Temporal sampling annealing schemes for receding horizon multi-agent planning

Aaron Ma[a,c], Mike Ouimet[b,d], Jorge Cortés[a,c]

[a] *University of California, San Diego*
[b] *Singular Genomics, San Diego*
[c] *{aam021,cortes}@ucsd.edu*
[d] *michaelo@singulargenomics.com*

Abstract

This paper deals with multi-agent scenarios where individual agents must coordinate their plans in order to efficiently complete a set of tasks. Our strategy formulates the task planning problem as a potential game and uses decentralized stochastic sampling policies to reach a consensus on which sequences of actions agents should take. We execute this over a receding finite time horizon and take special care to discourage agents from breaking promises in the near future, which may cause other agents to unsuccessfully attempt a joint action. At the same time, we allow agents to change plans in the distant future, as this gives time for other agents to adapt their plans, allowing the team to escape locally optimal solutions. To do this we introduce two sampling schemes for new actions: a geometric-based scheme, where the probability of sampling a new action increases geometrically in time, and an inference-based sampling scheme, where a convolutional neural network provides recommendations for joint actions. We test the proposed schemes in a cooperative orienteering environment to illustrate their performance and validate the intuition behind their design.

Keywords: Multi-agent planning, potential games, reinforcement learning, simulated annealing

1. Introduction

Interest in autonomous unmanned vehicles (UxVs) and their applications to real-world scenarios is increasing enabled by recent technological and computational advances. Problems of interest include mapping of unknown regions, monitoring, and surveillance. The use of UxVs in these scenarios is motivated by safety, improved task performance, and robustness through redundancy. The ability to complete tasks to satisfy these objectives is largely determined by the vehicle's capabilities for autonomy and planning. When considering multiple vehicles, centralized task planning can become infeasible because the size of the joint action space grows exponentially as the number of agents increases.

This paper proposes a multi-agent task planning algorithm by modeling the joint action selection process that agents face as a potential game . For reasons of computational efficiency, the game is solved iteratively over a moving time horizon and, to promote finding more globally optimal solutions, agents utilize stochastic policies for selecting plans. Both aspects of our strategy might lead to agents changing their action right before it is executed, potentially leading to the loss of some other agent's utility because of the lack of time for replanning. To address this, we investigate sampling schemes which discourage agents from switching actions in the near future and instead allow agents to change plans in the distant future to escape locally optimal solutions.

Literature review

Distributed coordination algorithms aim to achieve a global outcome when agents only have partial information about each other and the environment, see e.g., [1, 2, 3, 4, 5]. Many of these algorithms are naturally robust to individual failures and distribute the computational load across the group of agents, making them preferable over centralized ones. Their correctness guarantees are valid under specific set of assumptions about the task and the environment. However, in general there is a need for the development of multi-agent deployment capabilities that can deal with scenarios with varying degrees of uncertainty.

Markov decision processes (MDP) are often used to model systems where the probability of state transition is only dependent on the current state and action selected by an agent [6, 7, 8, 9]. *Semi-Markov decision process* (SMDP) [10] are a variation of MDPs where agents select between macro-actions that each take a probabilistic amount of time to complete [6]. Model-based reinforcement learning can be utilized to generate plans or select actions in MDPs and SMDPs. This is commonly done by utilizing tree search as a means to sample the action space with Monte-Carlo tree search (MCTS) [11]. One efficient method [12] for sampling the action space during Monte-Carlo tree search is to select actions with respect to the upper confidence bound of the expected discounted reward for taking that action. Along with the use of tree search methods comes the challenge of the size of the state and action spaces: as they become large, sampling for suitable actions becomes infeasible within a reasonable amount of time. The use of abstractions and hierarchical structures is an efficient way to deal with this, cf. [13, 14], where agents search for subspaces of the environment on one level and optimize their plans within that subspace on another level. An alternative approach to deal with this challenge is the use of neural networks to aid in tree search. Neural networks have been used to infer the state value [15], saving time during tree search. These networks have also been trained on the distribution of actions selected at a given state during offline sessions. The output distribution of these networks can then be used to influence the action selection of tree search [16, 17, 18] in the hope of improving the sampling efficiency. Recently, deep convolutional neural networks have been used in both ExIt [19] and AlphaZero [20] to generate a stochastic action selection policy which resembles the action selection distribution for previous iterations of UCT/MCTS. The work [21] extends these ideas to environments where the dynamics is not known a priori and must be learned. Our work draws inspiration from recommendation networks when considering sampling actions in simulated annealing.

