bayesian Networks. For example, on BN-Large, this new method would involve performing probabilistic inference on a parameter extended BN-PA (the surrogate network), greatly accelerating inference time. We start with a few motivating observations on the behavior of approximate inference algorithms such as Abstraction Sampling on TDBNAs in Section 4.1. This will be followed by a description of our basic surrogate network maximum likelihood estimation (MLE) algorithm in Section 4.3. We develop an improved algorithm termed KL-search for inference by devising a sample/search method for generating informative samples for our MLE algorithm in Section 4.4. We will empirically show in Section 4.5 that our methods produce superior results to existing approximate inference algorithms such as abstraction sampling when applied to a TDBNA.

4.1 Problem Statement

In Chapter 3, we presented TDBNAs, which were used to approximate the forward dynamics of spatial evolutionary game models. It was shown that larger input neighborhoods produce more accurate approximations of the underlying dynamics. However, it is computationally expensive to evaluate a TDBNA for larger neighborhoods. Running exact inference on a 25 node TDBNA[1] for a simple three strategy evolutionary game requires performing inference on a network with an induced width greater than 30. This becomes even more impractical for more complex evolutionary games with larger numbers of strategies.

Notation: In this chapter, we will use node/variable interchangeably. We will denote by $Nb(X)$ an arbitrary neighbor of a focal node denoted by X. A bold $\mathbf{X}$ refers to all variables in a network. Superscripts such as $\mathbf{x}^{(j)}$ will be used to index samples (full configurations) over a network.

[1]In this context, 25 node refers to the size of the input neighborhood

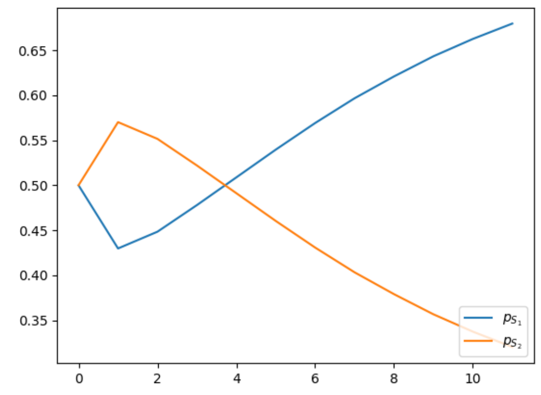
4.2 Behavior of Abstraction Sampling on TDBNA

When a TDBNA is too large, we can apply approximate inference techniques. One such approximate inference technique is Abstraction Sampling (AS) [47, 48]. While effective on multiple benchmarks in the previous literature in reducing the error when estimating the partition function of graphical models, it less effective when applied to our particular application setting. For our motivating example, we will study the 2 strategy Deadlock game (see Table 4.6 for the payoff matrix).

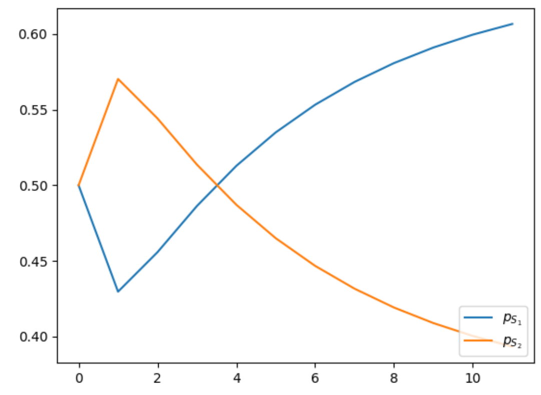
For this case study, we will use Abstraction Sampling with a WMBE heuristic (ibound of 10) as the inference routine for line 6 of Algorithm 2. For each time iteration of the TDBNA BN-Large, we generate 100 probes (roughly 75 seconds for 100 probes) on the corresponding search tree. Each probe is conditioned on a specific configuration of the two nodes X and $Nb(X)$ in the output neighborhood: $(X = s_1, Nb(X) = s_1), (X = s_1, Nb(X) = s_2), (X = s_2, Nb(X) = s_1), (X = s_2, Nb(X) = s_2)$. Probes conditioned on partial configurations will only expand branches of the search tree that are consistent with those partial configurations. The partition function estimates from these conditioned probes are used to compute $P(X = s_1, Nb(X) = s_2)$. We split the probes such that each partial configuration receives an equal number of probes, so that we have $\frac{L}{(|S|)^2}$ probes per strategy pair, where $L = 100$ is the number of samples.

4.2.1 Results on Deadlock

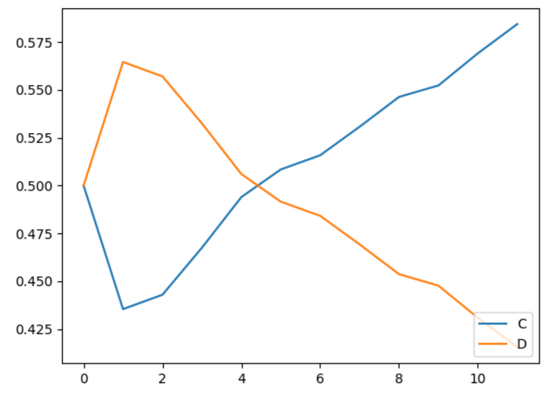
We evaluate time evolution of p_{s_i} in the game Deadlock starting from an initial condition of $p_{s_1}(t = 0) = 0.5$ that is randomly mixed $(p_{s_i s_j}(t = 0) = p_{s_i}(t = 0)p_{s_j}(t = 0)$ and so forth for higher order distributions). We show the performance of AS in Fig. 4.1. The figure



(a)



(b)



(c)

Figure 4.1: Simulation (3000×3000 grid) ground truth (a), compared with exact 8 node TDBNA result (b), and approximate Abstraction Sampling result on 25 node TDBNA (c)

depicts the time evolution of p_{s_i} for both strategies over time using three different methods of computation: agent based evolutionary game simulation, BN-PA TDBNA with exact inference, and BN-Large with AS. As can be seen in the time evolution curves of Fig. 4.1, when using Abstraction Sampling, there can be significant error when we apply this sampling method to TDBNAs. More specifically, in a method by method breakdown:

- **Simulation**: In the simulation on a 3000 by 3000 grid (top graph of Fig. 4.1), Deadlock shows a reversal pattern at $t = 1$. There is another discontinuous derivative at $t = 2$. After which, the graph is smooth and $p_{s_1}(t = 10) \approx 0.66$ (the probability of agents in the simulation playing the first strategy at time = 10). We will treat the simulation trajectory as the ground truth trajectory.

- **BN-PA**: In the result of BN-PA with bucket elimination (middle graph of Fig. 4.1), we have the same reversal pattern at $t = 1$, however the slope of p_{s_1} curves down much faster than in the simulation. This results in $p_{s_1}(t = 10) \approx 0.6$.

- **BN-Large with Abstraction Sampling**: In the result for Abstraction Sampling in the bottom graph of Fig. 4.1), we can qualitatively see a reversal at $t = 1$ and a second small discontinuous derivative at $t = 2$. However, afterwards, it seems the trajectory is unstable due to noise and we get $p_{s_1}(t = 10) \approx 0.57$.

Since we start from a randomly mixed initial condition, BN-PA provides an exact solution[2] for p_{s_1} at $t = 1$. Specifically, the ground truth value at $t = 1$ is $p_{s_1}(t = 1) = 0.4297$, which is obtained by both BN-PA and the simulation. However, there is a noticeable discrepancy in

[2]Since the initial distribution is randomly mixed, the tree approximation of the input distribution in BN-PA is exact at $t = 0$.

$p_{s_1}(t=1)$ for BN-Large with AS for which $p_{s_1}(t=1) \approx 0.435$. This discrepancy can largely explain the error in the BN-Large results. In general, errors at early time-steps will gradually carry over and produce larger errors later on. We can also quantify this error by looking at the difference between second order distributions at time $t=1$. In particular, if we calculate the KL divergences:

$$D_{KL}(P_{sim}(X(1), Nb(X)(1))||P_{abs}(X(1), Nb(X)(1))$$
$$D_{KL}(P_{sim}(X(1), Nb(X)(1))||P_{pa}(X(1), Nb(X)(1)) \tag{4.1}$$

where $P_{sim}(X(1), Nb(X)(1))$ is the distribution of an agent X at time $t=1$ and an arbitrary neighbor $Nb(X)$ at time $t=1$ as computed by the simulation, we get:

$$D_{KL}(P_{sim}(X(1), Nb(X)(1))||P_{abs}(X(1), Nb(X)(1)) = 0.011395$$
$$D_{KL}(P_{sim}(X(1), Nb(X)(1))||P_{pa}(X(1), Nb(X)(1)) = 0.008374 \tag{4.2}$$

The divergence for BN-PA (denoted by pa) is less than the divergence for BN-Large using abstraction sampling (denoted by abs). Higher divergence values are directly correlated with a larger difference in the trajectory curves. If we run Abstraction sampling with more samples, the KL-divergence will gradually approach zero[3]. Our goal will be to design a new method that reduces the error more effectively than just improving AS by running it with more samples.

[3]With a randomly mixed initial distribution, the input neighborhood tree approximation for BN-Large is exact. Thus, exact inference for any joint probability up to the size of the number of fully defined nodes (without the sided Fermi rule in Section 3.2.4) in the output neighborhood at $t=1$ is exact. We assume that AS will eventually approach exact inference given enough samples so the KL-divergence will also approach zero.

4.2.2 Approach

Our goal for this chapter will be to design an approximate inference algorithm to replace exact inference on line 6 of Algorithm 2. As seen in the results on Deadlock, simply replacing exact inference with Abstraction Sampling runs into limitations. Specifically:

- Accuracy: It can take many samples for Abstraction Sampling on BN-Large to produce better results compared to just running exact inference on BN-PA.

- Time: Running exact inference on BN-PA is much faster than AS on BN-Large. In particular, to compute the trajectories seen in Fig. 4.1, it takes around 778s using AS on BN-Large while only 1.4s using BE on BN-PA (see Table 4.5).

- Noise: Without sufficient sample convergence in AS, it is difficult to determine certain quantities such as the location of inflection points, which can also impact qualitative results. For example, in the Fig. 4.1, the trajectory computed using AS has multiple inflection points, when there is only 1 (at around t = 3) for the simulation and BN-PA.

To address these issues, we propose a method that combines the estimate from the smaller network with estimates from the larger network. The idea is to leverage the advantages inherent to smaller TDBNAs. Specifically, we can compute a reasonably close trajectory to the ground truth (simulation results) very fast using exact inference on BN-PA. However, we cannot improve the results from BN-PA by giving the algorithm more time since the inference is already exact (in contrast to running AS on BN-Large where the results will get better with more time).

The main idea will be to replace line 6 of Algorithm 2 with exact inference on a *surrogate network*. The surrogate network is a smaller TDBNA that has been extended with additional

Table 4.2: Summary of algorithm traits

Algorithm	Speed	Init Acc.	Improve w/ time?	Noise
BE on BN-Large	Intractable	High	N	N
AS on BN-Large	Slow	Poor	Y	Y
BE on BN-PA	Fast	Medium	N	N
BE on Surrogate Network	Medium	Medium	Y	(See Section 4.4.4)

learnable parameters. We will use samples from the larger TDBNA to learn the parameters on the smaller TDBNA. Running this new algorithm for more samples will let us learn a surrogate network that is closer to the larger TDBNA. The advantages of this approach are summarized by Table 4.2.

4.3 Surrogate Maximum Likelihood Models

Consider an 8 node TDBNA (BN-PA) and a 25 node TDBNA (BN-Large) for approximating a spatial evolutionary game located on a grid structure (each agent has four neighbors). We will illustrate the approach on 8 and 25 node TDBNAs (the number of nodes referring to the size of the input set including the focal node as in Fig. 4.2), but in principle this work can also be extended to other pairs of TDBNAs with the smaller network serving as the surrogate of the larger network.

A key observation is that the smaller 8 node network is a subset of the larger 25 node network. Our idea is to extend the 8 node TDBNA with additional parameterized nodes/connections so it can approximate the 25 node network while still being small enough to facilitate tractable exact inference.

4.3.1 Parameterizing the New Edges

In order to increase accuracy of our 8 node network B, we propose to add a new set of 4 dummy neighbor nodes $Nb(X)(t+1)$ that capture the output neighborhood (see footnote for why these 4 nodes[4]). We add new edges from the input neighborhood to the output neighborhood as shown in Figure 4.3. This will yield a new surrogate Bayesian network for which some new CPTs must be defined. For our example, we specify new conditional probability tables (CPT) conditioned on two nodes in the input neighborhood (a new neighbor node $Nb(X)(t+1)$ will depend on $Nb(X)(t)$ and $X(t)$). These new CPTs are parameterized by $\theta_{x,y,z} = p_{x|yz}$. Since we inherit the assumption that all neighboring nodes are indistinguishable from 25 node network, all CPTs are identical and we can share parameters those across the various CPTs. We denote a specific surrogate network parameterized by θ as B_θ. The CPTs will be learned using Maximum Likelihood Estimation (MLE) as explained next.

Given a 25 node TDBNA model and an 8 node extended model, the basic **Maximum Like-**

[4]This is because when we compute $p_{s_i s_j}$ in the 8 node network, we approximate it as $P(X(t+1) = s_i, Nb(X)(t) = s_j)$. By adding four additional neighbor nodes in the output neighborhood, we can use exact inference to compute the original probability $P(X(t+1) = s_i, Nb(X)(t+1) = s_j)$.

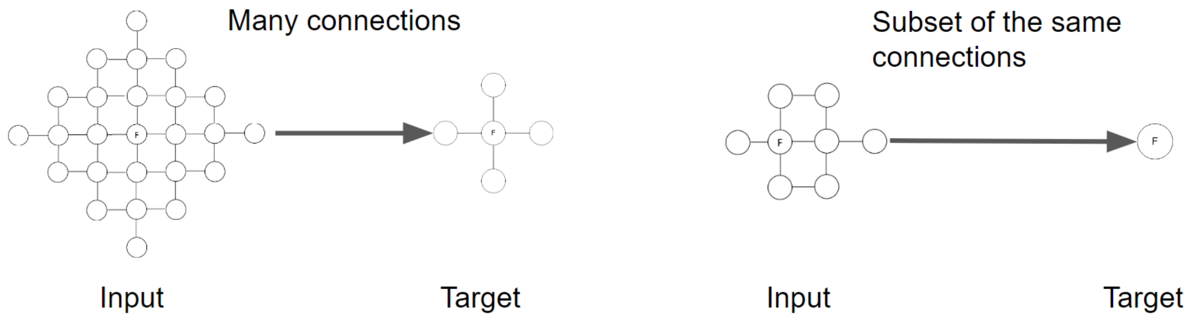


Figure 4.2: The input and target neighborhoods for a 25 node TDBNA (left) and 8 node TDBNA (right) for a spatial evolutionary game

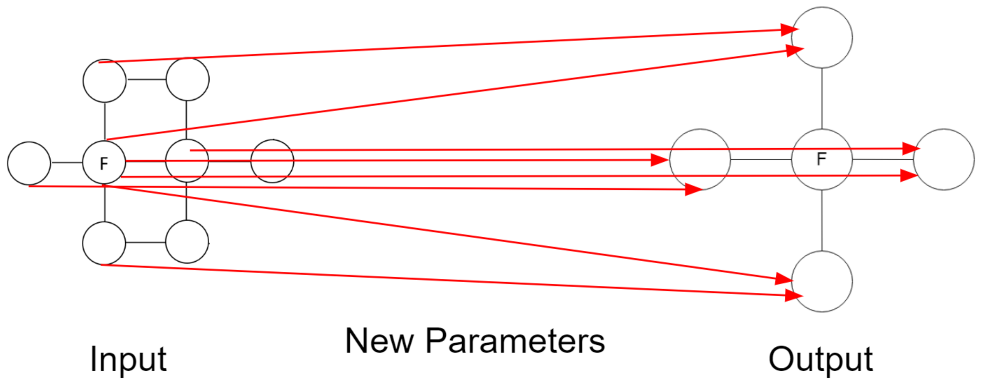


Figure 4.3: New parameters to add to 8 node network for MLE estimation. Two new edges connect each neighbor node in the input set to the corresponding new dummy neighbor node in the output set.

lihood Estimation (MLE) algorithm for learning the extended 8-node network is as follows:

1. Perform forward sampling on the 25 node TDBNA model for L samples. Each sample is a full configuration of all variables in the 25 node network. Denote each full configuration as an individual data point $\mathbf{x}^{(j)}$.

2. The extended 8-node network is fully observed for each data since its variables are subsumed in the large network. Consequently, we can derive the following likelihood expression for our surrogate network B_θ, $\mathcal{L}(B_\theta)$:

$$\begin{aligned}\mathcal{L}(B_\theta) &= \prod_{j=1}^{L} P_{B_\theta}(\mathbf{x}^{(j)}) \\ &= \prod_{j=1}^{L} P_{B_\theta}(Nb(x)^{(j)}(t+1)|x^{(j)}(t), Nb(x)^{(j)}(t)) \cdot C \\ &= \prod_{j=1}^{L} C \prod_{k=1}^{4} \theta(Nb_k(x)^{(j)}(t+1), x^{(j)}(t), Nb_k(x)^{(j)}(t)) \end{aligned} \tag{4.3}$$

where $x^{(j)}(t)$ denotes the assignment to X for the j-th data point and at time t (and $Nb_k(x)^{(j)}$ denotes the assignment to the k-th neighbor node). Because the likelihood of a Bayesian network is just a product of probabilities from each CPT, we can simplify P_{B_θ} for the 8-node extended network to just a product of the parameterized CPTs multiplied with a a constant term C that doesn't depend on θ. The constant term C can then be ignored when maximizing the corresponding sum of log likelihoods.

3. Recall from the construction of TDBNAs, the marginal distribution of an agent node is equivalent to the marginal distribution of any other agent node. This means $P(Nb_k(X)) = P(X)$ for any neighbor node $Nb_k(X)$. However, if we directly maximize Eq. 4.3, there is a chance that we end up with a model where $P(Nb_k(X))(t+1) \neq P(X)(t+1)$. To address this, we can define a set of symmetry constraints:

$$p_{s_i}(t+1) = \sum_{s_j, s_k \in S} \theta_{s_i, s_j, s_k} \cdot p_{s_j s_k}(t), \quad \forall s_i \in S \tag{4.4}$$

to ensure $P(Nb_k(X))(t+1) = P(X)(t+1)$ in B_θ. We pre-compute $p_{s_i}(t+1)$ on a non-parameter extended 8 node network and use this to constrain our log likelihood maximization derived from Eq. 4.3. Thus, we maximize the log likelihood $\log \mathcal{L}(B_\theta)$ subject to:

$$\theta^* = \arg\max_\theta \sum_{j=1}^{L} \sum_{k=1}^{4} \log \theta(Nb_k(x)^{(j)}(t+1), x^{(j)}(t), Nb_k(x)^{(j)}(t))$$

$$\text{subject to}$$

$$p_{s_i}(t+1) = \sum_{s_j, s_k \in S} \theta_{s_i, s_j, s_k} \cdot p_{s_j s_k}(t), \ \forall s_i \in S \tag{4.5}$$

This can be solved with existing constrained maximization solvers such as sequential least squares [56] or trust-region algorithms which can be found in the SciPy python package [57].

Once θ^* is learned, we can evaluate the forward dynamics (compute $p_{s_i s_j}(t+1)$) on the extended network B_{θ^*}. The algorithm is summarized in Algorithm 3.

Algorithm 3: Maximum Likelihood Estimation for Surrogate Network Learning
Input: BN-Large network A, BN-PA network B, surrogate network B_θ, initial distribution $p_{s_i s_j}(t)$, and pre-computed output distribution $p_{s_i}(t+1)$
Parameters: Number of samples L
Output: θ^*, parameters for surrogate network CPTs

```
1 for $j = 1 \to L$ do
2     Forward sample $\mathbf{x}^{(j)}$, a full configuration of $A$ using Monte Carlo forward sampling;
3 end
4 Solve $\theta^* = \arg\max_\theta \sum_{j=1}^{L} \sum_{k=1}^{4} \theta(Nb_k(x)^{(j)}(t+1), x^{(j)}(t), Nb_k(x)^{(j)}(t))$, subject to
5 $p_{s_i}(t+1) = \sum_{s_j, s_k \in S} \theta_{s_i, s_j, s_k} \cdot p_{s_j s_k}(t), \;\; \forall s_i \in S$;
6 Return $\theta^*$;
```

4.4 KL-based Search Tree Exploration

In the basic MLE algorithm, we used Monte Carlo forward sampling to generate the samples, however, we can devise a more intelligent way to generate samples. Our goal is to find samples that reduce the difference[5] between our surrogate model (the 8 node model) and the model we are approximating (the 25 node model). It is possible for the samples that we draw to not really inform us well on how to reduce this difference. This is because high probability areas in the 25 node model do not directly correspond to areas where the 8 and 25 node models differ greatly. **Key idea: instead of drawing random samples, we will combine sampling with search aiming for diverse samples that are likely to reduce the KL-divergence between**

[5]It is well known (see 6.4.1 in [58]) that maximizing likelihood is equivalent to minimizing KL-divergence. For the rest of this discussion, we talk in terms of minimizing the distance (KL-divergence) between the two models.

our surrogate model (the 8 node model) and the model we are approximating (the 25 node model).

From this point onward, we'll refer to the the 25 node model as model A and the 8 node model as model B for simplicity. Furthermore, we will assume bi-valued variables (corresponding to evolutionary games with two strategies) and use P_A, and P_B to denote distributions on model A and model B, respectively. We will use P_{B_θ} to denote the distribution of the extended model B parameterized by θ that we aim to learn.

4.4.1 Informative Sample Generation

In order to guarantee that all the samples are different[6] and aim towards meaningful samples we generate a fixed number L of partial configurations from the large network A using best-first search along some variable ordering guided by a heuristic function. For example, the first two nodes that are generated are associated with the root variable X_1 and represent the partial configurations $(X_1 = 0)$ and $(X_1 = 1)$. In the best-first search, we want a heuristic that will explore branches of high difference between the models. To do this, we define a new KL heuristic for the partial configuration $(X_1 = 0)$ as:

$$h_{kl}(X_1{=}0) = [\log(P_A(X_1{=}0)) - \log(P_B(X_1{=}0))] \cdot P_B(X_1{=}0) \tag{4.6}$$

This heuristic has an interesting property where if we expand all of the nodes at a given level and sum all of the KL heuristic values on that level, we obtain the KL divergence between the distributions represented by A and B on the subset of variables that have been expanded at that

[6]Having all samples be different means that all samples will provide information. We can compute the exact probability $P_A(\mathbf{x})$ for any sample $\mathbf{x}$, so duplicate samples do not give any meaningful information.

level. For example at the very first level expanded we have that:

$$\begin{aligned} D_{KL}(P_A(X_1)||P_B(X_1)) &= \sum_x [\log(P_A(X_1{=}x)) - \log(P_B(X_1{=}x))] \cdot P_B(X_1{=}x) \\ &= \sum_x h_{kl}(X_1 = x) \end{aligned} \tag{4.7}$$

where $\sum_x$ sums through all possible values for X_1 where each value corresponds to a node on the first level of the search tree. Intuitively, branches with a high heuristic value correspond to branches that contribute the most to the distance between the two networks.

Since it is difficult to directly compute $P_A(X_1 = 0)$ and $P_B(X_1 = 0)$ in the kl heuristic, we approximate these terms using the weighted mini-bucket (WMB) heuristic [46] with an i-bound of 10. Namely, WMBE is used to compute an estimate of the partition function conditioned on the node's partial configuration.

Once we have L leaf nodes (each representing a partial configuration) we extend each to a full configuration by forward sampling the rest of the variables using the 25 node network yielding a data point $\mathbf{x}^{(j)}$ conditioned on the node's partial configuration.

Using these L full samples, we solve for θ^* that minimizes the following loss function:

$$\theta^* = \arg\max_\theta \sum_{j=1}^{L} \log P_{B_\theta}(\mathbf{x}^{(j)}) \cdot P_A(\mathbf{x}^{(j)})$$

subject to

$$p_{s_i}(t+1) = \sum_{s_j, s_k \in S} \theta_{s_i, s_j, s_k} \cdot p_{s_j s_k}(t), \ \ \forall s_i \in S \tag{4.8}$$

with an additional term $P_A(\mathbf{x}^k)$ where each θ expression is also multiplied with the probability of the sample $P(\mathbf{x}^k)$. We can justify loss function because in the limit of when $L \to \infty$, this loss function is equivalent to minimizing the KL divergence. This is formally stated in the following Theorem. The full **KL-Search** algorithm is shown in Algorithm 4.

