Reinforcement Learning for Decentralized Stable Matching

Preprint · May 2020

3 authors, including:

Kshitija Taywade
University of Kentucky

Judy Goldsmith
University of Kentucky

REINFORCEMENT LEARNING FOR DECENTRALIZED STABLE MATCHING

A PREPRINT

Kshitija Taywade
Department of Computer Science
University of Kentucky
kshitija.taywade@uky.edu

Judy Goldsmith
Department of Computer Science
University of Kentucky
goldsmit@cs.uky.edu

Brent Harrison
Department of Computer Science
University of Kentucky
harrison@cs.uky.edu

ABSTRACT

When it comes to finding a match/partner in the real world, it is usually an independent and autonomous task performed by people/entities. For a person, a match can be several things such as a romantic partner, business partner, school, roommate, etc. Our purpose in this paper is to train autonomous agents to find suitable matches for themselves using reinforcement learning. We consider the decentralized two-sided stable matching problem, where an agent is allowed to have at most one partner at a time from the opposite set. Each agent receives some utility for being in a match with a member of the opposite set. We formulate the problem spatially as a grid world environment and having autonomous agents acting independently makes our environment very uncertain and dynamic. We run experiments with various instances of both complete and incomplete weighted preference lists for agents. Agents learn their policies separately, using separate training modules. Our goal is to train agents to find partners such that the outcome is a stable matching if one exists and also a matching with set-equality, meaning outcome is approximately equally likable by agents from both the sets.

Keywords Decentralized matching · Stable matching · Multi-agent system · Reinforcement Learning

1 Introduction

Many real-world two-sided matching markets are decentralized in the sense that no centralized agency exists which knows everyone's preferences in order to perform the matching between the two sides of the market. Having such a central entity that knows everyone's preferences is often not feasible. Furthermore, privacy issues can arise if preferences are confidential. While matching services are now-a-days available at one's expense, people may prefer that some matchings are formed privately, without public sharing of preferences. Even in the case of dating apps or marriage brokers, those central agencies cannot perfectly decide someone's actual preference over prospective matches. Rather than directly pairing up people, they only give suggestions and people then, in turn, choose partners after interacting with the suggested prospects.

If each person wants to find a match for themselves and if only they know their preferences, then it results in a system where people find matches in a decentralized manner by searching the world around them for prospective partners. Therefore we feel that it is appropriate to model the matching market as a decentralized, multi-agent system. A multi-agent system is a group of autonomous, interacting entities sharing a common environment. When agents are acting autonomously then, from the point of view of a single agent, the environment is dynamic and uncertain; not knowing the preferences of other agents and only the noisy version of own preferences adds on to the unpredictability. Moreover, the curse of dimensionality arises with an increasing number of agents. First finding a good enough match in this scenario, and then remaining with it, is quite a complex task. Therefore we try to solve this problem using reinforcement learning (RL).

In many matching markets, navigating (physically or virtually) for locating and approaching a potential match is crucial task. Several decentralized matching markets, such as worker-employer markets and buyer-seller trading markets, consist of locations at which potential matching agents may meet. Such locations can be present physically or online on the internet. This additional problem of locating potential matches that many approaches to solving matching problems, be they centralized or decentralized, overlook. But, as this is an important aspect of the matching process, to incorporate it, we model the problem by using a grid world environment in which agents must learn to navigate to other agents in order to form matches.

In the classic stable matching problem, members of one set have strictly ordered preferences over all members of the opposite set. In real-life situations, there are multiple factors involved in deciding a preferred match and having a score for each match is more expressive. Therefore, we have considered the case of stable matching problems with weighted preferences which are also discussed in [1, 2, 3]. The weighted preference of one agent for another agent in the opposite set is used as the utility value (or *reward*) obtained by that agent if both are in a match. While an instance of the conventional *stable matching problem (SM)* includes members of one set having strictly ordered preferences over all members of the opposite set, there are important extensions to this problem which are also widely studied in the literature, *stable matchings with incomplete lists (SMI)* where agents are allowed to declare one or more unacceptable partners, *stable matching with ties (SMT)* where agents have same preference for more than one agents and third one is *(*SMIT) which is a mixture of both previous extensions. In addition to classical *SM* problem, we also consider *SMI* problem. Moreover, both symmetric and asymmetric preference types are considered.

The two-sided matching theory generally assumes that agents know their true preferences over potential partners before engaging in a match. However, in many matching markets, knowledge acquisition is important: in labor markets, employers interview workers; in matching markets, men and women date; and in real estate markets, buyers attend open houses. Moreover, when two autonomous entities pair up in a partnership, in practice due to uncertain nature of those entities (for example, uncertain nature of human behavior in relationships and employee's inconsistent happiness at workplace depending on the work he/she is doing on a particular day), utilities obtained from the partnership can differ from time to time. We have incorporated uncertainty into the rewards that agents get while in a match. We have a noisy reward signal, noise being calculated from specific normal distribution, which in turn significantly increases the complexity of the problem.

Our goal is to train agents to find partners such that the whole system reaches stability if the stable solution exists. But as we have dynamic decentralized system with agents with incomplete knowledge of the world, it is hard to guarantee stability for every instance. Our additional aim is to have set-equality in the final outcome, meaning it should be approx. (we say approximately because obtainable utility values differ from agent to agent) equally good for both the sets, reflecting that every agent did well in managing to find a good match for it-self. Also, this is expected to happen as we are training each agent separately to deal with the environment.