The algorithmic strategy proposed here also builds on the field of game theory to formulate the multi-agent planning decision process. Game theory [22, 23] provides a powerful framework for enabling cooperative decision making through strategic interactions. We are particularly interested in potential games [24], where the agents' incentive to change strategies is aligned with a global potential function. Potential games have been utilized for cooperative control [25] as well as for cooperative task planning [26]. In potential games, agents will reach a pure Nash equilibrium in finite time by incrementally choosing actions that improve their utility, albeit this Nash equilibrium may only be a local global optimizer of the potential function. To address this, spatial adaptive play [25, 27, 28] has an agent change its strategy at random with respect to a stochastic policy that balances exploration vs exploitation. In this paper, we utilize stochastic sampling policies from *simulated annealing* (SA) [29, 30, 31, 32, 33]. Simulated annealing is an approach for finding a state with optimal value in a Markov chain that borrows the idea of annealing from nature and is capable of handling high-dimensional, nonconvex problems. Under appropriate temperature schedules, simulated annealing finds the globally optimal solution with probability 1 as the number of iterations grows to infinity.

How the paper advances the state of the art

Scenarios where agents can complete multiple tasks often require precise and timely sequences of joint actions to reach a desired outcome. Furthermore, it is difficult to find an optimal sequence of joint actions in a reasonable amount of time with a large number of agents because of the number of possible outcomes. Consequently, there is a broad range of approaches for multi-agent task planning that attempt to alleviate the computational challenges of multi-agent task planning, generally in a stochastic fashion.

One recent popular success in multi-agent task planning is demonstrated with AlphaStar [34], where a system of networks is trained to assign precise tasks to units in Starcraft II. Multiple deep neural networks work together and are trained for very particular operations in order to select unit(s) and assign them an action. AlphaStar is able to compete at a competive level against humans, but its extension to real-world robotics presents significant challenges. One challenge is that AlphaStar is centralized in assigning tasks for the units, which can pose a problem in environments with latency, missed packages, or lack of connectivity amongst agents. In those environments, it is more robust to deploy an algorithm where agents are able to plan locally. Another limitation AlphaStar for implementation on real-world robotic systems is that the structure of the system of neural networks is engineered specifically for StarCraft II. One of the benefits of using machine learning in robotics is the capability of extending an algorithm to a broad range of environments and scenarios. Machine learning is beneficial in this aspect because it allows the agents' policy to be automated as opposed to hand-crafted by an engineer. Unlike AlphaStar, whose system of networks make it narrow and dedicated in scope of environment and application, our algorithm is applicable to any environment that can be described as a multi-agent Markov decision process.

Another successful algorithm created by Deepmind is AlphaZero [20], which utilizes a multitude of techniques to augment and improve UCT to compete in games such as chess and go. When trying to implement techniques used for AlphaZero for multi-agent task planning, the most obvious limitation is that it is designed for a single agent. Implementing a real world multi-agent deployment algorithm that is similar to AlphaZero poses analogous challenges as AlphaStar. In particular, communication concerns such as latency, and connectivity would inhibit the implementation of AlphaZero in a centralized way. In the fully decentralized case, a possible implementation would have each agent running their own tree search and predicting the other agents actions to be able to function independently of communication. Comparatively, the decentralized algorithm that we introduce here is flexible in its requirements to connectivity. Agents that lose connectivity during deployment are still able to operate on their own, and resume joint planning when they regain connectivity.