Algorithm 4: KL-Search Minimization
Input: Two Bayesian networks: a large network A, a smaller network B, and a parameterized extended network B_θ such that all nodes in B_θ are in A, a variable ordering o over A, initial pair-wise distribution $p_{s_i s_j}(t)$, and pre-computed output distribution $p_{s_i}(t+1)$ for a single variable
Parameters: Number of samples L
Output: θ, estimated parameters that minimize the distance in Eq. 4.8 between A and B_θ

```
1  $T \leftarrow$ data structure for the OR-search tree over $A$ using ordering $o$;
2  $OPEN \leftarrow \{\langle root(T), 0\rangle\}$;
   // frontier nodes are ordered by the 2nd value
3  for $i = 1 \to L$ do
4      $v \leftarrow OPEN$.dequeue() ;          // remove the node of highest priority
5      for $u \in children(v)$ do
6          $h_{kl}(u) \leftarrow [\log(P_A(u)) - \log(P_B(u))] \cdot P_B(u)$ ;     // ($P_A(u)$ w/ WMBE)
7          Add $< u, h_{kl}(u) >$ to $OPEN$;
8      end
9  end
10 Let $X$ be an empty list;
11 for $v \in OPEN$ do                         // there will be $L$ leaf nodes in OPEN
12     Forward sample $x$, a full configuration of $A$ conditioned on the partial configuration
         represented by $v$;
13     Append $x$ to $X$;
14 end
15 Solve $\theta^* = \arg\max_\theta \sum_{j=1}^{L} \log P_{B_\theta}(\mathbf{X}^{(j)}) \cdot P_A(\mathbf{X}^{(j)})$, subject to
16 $p_{s_i}(t+1) = \sum_{s_j, s_k \in S} \theta_{s_i, s_j, s_k} \cdot p_{s_j s_k}(t), \;\; \forall s_i \in S$;
17 Return $\theta^*$;
```

Theorem 4.4.1.1 (Asymptotic Convergence of KL-Search Minimization)**.** *Let θ_L be the result of KL-Search Minimization [Algorithm 4] given L samples. Then given a family of extended* *networks B_θ parameterized by θ:*

$$\lim_{L\to\infty} \theta_L = \arg\min_\theta D_{KL}(P_A||P_{B_\theta}) \tag{4.9}$$

Proof: Since samples are generated from leaf nodes who are guaranteed to be different, it implies that no samples are identical. If the search tree is fully expanded, our optimization problem becomes:

$$\begin{aligned}\theta^* &= \arg\max_\theta \sum_{X\in A} \log P_{B_\theta}(X) \cdot P_A(X) \\ &= \arg\max_\theta \mathbb{E}_{X\sim P_A}[\log P_{B_\theta}(X)] \end{aligned} \tag{4.10}$$

which is just maximizing the log likelihood. With some additional derivation:

$$\begin{aligned}\theta^* &= \arg\max_\theta \sum_{X\in A} \log P_{B_\theta}(X) \cdot P_A(X) \\ &= \arg\max_\theta \sum_{X\in A} \log P_{B_\theta}(X) \cdot P_A(X) - \log P_A(X) \cdot P_A(X) \\ &= \arg\max_\theta \sum_{X\in A} \log \frac{P_{B_\theta}(X)}{P_A(X)} \cdot P_A(X) \\ &= \arg\min_\theta \sum_{X\in A} - \log \frac{P_{B_\theta}(X)}{P_A(X)} \cdot P_A(X) \\ &= \arg\min_\theta \sum_{X\in A} \log \frac{P_A(X)}{P_{B_\theta}(X)} \cdot P_A(X) \\ &= \arg\min_\theta D_{KL}(P_A(X)||P_{B_\theta}(X)) \end{aligned} \tag{4.11}$$

it can be shown that maximizing the expectation $\mathbb{E}_{X\sim P_A}[\log P_{B_\theta}(X)]$ is equivalent to minimizing the KL-divergence $D_{KL}[P_A(X)||P_B(X)]$, so our search procedure is guaranteed to eventually find the 8 node parameterized model having minimum KL-divergence as the search tree approaches full expansion.

4.4.2 Additional Computational Details

In practice it is difficult for black-box solvers to satisfy the marginal probability constraint. This is even more of an issue in games with a larger number of strategies. To address this, we turn the hard constraint into a soft constraint:

$$\theta^* = \arg\max_{\theta} \sum_{k}^{L} \log P_{B_\theta}(\mathbf{x}^k) \cdot P_A(\mathbf{x}^k) +$$
$$C \cdot \sum_{x} \left[p_x(t+1) - (\sum_{y,z \in S} \theta_{x,y,z} \cdot p_{yz}(t). \right]^2 \tag{4.12}$$

For our empirical tests, we solve this optimization problem with Sequential Least Squares Programming (SLSQP), a method for solving nonlinear optimization through a sequence of quadratic problems, through the minimize routine in the SciPy python library [56, 57].

4.4.3 Scaling up to More Strategies

To scale up the method to handle evolutionary games with more strategies, the major computational bottleneck is Equation 4.6 using our WMB approximation:

$$h_{kl}(X_1{=}0) = [\log(P_A(X_1{=}0)) - \log(P_B(X_1{=}0))] \cdot P_B(X_1{=}0)$$

While this equation can be evaluated in 25 node networks having 3 or less strategies, WMB heuristics become difficult to evaluate directly in 25 node networks with 4 or more strategies due to the size of the CPTs in the network. Therefore, instead of computing $P_A(X_1 = 0) \approx$ WMB$(X_1 = 0)$, we propose an easier to compute heuristic such as:

$$P_A(X_1 = 0) \approx \hat{Z}_A(X_1 = 0) = \frac{1}{N}\sum_{i=1}^{N} \hat{Z}_A(x^{(i)}), \qquad x^{(i)} \sim P_A(x|X_1 = 0) \tag{4.13}$$

where $\hat{Z}_A$ is a stochastic estimate of the partition function of A conditioned on $(X_1 = 0)$ for N samples. We propose a quick estimate for the partition function to be evaluated from just a single sampled configuration ($N = 1$ in Eq. 4.13)[7]. We'll call our algorithm using this heuristic **Fast KL-search**.

4.4.4 Learning and Inference

KL-Search (Algorithm 4) and Fast KL-search (step 6 in Algorithm 4 with our simplified heuristic) both work over one timestep of the evolutionary game. The θ^* obtained as the return value from these algorithms only minimizes the kl divergence between models for a single timestep. The optimal θ value for B_θ can change at each timestep and is dependent on the current values of $p_{s_i}(t)$ and $p_{s_i s_j}(t)$. Thus the full algorithm for our approach has two parts: learning (KL-Search) followed by inference (TDBNA in Algorithm 2) and there can be multiple ways to incorporate the two together. Some methods for doing this include:

- Static: Run KL-Search for only the first timestep (or first few timesteps) of the TDBNA algorithm.

[7]This Z-estimate would just the product of all the weights down a single path in an OR-search tree.

- Dynamic: Run KL-Search for every timestep of the TDBNA algorithm.
- Threshold: Run KL-Search if the values for p_{s_i} and $p_{s_i s_j}$ change beyond a certain threshold from the last timestep it was applied.

Determining what regime to run KL-Search with a TDBNA requires analyzing the trade-off between noise, speed and accuracy. For example, if KL-search is run using the static regime, we are guaranteed that the trajectory will not have noise later (since the remaining timesteps will be computed with only exact inference on a surrogate model). Running it in every iteration may produce more accurate results, but will naturally take longer to compute and expose the trajectory to noise (possibly obscuring qualitative results).

Finding the most effective method for interleaving learning with inference will be left to future work. Our goal in the empirical results section will be to determine what methods reduce the single iteration difference the most out of all learning methods.

4.5 Empirical Results

We will first summarize the algorithms we will compare in this section. The methods used in our comparisons will include:

- 8 node (exact): the estimate obtained from exact inference on the 8 node network
- AS: Abstraction Sampling on a 25 node TDBNA
- MC: Forward sampling on a 25 node TDBNA
- MLE: The surrogate network approach from Section 4.3 without doing KL-search to generate the data points.

- KL: The estimate obtained using θ^* from Algorithm 4
- FKL: The estimate obtained using θ^* from Algorithm 4 with step 6 using a simplified heuristic

In practical application, we are interested in querying $P(X(t+1), Nb(X)(t+1))$ to compute $p_{s_i s_j}$ for the next time step. Therefore, we are interested in reducing the distance between $P_{sim}(X(t+1), Nb(X)(t+1))$ and $P_{method}(X(t+1), Nb(X)(t+1))$ where P_{sim} denotes the distribution calculated using a simulation on a 3000×3000 grid. We will use the measure:

$$D_{KL}(P_{sim}(X(1), Nb(X)(1)) || P_{method}(X(1), Nb(X)(1))) \tag{4.14}$$

to evaluate how effective different approximate inference algorithms are as we show in Figure 4.4, 4.5, and 4.6. Specifically, we will evaluate the distribution distance at $t = 1$ in our spatial evolutionary game to compute the single iteration KL-divergence for each method.

4.5.1 Comparison of Methods

For an initial comparison, we use the Deadlock (DLK) game (see Table 4.6) from Fig. 4.1 that we employed as a motivating example. Figure 4.4 depicts KL-divergence between the method and simulation at $(t = 1)$ as a function of the chosen inference scheme. We see that on the DLK game, the KL-search approach outperforms all of the other methods in terms of KL-divergence, and notably outperforms AS (400 samples), MC (400 samples), and MLE (400 samples) with just 100 samples.

We next compare KL-search (Algorithm 4), Fast KL-search (Algorithm 4 with Eq. 4.13),

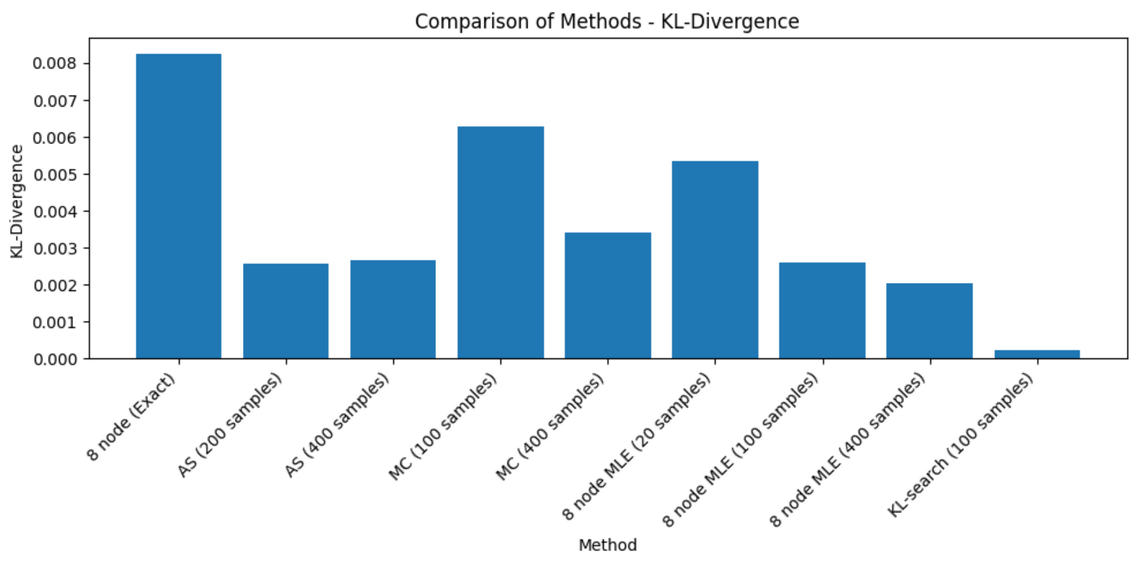


Figure 4.4: Sample efficiency comparison (measured using average KL divergence) on the Deadlock evolutionary game at $(t = 1)$. Results are averaged across 10 separate runs for each method

Pair Approximation (BN-PA), and Abstraction Sampling using the average KL divergence metric as in Fig. 4.4. For these set of tests, we average the the results of 30 separate runs and allow each method to generate 100 samples for each run. For inference on the surrogate network for KL-search/Fast KL-search we employ standard Bucket Elimination as described in Section 2.3.2.1.

The results are displayed in Fig. 4.5, which compares the three methods across 5 different evolutionary games (see Table 4.6). Both the Fast KL-search (denoted fkl in the figure) and the regular KL-search with WMB heuristic significantly outperforms Abstraction Sampling in all games.

4.5.2 Larger Games

The only method that can run on games with more than 3 strategies is Fast KL-Search. As mentioned earlier, with 4 or more strategies WMBE heuristic is hard to compute because of its inherent table representation of the CPTs which become too large. We next show a comparison

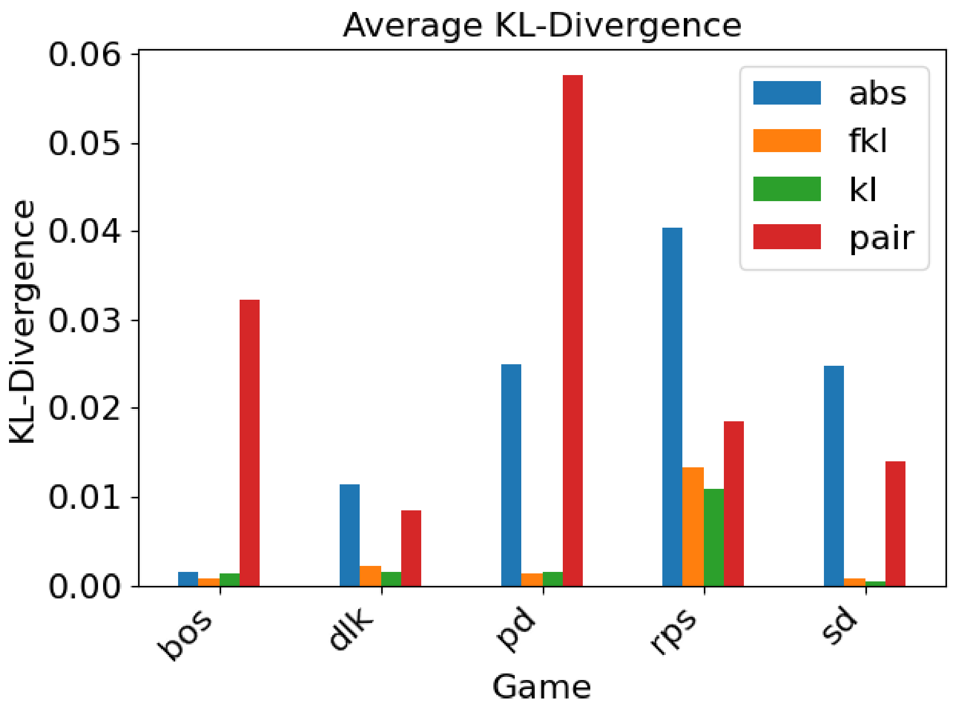


Figure 4.5: Average KL divergence at $(t = 1)$ between simulation and each of [abstraction sampling (abs), KL-search using WMB (kl), pair approximation (pair), and KL-search using a single configuration sample (fkl)].

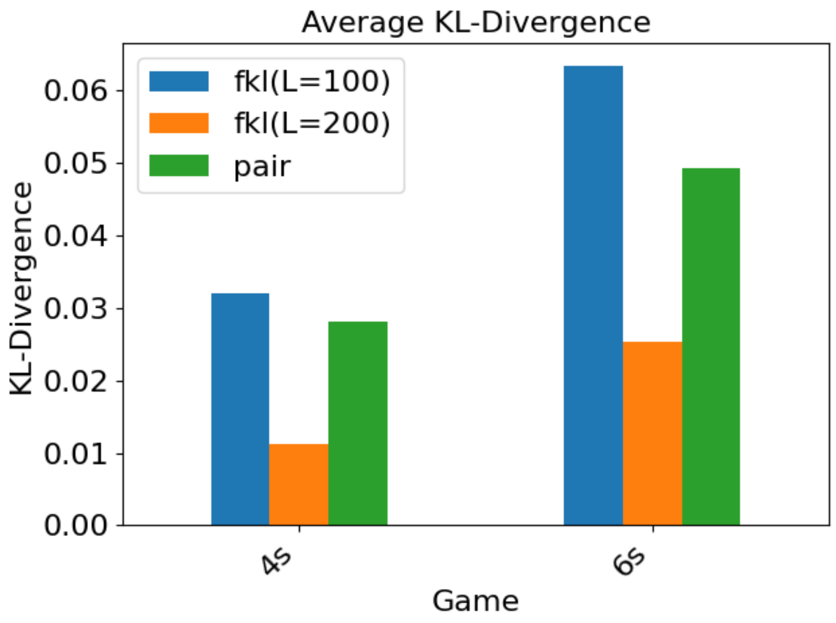


Figure 4.6: Average KL divergence at $(t = 1)$ between simulation and each of [pair approximation (pair), and KL-search using a single configuration sample (Fast KL-search) at 100 and 200 samples] on games with a high number of strategies.

with the method known as pair approximation [2] in Fig. 4.6 on these large games (a 4 strategy and 6 strategy game from Table 4.6. We observed that in these larger games, we need at least 100 samples in order to outperform pair approximation.

4.5.3 Time Comparison

A comparison of time per probe/sample is shown in Table 4.3. The table is generated by aggregating average time statistics from the experiments in Fig. 4.5 and 4.6 and dividing by the number of samples given (L = 100). We observe that Fast KL-Search actually takes more time on average than normal KL-Search per sample. This is because the time for the latter is frontloaded in computing the initial WMBE tree (which is not included in these time comparison results as it is pre-computed). For individual samples, computing a WMBE heuristic (e.g. WMB$(X_1 = 0)$) in KL-Search takes less time than evaluating the value of a single sample in Eq. 4.13 for Fast

Table 4.3: Comparison: Average time per sample (s)

Number of Strategies	Abstraction Sampling	KL-Search	Fast KL-Search
2	0.5464	0.01419	0.02019
3	0.7092	0.02829	0.03549
4	-	-	0.1697
6	-	-	0.7299

KL-Search. As expected, both methods take much less time than Abstraction Sampling. The time for computing a full trajectory is shown in Table 4.5. These time results include the time taken for computing the WMBE tree at each timestep in AS and KL-Search. With the WMBE tree computation step taken into account, we can observe that Fast KL-Search is indeed faster than normal KL-Search for generating a full trajectory.

One sample in AS is a full probe that expands $O(d \cdot h)$ nodes in a search tree where d is the number of abstract states and h is the tree's depth. This takes $O(d \cdot h \cdot L)$ time where L is the number of samples. For KL-Search and Fast KL-Search, only L nodes in the search tree need to expanded to get L samples, so the final time is just $O(L)$. This is summarized in the table 4.4.

Table 4.4: Algorithm complexity for AS, KL-Search and Fast KL-Search per sample

AS	KL-Search	Fast KL-Search
$O(d \cdot h \cdot L) \cdot O(1)$	$O(L) \cdot O(1)$	$O(h) \cdot O(L)$

Table 4.5: Comparison: Time (s) per full trajectory (with time horizon T = 10). The methods AS, KL, FKL all use $(L = 100)$ samples and the simulation is performed on a 3000×3000 grid.

# Strat	Sim	Matrix[8]	BN-PA w/ BE	BN-Large w/ AS	Alg. 2 w/ KL	Alg. 2 w/ FKL
2	849.091	60.996	1.429	777.921	64.872	29.703
3	=	=	2.203	1450.367	98.869	54.842

[8]On populations located on grids, it is possible to accelerate computation on an agent based simulation for simple games using matrix operations with the Python library numpy instead of normal loops. These operations will not work if the spatial evolutionary game is defined over non-grid spatial systems.

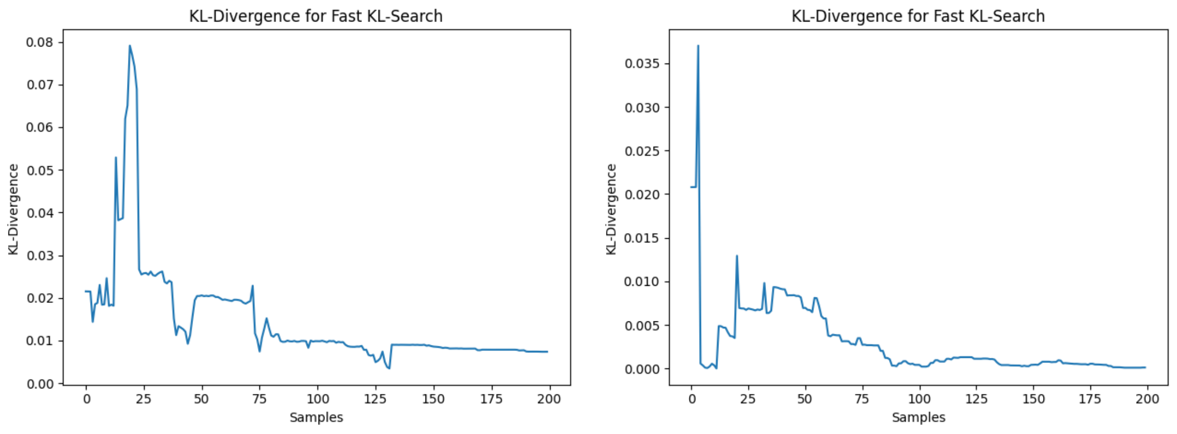


Figure 4.7: KL divergence for KL-search using a single configuration (Fast KL-search) on RPS (left) and Deadlock (right)

4.5.4 Individual Runs

In Fig. 4.7, we show two of the divergence curves of individual runs for the Fast KL-Search method (these runs are one of the 30 runs for Fast KL-Search that were evaluated on RPS and Deadlock for plotting Fig. 4.5). The behavior of our method is somewhat different from standard sampling algorithms as the final step of the method is an optimization problem. As a result, there are certain points in the graph where the optimization problem switches between different local maxima. This becomes less frequent as more samples are obtained.

4.5.5 Extended Comparison

For a longer comparison, we test Abstraction Sampling vs KL-search on Rock-Paper-Scissors for approximately 2.5 and 5 minutes in Fig. 4.8. This is quite long since this time limit is per timestep of the evolutionary game. Each test is performed over an average of 30 independent runs. We also provide a comparison to the 100 sample results from the earlier figure. The time taken for KL-search is not exactly 2.5 or 5 minutes due to the stopping criteria used. Since we

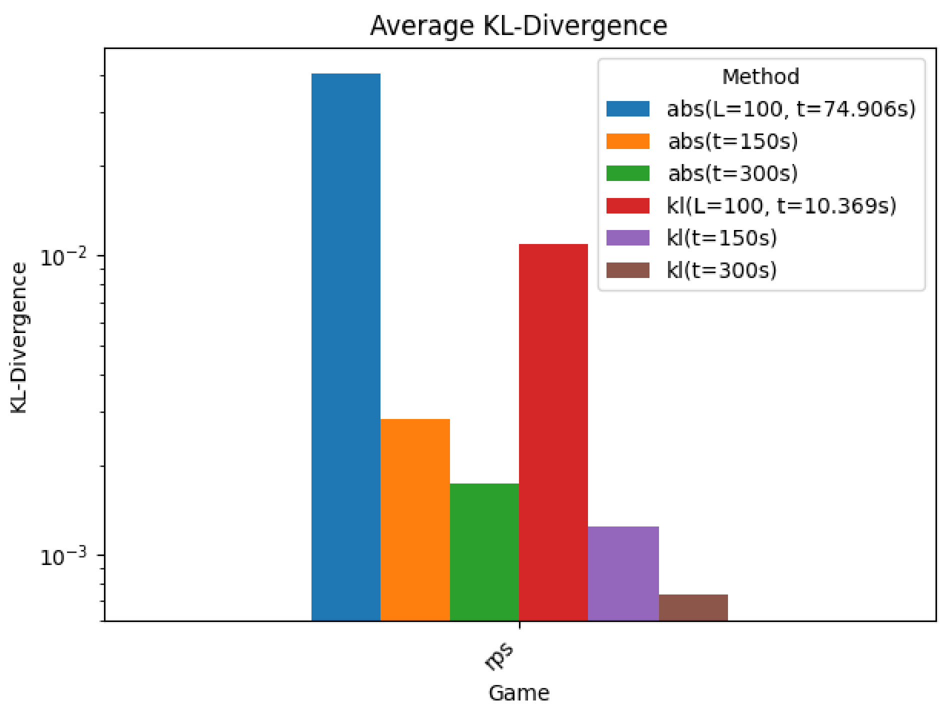


Figure 4.8: KL divergence at ($t = 1$) for KL-search (Alg. 4) (left 3 bars) vs Abstraction Sampling (right 3 bars) for longer sample time over the rps game

need to solve an optimization problem and an inference problem for $P(X(t+1)|Nb(X)(t+1))$ after sampling, we add an early stopping criteria of 10s before 2.5 or 5 minutes for post-sampling computation. The KL-divergence continues to converge towards 0 for both methods even beyond the initial 100-200 samples.

4.5.6 Discussion

In general, both KL-Search and Fast KL-Search outperform Abstraction Sampling in both sample and time efficiency. When performing approximate inference on a TDBNA it is preferable to use our new proposed methods. However, there are few considerations that need to be examined before determining how to apply KL-Search:

- Noise: In section Fig. 4.7, we show the KL-divergence of particular runs. Most notable are the spikes in the divergence graph. Due to these spikes, there can be considerable noise when running KL-Search for a small amount of samples. It is then recommended to run KL-Search for a sufficiently large amount of samples such that drastic changes in the optimization problem are less frequent. In the examples graphs shown, this would suggest at running for at least 150 samples.