We compare our results with several baselines such as Hoepman's algorithm [4] and decentralized algorithm by Comola and Fafchamps [5], both of which are decentralized approaches. Additionally, we compare our results with the bidirectional local search algorithm [6] and brute force, both of which are centralized approaches giving optimal results in terms of stability subject to set-equality. Our problem differs from the problems considered in these baselines because of the spatial (grid) formulation, which adds great complexity to the problem. Note that as no one has tried a spatial version of stable matching problem before, it was hard to find perfect comparison, so we have tried to extend the implementation of decentralized algorithm by Comola and Fafchamps on a grid, we do it in two ways as explained in details in experiments section. Our multi-agent RL method was able to successfully find stable matchings with set-equality for almost all the instances in our experiments.

2 Related Work

Reinforcement Learning has not been used for the decentralized stable matching problem but researchers have applied both RL and Deep RL mechanisms to solve coalition formation problems. Note that matching problems are a special case of coalition formation problems. In [7], the authors propose a framework for training agents to negotiate and form teams using deep reinforcement learning, they have also used a grid (spatial) environment in their experiments. Bayesian reinforcement learning has also been used for coalition formation problems [8, 9]. Note that unlike most of the work in multi-agent RL approaches for coalition formation and task allocation, our agents cannot communicate with each other at all. Agents can only perceive other agents from the opposite set which are present at the same cell location and also which ones of them are interested in forming a match.

Researchers have studied several decentralized matching markets, and have proposed frameworks for modeling them and techniques for solving them, and have also analyzed different factors that affect the results [10, 11, 12, 13, 14, 15, 16, 17, 18]. Most of these works focus on job markets. Echenique and Yariv in their study of one-to-one matching markets proposed a decentralized approach for which stable outcomes are prevalent, but unlike our formulation of a problem, agents have complete information on everyone's preferences. In our work, we apply reinforcement learning to decentralized one-to-one matching market (modeled spatially), while none of the above mentioned work have formulated this problem spatially and they have used different methods than reinforcement learning.

There are several distributed algorithms that find stable or almost stable matchings [19, 20, 21, 22, 23, 24, 25]. Distributed algorithms for weighted matching mainly include algorithms which are distributed in terms of agents acting on their own either synchronously or asynchronously [4, 26, 27], and algorithms which are distributed using parallel programming to find approximate maximum matching [28, 29, 30, 31, 32, 33, 34, 35, 33]. The latter case is essentially centralized. Most of the distributed algorithms are focused on networking and job scheduling problems. Although there are some decentralized methods where agents act on their own, the crucial assumption in these works is that agents already know their preferences/weights for all the members of opposite set and can directly contact other agents to propose matches. In our model, agents must explore the world to acquire such knowledge. Most of the decentralized methods mentioned here allow agents to make matching offers/proposals and also accept/reject such offers. In our approach, an agent chooses other agent from the opposite set that they want to be paired with, by selecting a relevant action. State space represents whoever agents from the opposite set are present at the same cell location and which of them have shown interest of pairing up (by selecting relevant action, as mentioned earlier). Agents get matched only when both of them have selected an action for pairing with each other. While this may seem similar to making, accepting or rejecting offers, it is not exactly the same.

3 Preliminaries

Definition 1 *An instance of the classical* stable matching problem (SM) *involves n agents in each of sets S_1 and S_2. Each agent has a strict preference order over the members of the other set. The aim is to find a matching M, i.e., a one-to-one correspondence between the two sets. Given matching M, an agent $x \in S_1$ and an agent $y \in S_2$, the pair (x, y) is a blocking pair for M if x prefers y and y prefers x to their partner in M. A matching is said to be stable if it does not contain blocking pairs.*

Definition 2 *The* stable matching problem with incomplete preference lists (SMI) *may involve incomplete preference lists for those involved. In this case, the members of the opposite set who are unacceptable to an agent simply do not appear in their preference list.*

In our problem formulation, an agent has negative weights for unacceptable agents of the other set.

Definition 3 *Some people involved in stable matching problem may be indifferent between two or more members of opposite set which gives another extension of stable matching problem called as* stable matching problem with ties (SMT).

Definition 4 *A stable matching problem including both incomplete lists and ties is another extension, we will call it* SMIT.

Definition 5 *A* stable matching problem with weighted preferences (SMW) *is a variant of classical stable matching problem where every agent gives a score to every member of the other set representing how much it prefers that member or the utility value it may get by being in the match with that member. Given agents $x_i \in S_1$ and $y_j \in S_2$, we write U_{ij} for the utility x_i receives at each time step (of episode in our RL approach) for being in a match with y_j, and U_{ji} analogously.*

Agents wish to maximize their utility over time.

Definition 6 *Let M be a match over (S_1, S_2), and $TU(S_i)$ be the summed utility of all agents in S_i. The* equality cost *$|TU(S_1) - TU(S_2)|$ [6] measures the* set-equality *of M. Lower equality cost means more equality between two sets. We use the equality cost to measure the quality of our results. By referring to the outcome as set-equal/egalitarian, we mean that it has minimum possible equality cost.*

A reinforcement learning agent learns by interacting with its environment. The agent perceives the state of the environment and takes an action, which causes the environment to transition into a new state at each time step. The agent receives a reward reflecting the quality of each transition. The agent's goal is to maximize cumulative reward over time [36]. In our system, although agents learn independently and separately, their actions affect the environment and in turn affect the learning process of other agents as well. As agents receive different intrinsic rewards, we modeled our problem as Markov game defined as follows and our approach can be considered as multi-agent reinforcement learning.