Spatial adaptive play (SAP) is game-theoretic algorithm used for multi-agent task planning. In SAP, agents individually sample actions. They accept and communicate the action if it yields higher expected reward or with some probability which decreases with time. One limitation of SAP is that it requires a heuristic in order to sample actions efficiently. As previously mentioned, one of the problems we face is the computational complexity in finding solutions in the large joint action space in multi-agent task planning. Without a heuristic, SAP resorts to sampling actions with respect to a flat distribution. Our algorithm deals with the large joint action space by sampling more efficiently with either a self-learned heuristic, or a temporal heuristic, both of which can be applied to a variety of scenarios. We also recycle our solutions and utilize a receding time horizon in order to make our sampling more efficient. Other algorithms aim to solve the problem of computational complexity by abstracting the state and action spaces. In the algorithm that we present, we sacrifice optimality by only planning over a finite time horizon as opposed to an infinite time horizon, but we reach an optimal multi-agent Nash equilibrium in that finite time horizon through the use of simulated annealing.

Statement of contributions

We strive to enable a swarm of UxVs to cooperate and complete a large variety of objectives in massive environments. The goal for the agents is to collectively generate plans in a receding horizon fashion to maximize future rewards gained from the completion of cooperative tasks. We model the problem as a potential game, where the potential function corresponds to the sum of future rewards over the time horizon. In the resulting potential game, agents select actions to increase their wonderful life utility, leading to a receding improvement path which terminates at a pure Nash equilibrium that, in general, is only a locally optimal solution of the potential function. With the aim of finding a globally optimal solution, we utilize simulated annealing, where we recycle solutions as the time horizon shifts and design sampling schemes which determine the probability that an agent will sample an action schedule given its current solution. Our sampling schemes are specifically tailored to discourage agents from breaking promises in the near future and instead allow them to change plans in the distant future. The first sampling scheme, termed 'geometric sampling', is novel in that the placement of action schedule elements is determined with respect to their position in the receding time horizon. This sampling scheme aims to be efficient in terms of the solutions that we recycle during the simulated annealing process, leading to quicker convergence of the solution to the stationary distribution. The second sampling scheme, termed 'inference-based' sampling, incentivizes the agents to select actions during the simulated annealing process that lead to globally optimal solutions. Inspired by recent work in tree searches guided by neural networks, we utilize convolutional neural networks to generate efficient sampling matrices given the state of the environment. We provide theoretical analysis for properties of the Markov chain induced by the simulated annealing process given our sampling schemes. Performance and theoretical results are validated with metrics tested in a cooperative orienteering environment.

2. Preliminaries

We introduce here essential concepts and tools for the rest of the paper, beginning with some notation. We use $\mathbb{Z}$ and $\mathbb{R}$ to denote integers and real numbers, respectively. An object-oriented approach with the use of tuples is present throughout the paper: for an arbitrary tuple $a = \langle b, c \rangle$, the notation $a.b$ means that b belongs to tuple a. Also, $|\mathcal{Y}|$ indicates the cardinality of a set $\mathcal{Y}$. A row-stochastic matrix P is a matrix with non-negative entries whose rows sum 1. Row-stochastic matrices have 1 as their largest absolute value eigenvalue.

2.1. Markov chains

A Markov chain describes a sequence of states such that the probability of transitioning from one state to another only depends on the current state. We define a Markov chain as a tuple $\langle \mathcal{S}, P \rangle$ of states $s \in \mathcal{S}$ and probability $P(s'|s)$ that s transitions to s'. If we assign states of the Markov chain to nodes, and positive state transition probabilities $P(s'|s) > 0$ to edges, we can create a directed graph representing the Markov chain.