- Scaling: When the dimensionality of the distribution increases (e.g. for games with more than 3 strategies), the efficiency of Fast KL-search decreases with respect to pair approximation. This can be observed in Fig. 4.5 and 4.6, the results of Fast KL-search at 100 samples is increasingly worse compared to pair approximation as the number of strategies increases. As a result, to retain an advantage over pair approximation, it is necessary to increase the number of samples used as a function of the number of strategies

As shown in the time comparison section, even with the approximate inference methods TDBNAs are not significantly faster compared to running an agent based simulation. In particular, due to the sampling effects mentioned above, the KL-Search method may need more than 100 samples to be effective even when compared to pair approximation. In many cases, this will take longer than a direct simulation (using matrix accelerated operations) on a 3000×3000 grid. However, there are a few situations where it may be beneficial to use a large TDBNA with KL-search:

- Static regime: As mentioned in Section 4.4.4, if we run Algorithm 2 with KL-Search only applied to the first few timesteps, the total time cost will be low. With a thresholding regime, it may be possible to improve the time efficiency of KL-Search based TDBNA in many situations, although this is left for future work.

- Rare events: If there are strategies with low proportion in the population but an outsized impact on population dynamics, it can be difficult to capture accurate dynamics with a simulation. It can be necessary to run a large simulation many times to get low variance results. In these case, it can be viable to analyze these low proportions with a TDBNA instead.

- Spatial structure and complex games: For complex games (games with more complex utility functions or update rules) or games defined on other spatial structures, it will no longer be possible to use accelerated matrix operations for computing agent based simulations. In these casse, the simulation on a large grid can take significantly longer (see Table 4.5 where it takes 850 seconds to generate a trajectory for a game with 2 strategies). Furthermore, on certain types of graphs (e.g. random regular graphs), TDBNAs (and other moment closure techniques in general) will perform better due to less short range loops compared to a grid spatial structure.

- Game-based scaling compared to population-based scaling: It is also clear that the agent based simulation scales as a function of population size, while the TDBNA scales as a function of game size.

4.6 Summary

We have introduced a novel approach for performing approximate inference on evolutionary game Bayesian networks. Our method, based on optimizing parameterized surrogate Bayesian Networks, leverages the symmetry present in TDBNAs to combine the efficient computation

Table 4.6: Evolutionary Game Payoff Matrices

Game Name	Payoff Matrix
Prisoner's Dilemma (pd)	$\begin{pmatrix} 2 & -1 \\ 3 & 0 \end{pmatrix}$
Snowdrift (sd)	$\begin{pmatrix} 2 & 1 \\ 3 & 0 \end{pmatrix}$
Battle of the Sexes (bos), symmetric version [54]	$\begin{pmatrix} 0 & 1 \\ 2 & 0 \end{pmatrix}$
Deadlock (dlk)	$\begin{pmatrix} 5 & -4 \\ 3 & -5 \end{pmatrix}$
Rock Paper Scissors (rps)	$\begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix}$
Random 4 strategy game (4s)	$\begin{pmatrix} 1 & 3 & 1 & 3 \\ -2 & 4 & 4 & 4 \\ 1 & 4 & 2 & 5 \\ -2 & 3 & 4 & 2 \end{pmatrix}$
Random 6 strategy game (6s)	$\begin{pmatrix} 1 & 3 & 1 & 3 & 3 & 1 \\ -2 & 4 & 4 & 4 & 1 & 1 \\ 1 & 4 & 2 & 5 & 2 & 3 \\ -2 & 3 & 4 & 2 & -1 & 2 \\ -4 & 1 & 3 & 3 & 0 & 3 \\ -2 & 1 & -1 & 1 & 1 & 3 \end{pmatrix}$

of smaller TDBNAs with the more accurate results of larger TDBNAs. We formulate the parameter learning problem as a maximum likelihood estimation problem. We introduce a novel sample-search approach to generate high value samples for this optimization problem, and empirically demonstrate its effectiveness when compared to existing approximate inference techniques such as Abstraction Sampling.

Chapter 5: Fast Fourier Transform Reductions for Bayesian Network Inference

Bayesian Networks are useful for analyzing the properties of such distributed systems for tasks such as load balancing [59], reliability analysis [60], or fault diagnosis [61]. These networks typically have large functions (CPTs), making even the specification of the BN itself intractable. Recall from Chapter 2 that Bayesian Network inference is exponential in the network's induced width (or tree-width), making exact inference intractable especially over large CPT's, as they lead to very high induced widths [41]. However, a subset of these distributed service models such as k-out-of-n reliability models include large summations of a distributed resource (e.g. number of computational hosts, unit of power available per generator in a smart grid) which have stochastic availability. The unique additive symmetry present in these models make them amenable to efficient inference and CPT reduction algorithms through the usage of the Fast Fourier Transform (FFT) [62].

More specifically, many distributed resource problems can be modeled using k-out-of-n Bayesian Networks [63]. The general idea is to model n different resource providers $\{S_1, \ldots, S_n\}$ which each provide some varying amount of distributed resource (server nodes, power generation, etc.) with stochastic availability and/or quantity. For example, in a distributed computation application, each server location can be modeled as a separate node S_i which provides some k_i amount of computing resource (e.g. S_1 is some server cluster with 300 total available comput-

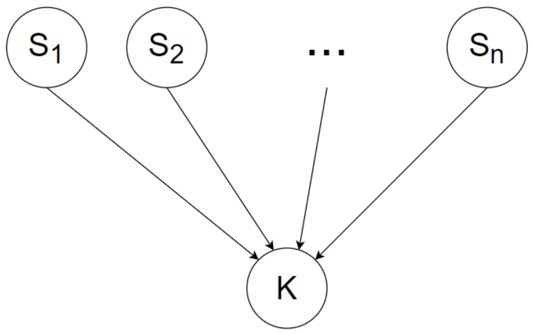


Figure 5.1: Bayesian Network for summation of random variables, $K = \sum_i S_i$

ing nodes). As mentioned in [63], the naive k-out-of-n model is one converging node K with n parents $\{S_1, \ldots, S_n\}$ with the structure in Fig. 5.1. The value of the K node is the sum of the variables $\{S_i, \ldots, S_n\}$. The CPT for node K is very sparse as a table and can be represented symbolically as the function:

$$P(K = k | S_1 = k_1, \ldots, S_N = k_n) = \mathbb{1}_{k=\sum_i k_i} \tag{5.1}$$

Similar and more complex models can be constructed for the evaluation of other distributed resource problems where there are multiple providers and the goal is to check whether the sum of resources sourced from individual providers satisfies a given constraint.

This type of convergent structure can also be found in Bayesian Networks used to represent Fault Trees [63]. Fault Trees are frequently used to analyze system reliability and a key component in many of them are k-out-of-n voting gates [64], which only activate if more than k out of n parents encounter a fault event. In the distributed resource model, each S_i's domain is simply $\{0, 1\}$ and we query the probability that the summation node K has a value greater or equal to k.

Recall the structure of the dynamic Bayesian network used to model a spatial evolutionary game. In an evolutionary game, the fitness of an agent is evaluated as the sum of payoff values obtained by playing normal form games with neighboring agents $N(i)$:

$$Pay(s_i) = \sum_{j \in N(i)} U[s_i, s_j] \tag{5.2}$$

These Bayesian networks can have large sets of nodes that are topologically identical with respect to their position in the network. The number of nodes in these sets is dependent on a model parameter d denoting the number of neighboring agents. Inference on the network can become cumbersome for large d since the size of the conditional probability table of a payoff valued node is exponential in d. Like in the aforementioned distributed resource models, these payoff valued nodes can be thought of as summation nodes.

In this chapter, we will describe a novel algorithm for accelerating probabilistic inference in networks with additive symmetry. To start off, in Section 5.1, we provide background on causal independence (CI) in Bayesian networks, and on computing sum of random variables using Fast Fourier Transform (FFT). We then show how FFT can be used for efficient reduction of a BN with large CI functions (Section 5.2.1) and subsequently (Section 5.2.2) show how FFT can speed up general probabilistic inference (e.g., Bucket elimination) when the network has summation-based CI fragments. Section 5.3 provides empirical evaluation of the above algorithms.

5.1 Background

Casual independence is a probabilistic relationship between a set of causes $\{c_1, \ldots, c_n\}$ and an effect e where the effect can be seen as a deterministic function of hidden variables $\{h_1, ..., h_n\}$ such that $e = h_1 * h_2 ... * h_n$ where each h_i is a probabilistic function of its corresponding c_i and $*$ is a commutative and associative binary operator [65, 66]. This paper is directed towards networks possessing causal independence where the $*$ operator is the addition operator $+$ and the effect e can then be expressed as: $e = \sum_i h_i$.

Causal independence in Bayesian Networks such as Fig. 5.1 enables efficient network transformations (e.g., temporal transformation [63, 67]) to significantly reduce the size of parent sets in the network.

Network Transformations. Given a CI Bayesian Network fragment $\{X, D, F\}$ with a set of causes $\{c_1, \ldots, c_n\}$, an effect e: $X = \{c_1, ..., c_n, e\}$ and a set of hidden variables $\{h_1, \ldots, h_n\}$, consider a computation ordering over the equation $e = h_1 * h_2 ... * h_n$. An example ordering for a *temporal transformation* is:

$$e = (\ldots(((h_1 * h_2) * h_3) * h_4) * \ldots) * h_n \tag{5.3}$$

Given an ordering, denote the quantities enclosed in each parenthesis set as intermediate variables y_i: $\{y_1 = h_1 * h_2,\ y_2 = y_1 * h_3,\ y_3 = y_2 * h_4 \ldots\}$.

A network transformation $\{X', D', F'\}$ is a network such that $X' = \{c_1, \ldots, c_n, h_1, \ldots, h_n,$ $y_1, \ldots, y_m, e\}$ is expanded to include hidden variables h_i and intermediate variables y_i defined over a valid computation ordering. The resulting network is also called a *decomposition network*. The goal is to transform large parent sets to small ones (e.g. 2 variables per set).

Further work on the topic [65, 66] exploit the decomposition graphs resulting from network transformations to accelerate bucket elimination (*ci-elim-bel* in Algorithm 5). It was found that using *ci-elim-bel* to exploit casual independence can significantly improve the performance of exact inference on polytrees (from $O(Nd^m)$ to $O(Nmd^3)$ where N is the number of nodes, d is the domain size and m is the size of the largest parent set) as well as in two layer *k-n-networks* (see Fig. 5.6) (from $O((k+n)d^k)$ to $O((k+n)d^{min\{k,2n\}})$).

```
Algorithm 5: ci-elim-bel [65]
Input: A Bayesian network $B = (X, D, F)$ where $F$ are CI, evidence $e$
Output: $P(x_1|e)$
1  Generate a decomposition network $B'$ from $B$ with pair-wise hidden variables
   $\{u_1, \ldots, u_m\}$
2  Generate ordering $o = \{Z_1, \ldots, Z_n\}$ using $B'$ s.t. $Z_1 = \{x_1\}$ and $Z_i = \{x_j\} \mid \{u_j; u_k\}$
   [for details, see [65]]
3  for $i = n \to 1$ do // create buckets
4      $\forall x \in Z_i$, put all network functions with $x$ as highest ordered variable in $bucket_i$
5  end
6  for $i = n \to 1$ do // process buckets
       // $h_1, \ldots, h_m$ are functions in $bucket_i$
7      if $(x = e_j) \in bucket_i$ for $e_j \in e$ then
8          replace $x$ by $e_j$ in each $h_i$ and put the result in appropriate lower bucket.
9      else
10         if $Z_i = \{x\}$ then // input variable
11             $h^{Z_i} = \sum_x \prod_j h_j$
12         else // $Z_i = \{u_l; u_k\}, u = u_l * u_k$
13             $h^{Z_i} = \sum_{u_l, u_k | u = u_l * u_k} \prod_j h_j$
14         Put $h^{Z_i}$ in the highest bucket that mentions $h^{Z_i}$ 's variable.
15 end
16 Return $\alpha h^{x_1}$, ($\alpha$ is a normalizing constant).
```

We next provide background into the theory of probability generating functions and the use of Fourier transforms for computing random variable sums that is the main tool we will use to speedup some computations applied to Bayesian Network inference and reductions.

5.1.1 Random Variable Sums

Suppose we have a set of independent identically distributed ($i.i.d$) random variables $X = \{X_1, X_2, \ldots, X_N\}$ with domain D and a random variable Z s.t:

$$Z = \sum_i X_i \tag{5.4}$$

Naively, we can find $P(Z=k)$ by enumerating all X_i values that sum to k, taking $O(|D|^N)$ time. However, we can calculate $P(Z=k)$ more efficiently through probability generating functions as described next.

Definition 5.1.1 (Probability Generating Functions)**.** *A probability generation function (or polynomial) $\mathbb{P}_X$ for a discrete random variable X, with $D_X \subset \mathbb{N}$, having a distribution P_X is defined as:*

$$\mathbb{P}_X(x) = \textstyle\sum_{k=0}^{\infty} x^k P_X(k) \tag{5.5}$$

The k-th coefficient of $\mathbb{P}_X$, $P_X(k)$, can be found by taking derivatives of the generating polynomial:

$$P_X(k) = \mathbb{P}_X^{(k)}(0)/i! \tag{5.6}$$

where $\mathbb{P}_X^{(k)}$ is the k-th derivative of $\mathbb{P}_X$.

Theorem 5.1.1.1 (Generating Function Multiplication, (8.37 in [68]))**.** *For a set of random variables $X = \{X_1, \ldots, X_N\}$ with distributions $\{P_{X_1}, \ldots, P_{X_N}\}$ and their corresponding generating functions $\{\mathbb{P}_{X_1}, \ldots, \mathbb{P}_{X_N}\}$, the distribution of $Z = \sum_i X_i$ obeys:*

$$P(Z=k) = \mathbb{P}_Z^{(k)}(0)/k!, \quad \textit{where} \;\; \mathbb{P}_Z = \textstyle\prod_j^N \mathbb{P}_{X_j} \tag{5.7}$$

Proposition 1. *Computing $(\mathbb{P}_Z)_k, \forall k$ using generating polynomial multiplication takes $O(|D|^2 \cdot N^2)$ time.*

Proof. For two generating polynomials $\mathbb{P}_{X_1}, \mathbb{P}_{X_2}$, the product is computed as:

$$(\mathbb{P}_{X_1} \times \mathbb{P}_{X_2})_k = \textstyle\sum_{j=0}^{k} (\mathbb{P}_{X_1})_k (\mathbb{P}_{X_2})_{k-j} \tag{5.8}$$

where subscripts denote the coefficient of the corresponding polynomial. This computation is quadratic in the size of the polynomials (number of coefficients). In the process of computing $\mathbb{P}_Z$, the size of intermediate polynomials is bounded by $N|D|$. Therefore the total time for computing $\mathbb{P}_Z$ is:

$$\textstyle\sum_{i=1}^{N} i|D| \cdot |D| = O(|D|^2 \cdot N^2) \tag{5.9}$$

□

Definition 5.1.2 (Convolution)**.** *Let $F_X(k) = P(X = k)$ and $F_Y(k) = P(Y = k)$ be two probability density functions for random variables X, Y. The convolution of functions F_X and F_Y is defined as:*

$$(F_X * F_Y)(k) = \textstyle\sum_j F_X(k) F_Y(k-j) \tag{5.10}$$

Clearly $(\mathbb{P}_{X_1} \times \mathbb{P}_{X_2})_k = (P_{X_1} * P_{X_2})(k)$. Consequently, we can use tools for performing convolutions to compute the distribution of a sum.

The Convolution Theorem. The convolution theorem provides an alternative method for the calculation of Eq. 5.10 using the Fourier Transform.

Definition 5.1.3 (Discrete Fourier Transform)**.** *The discrete Fourier transform $\mathcal{F}$ of a discrete* *probability distribution F_X defined over integers $[0, M-1]$ is:*

$$\hat{F}_X(x) = \mathcal{F}\{F_X\}(x) = \sum_{k=0}^{M-1} F_X(k) \cdot e^{-i2\pi xk/M} \tag{5.11}$$

$\hat{F}_X(x)$ is also defined over M values and is called the Fourier transform of F_X and its domain is called the frequency domain. The inverse Fourier transform is defined as:

$$F_X(k) = \mathcal{F}^{-1}\{\hat{F}_X\}(k) = \frac{1}{M} \sum_{x=0}^{M-1} \hat{F}_X(x) \cdot e^{i2\pi xk/M} \tag{5.12}$$

The original domain is referred to as the time domain.

Theorem 5.1.1.2 (The convolution theorem, (4.3.49 in [69]))**.** *For two discrete probability distributions $F_X(t)$ and $F_Y(t)$, it can be shown that:*

$$(F_X * F_Y)(k) = \mathcal{F}^{-1}(\mathcal{F}\{F_X\} \cdot \mathcal{F}\{F_Y\})(k) \tag{5.13}$$

where $\mathcal{F}\{F_X\} \cdot \mathcal{F}\{F_Y\}$ denotes the pointwise multiplication of the two frequency distributions.

Corollary 5.1.1.2.1 (Symmetry)**.** *For $i.i.d$ random variables $X = \{X_1, X_2, \ldots, X_N\}$ and their sum $Z = \sum X_i$, it is easy to show that:*

$$\begin{aligned} P(Z\!=\!k) &= \mathcal{F}^{-1}(\mathcal{F}\{F_X\}^N)(k) \\ \textit{where } F_X(k) &= P(X_i\!=\!k) \end{aligned} \tag{5.14}$$

Theorem 5.1.1.3 (Time Complexity)**.** *For $i.i.d$ random variables $X = \{X_1, X_2, \ldots, X_N\}$ where each variable has a domain size $|D|$ and $Z = \sum X_i$, computing $P(Z = k)$ using Eq. 5.14 will* *take time $O(N^2|D|^2)$. For non-$i.i.d$ variables, the time complexity is $O(N^3|D|^2)$.*

Proof. When computing the discrete Fourier transform, the size of the distributions before and after the transforms are applied must be the same. It is necessary to pad the starting distributions F_X with zeros up to the size of the target sum Z which is $O(N|D|)$. The direct computation of the Fourier Transforms is quadratic in $N|D|$. Sequentially, the computation consists of:

- Fourier transform $\mathcal{F}$: $O(\big(N|D|\big)^2)$
- Exponentiation $\mathcal{F}\{F_X\}^N$: $O(N|D|)$
- Inverse transform $\mathcal{F}^{-1}$: $O(\big(N|D|\big)^2)$

which in total is $O(N^2|D|^2)$. In the case of non-$i.i.d$ variables, we perform a Fourier Transform for each distribution P_{X_i}, $O(N\big(N|D|\big)^2)$ and perform point-wise multiplications $O(N^2|D|)$ in the frequency domain bringing the total time to $O(N^3|D|^2)$. □

This computation can be accelerated using the **Fast Fourier Transform** (FFT). The FFT is a collection of divide and conquer algorithms (e.g., the Cooley-Tukey algorithm [62]) that reduces the time required for computing the Fourier transform from $O(|D|^2)$ to $O(|D|\log|D|)$ where $|D|$ is the domain size of the discrete distribution being transformed.

Theorem 5.1.1.4 (FFT Time Complexity). *For $i.i.d$ random variables $X = \{X_1, X_2, \ldots, X_N\}$ where each variable has a domain size $|D|$ and their sum $Z = \sum X_i$, computing $P(Z = k)$ using the FFT will take time $O(N|D|\log(N|D|))$. For non-$i.i.d$ variables, the time complexity is $O(N^2|D|\log(N|D|))$.*

Proof. The proof is the same as the previous theorem with the Fourier transform computed instead using the FFT in $O(N|D|\log N|D|)$ time. Consequently the time required to calculate the distribution over the sum of N $i.i.d$ variables defined over consecutive integers is reduced to:

$$\text{Time}(i.i.d) = O(N|D|\log(N|D|)). \tag{5.15}$$

and for non-$i.i.d$ variables:

$$\text{Time}(\text{non-}i.i.d) = O(N^2|D|\log(N|D|)). \tag{5.16}$$

□

In practice, even if X_i is not defined over consecutive integers, it is still possible to calculate the sum distribution of Z using the FFT, however the time complexity is better expressed using the range of Z:

$$\begin{aligned} R &= \max(Z) - \min(Z) \\ \text{Time} &= O(R\log R) \end{aligned} \tag{5.17}$$

5.2 Application to Bayesian Networks

5.2.1 FFT Reduction

Up to this point, we have been working with sums of random variables. Now, we will apply the FFT theory towards the reduction of variables in a Bayesian Network.

Consider the Bayesian Network in fig. 5.2 which has 3 types of nodes.

- Source node S take a value in some domain D_S

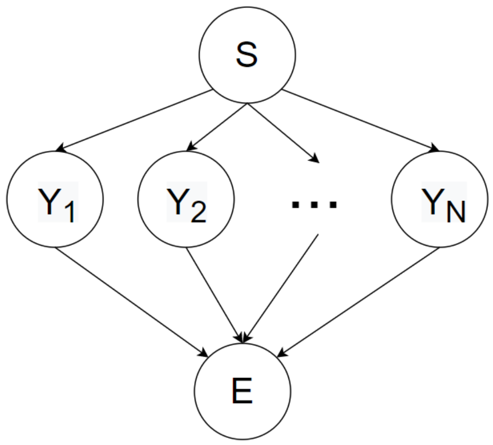


Figure 5.2: Symmetric Bayesian Network with i.i.d paths between source node S and sink node E

- A set of N $i.i.d$ nodes Y_i that take values in some domain D_Y
- Sink node E takes a value in a subset of the natural numbers $D_E \subset \mathbb{N}$

Let $U : D_Y \rightarrow \mathbb{N}$ be some cost function. We assume:

$$P(E = e \mid \{Y_i = y_i, \forall i\})) = \begin{cases} 1 & \text{if } e = \sum_i^N U[y_i] \\ \\ 0 & \text{otherwise} \end{cases} \tag{5.18}$$

The size of this function expressed as conditional probability table is exponential in N. However, using FFT theory, we can reduce the size of this CPT from $O(|D_E||D_Y|^N)$ to $O(|D_S||D_E|)$ by eliminating $Y_i, \ldots, Y_N$.

Theorem 5.2.1.1 (FFT Reduction). *Let $B = \{X, D, F\}$ with $X = \{S, Y_1, \ldots, Y_N, E\}$ be a source-sink network with N i.i.d paths as in Fig. 5.2. The network can be transformed into $\{X', D', F'\}$ such that $X' = \{S, E\}$ reducing the CPT for E from size $O(|D_E||D_Y|^N)$ to size $O(|D_S||D_E|)$ in $O(|D_S|R \log R)$ time where R is the numerical range of random variable E.*

Proof. Our goal is to define a new CPT for E of size $O(|D_S||D_E|)$ where each entry is:

$$P(E\!=\!e \mid S\!=\!s),\ \forall s \in D_S, e \in D_E \tag{5.19}$$

We use the FFT reduction to eliminate all $i.i.d$ nodes Y_i in between the S and E nodes. For each value of (s, e), we proceed as follows:

1. To calculate $P(\mathbf{U}[Y_i] = j|S = s)$, we transform the distribution on $Y's$ from the domain D_Y to a distribution on the natural numbers domain $\mathbb{N}$. This takes $O(|D_Y|)$ time as we simply iterate through $\mathbf{U}[y], \forall y \in D_Y$.

2. Let:

   $$Z = \textstyle\sum_i \mathbf{U}[Y_i] \tag{5.20}$$

   Let F_u be the discrete probability distribution of $U[Y_i]$:

   $$F_u(j) = P(U[Y_i]\!=\!j \mid S\!=\!s) \tag{5.21}$$

   Using the convolution theorem:

   $$\begin{aligned} P(E\!=\!e \mid S\!=\!s) &= P(Z\!=\!e \mid S\!=\!s) \\ &= \mathcal{F}^{-1}\{\mathcal{F}\{F_u\}^N\}(e) \end{aligned} \tag{5.22}$$

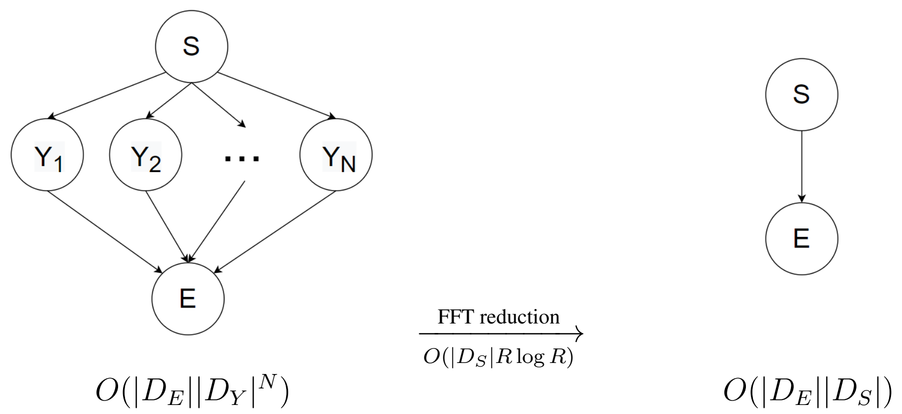


Figure 5.3: FFT Reduction

which takes $O(R \log R)$ where R is Z's range:

$$R = N(\max \mathbf{U} - \min \mathbf{U}) \tag{5.23}$$

We perform the above steps for each value of $s \in D_S$ giving a final time complexity of:

$$O(|D_S| R \log R)$$
$$R = N(\max \mathbf{U} - \min \mathbf{U}) \tag{5.24}$$

to reduce the CPT to $O(|D_E|\,|D_S|)$. □

Corollary 5.2.1.1.1 (Reduction on general CI Networks)**.** *For any set of random variables $X = \{X_1, \ldots, X_N\}$ (not necessarily identically distributed) and corresponding sum $Z = \sum_i X_i$, we can calculate all probabilities $P(Z = k)$ in $O(N^2|D|\log(N|D|))$ time.*

Proof. Simply apply the process for FFT reduction without conditioning on values of a source node S:

1. FFT for each X_i: $O(N \cdot N|D| \log(N|D|))$

2. N point-wise multiplications in Fourier Domain $O(N^2|D|)$

3. Inverse FFT for Z: $O(N|D|\log(N|D|))$

Adding these up results in $O(N^2|D|\log(N|D|))$ complexity. $\square$

5.2.2 Algorithm CI-Elim-Bel with FFT

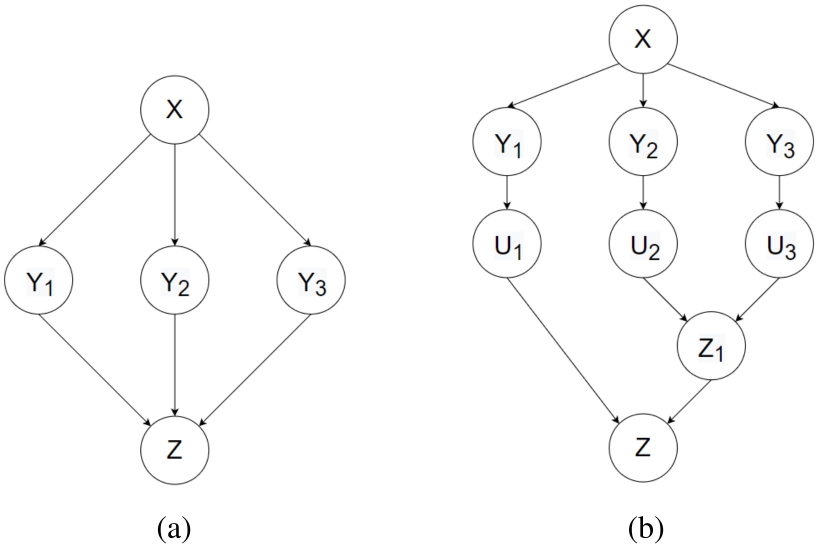


Figure 5.4: Five node source-sink Bayesian Network (a) and its decomposition graph (b)

In this section we will show how the use of FFT for processing summation can be incorporated into the bucket-elimination algorithm *ci-elim-bel*. We will illustrate this with an example. Consider the five node Bayesian Network in Fig. 5.4 where all Y_i's are i.i.d and are defined by $P(Y_i|X) = P(Y_k|X), \ \forall k.$ and assume $Z = \sum_i Y_i$.