Multi-Agent Reinforcement Learning

Stochastic/Markov games [37] are used to model multi-agent decentralized control where reward function is separate for each agent as each agent works only towards maximizing their own total reward.

A Markov game with n players specifies how the state of an environment changes as the result of the joint actions of n players. The game has a finite set of states S. The observation function $O : S \times 1, \ldots, n \rightarrow R_d$ specifies d-dimensional view of the state space for each player. We write $O_i = \{o_i | s \in S, o_i = O(s, i)\}$ to denote the observation space of player i. From each state, players take actions from the set $\{A_1, \ldots, A_n\}$ (one per player). The state changes as a result of the joint action $\langle a_1, ..., a_n \rangle \in \langle A_1, ..., A_n \rangle$, according to a stochastic transition function $T : S \times A_1 \times ... \times A_n \rightarrow \Delta(S)$, where $\Delta(S)$ denotes the set of probability distributions over S. Each player receives an individual reward defined as $r_i : S \times A_1 \times ... \times A_n \rightarrow R$ for player i. In our multi-agent reinforcement learning approach, each agent independently learns, through its own experience, a behavior policy π_i: $O_i \rightarrow \Delta(A_i)$ (denoted $\pi(a_i|o_i)$) based on its observation o_i and reward r_i. Each agent's goal is to find policy π_i which maximizes a long term discounted reward [36].

4 Method

We want to formulate the environment which takes into consideration the aspects of matching in real world scenarios and we seek to develop a decentralized approach for solving it.

As mentioned previously, we believe that there is a spatial element to finding suitable partners in that agents must first find each other before they can potentially match to each other. We have addressed this by formulating our problem on a grid where agents start exploring the environment from random cell locations. This approximates the spatial reality of meeting with individuals (at bars, parties, or dances, say) or organizations (at, e.g., job fairs). As such, we model this problem as a multi-agent reinforcement learning problem in a grid based environment.

Learning of each agent is independent of other agent's learning. We believe that finding a partner for oneself is an independent task where agents don't necessarily need to compete or even co-operate. Agents only need to learn to find their suitable partner, which, in this framework, they do by using SARSA [38, 36], a reinforcement learning algorithm.

Agents have prior knowledge of the size of the grid and the total number of agents from the opposite set, but they completely lack the knowledge of identities of other agents and also unaware of the weighted preferences/utility values of other agents as well as their own preferences for other agents. They cannot perceive any part of the environment other than the cell they are located at. While exploring, if an agent stumbles upon another agent from the opposite set in the same cell and both the agents show an interest in matching with each other then they get matched. Even if an agent is not interested in another agent encountered at the same location, it knows whether that agent is interested or not, provided that both the agents remain in the same cell. If both the agents show interest in matching with each other at the same time step, then they are matched.

Matched agents receive a noisy reward as a utility value at each time step as long as they are matched. Noise is added to the actual value from normal distribution (details are in experiments section). Since agents can only view their current grid cell, agents can only match with one another if they are in the same location. In our model, individuals may choose to be in a match until someone better comes along or may choose to leave a match in order to explore further and look for someone better. Thus, a time step in which all agents are paired is not necessarily *stable*, because agents may break off a partnership to explore, or another, more appealing agent may be willing to partner with them. This latter situation corresponds to a blocking pair in the classic *stable matching problem (SM)* [39]. Observation and action spaces and reward function are described in detail in the experiments section.

As it is multi-agent system, other agents' actions certainly have an effect on the rewards of individual agents, although each agent's rewards are completely intrinsic. Note that it is not necessary to learn other agents' policies in this setting, in order to optimize one's expected utility.

Through this approach, we want each agent to independently learn to navigate the environment in order to find a viable partner. If the final outcome is stable, we can conclude that no two agents agree that their position can be improved by their alliance only. For the *SM* problem, stable matching always exists and for the *SMI, SMT and SMIT* problem, at least a weakly stable matching exists. In the weak stability, a blocking pair is defined as (m, w) such that $M(m) \neq w, w \succ_m M(m)$, and $m \succ_w M(w)$ [40]. In *SMI and SMIT* instances, agents can end up without a partner as incomplete lists make some potential matches unacceptable.

Also, we don't want certain agents to have dominant policy over others as we also want to have set-equality in the end result (with minimum equality cost as explained in preliminaries).

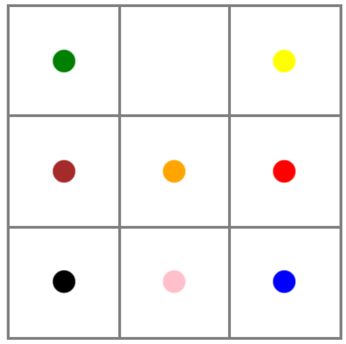


Figure 1: Sample grid world simulation; 8 agents in 3×3 grid.

5 Experiments

In experiments, we have simulated conventional stable marriage *(SM)* and stable marriage problem with incomplete lists *(SMI)*. Our simulation environment is a grid world where we randomly place agents from both the sets and allow them to seek suitable partners. Agents train independently, and our goal is to find a stable matching with set equality.