Like in [65], we can perform *ci-elim-bel* on this network using the decomposition graph in

Fig. 5.4. The full equation is:

$$
\begin{aligned}
P(Z|X) &= \sum_{Y_1}\sum_{Y_2}\sum_{Y_3} P(Y_1|X)P(Y_2|X)P(Y_3|X)P(Z|Y_1,Y_2,Y_3) \\
&= \sum_{Y_1,Y_2,Y_3} P(Y_1|X)P(Y_2|X)P(Y_3|X) \sum_{\{u_1^z,u_2^z,u_3^z,z=u_1^z+u_2^z+u_3^z\}} P(u_1^z|Y_1)P(u_1^z|Y_2)P(u_1^z|Y_3) \\
&= \sum_{\{u_1^z,z_1|z=u_1^z+z_1\}} \sum_{Y_1} P(Y_1|X)P(u_1^z|Y_1) \sum_{\{u_2^z,u_3^z|z_1=u_2^z+u_3^z\}} \sum_{Y_2} P(Y_2|X)P(u_2^z|Y_2) \sum_{Y_3} P(Y_3|X)P(u_3^z|Y_3)
\end{aligned}
\tag{5.25}
$$

The elimination steps are as follows for ordering $o = \{\{u_1, z_1\}, Y_1, \{u_2, u_3\}, Y_2, Y_3\}$:

1. bucket $Y_3 : h^{Y_3}(X, u_3) = \sum_{Y_3} P(Y_3|X)P(u_3|Y_3)$
2. bucket $Y_2 : h^{Y_2}(X, u_2) = \sum_{Y_2} P(Y_2|X)P(u_2|Y_2)$
3. bucket $\{u_2, u_3\} : h^{\{u_2,u_3\}}(X, z_1) =$

   $\sum_{\{u_2,u_3|z_1=u_2+u_3\}} h^{Y_2}(X, u_2)h^{Y_3}(X, u_3)$
4. bucket $Y_1 : h^{Y_1}(X, u_1) = \sum_{Y_1} P(Y_1|X)P(u_1|Y_3)$
5. bucket $\{u_1, z_1\} : P(Z|X) =$

   $\sum_{\{u_1,z_1|z=u_1+z_1\}} h^{y_1}(X, u_1)h^{\{u_1,u_2\}}(X, z_1)$

where h^{Y_i} denotes intermediate functions. The largest operation occurs in bucket $\{u_1, z_1\}$ which has a complexity of $|D_X||D_Z||D_Y|$.

Observe that the calculations performed in the buckets $\{u_2, u_3\}$ and $\{u_1, z_1\}$ are equivalent to multiplying the coefficients of the corresponding generating polynomials. For example, consider the following three polynomials:

$$
\begin{aligned}
\mathbb{P}_{h^{Y_2}} &= \textstyle\sum_i h^{Y_2}(X, u_2 = i)x^i \\
\mathbb{P}_{h^{Y_3}} &= \textstyle\sum_i h^{Y_3}(X, u_3 = i)x^i \\
\mathbb{P}_{h^{\{u_2,u_3\}}} &= \textstyle\sum_i h^{\{u_2,u_3\}}(X, z_1 = i)x^i
\end{aligned} \tag{5.26}
$$

We have that:

$$h^{\{u_2,u_3\}}(X, z_1) = \sum_{\{u_2,u_3|z_1=u_2+u_3\}} h^{Y_2}(X, u_2)h^{Y_3}(X, u_3)$$

is exactly the equation for the calculation of the coefficients in the polynomial product:

$$\mathbb{P}_{h^{\{u_2,u_3\}}} = \mathbb{P}_{h^{Y_2}} \times \mathbb{P}_{h^{Y_3}} \tag{5.27}$$

It follows then that for a given number N of Y_i nodes, variable elimination using a decomposition graph takes the same amount of time as the generating function polynomial multiplication. However, as we have shown earlier using the FFT reduction is computationally faster compared to the generating function approach yielding potentially a speedup compared to CI-based bucket elimination.

In general, suppose in the variable elimination calculation we have the following sequence of buckets in the *ci-elim-bel* algorithm for a temporal decomposition of N, Y_i nodes:

- bucket $\{u_1, u_2\}$: $h^{\{u_1,u_2\}}(X, z_1) =$
  $\sum_{\{u_1,u_2|z_1=u_1+u_2\}} h^{Y_1}(X, u_1)h^{Y_2}(X, u_2)$

- bucket $\{u_3, z_1\}$: $h^{\{u_3,z_1\}}(X, z_2) =$

  $\sum_{\{u_3,z_1|z_2=u_3+z_1\}} h^{Y_3}(X, u_3)h^{\{u_1,u_2\}}(X, z_1)$

- bucket $\{u_4, z_2\}$: $h^{\{u_4,z_2\}}(X, z_3) =$

  $\sum_{\{u_4,z_2|z_3=u_4+z_2\}} h^{Y_4}(X, u_4)h^{\{u_3,z_1\}}(X, z_2)$

- $\vdots$

- bucket $\{u_N, z_{N-2}\}$: $P(Z|X) =$

  $\sum_{\{u_N,z_{N-2}|z=u_N+z_{N-2}\}} h^{Y_N}(X, u_N)$

  $h^{\{u_{N-1},Z_{N-3}\}}(X, z_{N-2})$

Computationally, the bucket computations will take $O(N^2|D_U|^2|D_X|)$ time. Observe that if the Y_i's are identically distributed conditioned on X, we have that

$$h^{Y_i}(X, u_i) = h^{Y_j}(X, u_j), \;\; \forall i, j \in [1, N] \tag{5.28}$$

Theorem 5.2.2.1 (FFT bucket elimination)**.** *Given a sequence of N temporal decomposition buckets in ci-elim-bel for N $i.i.d$ Y_i's that conditionally depend on variable X with domain size $|D_X|$ and sum to variable Z, it is possible to compute the bucket sequence in $O(|D_X|R_Z \log R_Z)$ time where R_Z is the numerical range of the domain of the final bucket.*

Proof. We perform the following FFT reduction:

1. Define functions F_{Y_i}

$$F_{Y_i}(X, k) = h^{Y_i}(X, u_i = k)$$

	$i.i.d\, Y_i's$	non-$i.i.d\, Y_i's$
BE	$O(N\|D_X\|R_Z^2)$	$O(N\|D_X\|R_Z^2)$
FFT	$O(\|D_X\|R_Z \log R_Z)$	$O(N\|D_X\|R_Z \log R_Z)$

Table 5.1: Time complexity for computation of temporal decomposition buckets. Top row (Bucket Elimination), Bottom row (FFT)

2. Perform the FFT: $(O(|D_X|R_Z \log R_Z))$

$$\hat{F}_{Y_i}(X,t) = \mathcal{F}\{F_{Y_i}(X)\}(t)$$

3. Exponentiate the function $\hat{F}_{Y_i}$: $(O(R_Z))$

$$\hat{F}_{Y_i}(X,t)^N$$

4. Perform the inverse FFT: $(O(|D_X|R_Z \log R_Z))$

$$P(Z|X) = \mathcal{F}^{-1}\{\mathcal{F}\{F_{Y_i}(X)\}^N\}(Z)$$

Which in total takes $O(|D_X|R_Z \log R_Z)$ time. □

If the Y_i's are not identically distributed, we will need to perform an FFT for each F_{Y_i} and replace step 3 with pointwise multiplications which in total will take $O(NR_Z)$ time bringing the total time to $O(N|D_X|R_Z \log R_Z)$. A comparison of the different methods can be seen in Table 5.1. For comparison purposes, we can approximately substitute $N|D_X| \approx R_Z$.

Extending *ci-elim-bel*. Using *ci-elim-bel* as the baseline, we can replace the computation of $+$ in addition based networks with the FFT to create the *ci-elim-FFT* algorithm (see Algorithm 6). The FFT operation to compute h^{Z_i} is performed with respect to the additive variables y_l and y_k.

Algorithm 6: ci-elim-FFT
Input: A Bayesian network B, evidence e
Output: $P(x_1|e)$

```
1  Generate a decomposition network from $B$
2  Generate ordering $o = \{Z_1, \dots, Z_n\}$ with $Z_1 = \{x_1\}$ using $B'$
3  for $i = n \to 1$ do // create buckets
4      $\forall x \in Z_i$, put all network functions with $x$ as highest ordered variable in $bucket_i$
5  end
6  for $i = n \to 1$ do // process buckets
       // $h_1, \dots, h_m$ are functions in $bucket_i$
7      if $Z_i = \{u_l; u_k\}, u = u_l + u_k$ then
8          $h^{u_l} = \prod_{j, u_l \in h_j} h_j$ ;
9          $h^{u_k} = \prod_{j, u_k \in h_j} h_j$ ;
10         $h^{Z_i} = \mathcal{F}^{-1}\{\mathcal{F}\{h^{u_l}\} \cdot \mathcal{F}\{h^{u_k}\}\}$
11     else
12         use regular ci-elim-bel to compute $h^{Z_i}$
13     Put $h^{Z_i}$ in the highest bucket that mentions $h^{Z_i}$'s variable.
14 end
15 Return $\alpha h^{x_1}$, ($\alpha$ is a normalizing constant).
```

This means that we evaluate:

$$h^{Z_i} = \mathcal{F}^{-1}\{\mathcal{F}\{h^{y_l}\} \cdot \mathcal{F}\{h^{y_k}\}\} \tag{5.29}$$

for every value of the variables of h^{Z_i} except y, y_l, y_k.

5.3 Experimental Evaluation

We evaluate the effectiveness of both approaches proposed for accelerating inference in Bayesian Networks. Specifically we evaluate the FFT reduction technique [as in section 5.2.1]

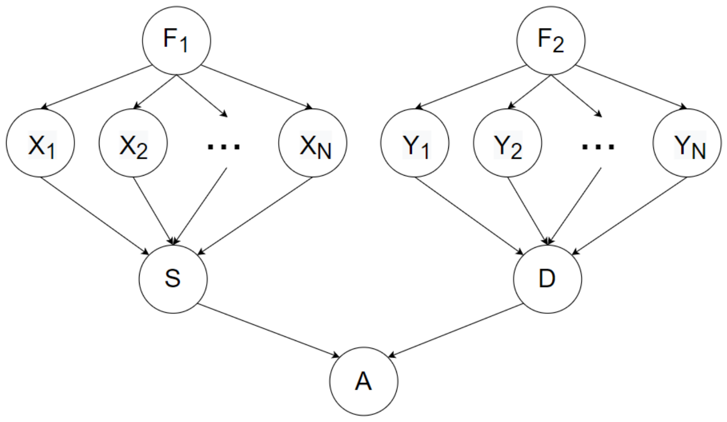


Figure 5.5: Two branch supply-demand network

for network transformation on a selection of common substructures that may be present in larger networks. We also evaluate the performance of *ci-elim-FFT* compared to *ci-elim-bel* on general two layer additive networks also known as $k - n$ networks. It has been shown that *ci-elim-bel* can speed up inference exponentially on $k - n$ networks when $k > 2n$ [70]. We test the scaling efficiency of *ci-elim-FFT* in cases where $k > 2n$ where one might want to use *ci-elim-bel*.

5.3.1 Experimental Setup

We evaluate all inference tasks on a 64-bit machine with an Intel i7-10870H 2.2 GHz CPU and 32 GB of RAM. The models are written in python with the library pomegranate and bucket elimination is performed using the library's pgmpy's built-in variable elimination. The FFT is computed using the library numpy. We also compare python implementations (not using pgmpy) of *ci-elim-FFT* with *ci-elim-bel* where the only difference is the method for computing buckets with two hidden variables.

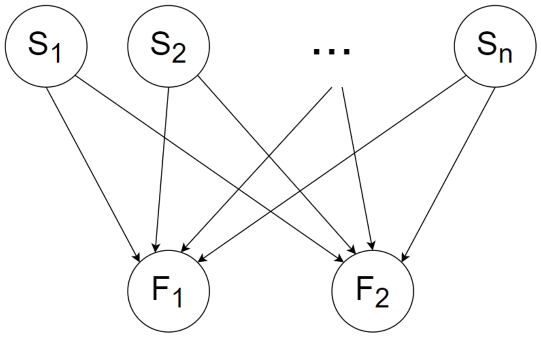


Figure 5.6: Two layer additive network

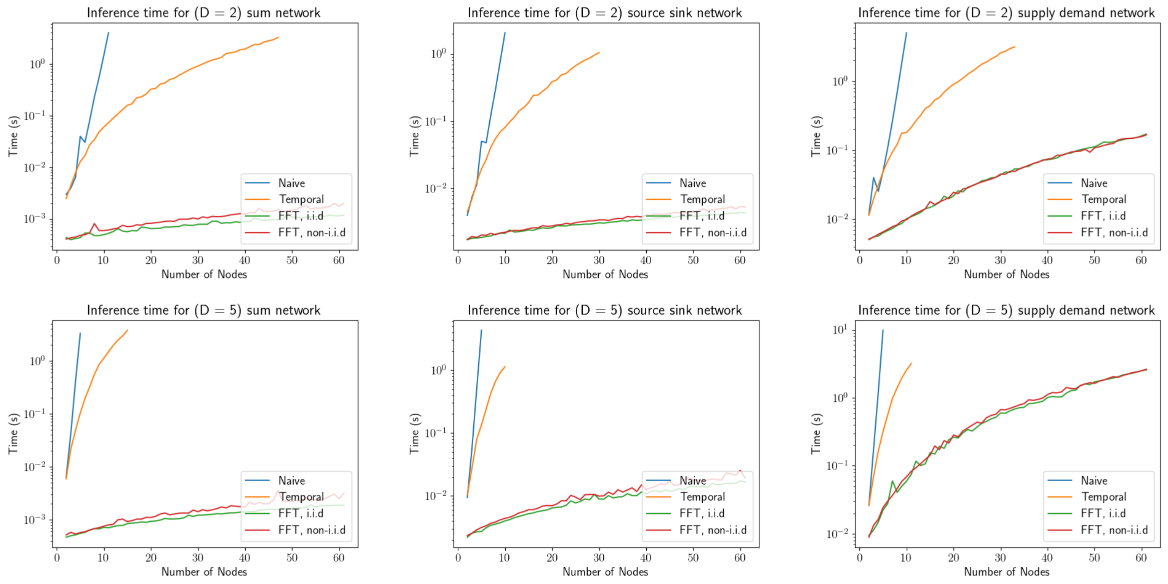


Figure 5.7: Inference times for three scenarios controlled for domain sizes (D = 2, D = 5)

5.3.2 FFT Reduction Evaluation

We evaluate the FFT reduction for accelerating inference on a selection of networks with casually independent summation nodes. For each network scenario, we test three different approaches:

1. vanilla bucket elimination (Naive)
2. bucket elimination on a temporal network decomposition as in [63] (Temporal)
3. FFT reduction [as in Section 5.2.1] followed by bucket elimination (FFT)

We test each approach in finding the marginal distribution of a specific variable in three scenarios:

1. $P(K=k)$ in an n-parent convergent network (Fig. 5.1) found in the distributed resource applications described in [63]
2. $P(E=e)$ in an n-path source-sink graph (Fig. 5.2) which can be found as substructures of the Bayesian Networks for evolutionary games
3. $P(A=a)$ in a supply and demand model for distributed resource production and consumption shown in Fig. 5.5. This consists of two n-parent convergent networks that feed into a boolean-valued node that is True if supply is larger or equal to than demand and False otherwise.

For each network class, we evaluate end-to-end inference time (construction time and inference time) for increasing numbers of casually independent nodes (S_i in Fig. 5.1, Y_i in Fig. 5.2, and X_i, Y_i in Fig. 5.5). We also evaluate inference time for the FFT approach on networks with $i.i.d$ intermediate nodes vs. non-$i.i.d$ nodes.

5.3.3 Results of FFT Reduction

In the three scenarios tested, as seen in Fig. 5.7, the FFT reduction method obtains a significant computational advantage over the naive approach as well as inference on the temporal decomposition. In Fig. 5.7, it is easy to see how each method scales relative to their theoretical complexity. The naive bucket elimination approach scales exponentially while the temporal decomposition method scales at a cubic rate. In comparison, employing the FFT reduction to remove $i.i.d$ nodes before performing bucket elimination significantly decreases the computational cost of subsequent inference.

Surprisingly, there is not much of a difference between networks with $i.i.d$ nodes versus non-$i.i.d$ nodes for the FFT approach, suggesting that the inference time for the reduced network overshadows the complexity of performing the FFT reduction.

5.3.4 Evaluating Ci-elim-FFT

We compare the computational efficiency of *ci-elim-bel* with *ci-elim-FFT* on 2-layer additive networks (Fig. 5.6). This is a generalization of the Binary 2-layer Noisy-Or (BN2O) networks commonly used in medical diagnosis. Instead of binary variables, we assume that the sources can be stochastically transformed into integer-valued variables which sum to an integer valued result. These types of networks can be used to model numerical medical findings such as body temperature or blood test results where separate causes (diseases and medications) have additive effects on the observed findings. We measure the efficiency of finding

$$P(S_1|F_1 = e_1, F_2 = e_2) \tag{5.30}$$

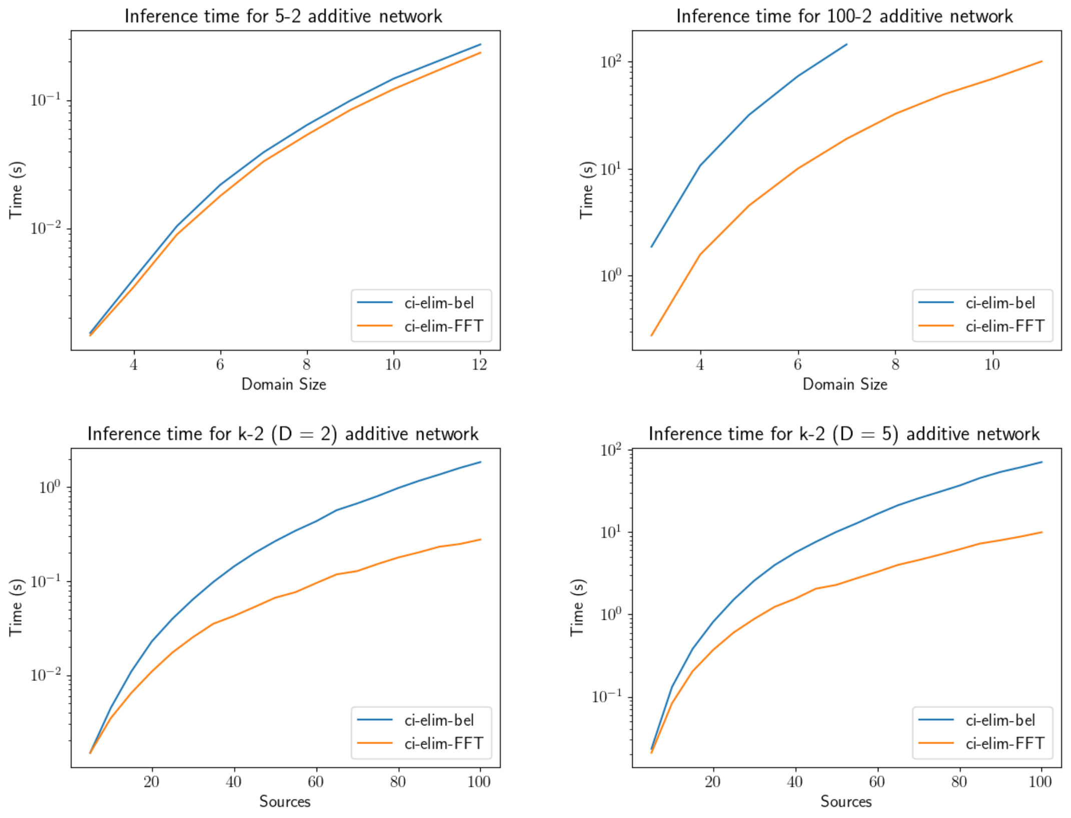


Figure 5.8: Inference times for two layer additive networks controlling for domain size (top) and number of source nodes (bottom)

given findings (e_1, e_2). We control for the number of causes ($\{S_1, \ldots, S_N\}$ in Fig. 5.6) as well as the domain size of each variable.

Fig. 5.8 demonstrates *ci-elim-FFT*'s clear advantage over *ci-elim-bel*. This advantage increases both as the number of source nodes increases and as well as when the domain size of variables increases.

5.4 Related Work and Limitations

Our work is not the first work which has attempted to apply the Discrete Fourier Transform (DFT) to exact probabilistic inference on Bayesian Networks. The DFT can be thought of as a special case of the tensor decomposition approach first described in [71]. In [72], the DFT was applied as a special case of the tensor decomposition towards test score prediction on a subset of the source-sink type networks that we discussed in section 3. However, in their approach, a naive application of the DFT was used that relies on matrix multiplication which is equivalent to computing Eq. 5.11 directly. Due to this, we believe that our FFT based approach can provide a method for lowering the computational cost of test score prediction even further beyond the existing DFT approach using matrix multiplication. Furthermore, our work also improves upon prior work by directly integrating the FFT into a general bucket elimination algorithm.

The FFT itself has also been applied towards the calculation of random variable sums in [73], but not directly towards the similar problems as found in summation type nodes in Bayesian Networks or combined with algorithms such bucket elimination.

Conceptually, the problem of accelerating the computation of summation-based CPTs is also similar to the problem addressed through the use of counting factors in the literature on lifted variable elimination [74]. However, there are a few key differences. Most notably, while lifting is said to work on general models, in practice several restrictions must be satisfied for lifting to provide a computational advantage. In particular, it is necessary for the factors present in the equation to be identical. This allows work to be exponentiated out through symmetry and is equivalent to the case of $i.i.d$ variables in our work. If the factors are not identical, then it is necessary to perform grounding, defaulting to basic variable elimination.

While our FFT method provides the greatest computational advantage when all variables are $i.i.d$, we note in corollary 3.3.1 and in the paragraph right after theorem 3.2, that the method can still be applied even when the variables are not $i.i.d$, providing theoretical speedup (as shown in Table 5.1) and an empirical speedup as observed (in Figure 5.7, 5.8). Importantly, CI-elim-FFT can be applied to any Bayesian Network that include summation nodes, and not just to the four types of networks demonstrated in the empirical results. For general networks, the only requirement for ci-elim-FFT to provide a speedup over a temporal decomposition/ci-elim-bel is the presence of summation type nodes.

It may be interesting to incorporate our FFT method as a type of lifted operator for counting factors over asymmetric variables, extending ci-elim-FFT into an analagous lifted version. We leave this problem for future research in this area.

5.5 Summary

In this chapter, we have presented an efficient method for reducing the size of summation-based Conditional Probability Tables (CPTs) in Bayesian Networks having *causal independence* (CI). We also have shown how to apply this reduction directly towards the acceleration of Bucket

Elimination. We have provided experimental results showing the FFT reduction's advantage for inference on a selection of common sub-networks found in Bayesian Networks for modeling distributed resource. We have developed an extension to *ci-elim-bel* called *ci-elim-FFT* and provided empirical results that demonstrate its scaling advantages.