We compare our results with both centralized and decentralized algorithms. Comparison with centralized algorithms shows us the quality of our results with respect to optimal results. We compare with brute force approach and bidirectional local search [6]. Not only brute force approach, but also birectional local search gives optimal results in terms of stability with respect to set-equality. As our decentralized approach is fundamentally different from centralized approaches, comparison with other decentralized approaches is more suitable. We compare with Hoepman's algorithm [4], and the decentralized algorithm by Comola and Fafchamps [5]. Although these are decentralized approaches, they are not set in the spatial environment, unlike ours, therefore, we also implement grid world version of the decentralized algorithm by Comola and Fafchamps.

5.1 Grid world Environment

Our method assumes that agents are placed in a grid world and actions available can be categorized into two types: move one space in the grid environment and stay by trying to form a match with agent from opposite set.

The grid world environment as shown in Figure 1 is a 3×3 grid with 8 agents randomly placed on it. The agents are evenly divided between S_1 and S_2. They are *myopic,* in the sense that they can only observe their current cell location. They can perceive the location of their current cell on the grid, which individuals of the opposite set are in the current cell, and which one of those are interested in matching with them. Further details of the observation space are given in the next section.

5.2 Environment specifications

5.2.1 Observation space:

An observation of an agent i, O_i, consist of its location on the grid, who (from the opposite set) is present in the cell with it, and of those agents, which ones are showing an interest in matching with that agent. The size of O_i is $R \times C + 2 \cdot N$, where R and C are the number of rows and columns in the grid and N is the total number of members of the opposite set. An observation at time step t, $O_i[t]$ consists of three one-hot vectors. The first one represents the agent's position on the grid and the second and third ones represent information about agents of the opposite set in the current cell location. The size of the first hot vector is equal to the total number of grid cells. The sizes of the second and third vectors are the total number of members of the opposite set; each value is associated with a single member. The second vector represents which members of the opposite set are present in the current cell. The third vector represents which of those members have shown interest in matching with the observing agent. Thus, an agent initially starts out knowing only the dimensions of the grid, and the total number of members of the opposite set.

5.2.2 Action space:

Each agent's possible actions are moves in the grid and expressing interest in a match with an agent from the opposite set. The action space is of size $N + 4$ where N is the number of members of the opposite set and each member has an action associated with it for showing an interest in matching with that member. There are 4 additional actions for navigating the grid by moving up, down, left and right. There is no specific action for staying in the same grid cell because whenever an agent is interested in forming a match with another agent, it automatically stays in the same cell. When two collocated agents show an interest in forming a match with one another, then the match is considered to be formed. Note that once a match is formed, the agents must continue to express interest in each other at each time step in order to maintain the match. If at some point, one ceases to express interest, the match is dissolved.

5.2.3 Transition Function:

Our environment is deterministic.

5.2.4 Reward Function:

We have a noisy reward function described as follows:

- 0 reward for navigation actions.
- The immediate reward received by an agent i for a matching of i and j is $R_{ij} = U_{ij} \cdot Z$, where Z is the noise, sampled from a normal distribution with mean $M = 1$ and standard deviation $SD = 0.1$ and U_{ij} is the actual utility value that agent i can get from a match with agent j.

5.3 Simulation Instances

As mentioned before, we are considering a stable matching problem with weighted preferences ***(SMW)***, where the weights represent the utility values U_{ij}. One constraint of our model is that, for each agent, the weights/utilities determine a strict preference order over members of the opposite set. In the stable matching problem literature, some variations of the stable matching problem are considered. In preliminaries, we described one of those variations, which is stable matching with incomplete lists. We have implemented two types of instances. The first is based on conventional stable matching *(SM)* problem with strictly ordered weighted preferences and the second one is related to incomplete lists *(SMI)*. In our formulation, for the instances of stable matching problem with incomplete lists, negative weights are used to represent unacceptability.

Moreover, as per the literature, there are two types of preferences among agents: Symmetric and Asymmetric. We have considered symmetric preferences as the ones in which any two agents from opposite sets have the same weighted preference for each other. In the asymmetric case, this may not be the case. In either case, agents do not have any prior knowledge of these weights/obtainable utility values. Note that there are no ties present in any of the instances.

For *SM* instances, the utility values U_{ij} are generated from a uniform random distribution in the range $(1, 10)$, and those for *SMI* instances are generated from uniform random distribution in the ranges $(-5, 10)$ and $(-10, 10)$. By considering both of the ranges, we want to see the effect of the difference in acceptance rates (the number of acceptable matches for an agent) on the convergence of the whole system as well as on the success rate of finding a match by individual agents.

5.4 Parameter Settings

Each agent independently learns a policy using SARSA [38, 36] with a multi-layer perceptron as a function approximator to learn a set of Q-values. Each network consists of 2 hidden layers with hidden units 50 and 25 respectively. We train using the Adam optimizer [41] with learning rate $1e-4$ to minimize TD-control loss. Each agent is trained with 60000 episodes each consisting of 300 steps and discount-factor $\gamma = 0.9$. We have combined SARSA with experience replay for better results. Our experience replay buffer size is 5000 and we use a batch size of 200 in each experiment.

When there are multiple suitable partners available in the environment for an agent, a proper exploration strategy is needed to find the best among them. Therefore we have exploration rate with non-linear decay, such that it is very high in the beginning but decays sharply (with a minimum exploration rate, $\epsilon = 0.05$). For a certain episode, we can calculate it using a formula: $e^{-(0.3+ep \cdot 0.00008)}$, where ep is the episode number. Here, 0.3 and 0.00008 are the values hand-picked by us on the basis of performance relative to the other values we tried.