Chapter 6: A Mean Field Game Model of Spatial Evolutionary Games

So far, we have mainly discussed evolutionary games, their spatial extensions, the methods used for approximating their behavior, and methods for improving the quality of those approximations. A Mean Field Game (MFG) is a different type of game altogether from evolutionary games, but is likewise used to analyze the interactions between a large system of agents. In evolutionary game theory, mean field approximations are used to approximate the dynamics of the population by only analyzing a set of mean field terms.

Due to their properties, existing Mean Field Games are not capable of modeling an evolutionary game and similar spatial systems. To address this, we have developed our novel Pair Approximation Mean-Field game (Pair-MFG) model that is a generalization of the spatial evolutionary game model such that the behavior of a given spatial evolutionary game (or more specifically the behavior of its pair approximation) is a special case trajectory of the corresponding Pair-MFG. In Section 6.1, we will start by giving a brief background on Mean Field Game Theory. This is followed in 6.2, where we describe the process for defining a Pair-MFG building off of existing discrete MFGs. We then discuss our novel fixed point methods in Section 6.3 for solving these models and then empirically demonstrate that these methods are capable of solving the models in Section 6.4.

6.1 Background on Mean Field Games

Mean Field Games (MFG) are a game theoretic model of large populations first developed independently by Huang et al. [75] and Lasry and Lions [76] as a formalism for approximating the behavior of a large number of rational agents. The central idea behind this model is the notion of a mean field, or an aggregate population distribution. The combinatorial number of interactions between players is approximated as a single interaction between a representative agent and the aggregate mean field distribution. This approximation error tends to zero as the number of agents becomes large. In MFGs, each agent independently optimize some cost function that depends on a mean field term. The goal of each agent in an MFG is to optimize their control (strategy) with respect to the cost (payoff) they receive. It is important to note that this is different from evolutionary games where the strategy of each agent simply evolves according to some evolutionary update rule.

Since its initial inception, many types of MFGs have been developed that encompass a wide variety of mathematical models that allow for the simulation and analysis of large populations of rational agents. In principal, like mean field approximations for evolutionary games, MFG's can be thought of as a type of macroscopic model. In this case, the macroscopic model is used to approximate the behavior of a corresponding microscopic model which consists of a multi-agent optimal control problem. An MFG consists of two components:

- A state evolution equation or **forward equation**: each agent has some state that evolves over time. As an example, let x_i be the state of an agent in some continuous state space, one possible forward equation is:

$$\frac{\partial x_i}{\partial t} = M(x_i(t), \alpha(t)) + w(t) \tag{6.1}$$

where α is some control variable and w is additional random noise.

- An objective function or **backward equation**: each agent has some objective function that they are tasked with optimizing over the time horizon. As an example, consider a finite time horizon $[0, T]$, one possible backward equation is:

$$\min_{\alpha} \int_0^T F(x_i(t), \theta(t), \alpha(t)) + G(x_i(T), \theta(T)) \tag{6.2}$$

  where θ is a **mean field** term that denotes the average state of agents in the population, F is the running cost function, and G is a terminal cost function.

Like the mean field approximation in spatial evolutionary games, the forward equations of each individual agent can be rewritten to be formulated as a time evolution equation defined on the population profile θ, instead of the individual x_i's. The primary difference between spatial evolutionary games and mean field games is the presence of the backward equation which allows for each individual in the population to have agency.

In existing literature, there are many types of mean field games. For the purpose of this discussion, we'll give an example of three types of MFGs that each deal with spatial systems differently:

- *Linear Quadratic Gaussian Mean Field Games (LQG-MFG)*: These types of games [77] consider well mixed populations in which each agent interacts equally with every other agent in the population. In these types of games, constraints are placed on the format of the cost functions (quadratic) and the forward equation (linear). This allows for efficient algorithms for solving these games. As a result of being a well-mixed model, these types of games lack a spatial component.

- *Optimal Transport Mean Field Games*: In the optimal transport mean field game [78], the state space of each agent is their location in some real space. The control of each agent is their velocity in this real space. While these types of models have a spatial component, it is equivalent to their strategy component and thus these MFGs can't be used to model strategy evolution on networks.

- *Graphon Mean Field Games (GMFG)*: A graphon mean field game GMFG [79] [80] is a type of multi-population mean field game where a population of agents is divided into subclusters located on nodes within a network. In this model, the dynamics and/or cost function of an agent decompose into two components: a local well-mixed component and an averaged effect over other population clusters. To our knowledge, this is the only type of currently existing MFG model that can support distinct spatial and strategy components. However, GMFGs differ from our approach in how the spatial component is handled. In GMFGs, the spatial component is analyzed using a graphon, which is essentially the infinite limit of an adjacency matrix. Since the effect of the spatial component is computed using a pair-wise kernel integrated over the graphon, it is impossible to model nonlinear effects of local configurations.

Due to their properties, none of these prior formulations are capable of modeling an evolutionary game and similar spatial systems. To address this, in the following sections, we will describe our novel Pair Approximation Mean-Field game (Pair-MFG) that is a generalization of the spatial evolutionary game model such that the behavior of a given spatial evolutionary game (or more specifically the behavior of its pair approximation) is a special case trajectory of the corresponding Pair-MFG.

6.2 Mean-Field Game Formulation

In this section, we formulate the Pair Approximation Mean-Field Game (Pair-MFG), a Mean Field Game (MFG) model that generalizes the behavior of spatial evolutionary games. First, we will give an example of how a continuous MFG is formulated through the LQG-MFG followed by a discrete version of that game.

6.2.1 Continuous Linear Quadratic Gaussian MFG

A Linear Quadratic MFG models [81] a population of agents $\{1, \dots, n\}$ where each agent i individually minimizes a cost function:

$$\begin{aligned} E\big[\frac{1}{2}\int_0^T & x_t^T F_t x_t + (x_t - \overline{x}_t)^T \overline{F}_t (x_t - \overline{x}_t) + v_t^T L_t v_t dt \\ & + \frac{1}{2} x_T^T G_T x_T + (x_T - \overline{x}_T)^T \overline{G}_t (x_T - \overline{x}_T)\big] \end{aligned} \tag{6.3}$$

where:

- x_t is a vector specifying the agent's state
- $\overline{x}_t$ is a vector specifying the average of all agent states
- v_t is a vector specifying the agent's control

- F_t is a matrix specifying the quadratic running cost of being at state x_t
- L_t is a matrix specifying a quadratic transport cost of using control v_t
- $\overline{F}_t$ is a matrix specifying the quadratic running cost of being at state x_t with respect to the population average $\overline{x}_t$
- G_T and $\overline{G}_T$ being the terminal cost versions of F

The position of each agent changes according to a state evolution equation:

$$dx_t = \big(A_t x_t + B_t v_t + \overline{A}_t \overline{x}_t\big)\, dt + \sigma_t dW_t \tag{6.4}$$

where $A_t, \overline{A}_t$, B_t are some matrices that describe the movement of the agent as a function of its control v_t, current state x_t and the average of other agent states $\overline{x}_t$. The last term $\sigma_t dW_t$ can be used to describe additional Gaussian noise present in the state evolution equation.

6.2.2 Discrete Mean Field Game

Consider the discrete version of the LQG-MFG model presented in section 2 of [82]. Suppose each agent x can be in a set of states $s_i \in S$ and the state of a player evolves according to a controlled Markov process:

$$P\big(x(t+1) = s_j \mid x(t) = s_i\big) = \alpha_{ij}(t) + \eta_{ij} \tag{6.5}$$

Each agent specifies *controls* (or actions [1]) α_{ij} that correspond to transition probabilities between all $s_i, s_j \in S, i \neq j$ and each agent tries to minimize the objective function:

$$\min_{\alpha} E\left[\int_0^T C(x(t), \theta(t), \alpha(t))dt + G(x(T), \theta(T))\right] \tag{6.6}$$

where $\theta(t)$ describes the distribution of other players in each state at time t and the cost function C is separated into the running cost F and a quadratic energy cost:

$$C(x(t), \theta(t), \alpha(t)) = F(x(t), \theta(t)) + \frac{1}{L}\frac{\|\alpha(t)\|^2}{2} \tag{6.7}$$

where L is some constant and

$$\|\alpha(t)\|^2 = \sum_{i \neq j} \alpha_{i,j}(t)^2 \tag{6.8}$$

6.2.2.1 Well-mixed EGT

Given the above discrete framework, it is easy to formulate an MFG model analogous to an EGT model for a well mixed population. We define $F(x(t), \theta(t))$ to be:

$$F(x(t), \theta(t)) = \sum_i \mathbf{P}_{xi}\theta_i \tag{6.9}$$

where $\mathbf{P}_{xi}$ is the (x, i)-th entry of the payoff matrix $\mathbf{P}$. The running cost is computed against $\theta(t)$ which describes the average state distribution of the entire population. In essence, an agent will

[1]In reinforcement learning literature, this would correspond to actions taken at each timestep

receive a cost from playing a game with a well-mixed population.

6.2.3 Pair Approximation Mean Field Game

We make an observation that, in the discrete MFG framework, the mean field $\theta(t)$ represents the distribution of a single agent. In EGT literature, this is the same property as evolutionary dynamics defined on a well-mixed population such as the replicator dynamics. As mentioned in Section 2.2.1, pair approximation is a natural extension of evolutionary dynamics to structured populations where the equations use higher order distributions. Suppose then, in a manner similar to pair approximation, we define a pair-level mean field distribution $\theta(t)$ over $S \times S$. This will be the basis for our Pair-MFG model.

The Pair-MFG builds on top of the discrete MFG through the addition of a local density mean field. We denote this local density by θ_{ij}. We define our running cost F for a single agent in the state x to be:

$$\begin{aligned} F(x(t), \theta(t)) &= \sum_i \mathbf{P}_{xi} \frac{\theta_{xi}(t)}{\theta_x(t)} \\ \theta_x(t) &= \sum_j \theta_{xj}(t) \end{aligned} \tag{6.10}$$

Like in the discrete MFG model we define a state evolution equation based on a Markov process, this time using a more complicated equation using the pair-level mean field:

$$P(x(t+1) = s_j | x(t) = s_i) = \sum_{cf \in C_d} P_{cf}(i, t) \cdot (\alpha_{ij}(t, cf) + \eta_{ij}) \cdot \gamma \tag{6.11}$$

where C_d is the set of all possible local neighborhood assignment configurations for an agent with d neighbors and γ is the percentage of agents that can change their strategies during each generation. More rigorously, we define C_d as the set of tuples:

$$C_d = \{(x_1, x_2, ..., x_{|S|}) \;\; s.t. \sum x_i = d\} \tag{6.12}$$

where x_i denotes the number of neighbors of the agent x that are playing strategy i and d is the degree of the network. We constrain the control variables $\alpha_{ij}(t, \mathit{cf}) \in [0, 1]$ to ensure that equation 6.11 produces a valid probability distribution. $P_{\mathit{cf}}(i)$ denotes the probability of the configuration *cf* and is a function of $\theta(t)$ and we can approximate it by assuming all neighbors are independently distributed:

$$\begin{aligned} P_{\mathit{cf}}(i,t) &= \binom{d}{x_1, x_2, ..., x_{|S|}} \prod_j \left(\frac{\theta_{ij}(t)}{\theta_i(t)}\right)^{x_i} \\ \mathit{cf} &= (x_1, x_2, ...x_{|s|}) \end{aligned} \tag{6.13}$$

In this setting, we define controls $\alpha_{ij}(\mathit{cf})$ for each transition $i \to j$ that is some function of *cf*. From Eq. 6.11, using pair approximation, we can derive the time evolution of joint probabilities:

$$\begin{aligned} \theta_{ij}(t+1) &= \theta_{ij}(t) + \Delta\theta_{ij}(\alpha, t) \\ \Delta\theta_{ij}(\alpha, t) &= \sum_k \sum_{\mathit{cf} \in C_d} \Big[\theta_k(t) P_{\mathit{cf}}(k,t) \frac{N(j, \mathit{cf})}{d} \alpha_{ki}(t, \mathit{cf})\gamma \\ &\quad - \sum_{\mathit{cf} \in C} \theta_i(t) P_{\mathit{cf}}(i,t) \frac{N(j, \mathit{cf})}{d} \alpha_{ik}(t, \mathit{cf})\gamma\Big] \end{aligned} \tag{6.14}$$

where $N(j, \mathit{cf})$ denotes the number of nodes assigned j in neighborhood configuration *cf* or the x_j value in the *cf* tuple. To simplify notation for further discussion, let M denote the mean field evolution equations that produce $\theta(t)$ according to Eq. 6.14:

$$\theta = M(\alpha, \theta(0)) \tag{6.15}$$

6.2.4 An example

Let us define a pair-MFG for modeling an evolutionary game on a square lattice grid ($d = 4$) with two strategies $[0, 1]$ and the following payoff matrix:

$$\mathbf{PAY} = \begin{bmatrix} 2 & -1 \\ 3 & 0 \end{bmatrix} \tag{6.16}$$

For our MFG model we want low costs for an agent to correspond to high payoffs in the original EGT. A valid transformation for this mapping is to subtract $\mathbf{PAY}$ from the maximum value of $\mathbf{PAY}$ and add a small constant cost, $\mathbf{P} = \max(\mathbf{PAY}) - \mathbf{PAY} + 1$:

$$\mathbf{P} = \begin{bmatrix} 2 & 5 \\ 1 & 4 \end{bmatrix} \tag{6.17}$$

Our mean field distribution is defined as four values: $\theta_{00}, \theta_{01}, \theta_{10}, \theta_{11}$ that sum to 1 and two additional computed values: $\theta_0 = \theta_{00} + \theta_{01}$, $\theta_1 = \theta_{10} + \theta_{11}$ Our running cost F is:

$$F(x(t), \theta(t)) = \begin{cases} \frac{1}{\theta_0}\left(2\theta_{00} + 5\theta_{01}\right) & x = 0 \\ \frac{1}{\theta_1}\left(1\theta_{10} + 4\theta_{11}\right) & x = 1 \end{cases} \tag{6.18}$$

and our terminal cost is the same: $G(x(T), \theta(T)) = F(x(T), \theta(T))$

Since our model only has two strategies, a configuration *cf* can be represented with just one variable $c \in [0,4]$ that denotes the number of agents in the neighborhood playing the first strategy. As an example, we have that θ_{00} evolves as such:

$$\begin{aligned} P\big(x(t+1){=}1 \mid x(t){=}0\big) &= \sum_{c=0}^{4} P_c(0) \cdot \alpha_{01}(t,c)\gamma \\ \Delta\theta_{00} &= \sum_{c=0}^{4} \theta_1 P_c(0) \frac{c}{4} \alpha_{10}(t,c)\gamma - \theta_0 P_c(0) \frac{c}{4} \alpha_{01}(t,c)\gamma \\ P_c(0) &= \binom{4}{c} \left(\frac{\theta_{00}}{\theta_0}\right)^{c} \left(\frac{\theta_{01}}{\theta_0}\right)^{4-c} \end{aligned} \tag{6.19}$$

The transition probabilities and equations for other θ_{ij} are similarly defined.

6.2.5 Features

6.2.5.1 Generalization of best response

The model is a generalization of a spatial evolutionary game using the best response update rule. To see how this is the case, suppose we limit our pair-MFG model to one generation $t \in \{0,1\}$.

Theorem 6.2.5.1 (Generalization of best response)**.** *There exists a set of controls α (and cost functions C, G) such that the pair-MFG model is equivalent to a pair approximation model of a best response spatial evolutionary game over the time interval $t \in \{0,1\}$.*

Proof. Consider a best response spatial evolutionary game. A given agent x playing strategy i will change its strategy based on the strategies of its neighbors. This is a deterministic function R of a given neighborhood configuration *cf*:

$$R(i,j,cf) = \begin{cases} 1 & Pay(j,cf) > Pay(k,cf) \;\; \forall k \in S \\ \\ 0 & \text{otherwise} \end{cases} \tag{6.20}$$

The above function is the probability in the best response model that a given agent x playing i will change its strategy to j. In our pair-MFG formulation, the control term α models the probability of an agent changing its strategy given a neighborhood configuration *cf*. If we let our control terms be:

$$\alpha_{ij}(cf) = R(i,j,cf) \tag{6.21}$$

the equations 6.11 and 6.14 are then equivalent to the pair approximation equations for a best response spatial evolutionary game. Now consider the cost function:

$$\alpha^* = \arg\min_{\alpha} E\left[C(x(0),\theta(0),\alpha(0))dt + G(x(1),\theta(1))\right] \tag{6.22}$$

Replacing the C and G terms:

$$\begin{aligned} \alpha^* &= \arg\min_{\alpha} E\left[F(x(0),\theta(0)) + \frac{1}{L}\frac{\|\alpha(t)\|^2}{2} + G(x(1),\theta(1))\right] \\ &= \arg\min_{\alpha} E\left[\frac{1}{L}\frac{\|\alpha(t)\|^2}{2} + F(x(1),\theta(1))\right] \end{aligned} \tag{6.23}$$

If we make two additional assumptions for the terms in C, G:

- let L be a large enough such that $\frac{1}{L}\frac{\|\alpha(t)\|^2}{2} << G(x(1),\theta(1))$ or equivalently that the energy term does not significantly impact the minimum of our cost function.

- let $\gamma << 1$, so that $\theta(1) \approx \theta(0)$

we have that

$$\alpha^* \approx \arg\min_{\alpha} E\left[F(x(1), \theta(0))\right] \tag{6.24}$$

which is the decision rule for a best response spatial evolutionary game. As $\gamma \to 0$, the closer the pair-MFG model approaches the best-response spatial evolutionary game.

6.2.5.2 Beyond best response

A key concept in the pair-MFG model is that the agent's control is their transition probability given a certain neighborhood configuration. The original spatial evolutionary game is a special case of the pair-MFG where the "control" probabilities are determined using the update rule. Following this logic, we can model any update rule that can be specified by local configurations of strategy assignments. In section 6.2, we defined our configuration space as:

$$cf \in C_d, \;\; C_d = \{(x_1, x_2, ..., x_{|S|}) \;\; s.t. \sum x_i = d\} \tag{6.25}$$

For the Fermi rule, we can define a configuration space:

$$cf \in S \times C_{d-1}^2, \;\; S \times C_{d-1}^2 = \{(o, n_1, n_2) | o \in S, n_1, n_2 \in C_{d-1}\} \tag{6.26}$$

The new *cf* defines a neighbor o and the neighborhood distributions of the other $d-1$ neighbors of our agent and the $d-1$ neighbors of o. By replacing C_d in equations 6.11 and 6.14 with the new configuration space, we obtain a pair-MFG model that generalizes spatial evolutionary games that use the Fermi rule. Like for best response, there exists values of the control $\alpha_{ij}(cf)$ such that the pair-MFG model is equivalent to the pair approximation equations for the Fermi rule spatial evolutionary game.

6.3 Solving the Model

The Pair-MFG model is not necessarily monotone. As a result, the solution to a Pair-MFG may not necessarily be unique. We define a solution to our pair-MFG model as a pair: (θ^*, α^*) such that:

$$\alpha^* = J(\theta^*) = \text{arg}\min_{\alpha} E\left[\sum_{t}^{T-1} C(x(t), \theta^*(t), \alpha(t)) + G(x(T), \theta(T))\right]$$

$$P(x(t+1){=}s_j \mid x(t){=}s_i) = \sum_{cf \in C_d} P_{cf}(i,t) \cdot \alpha_{ij}(t, cf) \cdot \gamma$$

$$\theta^* = M(\alpha^*, \theta(0)) \tag{6.27}$$

given an initial condition $\theta(0)$.

6.3.1 Existence

Our optimization problem is not necessarily convex given certain specifications of the payoff matrix $\mathbf{PAY}$ and subsequently the cost function C, but because both θ and α are defined on convex compact sets there exists some (θ, α) pair such that the above equations are satisfied.

Using the Brouwer/Kakutani fixed point theorem, it is easy to see that (θ^*, α^*) is a fixed point of the function:

$$(\theta, \alpha) \to (M(J(\theta)), J(\theta)) \tag{6.28}$$

which ensures that there exists a solution to the Pair-MFG model.

6.3.2 Fixed Point Iteration

Since we know that the optimal pair (θ^*, α^*) must be a fixed point of the mapping in Eq. 6.28, we can try iterating the function until convergence. However, this approach is not guaranteed to converge, since we have no guarantee that Eq. 6.28 is a contraction mapping. In fact, in [83], it was shown that in general, Eq. 6.28 is not a contraction mapping for any finite MFGs. As a result, there are many situations where fixed point iterations will get stuck alternating between two different paths.

6.3.2.1 Adjustments

We developed several novel adjustments that can be made to the naive fixed point iteration method that can improve its convergence:

- *Step-size*: The problem encountered by the fixed-point iteration is similar to the problem occasionally encountered by hill-climbing optimization routines without a step-size adjustment (eg. line search or backtracking). Assume that some black-box iterative optimization routine is used to optimize $J(\theta)$. Instead of running this optimization routine until convergence for each fixed point iteration, we can limit the number of iterations in the inner optimization loop and instead re-evaluate M more often. The number of inner black-box optimizer iterations then serves as the ”step-size” for the outer fixed point iteration.

- *Proximal terms*: Suppose we consider each fixed point iteration as a separate sub-problem in an algorithm for finding the solution of Eq. 6.27. At each iteration we add a proximal term to the objective function J that penalizes iterations α^n that move too far from the previous iteration α^{n-1}:

$$\begin{aligned}\alpha^{n+1} = J'(\theta^n) = \arg\min_{\alpha} E\big[\big(\sum_{t}^{T-1} C(x(t), \theta^n(t), \alpha(t)) \\ + \frac{1}{2}\|\alpha(t) - \alpha^n(t)\|\big) + G(x(T), \theta(T))\big]\end{aligned} \tag{6.29}$$

  However, since we apply a proximal term at each iteration, the proximal terms closer to $t = 0$ have a much larger effect than those later in time. Small changes in the trajectory early on can disproportionally impact the later trajectory and cost obtained.

- *Time-dependent Proximal terms*: Due to the property where small perturbations near $t = 0$ can drastically change later trajectories, simply adding a proximal term at each iteration is not likely to help the method converge. Since changes earlier in the time interval have more impact on the final trajectory, we will want to penalize early changes more than later changes. We define a new time-dependent proximal term:

$$\frac{1}{2W(t)}\|\alpha(t) - \alpha^n(t)\| \tag{6.30}$$

  such that the function $W(t)$ is a monotonically decreasing function of time.

- *Accumulating Proximal terms*: An alternative approach for penalizing drastic changes is to accumulate differences backwards in time. If large changes have already been made at a later time, earlier changes should be penalized more. Several approaches are possible such as:
  - A new function $W(t, \delta(i)\ \forall i > t)$ where $\delta(i) = \alpha(t) - \alpha^{n-1}(t)$ are the differences computed from the current backward pass
  - A new function $W_A(acc)$ where $acc = \sum_{\delta}(i)$ multiplied with the proximal term: $\frac{1}{2W(t)}\|\alpha(t) - \alpha^n(t)\| W_A(acc)$

Algorithm 7: Fixed-Point Iteration with Time-dependent Proximal terms
Input: α, θ_0
Output: α^*, θ^*

```
for $t \leftarrow 1$ to $T$ do
    $\theta_t = M(\alpha_{t-1}, \theta_{t-1})$ ;
end
Initialize $V(x,t), \forall t$ ;
while $\alpha, \theta$ not converged do
    $V(x,T) = G(x, \theta_T)$ ;
    $\alpha_p = \alpha$, $acc = 0$ ;
    for $t \leftarrow T-1$ to $0$ do // Bellman backwards
        let: $J_{x,t}(\alpha) = C(x, \theta_t, \alpha) + \sum_y P(y|x,\alpha)V(y,t+1) + \frac{1}{2W(t)}\|\alpha - \alpha_{t,p}\| W_A(acc)$ ;
        $\alpha_t = \arg\min_\alpha J_{x,t}(\alpha)$ ;
        $V(x,t) = J_{x,t}(\alpha_t)$ ;
        $acc = acc + \frac{1}{2}\|\alpha_t - \alpha_{t,p}\|$
    end
    for $t \leftarrow 1$ to $T$ do // Pair Approximation forwards
        $\theta_t = M(\alpha_{t-1}, \theta_{t-1})$ ;
    end
end
```

6.4 Empirical Results

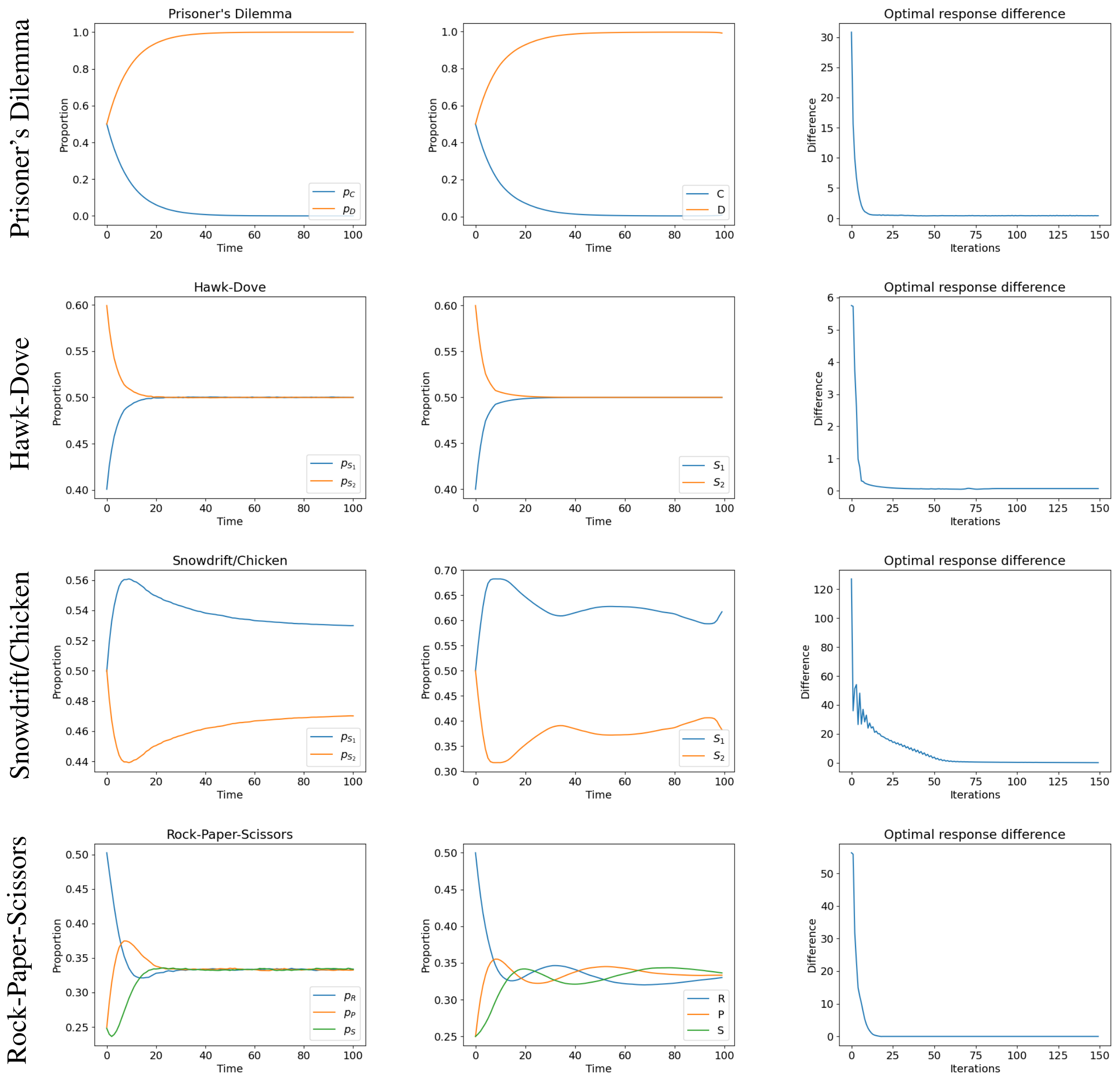
We evaluate the fixed point iteration method with proximal heuristics on the Pair-MFG model for several common games used in EGT models. In order to evaluate whether the fixed point iteration method can obtain a solution we compute the optimal response difference Δ_{opt}. Given iterates (θ^i, α^i) at iteration i of the fixed point algorithm, the optimal response difference Δ_{opt} is the difference between the two costs:

- Cost O_m obtained by an agent following the current controls α^i with respect to the induced mean field θ^i
- Cost O_d obtained by an optimal defecting agent who uses strategy $J(\theta^i)$ (where J is the Bellman equation with no proximal heuristics) against the mean field θ^i

$$\Delta_{opt} = O_m - O_d \tag{6.31}$$

It is easy to see that $\Delta_{opt} = 0$ only when a solution is found as this means a defecting agent has no incentive to unilaterally deviate from the current control iterate (the same concept as a Nash equilibrium). This quantity is similar to the notion of policy exploitability in section 6 of [83]. In Fig. 6.1, we demonstrate empirical results where the fixed point iteration method with proximal heuristics can reduce this quantity.

A solution to a MFG model (and thus the Pair-MFG model) is different compared to the solution of a spatial evolutionary game. Agents in the evolutionary game do not directly optimize their payoff over the entire time-horizon. Intuitively, the trajectories obtained by the Pair-MFG



(a) EGT Simulations (b) Pair-MFG Trajectory (c) Optimal Response Difference

Figure 6.1: Comparisons of (a) best-reponse EGT simulations with (b) Pair-MFG trajectories after 150 iterations of the fixed-point algorithm, in four evolutionary games. To demonstrate convergence of the fixed-point algorithm, column (c) shows the optimal-response difference at each of its iterations. The x-axes in column (c) denote iterations of the fixed-point algorithm (as opposed to iterations of the evolutionary game in parts (a) and (b)).

model are more optimal with respect to the possible trajectories of a rogue defecting/invading agent.

It is well known that Evolutionary Stable Strategies (ESS) are an equilibrium refinement of Nash Equilibrium. It is the same case here where only under certain conditions is it true that the models have equivalent trajectories. However, many of the behaviors of the spatial EGT simulation are preserved in the corresponding MFG model.

For example, in the Snowdrift game, a pull back after an initial increase in the proportion of the first strategy is observed in the Pair-MFG trajectory. This pull back is a result that cannot be obtained in a well-mixed population model for two strategies, but can frequently appear in spatial models. Another example of the preservation of spatial effects can be seen in the Rock-Paper-Scissors game. For the specific RPS payoff matrix used, the replicator equation and other well-mixed models predict a limit cycle where the magnitude of the cyclic waves for each strategy remains constant. In a spatial population, the population will instead converge to the Nash Equilibrium of $\{1/3\, R, 1/3\, P, 1/3\, S\}$ in which the magnitude of the cycle waves quickly dampen as the population evolves towards equilibrium.

As seen in the optimal response difference curves in the bottom row of Fig. 6.1, the proximal heuristics ensure that the optimal response difference values do not get stuck in a limit cycle. By adjusting the optimization rate for the control variables, we can avoid one of the major problems found in the application of fixed point iteration to solving mean field games.

6.5 Mean Field Type Control

Suppose we directly optimize Eq. 6.27 for α under the assumption that the joint state θ is controlled by the optimizer:

$$\begin{aligned}\alpha^* &= \arg\min_\alpha E\left[\sum_t^{T-1} C(x(t), \theta(t), \alpha(t)) + G(x(T), \theta(T))\right] \\ \theta &= M(\alpha, \theta(0))\end{aligned} \tag{6.32}$$

This transforms the problem into a single agent optimal control problem where the state consists of both the agent state x and the global joint state θ. This formulation is also known as a mean field type control (MFTC) problem. Note that if we let α' be the solution to the above optimal control problem and $\theta' = M(\alpha')$, we are not guaranteed that $\alpha' = J(\theta')$. The cost quantity in the objective function is likely to obtain a lower value than the one in Eq. 6.27. The MFTC model is analogous to having a central planner dictating individual agent strategies and is more akin to the idea of a pareto optimality rather than a nash equilibrium.

This relaxation of a Pair-MFG model, which we would denote Pair-MFTC, is applicable to many multi-agent optimization problems. One possible application is reactive risk management for large-scale intrusion detection. For example, using a Pair-MFTC model, it becomes possible to efficiently optimize over a spatial versions of the susceptible-infected-susceptible (SIS) model that more accurately captures network connectivity compared to existing well-mixed SIS models. A solution to the Pair-MFTC can be used to find optimal decentralized security policies for each agent to follow based on the infection status of their local network dependencies.

6.6 Summary

In this chapter, we have described Pair-MFG, a MFG generalization of the spatial evolutionary game models. The Pair-MFG model allows for the formulation of the spatial evolutionary game as a control problem, opening up additional avenues of research into controlling the outcomes of these games. We have provided a walkthrough of the derivation of a Pair-MFG model from the equivalent EGT model and shown that the behavior of a given spatial evolutionary game (or more specifically the behavior of its pair approximation) is a special case trajectory of the corresponding MFG. We have provided a method for solving this new Pair-MFG model using fixed point iteration with time-dependent proximal terms and show empirically that this method is capable of finding a solution to a selection of EGT games.

Chapter 7: Bayesian Mean Field Games, Control Problems, and Applications

In Chapter 6, we defined the Pair-MFG model which approximates the forward dynamics of a spatial evolutionary game using pair approximation. The transition probabilities in the pair approximation equations correspond to the control variables to be optimized over in the control problem. We have previously shown in Section 3.2.5 that a special case of the Truncated Dynamic Bayesian Network Approximation is computationally equivalent to pair approximation. The transition probabilities in the pair approximation equations are directly encoded into the Bayesian network itself.

In this chapter, we replace the forward equations in the Pair-MFG framework with an equivalent Truncated Dynamic Bayesian Network Approximation that is parameterized by the control variables of the MFG model. By using TDBNA we can define *Bayesian-MFG* models with higher accuracy than Pair-MFG addressing the inherent limits of the pair approximation based model. We expect that these more accurate models will also lead to better *policies* in the corresponding Bayesian-MFTC.

In Section 7.1, we will define our new Bayesian-MFG framework and how it can be relaxed into a Bayesian-MFTC in 7.1.1. We will then provide the mathematical approach for solving a Bayesian-MFTC from Section 7.2 to Section 7.2.3. In the rest of the chapter, starting from Section 7.3, we will verify that our hypothesis from Section 7.1.1 holds in a variety of application domains. Our goal is to confirm that using approximations that capture more accurate forward dynamics will lead to better control policies. Numerically, we will compare the objective function results for well-mixed, pair, and Bayesian MFTC problems and evaluate which method produces the policies that obtain the lowest costs.

It is important to note that by the objective function here, we mean the objective function results on the simulation. This is a two step process:

1. First, we solve the corresponding MFTC problem (well-mixed/pair/Bayesian) for a control policy π. This control policy takes in the mean field of the system θ and outputs a control α. The dimension of the state space is dependent on the complexity of the MFTC model. In the case of the Pair-MFG model, the control policy would take in the second order mean field $\theta_{ij}, \forall i, j$ and output an α for every configuration cf.

2. After solving the MFTC problem, we run an agent based stochastic simulation where the behavior (value of their α parameter) of all agents in the population are computed using the control policy obtained from the corresponding MFTC problem. We can compute the objective function directly on the simulation and that is the quantity we will use to compare the different MFTC problems.

For our empirical evaluation, we will consider spatial models originating from two domains: reaction-diffusion and network security.

7.1 A Unified Approximation Framework for Spatial Mean Field Games

Given a multi-agent system with a population of agents x_i, each optimizing an objective function $J(\alpha, \theta)$ with respect to a set of controls α and a mean field θ, suppose the state of

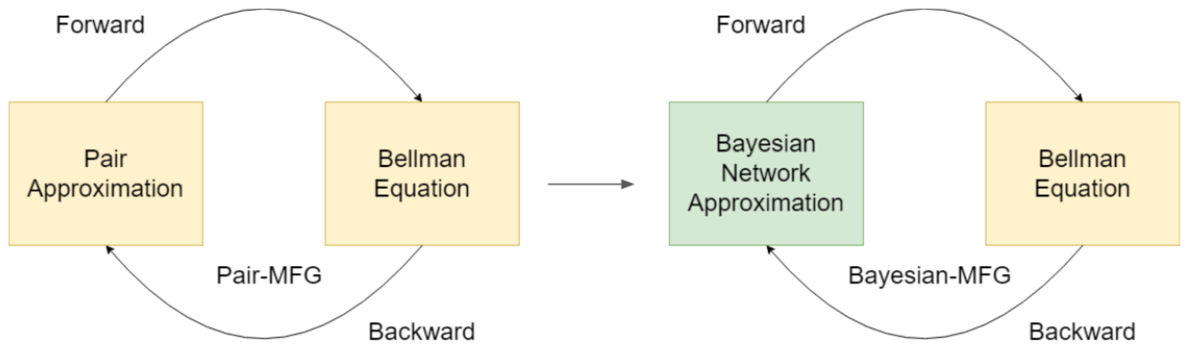


Figure 7.1: Bayesian-MFG structure. Compared to Pair-MFG, we replace the pair approximation equations and instead couple the Bellman equation with a Truncated Dynamic Bayesian Network Approximation.

each agent x_i follows some Markov process dependent on control α and the states of agents in a neighborhood $N(x_i)$:

$$P(x_i(t+1) = s'|x_i(t) = s) = F(s, s', x_j(t), j \in N(x_i))$$

We define a Bayesian-MFG as the following Mean-Field game:

$$J(\alpha, \theta), B(\alpha) \tag{7.1}$$

where $J(\alpha, \theta)$ is an objective function and $B = (I, T, O)$ is a Truncated Dynamic Bayesian Network Approximation of the spatial multi-agent Markov process parameterized by the control α. We can expect that like in Section 3.3, TDBNAs with larger input neighborhoods and/or target neighborhoods will produce more accurate approximations of a given MFGs forwards dynamics. In turn, the solution α^* of a larger *Bayesian-MFG* will be closer to the solution α^* of the original multi-agent system.

7.1.1 Bayesian Mean Field Type Control

Recall from Section 6.5 that we can relax a Pair-MFG model into a Pair-MFTC problem. Likewise, we can relax a Bayesian-MFG into a Bayesian-MFTC problem. Furthermore, we can expect that the resulting solution α^* obtained from a Bayesian-MFTC problem can give a lower cost with respect to the original multi-agent system compared to the cost obtained from the Pair-MFTC problem. More concretely, consider the following multi-agent control problem:

- N agents $\{x_1, ...x_N\}$, a spatial component G, and an state space S. Let s be the state profile of the N agents
- Each agent follow a central shared policy π that produces a sequence of controls α which controls the time evolution of their state in S over time: $P(x_i(t+1) = s|\mathbf{x}(t)) = F(\alpha, \mathbf{x}(t))$ for some function F
- The goal of a central planner is to minimize a population cost $J(\alpha)$, over a time horizon $[0, T]$

In deterministic systems, the policy π is equivalent to a sequence of controls α. In stochastic systems π is a function of θ that produces a given control α when the population is in a given state θ. In both cases, the central planner solves for an optimal policy π that minimizes the objective function over the controls α generated by the policy π. We consider three approximations of the above multi-agent control problem:

1. Well-mixed MFTC problem: A central planner minimizes a population cost $J_W(\alpha, \theta)$ where θ is simply the population profile of the agents over the state space S, and the time evolution of the state of an agent is expressed as $(x_i(t+1) = s|\mathbf{x}(t)) = F(\alpha, \theta)$.

2. Pair-MFTC problem: A central planner minimizes a population cost $J_P(\alpha, \theta)$ where θ denotes a second order mean field over joint states in the population. The time evolution of the states of an agent and the joint states are expressed using pair approximation.

3. Bayesian-MFTC problem: A central planner minimizes a population cost $J_B(\alpha, O)$ where O denotes the set of mean field quantities in the output query of the TDBNA. The time evolution of the quantities in O is specified using a TDBNA $B(I, T, O)$.

Let $\alpha_W, \alpha_P, \alpha_B$ be the controls generated by solution to each of the above problems. Our hypothesis is that:

$$J(\alpha_B) \leq J(\alpha_P) \leq J(\alpha_W) \tag{7.2}$$

if the TDBNA used for the Bayesian-MFTC problem increases beyond the size used for pair approximation. To make our comparison easier, we will simply use BN-Large (25 node input neighborhood) as the representative TDBNA for the Bayesian-MFTC problem.

Intuitively, this hypothesis suggests MFGs that have more accurate approximations of the forward dynamics/time evolution of a spatial system can be relaxed into MFTCs which can be solved for better policies in the original spatial system. This will allow us to apply the techniques from Chapter 3 and 4 for improving the approximation of the time evolution of spatial systems towards finding better policies for controlling them. Future work on improving approximation efficacy can also be leveraged towards improving the quality of control policies.

7.2 Mathematical Approach

In Chapter 6, we defined the concept of a Pair-MFTC problem, but we did not provide an algorithm to solve it. Since the Pair-MFTC problem relaxes the connection between the forward dynamics and the Bellman equation, we no longer need to use a fixed point approach and can instead solve the Bellman equation directly. However, many existing algorithms struggle with solving a Pair-MFTC problem. After relaxing the Pair-MFG, the resulting Bellman equation has both continuous state and action spaces. Furthermore, the Pair-MFTC problem retains the chaotic behavior from the Pair-MFG where small perturbations near initial timesteps can greatly change later trajectories.

Unlike in the Pair-MFG where we solve for a sequence of α terms, in a Pair-MFTC problem we solve for a controller or a policy π. An easy way to parameterize such a controller is to use a neural network. In this chapter, we will apply an evolutionary method for learning neural policies to our Pair-MFTC and Bayesian-MFTC problems.

7.2.1 Pair Approximation Mean Field Type Control

The Pair-MFG can be relaxed to a pair approximation mean field type control problem (Pair-MFTC) under the assumption that instead of having individually optimizing agents in a MFG, all agents will follow one policy decided by central planner. In [80], this type of relaxation is known as the mean field teams solution. This means that the mean field θ is no longer an argument to the backward equation $J(\theta)$ and we instead solve for directly for an optimal α that induces a mean field θ starting from an initial condition $\theta(0)$:

$$\begin{aligned}\alpha^* = \min_{\alpha} \sum_{t=0}^{T} \sum_{x,cf} F(x_i(t), \theta(t), \alpha(x, t, cf)) \Pr(cf) \\ + \sum_{x,cf} G(x_i(T), \theta(T)) \Pr(cf) \\ \theta(t) = M_{PA}(\alpha, \theta(0), t)\end{aligned} \tag{7.3}$$

where M_{PA} describes the pair approximation equations applied until time t. The problem is transformed into a single agent optimal control problem where the state of the population is described using the global joint state θ. Essentially, the population state is controlled using a local policy $\alpha(x, t, cf)$. To formulate a Bayesian-MFTC, we simply replace the M_{PA} with an M_{Bayes}:

$$\theta(t) = M_{Bayes}(\alpha, \theta(0), t) \tag{7.4}$$

where the M_{Bayes} describes a Truncated Dynamic Bayesian Network Approximation applied until time t.

7.2.2 Finding a Solution

Let $R(\alpha)$ be the cost of a given sequence of controls:

$$\begin{aligned} R(\alpha) = &\sum_{t=0}^{T} \sum_{x,cf} F(x_i(t), \theta(t), \alpha(x, t, cf)) \Pr(cf) \\ &+ \sum_{x,cf} G(x_i(T), \theta(T)) \Pr(cf) \\ &\theta(t) = M_{PA}(\alpha, \theta(0)) \end{aligned} \tag{7.5}$$

A sequence of controls α^* for every $x \in S, t \in T, cf$, is a solution to the PMTFC defined in equation 7.3 if:

$$R(\alpha^*) \leq R(\alpha) \tag{7.6}$$

for every possible sequence of controls α. Note that the above solution assumes that the time evolution of θ is deterministic. However, our goal is to use the solution to a PMTFC model to control policies in a finite stochastic environment. Since the number of nodes in the real world environments are not infinite, there is additional error in the forward equations aside from the approximation error introduced by moment closure. Consequently, the model becomes a stochastic optimal control problem instead where we need to solve for the optimal policy π, where $\alpha(x, t, cf) = \pi(x, cf, \theta)$. As previously noted in Chapter 6, due to the properties of the pair approximation equations, the forward and backward equations are nonlinear and possess many couplings between mean field θ terms. This makes it difficult to apply existing techniques for solving stochastic optimal control problems that typically assume convexity and/or linearity.

To address this, we propose to use an evolution strategies approach which has been previously observed to have success with similarly hard problems.

7.2.3 Evolution Strategies

First developed for continuous parameter optimization in the control of nonlinear systems, evolution strategies are a type of heuristic local search inspired by evolutionary computation. The routine consists of an iterative process applied to a population of "individuals" where each individual represents a set of continuous parameters. At each iteration of the routine, a function is evaluated using each individual's parameters to compute a real fitness value. Then the most individuals with the highest fitness are selected from the current population and perturbed to generate the next population of individuals.

Evolution strategies appear to have some success in application to difficult reinforcement learning environments in which the state and action dynamics are highly nonlinear [84]. Because of these appealing properties, we propose to employ an evolutionary strategy to learn a population of neural networks that approximate the optimal policy σ in the PMFTC model. The algorithm we propose is as follows:

1. Let $\{\gamma_1, ...\gamma_L\}$ be a population of L parameter sets. Each γ_i is a full instantiation of all the weights for a neural network mapping from $(x, cf, \theta) \to [0, 1]$.

2. Evaluate each γ_i as an policy σ using the forward dynamics. To model the stochasticity of the real system, we perturb $\theta(0)$ with ϵ noise. A sequence of α_i is generated from time 0 to T.

3. Evaluate the cost of the sequence of controls α_i and let this be the fitness of the individual

γ_i

4. Choose the top r networks of the the current population to be part of the next population.

5. Generate $N-r-1$ children $\gamma_j = \gamma_i + \mathcal{N}(0, \sigma), i \sim [1, r]$ where a randomly chosen parent γ_i is perturbed with Gaussian noise $\mathcal{N}(0, \sigma)$ to produce a child γ_j.

6. For the last individual in the next generation, place a copy of the individual with the highest fitness through all population iterations (this is a convention known as elitism).

7. Repeat from step 1.

This algorithm is based off a standard $(r/1+\lambda)$-ES scheme [85] (where $\lambda = N-r-1$) with an additional slot for elitism [86] to learn a neural network γ that we can use as a policy σ to generate our control sequence α.

7.3 Network Security

Compared to earlier research in the optimal control of pair approximation models [87], the capability of marginalizing over a set of configurations *cf* in the Pair-MFG/MFTC model allows for the arbitrary specification of network topology and controls over local configurations. More specifically, traditional pair approximation for SIR/SIS models, typically designed for optimal vaccination, are limited to considering global controls (where the vaccination rate is uniform and independent of an agent's location in the network) [87] or pairwise controls (where the vaccination rate can additionally depend on 2nd order terms) [88]. Pair-MFTC allows for the consideration of controls over arbitrary local configurations. For example, in a SIS model with demographics for pandemic spread, a local configuration consisting of a nurse taking care of a large group of high-risk patients should be given high priority for vaccination. With a global control, such a configuration will not be considered at all while a pairwise control may not have sufficient granularity for the targeted application of vaccination measures.

We propose that these advantages will allow for the derivation of more optimal solutions towards many security policy optimization problems that occur on computer networks. An example of such a problem is the prevention of a network contagion.

7.3.1 Network Contagion Scenario

Consider a network of N interconnected servers or computing resources. Nodes in this network may connect to other nodes for various services. It is also reasonable to assume that in a sufficiently large network a single node does not necessarily communicate with every other node in the network suggesting that a well mixed model can have significant prediction errors.

A commonly studied network security threat is the propagation of zero-day attack vectors through this interconnected network. As an example, consider the recent Remote Code Execution (RCE) vulnerability Log4Shell in the Log4j java library, an ubiquitous logging library common in many internet applications. This exploit allows attackers to execute arbitrary code through injecting malicious text into any application the employs the compromised library. In extreme scenarios, attackers may even develop self-propagating worms that can spread automatically as a network contagion by exploiting interconnected service interactions on the network.

We consider the task of optimal resource allocation by a server admin to address the spread of these zero-day attack vectors. Suppose the server admin has some finite resource M that can be distributed to nodes in the network to perform different fixes or interventions to prevent the spread of a given network contagion. For example:

- Placing active agents for intrusion monitoring
- Perform system rollbacks
- Update system code
- Temporarily shutting down or isolating key systems

Each intervention may cost the server admin a different amount of the finite resource M and can be treated as different strategies that the server admin can employ to address the network contagion. In general, if the number of network nodes N is large, solving for optimal resource allocation can be intractable.

7.3.2 Pair-MFTC Model

We propose a game-theoretic formulation of the above task that can be solved approximately as a PMFTC problem. Our initial model will be based on the cyber-security example discussed in [80]. Consider a system of N nodes that are faced with a malware attack:

- Each agent can have two states: $S = \{0, 1\}$, healthy and infected.
- Each agent has control over a parameter $\alpha(cf, \theta)$ which is a choice of whether to get repaired or not during a given iteration. In this model, each agent has the ability to look at the infection status of its local neighborhood *cf* and the overall infection status of the network θ to control a continuous parameter α which determines the probability of a successful repair.

- Each agent accrues a cost k for being infected and a cost $F(\alpha) = \lambda\alpha$ as the cost of a given repair policy alpha.

The goal of central planner is to deduce optimal policy for α that minimizes average cost over the population on a finite time horizon T. The setup of this model is similar to that of the SIS model in existing literature where each network can be in on of two states: susceptible or infected. The repair function in our model can be thought of as similar to vaccination in an epidemic model. Instead of solving for an optimal control over a SIS model, we will solve a Pair-MFTC problem. To define this Pair-MFTC problem, we first need to define the corresponding forward and backward equations.