5.5 Baselines for Comparison

We want agents to settle with matches such that the whole system reaches stability if there exists a stable solution for that problem instance. Moreover, we want our final outcome to have *set-equality* i.e., the matching is equally good for both the sets.

Utility Range	Preference Type	Hoepman's			Greedy			Multi-agent RL		
		Stable Outcomes	Equality cost	Total utility	Stable Outcomes	Equality cost	Total utility	Stable Outcomes	Equality cost	Total utility
(1,10)	Sym	Few	0	60.1	Few	0	61.5	All	0	69.1
	Asym	Few	5.9	52.3	Few	3.9	51.5	Almost all	3.7	56.9
(-5,10)	Sym	Few	0	47.8	Few	0	50.4	All	0	50.4
	Asym	Few	5.4	31.0	Few	3.5	25.4	All	3.3	31.4
(-10,10)	Sym	Few	0	37.2	Few	0	42.8	All	0	42.2
	Asym	Few	4.9	27.5	Few	5.2	24.8	All	3.9	27.4

Table 1: Comparison of our Multi-agent RL approach with Hoepman's algorithm and Greedy version of decentralized algorithm by [5]. We show how all three perform for six unique scenarios consisting of three different utility ranges and two types of preferences; 10 instances of each scenario. We compare quality of solutions in terms of stability and total utility as well as equality cost averaged over 10 instances.

We have a very dynamic and uncertain environment and autonomous agents that care only about maximizing their own rewards. In such a scenario, it is hard for agents to pair up with their ideal/best partner. It is very likely that agents may settle for a partner which has weight slightly lower than ideal partner, it can occur in cases where they find former partner way earlier than later while exploring the environment. Also in the case of asymmetric preferences, it is unlikely that ideal partner equally prefers an agent, making it harder for agents to maximize their own rewards.

This formulation of the problem is unique in the computational matching literature. We have modeled this as a spatial problem in which agents have no prior knowledge of their preferences over others or other agents' preferences for them. In addition, our approach is to solve this in a decentralized manner. Because of this, it is difficult to select reasonable baselines for comparisons. We have settled on what we feel are informative and reasonable set of baseline algorithms for comparison. They will be discussed in more detail below.

5.5.1 Brute Force Approach

This approach is designed in such a way that the result is always stable with subject to set-equality (with optimal set-equality cost).

5.5.2 Bidirectional Local Search Algorithm [6]

This is a centralized local search algorithm for finding a stable matching with set-equality. It simultaneously searches forward from the S_1-optimal stable matching and backward from the S_2-optimal stable matching until the search frontiers meet. We have chosen this comparison to check the quality of our results with those produced by a centralized algorithm. We have observed that for instances with a small number of agents, as in the instances discussed here, this algorithm produces a stable matching with minimum equality cost i.e. maximum set-equality.

5.5.3 Hoepman's Algorithm [4]

This is a variant of the sequential greedy algorithm [42] which computes a weighted matching at most a factor of two away from the maximum. It is a distributed algorithm, in which agents asynchronously message each other. An agent greedily sends a request to their current favorite from all connected agents (in a bipartite graph) of the opposite set. If their favorite one prefers them as well over all other available choices, they form a match. Otherwise, the connection is dropped and the agent moves on to the next best choice.

This algorithm performs significantly better than a random algorithm we implemented as a baseline, so we use Hoepman's algorithm instead. Note that it is a non-spatial algorithm where agents already have knowledge of every other agent present in the system. This gives them a significant advantage over the agents in our system, both because the agents know whom they prefer and because they have instantaneous contact, rather than having to wander around in a grid world.

5.5.4 Decentralized Algorithm by Comola and Fafchamps [5]

This algorithm is designed to investigate matching in a decentralized market with deferred acceptance. Deferred acceptance means an agent can be paired with several other partners in the process of reaching their final match. While Comola and Fafchamps focused on many-to-many matching, the method can be easily adapted for one-to-one matching. This algorithm includes a sequence of rounds in which agents take turns in making proposals to other agents, who can accept or reject them. Agents are picked randomly for a chance to propose a match and each agent is guaranteed to get a chance in every round. The system stabilizes after a certain number of rounds. We have observed that the outcome is usually egalitarian as agents are chosen randomly in every round and no particular set is favored over the other.

This algorithm is not designed for the problems formulated spatially. Along with the implementation of the original algorithm, we have implemented two versions of this decentralized algorithm for the grid world. In both the versions, agents are randomly placed on the grid and they wander randomly in order to find other agents and form matches. In these settings, agents are allowed to have prior knowledge about their utilities for the agents of the opposite set similar to the original non-grid version.

1. Greedy: In this version, we deployed agents at random places on the grid and allowed them to wander until everyone gets settled. For successful implementation on the grid, we could not let agents explore for more than one round. Also, it is not necessary to visit every other agent of the opposite set before settling into the match. The only case where agent seeks for a new partner is when the current partner leaves them for better choice. For adapting to a grid, we needed to make major changes, which resulted in a very different approach than the original one.

2. Frozen Grid World: To keep the rounds part from Comola and Fafchamps algorithm for the grid version as well, we implemented this version in such a way that agents freeze when it's not their turn in the round to form a new match. And we can add an additional constraint that each agent should visit every other agent from the opposite set before making a proposal, to know about the best available option. With these constraints, this version becomes similar to the original method. But we can see that this is not how real-world works and everyone in the system usually works simultaneously.