- *Forward Equation*: The state of each node on the network evolves stochastically through a Markov Process as a function of its local neighborhood and the interventions performed on a given node. In a given iteration, if a node's state is healthy ($x = 0$), we have:

  $$\Pr(x(t+1) = 1 \mid x(t) = 0) = q(1-\alpha)\frac{1}{|cf|}\sum_{j \in cf} \mathbb{I}_{x_j(t)=1} \tag{7.7}$$

  where the parameter q denotes the maximum rate of spread given all neighbors are infected. For example, if a node is healthy, all of its neighbors are infected, and the node has opted to spend 0 resources on repair ($\alpha = 0$), then the node has a probability of q to be infected in the next timestep. Note that this formulation allows for an extension to nonlinear infection dynamics as well:

  $$\Pr(x(t+1) = 1 \mid x(t) = 0) = q(1-\alpha)F_S(cf) \tag{7.8}$$

with a function F_S which can be used to model situations where the probability of infection grows non-linearly in the number of infected neighbors.

- *Backward equation*: The running cost obtained by the central planner can be written as:

$$F(x, \theta, \alpha(x, cf)) = k\theta_x + \sum_{cf} \lambda\alpha(x, cf)Pr(cf) \tag{7.9}$$

  By modeling our problem as a Pair-MFTC, we can also model cost functions that depend locally on the configuration state *cf* of a given node. For example, if a server node requires the services of neighboring nodes and an attacker has compromised the effectiveness of those services, it can be practical to model this interference as an additional cost that susceptible nodes accrue by being next to infected nodes.

7.3.3 A Mean Field Model

To compare with the PMFTC problem, we can design a similar MFTC where α does not depend on the configuration. In this case, we have that the transition probability is simply:

$$\Pr(x(t+1) = 1 \mid x(t) = 0) = q(1-\alpha)\theta_1 \tag{7.10}$$

where the agent's transition probability is modeled against the mean field θ_1 and the central planner minimizes:

$$\min_{\alpha} \sum_{t=0}^{T} k\theta_1 + \lambda\alpha(\theta) \tag{7.11}$$

Because this model simply optimizes against a mean field θ_1 and does not take into account local configurations, we can expect that the policy obtained from solving this model will be worse than the one obtained from the PMFTC. In the next section, we'll verify this behavior by solving MFTC and PMFTC problems for the network contagion scenario.

7.3.4 Empirical Results

To show that policies obtained from PMTFC problems are better than ones obtained from MFTC problem, we numerically solve the aforementioned network contagion scenario where an administrator is tasked with repairing an infected network. First, we construct a stochastic network simulation to be our ground truth environment. At the start of this simulation, a percentage of the population $\theta_1(0)$ starts off as infected. Each node in the network can control their repair rate α. To derive policies for controlling α, we construct MFTC and PMFTC problems that use the same parameters as the stochastic simulation. Solving these control problems will give us approximately optimal policies for α that we can then evaluate in the stochastic simulation to obtain an observed cost.

For our experiments, the model parameters are assumed to be: $q = 0.9, k = 0.3, \lambda = 0.2$. We consider a square toroidal grid for our spatial domain with a Von Neumann neighborhood. For evaluation, we use a stochastic simulation populated by 100×100 agents on the toroidal network.

As a simple baseline and sanity check, we provide the cost obtained from a baseline constant rate policy that assigns the repair rate $\alpha = c$ for some $c \in [0, 1]$. The results of this constant rate policy can be found in Figure 7.2 when starting from a network with low infection rate

$\theta_1(0) = 0.1$ and a highly infected network $\theta_1(0) = 0.8$. As expected, a more aggressive repair policy is needed for a population that starts out more infected.

As mentioned before, solving a 10000 dimensional optimal control problem is intractable, so our goal is to instead efficiently derive an approximately optimal policy. We compare our proposed Evolutionary Strategies approach with several common reinforcement learning algorithms [89] in Figure 7.4. Notably, our approach is superior to all four reinforcement learning approaches tested in convergence rate, final cost obtained, and computational efficiency. For reference, the best competing baseline, DDPG takes 21 minutes to train on 500 episodes compared to less than 5 minutes for our ES approach. Furthermore, due to the nature of the PMTFC problem, many existing reinforcement learning algorithms fail to converge to any reasonable solution and even DDPG converges to an approximate policy that has a higher cost (cost = 4.45) compared to the approximate policy obtained the ES approach (cost = 2.8).

By solving the MFTC problem and the PMFTC problem we can obtain far more optimal solutions than a constant rate policy as shown in Table 7.1. The trajectories generated by these approximately optimal policies can be seen in Figure 7.3 for initial infected rates of $\theta_1(0) = 0.1$ and 0.8. Notably, the policy obtained from the PMFTC problem allow the population to remove the infection much faster than the policy obtained from the MFTC problem. Both models allow for the derivation of a more optimal policy than the constant rate policy and the PMFTC policy obtained has a lower cost than the MFTC policy.

7.3.4.1 Comparing with Bayesian-MFTC

In Table 7.1, a comparison with Bayesian-MFTC (BMFTC) is also provided. What is important to note is that **the order of cost exactly satisfies our hypothesis from Eq. 7.2**. The solution obtained from BMFTC is always better than PMFTC which is always better than MFTC regardless of the value of the initial condition θ_1. There are two main factors for this decrease in cost:

- As mentioned previously, the control α in a PMFTC has additional granularity when compared to a MFTC model. Compared to the MFTC model where the control α is only a function of θ, the control α is a function of the configuration variables *cf* as well. However, this does not explain the gap between PMFTC and BMFTC as both models have the same level of granularity for their control α.

- The second factor is the accuracy of the forward dynamics. Notice that the gap between the three models increases as the initial condition θ_1 increases. This is directly due to the chaotic behavior present in the model's forward dynamics. From a modeling perspective, the goal of any policy in the network contagion scenario is to reduce θ_1 to zero as efficiently as possible. When θ_1 is low, the system is already close to the target state and the problem can be solved within a few timesteps. When θ_1 is high, it takes much more timesteps and thus a longer trajectory to reduce the network's infection to zero. The error arising from the approximation of the model's forward dynamics in the MFTC/PMFTC/BMTFC will be much more noticeable over longer time frames and this causes the solution to the corresponding control problem to drift away from the optimal policy on the simulation.

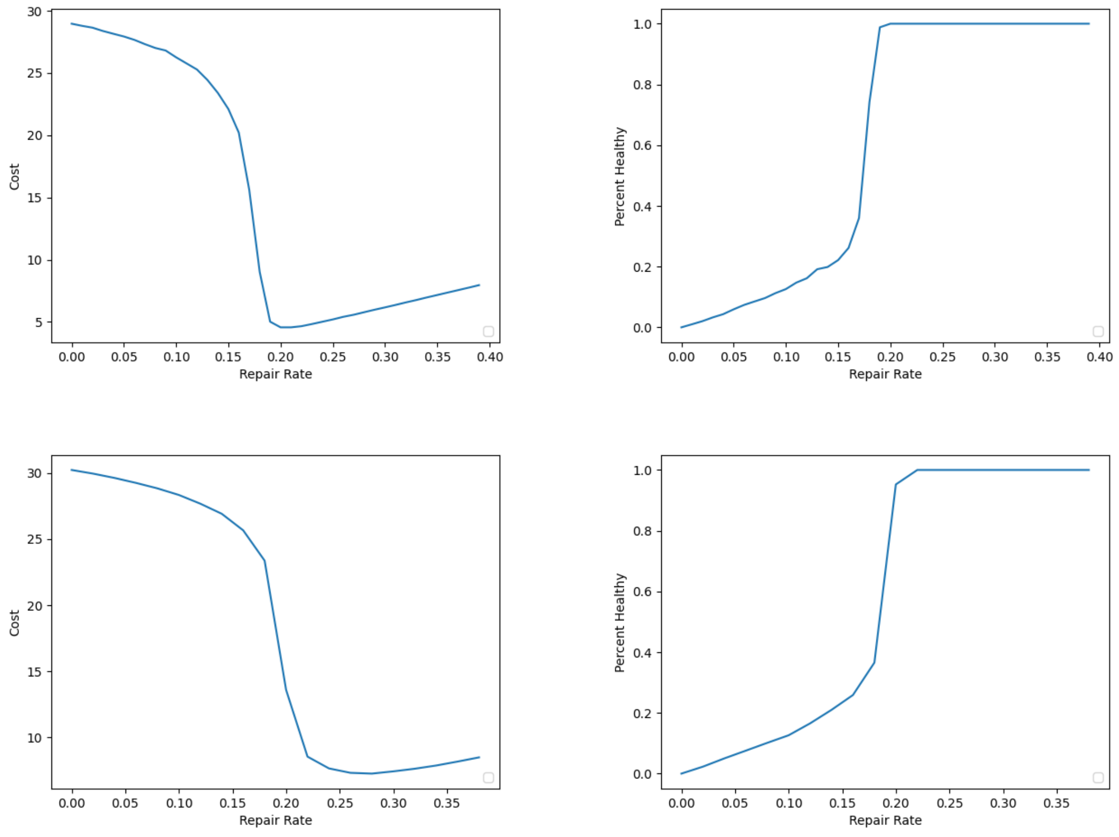


Figure 7.2: Right: number of healthy individuals at time T as a function of constant repair rate, Left: cost obtained as a function of constant repair rate, Top: $\theta_1(0) = 0.1$, Bottom: $\theta_1(0) = 0.8$.

Table 7.1: Average cost obtained on network security stochastic simulation using different policies.

$\theta_1(0)$	Constant Rate	MFTC	PMFTC	BMFTC
0.1	2.247	0.385	0.114	**0.113**
0.5	2.816	1.012	0.643	**0.497**
0.8	3.032	2.139	1.138	**0.797**

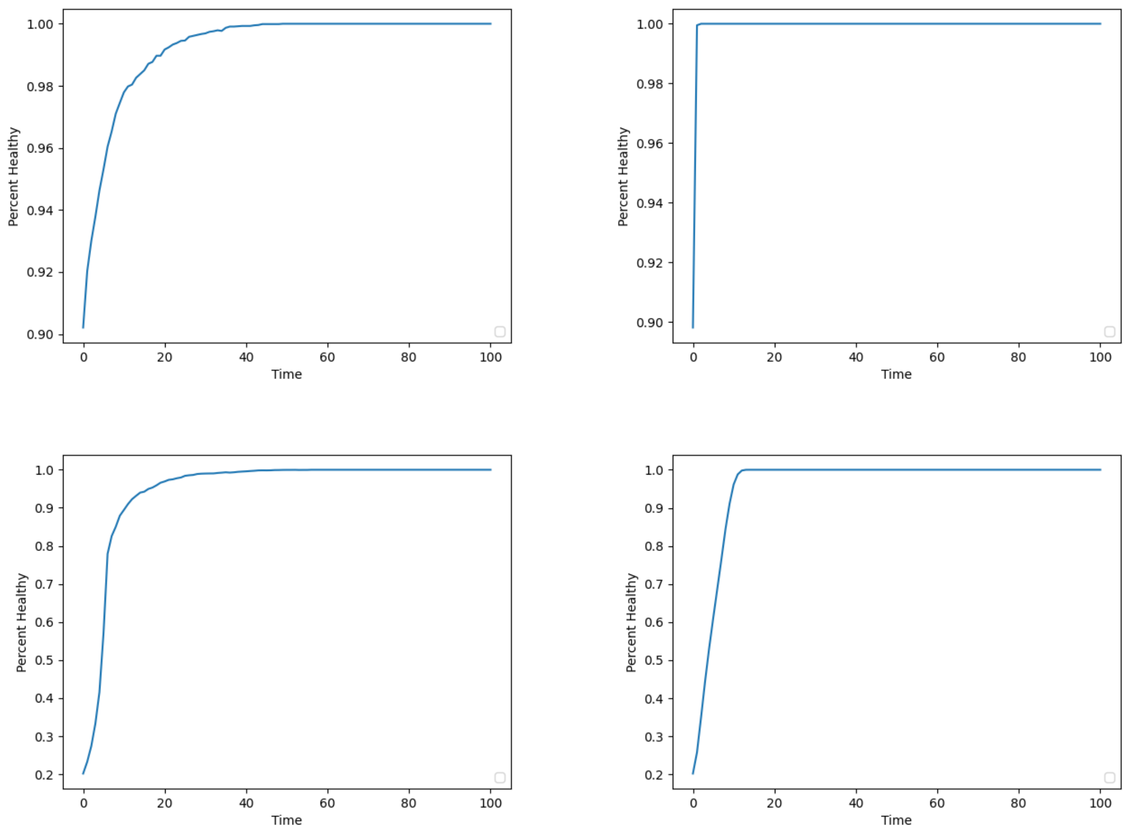


Figure 7.3: Right: number of healthy individuals as a function of time using the MFTC optimal policy, Left: number of healthy individuals as a function of time using the PMFTC optimal policy, Top: $\theta_1(0) = 0.1$, Bottom: $\theta_1(0) = 0.8$

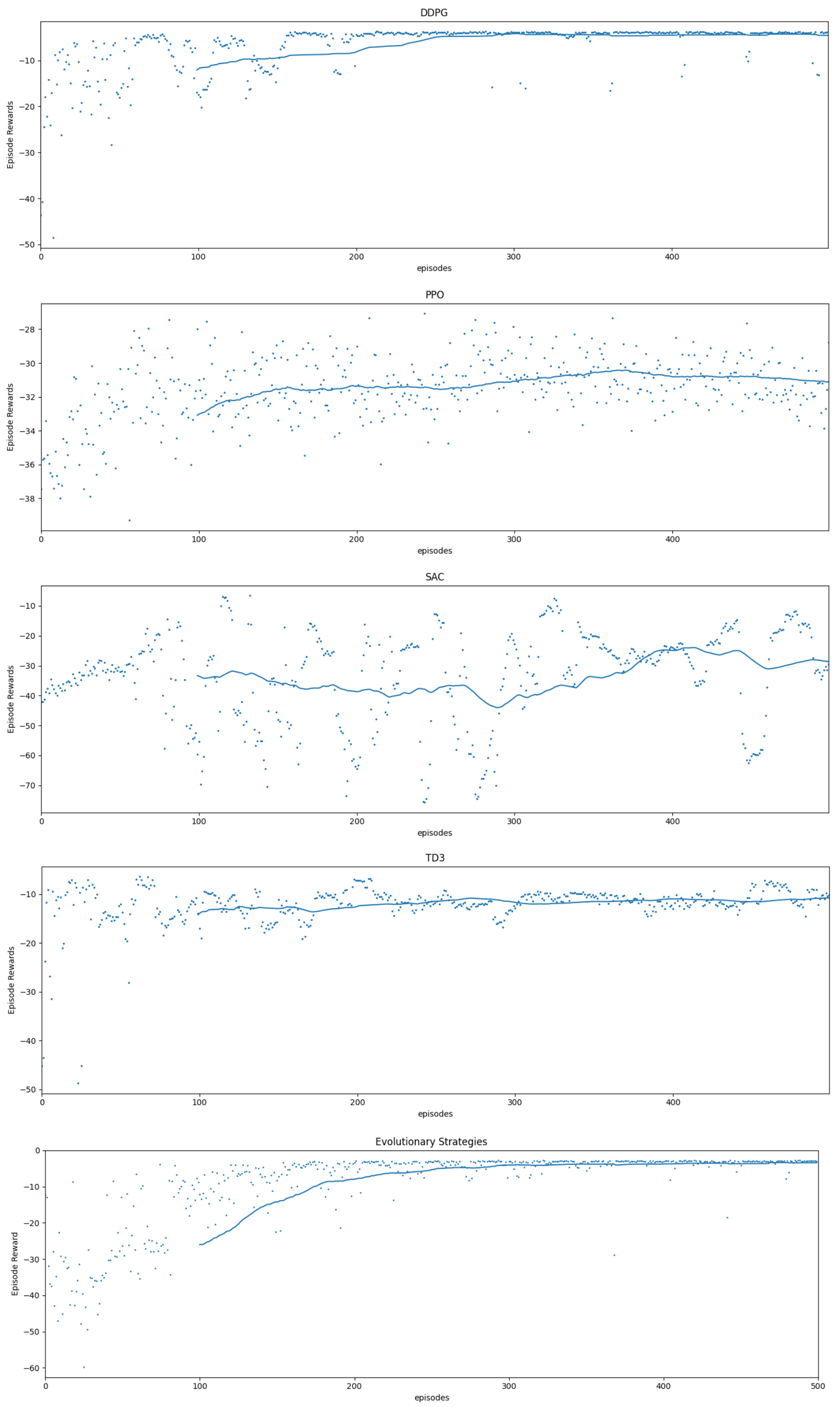
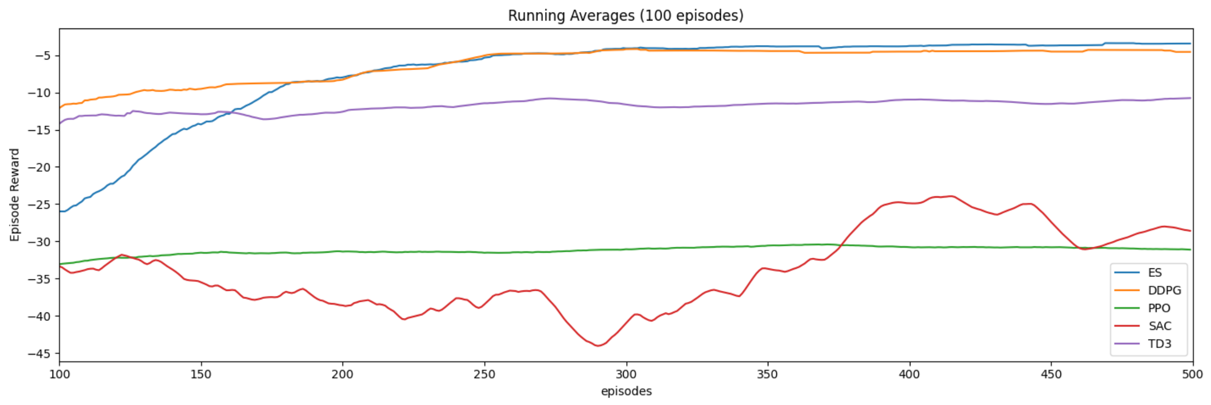


Figure 7.4: Convergence plots for common reinforcement learning (DDPG, PPO, SAC, TD3) algorithms compared to our proposed evolutionary strategies (ES) approach. Solid lines indicate a running average of 100 episodes. Only DDPG and our ES approach converge towards the optimal policy and DDPG reaches a worse final cost at a slower rate (4.55) compared to our ES algorithm (2.8).



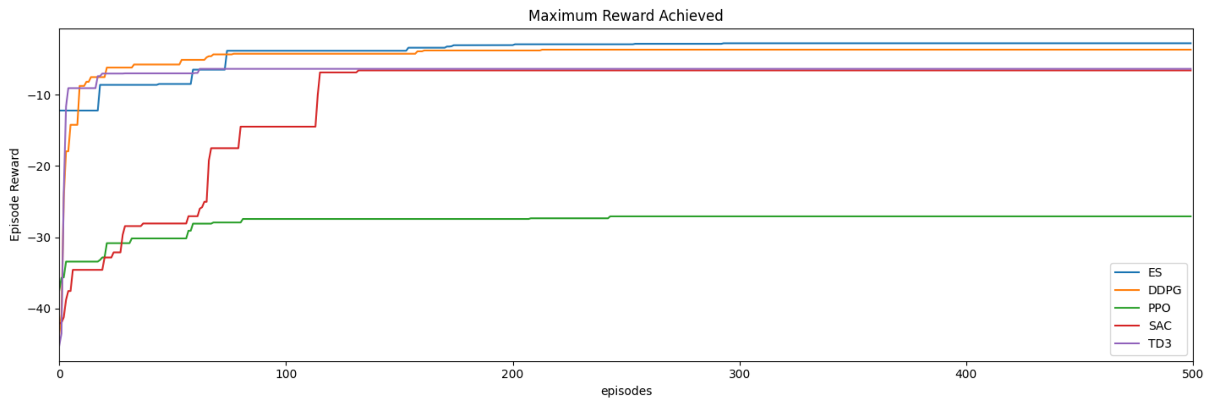


Figure 7.5: Running averages (top) and maximum achieved reward (bot) from Figure 7.4 for common reinforcement learning (DDPG, PPO, SAC, TD3) algorithms compared to our proposed evolutionary strategies (ES) approach.

7.4 Reaction Diffusion Equations

The second domain we will apply our control problems to is a spatial model based on reaction-diffusion systems. A reaction-diffusion system is a set of differential equations that describe the time evolution of a system of different types of particles on a spatial domain. The system includes two independent stochastic processes:

- A reaction process, typically used to model chemical interactions, that describes how certain types of particles may be transformed into other types through local particle interactions.
- A diffusion process that describes how particles spread over the spatial surface.

As a model over types of particles, reaction diffusion systems naturally occur in chemistry but examples can also be found in social models, opinion dynamics, biological and ecological systems (morphogenesis, tumor growth, etc.). Reaction diffusion systems have been modeled as discrete spatial models in previous literature such as [90] where a stochastic cellular automata was devised to study chemical reactions on a platinum catalyst. The goal of this section will be to devise a discrete model based on a set of reaction-diffusion equations.

For this analysis, we will consider the reaction diffusion equation formulated in section 3.1 of [91] which is given by:

$$\partial_t u(t,x) = \Delta F(u(t,x)) + R(u(t,x)) \tag{7.12}$$

This equation is defined over $x \in \Omega$ which is some convex compact spatial domain. Notice that in this equation, there is only one particle type and the variable u denotes its density function over Ω.

Given this reaction diffusion equation, we will construct a comparable discrete stochastic spatial model defined over individual stationary agents that play strategies from a strategy set. After defining this spatial model, we can construct a Pair-MFG model that approximates its behavior. For this example, we will use a 2D lattice grid with periodic boundary conditions to approximate Ω.

Let us define a strategy set $S = \{s_1, s_\varnothing\}$. The density variable u is analagous to the probability that a node on the 2D lattice grid is playing strategy s_1. We have an additional null strategy $s_\varnothing$ that denotes empty spaces in the grid. Using this convention, we can model particle motion in the original reaction-diffusion system as strategy evolution in our analagous system. Let θ be the mean-field variable of our new system. From the application of the pair approximation we have that:

$$\dot{\theta}(t) = \sum_{cf} P(cf)\dot{\theta}(t, cf) \tag{7.13}$$

where $\dot{\theta}(t, cf)$ denotes the rate of change of the mean field θ conditioned on the local configuration. On a 2D grid, a configuration *cf* for a reaction diffusion equation can be defined as a single focal node and four neighbors. We denote this as:

$$cf = \begin{pmatrix} x_{i-1,j} \\ x_{i,j-1} \\ x_{i,j} \\ x_{i+1,j} \\ x_{i,j+1} \end{pmatrix} \tag{7.14}$$

A laplacian operator is simply a pointwise multiplication with the filter:

$$\Delta = \begin{pmatrix} 1 \\ 1 \\ -4 \\ 1 \\ 1 \end{pmatrix} \tag{7.15}$$

We may also want R to depend on the local neighbors of the focal agent, so we write R as a function of *cf*. We can then write our equation as:

$$\dot{\theta}(t) = \sum_{cf} P(cf) \left[\Delta^T F(cf) + R(cf) \right] \tag{7.16}$$

Jump Process Currently our model is still a continuous model over a density variable θ. To make the model discrete, we will model the reaction-diffusion system as a Markov jump process. To do this, we first need to make an assumption on the density variable θ. In contrast to u in Eq. 7.12, we will limit θ to be within $[0, 1]$. First we consider the case where the reaction and the diffusion components of the system are separable.

Separable Reaction/Diffusion Let x denote a focal agent and $N(x)$ its local neighborhood. We consider a situation where reaction and diffusion effects do not occur at a given node during a single iteration at the same time (the effects are separable). For this, denote a coefficient k which determines the probability that a reaction will occur, otherwise diffusion will occur.

- Local Reactions: Let $R(i, j, cf) : S \times S \times cf \to [0, 1]$ be a reaction function that gives the probability of a focal agent transitioning from state i to state j given the local configuration *cf*. We can also parameterize this R function with a control α that can control the reaction rates conditioned on the local neighborhood configurations.

- Diffusion: In this two state model, diffusion consists of two transitions $s_1 \to s_\varnothing$ and $s_\varnothing \to s_1$. In practice, we can model diffusion behavior using an equation similar to the Fermi rule/Glauber dynamics. Because of how the discrete Laplacian is formulated, we can treat $F(cf)$ as a payoff function in an EGT sense:

$$\Delta^T F(cf) = \sum_{y \in N(x)} F(y) - F(x) = 4(\bar{F}(y) - F(x))$$

$$P_{diffusion}(x(t+1) = s_1 | x(t) = s, cf) = \frac{1}{1 + e^{-4(\bar{F}(y) - F(x))}} = \frac{1}{1 + e^{-\Delta^T F(cf)}} \qquad (7.17)$$

The final probability is then: θ is:

$$P(s_\varnothing \to s_1, cf) =$$
$$P(x(t+1) = s_1 | x(t) = s_\varnothing, cf) = \sum_{cf} P(cf) \left((1-k) \frac{1}{(1+e^{-\Delta^T F(cf)})} + k(R(\varnothing, 1, cf)) \right) \tag{7.18}$$

We can apply pair approximation equations to derive the time evolution equations for the joint mean field θ_{ij} as well:

$$\theta_{ij}(t+1) = \theta_{ij}(t) + \Delta\theta_{ij}(t)$$
$$\Delta\theta_{ij}(t) = \sum_k \sum_{cf} \left[\theta_k(t) P(cf) \frac{N(j, cf)}{d} P(k \to i | cf) - \theta_i(t) P(cf) \frac{N(j, cf)}{d} P(i \to k | cf) \right] \tag{7.19}$$

where $N(j, cf)$ is the number of nodes in the configuration *cf* that have the state j. In this model, we can parameterize the function $R(i, j, cf)$ with a control α:

$$R(i, j, cf) = U(\alpha_{i,j}(cf)) \tag{7.20}$$

If U is the identity function and we define some objective function, the resulting model is a standard Pair-MFTC model with added Laplacian dynamics.