Note that we do not compare our results with the Gale-Shapley algorithm because it finds stable matches that are optimal only for one side. We also do not compare our results with the optimal total utility outcomes for the whole system, because agents do not work towards the common goal of optimizing rewards for the whole system, rather agents are only self-interested and work to optimize their own rewards.

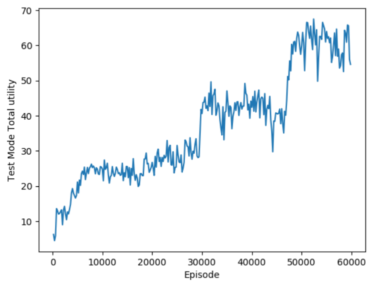
6 Results and Discussion

As mentioned earlier, our experiments contain six unique scenarios, the combination of both symmetric and asymmetric types of preferences with three different utility ranges $(1, 10)$, $(-5, 10)$ and (-10,10). We ran 10 different instances each for all six unique scenarios. Note that equality cost for measuring set-equality can only be considered for evaluating asymmetric preference cases because it is always zero for symmetric preference cases as we can see in table 1. As any two agents from opposite sets always have equal utility value for pairing up with each other, equality cost will always be zero. For symmetric instances, the quality of final outcomes can be evaluated by the total utility value (addition of the utility values obtained by all the agents in the system), it will be optimal because if agent x_i is the best choice for agent y_j then it is also vice-versa.

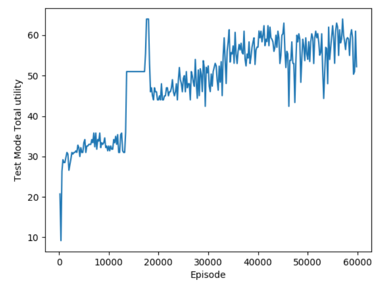
For all the instances (exception discussed later), we were successful in training agents to find partners such that the end result is a stable solution with set equality. This means that in terms of final outcomes, agents had no better choice available than their final partners. Equality costs were also comparatively low, meaning agents were able to find viable partners for themselves, resulting in final solutions favourable for everyone, which was our main goal.

Our final results exactly match those by Bidirectional Local Search [6], the centralized algorithm which produces stable matchings with set-equality. For asymmetric instances, matchings are with set-equality having minimum equality cost. As equality cost is always zero for the results of symmetric preferences cases, we measured total utility values for the whole system which is a suitable measure here and found that our results produce optimal total utility for instances with symmetric preferences. As our instances are very small, only containing 8 agents, bidirectional local search always gives the same results each time. Our results are also very similar to the decentralized algorithm by Comola and Fafchamps, but there is some randomness in this algorithm as agents are chosen randomly in rounds, which results in slightly different outcomes for different runs of same instance, all outcomes are stable but equality cost varies a little bit for different runs.

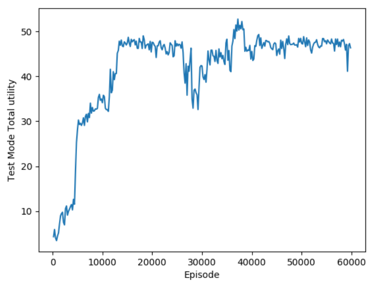
We show convergence graphs for all the unique cases. Note that although we ran our experiments for 10 different instances of each unique case, we are not able to show convergence graph for every instance due to space issues. But we have chosen graphs such that they show average behavior of those type of instances. All the simulation instances show convergence in about 60000 episodes. We think that the very uncertain and dynamic nature of the environment causes non-smoothness in convergence graphs.



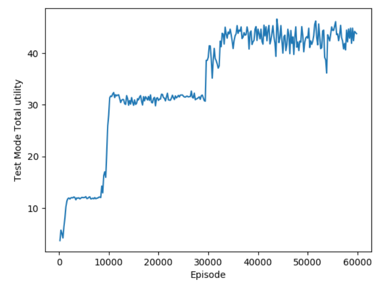
(a) Convergence graph for an instance of symmetric preference case with (1,10) utility range



(b) Convergence graph for an instance of asymmetric preference case with (1,10) utility range



(a) Convergence graph for an instance of symmetric preference case with (-5,10) utility range



(b) Convergence graph for an instance of asymmetric preference case with (-5,10) utility range

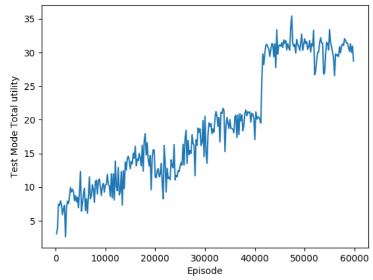
We have observed that, all agents in *SM* instances (with utility range $(1, 10)$) are able to find partners but when it comes to *SMI* instances (with utility range $(-5, 10)$ and $(-10, 10)$) then for some agents, being single is the only option left in the end, because pairing up with unacceptable agents results in negative rewards.

In table 1, we compare our method with Hoepman's algorithm and greedy grid world implementation of the decentralized algorithm by Comola and Fafchamps. While running these algorithms, we do not allow pairs to form if any one of the agents from the pair gets a non-positive utility. This makes them easier to compare with our approach as agents in our approach avoid getting negative rewards and never pair up with unacceptable agents. A frozen grid world implementation of the decentralized algorithm by Comola and Fafchamps produces the same results as the original non-grid version but by taking much more time because of the grid exploration part. Therefore, we did not include its results in the table.

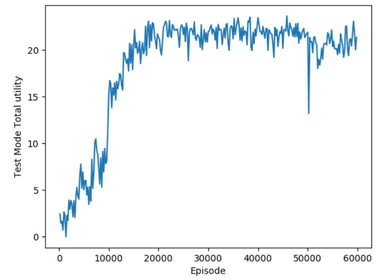
Hoepman's algorithm focuses on finding an optimal solution but does not consider stability at all and we can see from table 1 that we don't always get a stable end result. By observing the results from 10 different instances of each unique case, we can say that the end result is stable for less than the half of the instances. Note that in every run, we can get slightly different results for the same instance due to randomness inherent in this algorithm. Most importantly, we can see that our method performed better than Hoepman's algorithm even in obtaining overall total utility. Also, we get equality costs slightly higher than our method which means that our method produces outcomes with better set-equality than this approach.

For the grid world greedy implementation of the decentralized algorithm by Comola and Fafchamps, as agents do not propose to each other in rounds but rather simultaneously in a greedy way, it is not possible to always get a stable final outcome as we can also see in 1. Also, as agents explore the grid randomly, every time we get slightly different results while running an instance. Important thing to note here is that obtained total utilities are almost equal to the Hoepman's algorithm and equality costs are slightly more than our approach. Nonetheless, our approach still performs better in all three areas; stability, set-equality, and total utility.

For all the three comparisons, we get the equality cost for symmetric cases as zero. It is easy to deduce as in the symmetric case, both agents in the pair get the same utility value, resulting in the equal total utilities obtained by two sets.



(a) Convergence graph for an instance of symmetric preference case with (-10,10) utility range



(b) Convergence graph for an instance of asymmetric preference case with (-10,10) utility range

When it comes to our method, table 1 provides important takeaways. The final outcome is always stable for all the instances of all unique cases, expect one type of scenario, asymmetric instances for utility range of $(1, 10)$. For this case, 2 out of 10 instances resulted in unstable matching, but it is because the difference in obtained utility from the current partner and the partner in the blocking pair was very small. Some of the agents had a problem in making a choice between the two or more prospective matches if the obtainable utility values varied by a very small difference. Also, remember that we have a noisy reward signal which makes such differentiation even harder. It did not happen for asymmetric case instances of utility ranges $(-5, 10)$ and $(-10, 10)$, we think that as some of the agents from opposite set are unacceptable for an agent, it is easier for them to learn making choices between prospective matches and also as these utility ranges are more wide than $(1, 10)$, it is less likely to have prospective matches with weighted preferences near to each other, making it easier to differentiate between them. We can also see that the equality costs are reasonably low, meaning no particular set of agents has an advantage over the other and final result is fair for everyone. Our approach is able to obtain comparatively good results in terms of total utility values. For symmetric cases, we can say that the outcomes are optimal in terms of overall total utility. But asymmetric cases are tricky as agents don't necessarily have same utility values for being in a match with each other, therefore, final total utility outcomes are not necessarily optimal for the whole system.

7 Conclusion and Future Work

We have formulated a stable matching problem spatially as a grid world with independent and autonomous agents having no prior knowledge of obtainable utility values, acting in a decentralized manner. By modeling in this way, we wanted to focus on making agents learn to find and form matches and also stay in them if suitable. We have been successful in achieving our aim of training individual agents using multi-agent reinforcement learning such that everyone finds a suitable partner, by suitable we mean that whole system comes to stable outcome so we know that no two agents agree that their position can be improved by their alliance only and moreover there is a set-equality in the final outcome with minimum equality cost, so we know that the agents in both the sets are almost equally happy in the end (note that for symmetric instances we measure it with optimality in total utility value of all agents). We plan to extend our experiments on larger instances and for the instances with ties in the preferences of agents.

References

[1] Dan Gusfield. Three fast algorithms for four problems in stable marriage. *SIAM Journal on Computing*, 16(1): 111–128, 1987.

[2] Robert W Irving, Paul Leather, and Dan Gusfield. An efficient algorithm for the "optimal" stable marriage. *Journal of the ACM (JACM)*, 34(3):532–543, 1987.

[3] Maria Silvia Pini, Francesca Rossi, Kristen Brent Venable, and Toby Walsh. Stability and optimality in matching problems with weighted preferences. In *Agents and Artificial Intelligence*, pages 319–333. Springer Berlin Heidelberg, 2013.

[4] Jaap-Henk Hoepman. Simple distributed weighted matchings. *arXiv preprint cs/0410047*, 2004.

[5] Margherita Comola and Marcel Fafchamps. An experimental study on decentralized networked markets. *Journal of Economic Behavior & Organization*, 145:567–591, 2018.

[6] H H Viet, L H Trang, S Lee, and T Chung. A bidirectional local search for the stable marriage problem. In *2016 International Conference on Advanced Computing and Applications (ACOMP)*, pages 18–24. ieeexplore.ieee.org, November 2016.

[7] Yoram Bachrach, Richard Everett, Edward Hughes, Angeliki Lazaridou, Joel Leibo, Marc Lanctot, Mike Johanson, Wojtek Czarnecki, and Thore Graepel. Negotiating team formation using deep reinforcement learning. 2018.

[8] Georgios Chalkiadakis and Craig Boutilier. Bayesian reinforcement learning for coalition formation under uncertainty. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '04, pages 1090–1097, Washington, DC, USA, 2004. IEEE Computer Society.

[9] Tim Matthews, Sarvapali D Ramchurn, and Georgios Chalkiadakis. Competing with humans at fantasy football: Team formation in large partially-observable domains. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*. aaai.org, 2012.