Objective Function   For some applications, one might be interested controlling the final distribution of reaction diffusion equation on a lattice (for example to get a particular clustering pattern). In this situation, the control α could be the application of heat/energy over time to the lattice that modifies the reaction rate R through the function U. The reaction rates can be non-linear in the local configuration (e.g. adding heat of a certain amount makes local configurations with 3 or more s_1's accelerate, but does not change the reaction rates of configurations with 2 or less s_1's). This may be useful in modeling certain chemical reactions.

Given a target clustering pattern, first we acquire the θ_{ij} distributions θ' that we want to optimize towards and define the following Pair-MFTC objective function:

$$\arg\min_{\alpha} E\left[\sum_{t}^{T-1} C(x(t),\theta(t),\alpha(t)) + G(x(T),\theta(T))\right]$$
$$\theta = M(\alpha,\theta(0)) \tag{7.21}$$

Using this objective function, we can design cost functions C, G to avoid undesirable local patterns during the trajectory. A simple model can be defined using the following functions:

- $G(x(T,\theta(T)) = L\|\theta(T) - \theta'\|^2$
- $C(x(t),\theta(t),\alpha(t)) = \frac{1}{2}\|\alpha(t)\|^2$

where L is some scaling constant that denotes the importance of reaching the target distribution compared to the energy spent reaching it. Alternatively, one can also formulate the target distribution as a constraint if it is known that the target distribution is feasible.

7.4.1 Empirical Results

Consider a discrete separable reaction-diffusion lattice model as described in the previous section. Suppose through experimentation on the model and adjusting a set of controls we find a desired target distribution or pattern. In a chemistry application, the control might correspond to heat added to the system which affects the rates of each reaction in Eq. 7.20. For simplicity, we will assume that U is the identity function in Eq. 7.20.

For our empirical experiments, we perform a three step process:

1. We first set $R(i, j, cf)$ to some predetermined value. We run the discrete stochastic simulation with $R(i, j, cf)$ for T iterations. At the end of the simulation we record p_{ij} for the grid and set that to be our target distribution.

2. Like in the network security model, we solve a MFTC problem (Pair or Bayesian) for an optimal policy π using evolution strategies. We will define our cost function as:

$$
\begin{aligned}
G(x(T, \theta(T)) &= L\|\theta(T) - \theta'\|^2 \\
C(x(t), \theta(t), \alpha(t)) &= L\|\theta(T) - \theta'\|^2 + K \cdot \frac{1}{2}\|\alpha(t)\|^2
\end{aligned} \tag{7.22}
$$

   Based on how the model is set up, the solution to the MFTC problem will be a policy for $R(i, j, cf)$ that aims to efficiently bring the distribution of the system close the target distribution as fast as possible. The speed vs cost tradeoff can be adjusted using the L and K parameters.

3. After obtaining the policy π from the MFTC problems, we can then evaluate on the simulation to obtain the final cost for comparison. We expect to see the same cost ordering with $J(\alpha_B) \leq J(\alpha_P)$ as in the network security model.

For the purpose of this comparison, we set $k = 0.2$ and initial distribution $\theta_1(0) = 0.001$ with a target distribution of $\theta = [0.2, 0.31, 0.31, 0.18]$. An example of the initial and target distributions can be seen in Fig. 7.6.

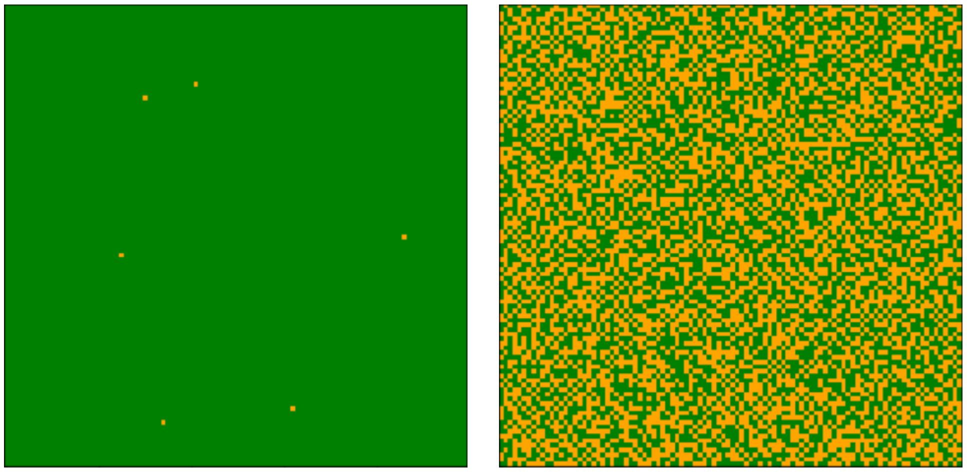


Figure 7.6: Left: an example initial distribution for the reaction diffusion model, Right: an example target distribution for the reaction diffusion model.

Table 7.2: Average cost obtained on reaction diffusion stochastic simulation using different policies.

Original setting	PMFTC	BMFTC
5.505	4.3818	**4.2710**

We run 100 simulations with each policy to evaluate the average cost of the policy. As can be seen in Table 7.2, we again get the result where the cost for the BMFTC policy is better than the PMFTC policy. Both policies obtain a better cost than the original setting for R used to generate the target distribution.

7.5 Summary

In this chapter, we devised an evolutionary method for learning neural policies to solve novel Pair Mean Field-Type Control Problems (PMFTC) and Bayesian Mean Field-Type Control Problems (BMFTC). The non-linearity and non-convexity of PMFTC problems breaks many of the conditions typically assumed by conventional optimal control algorithms.

We designed a network contagion problem and reaction-diffusion problem using the PM-FTC paradigm and demonstrate empirically that our proposed evolutionary algorithm can solve the resulting PMFTC system. Furthermore, the evolutionary method obtains better results than existing reinforcement learning algorithms while also having higher sample and time efficiency.

We showed that that the policies obtained from the PMFTC problem are more optimal compared to other existing mean field control models by evaluating them in stochastic agent based simulations. This extends to policies from Bayesian-MFTC problems which are in turn better than ones obtained from PMFTC problems.

Chapter 8: Conclusion

Even though both evolutionary games and mean field games have already been applied to a wide variety of problems in the existing literature, there are still areas of improvement that can be made to the methods for solving these games. This thesis presents a foray into new methods that first serve as an initial springboard for alternative probabilistic methods for solving the prediction problem in evolutionary games and the control problem in mean field games.

8.1 Summary of Contributions

The main contributions of this thesis are 1) a novel method using Bayesian networks for approximating the dynamics of spatial evolutionary games, 2) a novel algorithm for performing approximate inference on evolutionary game Bayesian networks, 3) a novel mean field game model called Pair-MFG that can be relaxed into a control problem framework, and 4) an extension of this pair-MFG framework with Bayesian networks instead of pair approximation allowing for models that have more accurate forward dynamics. More specifically:

1. We described Truncated Dynamic Bayesian Network Approximations (TDBNA), a method for approximating the forward dynamics of spatial markov processes. In general, any forward equation arising from a Markov process defined on spatial models with distinct strategy spaces (beyond just spatial evolutionary games) can be approximated using

Bayesian networks to desired accuracy by adjusting the size of the resulting TDBNA. We show empirically on several evolutionary games that adjusting the size of a TDBNA can result in better approximations of the game dynamics.

2. We introduce a novel method using surrogate Bayesian networks to perform approximate inference on large TDBNAs. The new method, termed KL-search is a hybrid sampling/ search scheme that provides informative samples for MLE estimation on a surrogate network. We show empirically that this method is superior to existing approximate inference approaches such as Abstraction Sampling in both accuracy and computational efficiency.

3. We have devised an efficient method for reducing the size of summation-based Conditional Probability Tables (CPTs) in Bayesian Networks having *causal independence* (CI). We also have shown how to apply this reduction directly towards the acceleration of Bucket Elimination. We have provided experimental results showing the FFT reduction's advantage for inference on a selection of common sub-networks include sub-networks found in Bayesian networks for spatial evolutionary games. We have developed an extension to *ci-elim-bel* called *ci-elim-FFT* and provided empirical results that demonstrate its scaling advantages.

4. We described Pair-MFG, a mean field game model designed to address the lack of spatial models with distinct strategy and spatial components in Mean Field Games. We devised a novel fixed point algorithm to solve these Pair-MFG problems and demonstrated empirically that it can solve Pair-MFGs modeled after a variety of evolutionary games.

5. We presented a unified framework for the approximation and control of large-scale spatial games. We extend the Pair-MFG model into a Bayesian-MFG model by replacing the

pair approximation equations with a TDBNA. We show how Pair-MFGs/Bayesian-MFGS can be relaxed into Pair-MFTC/Bayesian-MFTC problems which can then be applied to a variety of interesting domains.

6. We demonstrate how Bayesian-MFTC problems can be solved to obtain policies that are better than those obtained from Pair-MFTC problems (which are better than policies from normal MFTC problems) on problems that originate from network security and reaction-diffusion equations.

8.2 Limitations and Future Work

One limitation in this work is the assumption that the strategies of all agents have to be discrete. In domains such as evolutionary game theory, one might want to study models where agents can have a continuous strategy. In the reaction-diffusion model, it would be more accurate to develop a model that allowed for a continuous strategy space. There are a few ways to address this, such as bucketing a continuous strategy into a set of discrete strategies or using continuous or hybrid Bayesian networks. For future work, it would be interesting to explore what methods are effective ways to let TDBNAs handle continuous strategy spaces.

While not the main focus of this dissertation, it was necessary to propose new algorithms and use non-standard solutions such as evolution strategies to solve pair-MFG/Bayesian-MFG models and their MFTC counterparts. In many of these models, evaluating the forward dynamics maybe take a significant amount of computational resources (relative to a simpler approximation like a mean field approximation). An interesting direction would be to see if there are algorithms that are more sample efficient for solving these problems. An alternative approach is to consider techniques that accelerate probabilistic inference. Since the forward direction of a Bayesian-MFG model requires the evaluation of many probabilistic inference problems on the same model with just some changed parameters, it may be possible to compile the network instead of running something like bucket elimination each time.

Bibliography

[1] Richard Durrett and Simon Levin. The importance of being discrete (and spatial). *Theoretical population biology*, 46(3):363–394, 1994.

[2] Feng Fu, Martin A Nowak, and Christoph Hauert. Invasion and expansion of cooperators in lattice populations: Prisoner's dilemma vs. snowdrift games. *Journal of theoretical biology*, 266(3):358–366, 2010.

[3] Hobart Peyton Young. *Individual strategy and social structure: An evolutionary theory of institutions*. Princeton University Press, 2001.

[4] William H Sandholm. Evolutionary game theory. In *Encyclopedia of Complexity and Systems Science*, pages 3176–3205. Springer, 2009.

[5] Soham De, Dana S Nau, and Michele J Gelfand. Understanding norm change: An evolutionary game-theoretic approach. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 1433–1441, 2017.

[6] Robert L Axtell. Non-cooperative dynamics of multi-agent teams. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3*, pages 1082–1089, 2002.

[7] Karl Tuyls and Simon Parsons. What evolutionary game theory tells us about multiagent learning. *Artificial Intelligence*, 171(7):406–416, 2007.

[8] Marc Ponsen, Karl Tuyls, Michael Kaisers, and Jan Ramon. An evolutionary game-theoretic analysis of poker strategies. *Entertainment Computing*, 1(1):39–45, 2009.

[9] Steve Phelps, Peter McBurney, and Simon Parsons. Evolutionary mechanism design: a review. *Autonomous agents and multi-agent systems*, 21(2):237–264, 2010.

[10] Javier Morales, Michael Wooldridge, Juan A Rodríguez-Aguilar, and Maite López-Sánchez. Off-line synthesis of evolutionarily stable normative systems. *Autonomous agents and multi-agent systems*, 32(5):635–671, 2018.

[11] François Mériaux, Vineeth Varma, and Samson Lasaulce. Mean field energy games in wireless networks. In *2012 Conference Record of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, pages 671–675, 2012.

[12] Boualem Djehiche, Alain Tcheukam, and Hamidou Tembine. A mean-field game of evacuation in multilevel building. *IEEE Transactions on Automatic Control*, 62(10):5154–5169, 2017.

[13] Wonjun Lee, Siting Liu, Wuchen Li, and Stanley Osher. Mean field control problems for vaccine distribution, 2021.

[14] Karthik Elamvazhuthi and Spring Berman. Mean-field models in swarm robotics: A survey. *Bioinspiration & Biomimetics*, 15(1):015001, 2019.

[15] Benjamin Herd, Simon Miles, Peter McBurney, and Michael Luck. Verification and validation of agent-based simulations using approximate model checking. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 53–70. Springer, 2013.

[16] Gianluca Manzo and Toby Matthews. Potentialities and limitations of agent-based simulations. *Revue française de sociologie*, 55(4):653–688, 2014.

[17] György Szabó and Gábor Fáth. Evolutionary games on graphs. *Physics Reports*, 446(4-6):97–216, Jul 2007.

[18] Rasmus Ibsen-Jensen, Krishnendu Chatterjee, and Martin A Nowak. Computational complexity of ecological and evolutionary spatial dynamics. *Proceedings of the National Academy of Sciences*, 112(51):15636–15641, 2015.

[19] Josef Hofbauer and Karl Sigmund. Evolutionary game dynamics. *Bulletin of the American mathematical society*, 40(4):479–519, 2003.

[20] Carlos P Roca, José A Cuesta, and Angel Sánchez. Evolutionary game theory: Temporal and spatial effects beyond replicator dynamics. *Physics of life reviews*, 6(4):208–249, 2009.

[21] Soham De, Michele J Gelfand, Dana Nau, and Patrick Roos. The inevitability of ethnocentrism revisited: Ethnocentrism diminishes as mobility increases. *Scientific reports*, 5(1):1–7, 2015.

[22] Antonio F Peralta, Adrián Carro, Maxi San Miguel, and Raúl Toral. Stochastic pair approximation treatment of the noisy voter model. *New Journal of Physics*, 20(10):103045, 2018.

[23] Arne Traulsen and Christoph Hauert. Stochastic evolutionary game dynamics. *Reviews of nonlinear dynamics and complexity*, 2:25–61, 2009.

[24] P. Shakarian, P. Roos, and A. Johnson. A review of evolutionary graph theory with applications to game theory. *Biosystems*, 2012.

[25] Li-Min Ying, Jie Zhou, Ming Tang, Shu-Guang Guan, and Yong Zou. Mean-field approximations of fixation time distributions of evolutionary game dynamics on graphs. *Frontiers of Physics*, 13(1):1–10, 2018.

[26] Jorge Peña, Bin Wu, Jordi Arranz, and Arne Traulsen. Evolutionary games of multiplayer cooperation on graphs. *PLoS computational biology*, 12(8):e1005059, 2016.

[27] Satoru Morita. Extended pair approximation of evolutionary game on complex networks. *Progress of theoretical physics*, 119(1):29–38, 2008.

[28] György Szabó and Csaba Tőke. Evolutionary prisoner's dilemma game on a square lattice. *Physical Review E*, 58(1):69, 1998.

[29] Christoph Hauert and György Szabó. Game theory and physics. *American Journal of Physics*, 73(5):405–414, 2005.

[30] Christopher E Overton, Mark Broom, Christoforos Hadjichrysanthou, and Kieran J Sharkey. Methods for approximating stochastic evolutionary dynamics on graphs. *Journal of Theoretical Biology*, 468:45–59, 2019.

[31] Christoforos Hadjichrysanthou, Mark Broom, and Istvan Z Kiss. Approximating evolutionary dynamics on networks using a neighbourhood configuration model. *Journal of theoretical biology*, 312:13–21, 2012.

[32] Christian Kuehn. Moment closure—a brief review. In *Control of self-organizing nonlinear systems*, pages 253–271. Springer, 2016.

[33] Hirotsugu Matsuda, Naofumi Ogita, Akira Sasaki, and Kazunori Satō. Statistical mechanics of population: the lattice lotka-volterra model. *Progress of theoretical Physics*, 88(6):1035–1049, 1992.

[34] Christoph Hauert and Michael Doebeli. Spatial structure often inhibits the evolution of cooperation in the snowdrift game. *Nature*, 428(6983):643–646, 2004.

[35] Xiaofeng Wang, Matjaž Perc, Yongkui Liu, Xiaojie Chen, and Long Wang. Beyond pairwise strategy updating in the prisoner's dilemma game. *Scientific Reports*, 2(1):1–8, 2012.

[36] Emanuele Pugliese and Claudio Castellano. Heterogeneous pair approximation for voter models on networks. *EPL (Europhysics Letters)*, 88(5):58004, 2009.

[37] Jerome Benoit, Ana Nunes, and M Telo da Gama. Pair approximation models for disease spread. *The European Physical Journal B-Condensed Matter and Complex Systems*, 50(1):177–181, 2006.

[38] Xiao-Feng Luo, Xiaoguang Zhang, Gui-Quan Sun, and Zhen Jin. Epidemical dynamics of sis pair approximation models on regular and random networks. *Physica A: Statistical Mechanics and its Applications*, 410:144–153, 2014.

[39] Kazuki Kuga, Masaki Tanaka, and Jun Tanimoto. Pair approximation model for the vaccination game: predicting the dynamic process of epidemic spread and individual actions against contagion. *Proceedings of the Royal Society A*, 477(2246):20200769, 2021.

[40] Adnan Darwiche. *Modeling and reasoning with Bayesian networks*. Cambridge university press, 2009.

[41] Rina Dechter. Reasoning with probabilistic and deterministic graphical models: Exact algorithms. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 7(3):1–191, 2013.

[42] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[43] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann, 1988.

[44] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799, 2008.

[45] Rina Dechter and Robert Mateescu. And/or search spaces for graphical models. *Artificial intelligence*, 171(2-3):73–106, 2007.

[46] Qiang Liu and Alexander T Ihler. Bounding the partition function using holder's inequality. In *ICML*, 2011.

[47] Filjor Broka, Rina Dechter, Alexander Ihler, and Kalev Kask. Abstraction sampling in graphical models. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[48] Kalev Kask, Bobak Pezeshki, Filjor Broka, Alexander Ihler, and Rina Dechter. Scaling up and/or abstraction sampling. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence,{IJCAI} 2020*, 2020.

[49] Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[50] Rina Dechter and Irina Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM (JACM)*, 50(2):107–153, 2003.

[51] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50:5–43, 2003.

[52] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in bayesian networks. *arXiv preprint arXiv:1302.3562*, 2013.

[53] Vincent Hsiao, Xinyue Pan, Dana Nau, and Rina Dechter. Approximating spatial evolutionary games using bayesian networks. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1533–1535, 2021.

[54] Russell Cooper, Douglas V DeJong, Robert Forsythe, and Thomas W Ross. Communication in the battle of the sexes game: some experimental results. *The RAND Journal of Economics*, pages 568–587, 1989.

[55] Carlos Pérez Roca. *Cooperation in evolutionary game theory: effects of time and structure*. PhD thesis, Universidad Carlos III de Madrid, 2009.

[56] Deiter Kraft. A software package for sequential quadratic programming. Technical report, DLR German Aerospace Center – Institute for Flight Mechanics, 1988.

[57] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[58] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[59] Nastaran Bassamzadeh and Roger Ghanem. Multiscale stochastic prediction of electricity demand in smart grids using bayesian networks. *Applied energy*, 193:369–380, 2017.

[60] Yu Jiang, Hehua Zhang, Xiaoyu Song, Xun Jiao, William NN Hung, Ming Gu, and Jiaguang Sun. Bayesian-network-based reliability analysis of plc systems. *IEEE transactions on industrial electronics*, 60(11):5325–5336, 2012.

[61] Baoping Cai, Yonghong Liu, Qian Fan, Yunwei Zhang, Zengkai Liu, Shilin Yu, and Renjie Ji. Multi-source information fusion based fault diagnosis of ground-source heat pump using bayesian network. *Applied energy*, 114:1–9, 2014.

[62] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[63] Otto Bibartiu, Frank Dürr, Kurt Rothermel, Beate Ottenwälder, and Andreas Grau. Towards scalable k-out-of-n models for assessing the reliability of large-scale function-as-a-service systems with bayesian networks. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 514–516. IEEE, 2019.

[64] Luigi Portinale and Andrea Bobbio. Bayesian networks for dependability analysis: an application to digital control reliability. *arXiv preprint arXiv:1301.6734*, 2013.

[65] Irina Rish and Rina Dechter. On the impact of causal independence. Technical report, AAAI SS-98-03. Dept. Information and Computer Science, UCI, 1998.

[66] Nevin Lianwen Zhang and David Poole. Exploiting causal independence in bayesian network inference. *Journal of Artificial Intelligence Research*, 5:301–328, 1996.

[67] David Heckerman and John S Breese. A new look at causal independence. In *Uncertainty Proceedings 1994*, pages 286–292. Elsevier, 1994.

[68] Ronald L Graham, Donald E Knuth, Oren Patashnik, and Stanley Liu. Concrete mathematics: a foundation for computer science. *Computers in Physics*, 3(5):106–107, 1989.

[69] John G Proakis. *Digital signal processing: principles algorithms and applications*. Pearson Education India, 2001.

[70] Irina Rish. *Efficient reasoning in graphical models. PhD thesis*. University of California, Irvine, 1999.

[71] Petr Savicky and Jiří Vomlel. Exploiting tensor rank-one decomposition in probabilistic inference. *Kybernetika*, 43(5):747–764, 2007.

[72] Martin Plajner and Jiří Vomlel. Bayesian networks for the test score prediction: A case study on a math graduation exam. In *European Conference on Symbolic and Quantitative Approaches with Uncertainty*, pages 255–267. Springer, 2021.

[73] Joseph Beyene. *Uses of the fast Fourier transform (FFT) in exact statistical inference*. PhD thesis, National Library of Canada= Bibliothèque nationale du Canada, 2001.

[74] Nima Taghipour, Daan Fierens, Jesse Davis, and Hendrik Blockeel. Lifted variable elimination: Decoupling the operators from the constraint language. *Journal of Artificial Intelligence Research*, 47:393–439, 2013.

[75] Minyi Huang, Roland P Malhamé, Peter E Caines, et al. Large population stochastic dynamic games: closed-loop mckean-vlasov systems and the nash certainty equivalence principle. *Communications in Information & Systems*, 6(3):221–252, 2006.

[76] Jean-Michel Lasry and Pierre-Louis Lions. Mean field games. *Japanese journal of mathematics*, 2(1):229–260, 2007.

[77] Alain Bensoussan, Joseph Sung, Phillip Yam, and Siu Pang Yung. Linear-quadratic mean field games, 2014.

[78] Lars Ruthotto, Stanley J Osher, Wuchen Li, Levon Nurbekyan, and Samy Wu Fung. A machine learning framework for solving high-dimensional mean field game and mean field control problems. *Proceedings of the National Academy of Sciences*, 117(17):9183–9193, 2020.

[79] Peter E Caines and Minyi Huang. Graphon mean field games and the gmfg equations: ε-nash equilibria. In *2019 IEEE 58th conference on decision and control (CDC)*, pages 286–292. IEEE, 2019.

[80] Deepanshu Vasal, Rajesh Mishra, and Sriram Vishwanath. Sequential decomposition of graphon mean field games. In *2021 American Control Conference (ACC)*, pages 730–736. IEEE, 2021.

[81] Alain Bensoussan, KCJ Sung, Sheung Chi Phillip Yam, and Siu-Pang Yung. Linear-quadratic mean field games. *Journal of Optimization Theory and Applications*, 169(2):496–529, 2016.

[82] Bruce Hajek and Michael Livesay. On non-unique solutions in mean field games, 2019.

[83] Kai Cui and Heinz Koeppl. Approximately solving mean field games via entropy-regularized deep reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1909–1917. PMLR, 2021.

[84] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[85] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies–a comprehensive introduction. *Natural computing*, 1(1):3–52, 2002.

[86] Shumeet Baluja and Rich Caruana. Removing the genetics from the standard genetic algorithm. In *Machine Learning Proceedings 1995*, pages 38–46. Elsevier, 1995.

[87] Li Liu, Xiaofeng Luo, and Lili Chang. Vaccination strategies of an sir pair approximation model with demographics on complex networks. *Chaos, Solitons & Fractals*, 104:282–290, 2017.

[88] Notice Ringa and Chris T Bauch. Impacts of constrained culling and vaccination on control of foot and mouth disease in near-endemic settings: A pair approximation model. *Epidemics*, 9:18–30, 2014.

[89] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[90] Olga Leonidovna Bandman and Anastasiya Evgen'evna Kireeva. Stochastic cellular automata simulation of oscillations and autowaves in reaction-diffusion systems. *Numerical Analysis and Applications*, 8:208–222, 2015.

[91] Wuchen Li, Wonjun Lee, and Stanley Osher. Computational mean-field information dynamics associated with reaction diffusion equations. *arXiv preprint arXiv:2107.11501*, 2021.