[10] Effrosyni Diamantoudi, Eiichi Miyagawa, and Licun Xue. Decentralized matching: The role of commitment. *Games and Economic Behavior*, 92:1–17, 2015.

[11] Guillaume Haeringer and Myrna Wooders. Decentralized job matching. *International Journal of Game Theory*, 40(1):1–28, 2011.

[12] Muriel Niederle and Leeat Yariv. Decentralized matching with aligned preferences. Technical report, National Bureau of Economic Research, 2009.

[13] Federico Echenique and Leeat Yariv. An experimental study of decentralized matching. 2012.

[14] Muriel Niederle and Leeat Yariv. Matching through decentralized markets. *Discussion Paper, Stanford University*, 2007.

[15] Joana Pais, Agnes Pintér, and Robert F Veszteg. Decentralized matching markets: a laboratory experiment. 2012.

[16] Joana Pais, Ágnes Pintér, and Róbert F Veszteg. Decentralized matching markets with (out) frictions: a laboratory experiment. *Experimental Economics*, pages 1–28, 2017.

[17] Jack Ochs. The coordination problem in decentralized markets: An experiment. *The Quarterly Journal of Economics*, 105(2):545–559, 1990.

[18] Mark Satterthwaite and Artyom Shneyerov. Dynamic matching, two-sided incomplete information, and participation costs: Existence and convergence to perfect competition. *Econometrica*, 75(1):155–200, 2007.

[19] Ismel Brito and Pedro Meseguer. Distributed stable matching problems. In *Principles and Practice of Constraint Programming - CP 2005*, pages 152–166. Springer Berlin Heidelberg, 2005.

[20] Ismel Brito and Pedro Meseguer. Distributed stable matching problems with ties and incomplete lists. In *Principles and Practice of Constraint Programming - CP 2006*, pages 675–679. Springer Berlin Heidelberg, 2006.

[21] Nir Amira, Ran Giladi, and Zvi Lotker. Distributed weighted stable marriage problem. In *Structural Information and Communication Complexity*, pages 29–40. Springer Berlin Heidelberg, 2010.

[22] Patrik Floréen, Petteri Kaski, Valentin Polishchuk, and Jukka Suomela. Almost stable matchings by truncating the Gale-Shapley algorithm. *Algorithmica*, 58(1):102–118, 2010.

[23] Rafail Ostrovsky and Will Rosenbaum. Fast distributed almost stable matchings. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC '15, pages 101–108, New York, NY, USA, 2015. ACM.

[24] Fredrik Manne and Morten Mjelde. A self-stabilizing weighted matching algorithm. In *Stabilization, Safety, and Security of Distributed Systems*, pages 383–393. Springer Berlin Heidelberg, 2007.

[25] Subhendu Chattopadhyay, Lisa Higham, and Karen Seyffarth. Dynamic and self-stabilizing distributed matching. In *Proceedings of the Twenty-first Annual Symposium on Principles of Distributed Computing*, PODC '02, pages 290–297, New York, NY, USA, 2002. ACM.

[26] Mirjam Wattenhofer and Roger Wattenhofer. Distributed weighted matching. In *Distributed Computing*, pages 335–348. Springer Berlin Heidelberg, 2004.

[27] Arif Khan, Alex Pothen, Md Mostofa Ali Patwary, Nadathur Rajagopalan Satish, Narayanan Sundaram, Fredrik Manne, Mahantesh Halappanavar, and Pradeep Dubey. Efficient approximation algorithms for weighted b-matching. *SIAM Journal on Scientific Computing*, 38(5):S593–S619, 2016.

[28] Zvi Lotker, Boaz Patt-Shamir, and Adi Rosén. Distributed approximate matching. *SIAM Journal on Computing*, 39(2):445–460, 2009.

[29] Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. Improved distributed approximate matching. In *Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures*, SPAA '08, pages 129–136, New York, NY, USA, 2008. ACM.

[30] Stefan Hougardy and Doratha E Vinkemeier. Approximating weighted matchings in parallel. *Information Processing Letters*, 99(3):119–123, 2006.

[31] Manuela Fischer and Mohsen Ghaffari. Deterministic distributed matching: Simpler, faster, better. *arXiv preprint arXiv:1703.00900*, 2017.

[32] Yishay Mansour and Shai Vardi. A local computation approximation scheme to maximum matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 260–273. Springer, 2013.

[33] Fredrik Manne and Rob H Bisseling. A parallel approximation algorithm for the weighted maximum matching problem. In *Parallel Processing and Applied Mathematics*, pages 708–717. Springer Berlin Heidelberg, 2008.

[34] Yishay Mansour, Boaz Patt-Shamir, and Shai Vardi. Constant-time local computation algorithms. *Theory of Computing Systems*, 62(2):249–267, 2018.

[35] Ryuhei Uehara and Zhi-Zhong Chen. Parallel approximation algorithms for maximum weighted matching in general graphs. In *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics*, pages 84–98. Springer Berlin Heidelberg, 2000.

[36] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[37] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

[38] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, England, 1994.

[39] David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[40] Kazuo Iwama and Shuichi Miyazaki. A survey of the stable marriage problem and its variants. In *International Conference on Informatics Education and Research for Knowledge-Circulating Society*, pages 131–136. IEEE Computer Society, January 2008.

[41] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[42] Robert Preis. Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In *STACS 99*, pages 259–269. Springer Berlin Heidelberg, 1999.