The transition probabilities can be used to construct the *transition matrix* P^{tr} such that

$$\mathrm{P}^{tr}_{ij} = P(j|i).$$

The transition matrix is row-stochastic, and hence it has an eigenvalue $\lambda = 1$. Let $\{S_0, \ldots, S_k\}$ denote a series of random variables representing state transitions of a Markov chain across time. Let the vector $\mu_k \in \mathbb{R}^{|\mathcal{S}|}$ be the *distribution* of random variables at time k such that

$$\mu_k(s) = P(S_k = s).$$

A Markov chain is *irreducible* if for any state $j \in \mathcal{S}$ there exist k such that $\mu_k(j) > 0$ from $S_0 = s$, for all $s \in \mathcal{S}$. As k tends to infinity, the distribution of states in the Markov chain converges to the *stationary distribution*

$$\lim_{k \to \infty} \mu_k = v.$$

The stationary distribution can be determined by finding the row-eigenvector of P^{tr} associated with the eigenvalue $\lambda = 1$, $v\mathrm{P}^{tr} = v$. The total variation distance

$$d(v, \mu) = \sup_{s \in \mathcal{S}} |v(s) - \mu(s)|,$$

measures the difference between the stationary distribution v and an arbitrary distribution μ. The *mixing time* [35] t_{mix} of a Markov chain, $t_{\mathrm{mix}} = \min\{k : d(v, \mu_k) \leq \epsilon\}$ captures the time required for the stationary distribution and the distribution μ_k at time k to become smaller than ϵ. The mixing time can be approximated via the *relaxation time* of a Markov chain,

$$t_{\mathrm{rel}} = \frac{1}{1 - \lambda_2}, \tag{1}$$

where λ_2 is the second largest eigenvalue of P^{tr} ($1 - \lambda_2$ is known as the *spectral gap*). The relaxation time gives the following bounds on the mixing time,

$$(t_{\mathrm{rel}} - 1) \log\left(\frac{1}{2\epsilon}\right) \leq t_{\mathrm{mix}}(\epsilon) \leq \log\left(\frac{1}{\epsilon v_{\min}}\right) t_{\mathrm{rel}}, \tag{2}$$

where $v_{\min} := \min_{s \in \mathcal{S}} v(s)$. In general, a small mixing time is desirable when we want the distribution of the Markov chain to reach steady state quickly, which is common in computational applications. This in turn means that a small relaxation time t_{rel} is also desirable.

2.2. Simulated annealing

Let $V : \mathcal{S} \to \mathbb{R}$ be a function that denotes the value of each state in the Markov chain. Simulated annealing seeks to determine the *optimal state* $\operatorname{argmax}_s V(s)$, which is generally a combinatorial problem [29, 30, 32, 33]. In order to find it, simulated annealing is a probabilistic algorithm that samples states from the current one and accepts them with a probability that is increasingly determined, as the temperature of the annealing process decreases, by whether they lead to a larger value of V.

Simulated annealing employs a row-stochastic *sampling matrix* P^s. Here P^s_{ij} represents the probability of considering candidate solution j when the current solution is i. The candidate solution is accepted with probability $e^{(V(j)-V(i))/T_k}$, where T_k is the temperature of the process at time k. For a given temperature, we can construct the *acceptance matrix*

$$\mathrm{P}^a_{ij} = \begin{cases} 1 & \text{if } i = j \\ \min(1, e^{\frac{V(i)-V(j)}{\mathcal{T}_k}}) & \text{else.} \end{cases} \tag{3}$$

Algorithm 1: Simulated annealing

```
1 $s$ is sampled uniformly from $S$
2 for $k = 1 \to K$:
3     select $s'$ with probability $\mathrm{P}^s_{ss'}$
4     if $V(s') > V(s)$ or rand()> $\mathrm{P}^a_{ss'}$:
5         $s = s'$
```

Algorithm 1 outlines the process of simulated annealing with respect to P^s and P^a. Given the sampling and acceptance matrices, the probability that solution i transitions to solution j under a single iteration of the simulated annealing process are captured by the elements of the *transition matrix*,

$$\mathrm{P}^{tr}_{ij} = \begin{cases} \mathrm{P}^s_{ij} + \sum\limits_{k \neq i} \mathrm{P}^s_{ik}(1 - \mathrm{P}^a_{ik}) & \text{if } i = j \\ \mathrm{P}^s_{ij}\mathrm{P}^a_{ij} & \text{else.} \end{cases} \tag{4}$$

which is row-stochastic, cf. [36]. This transition matrix defines a Markov chain. Simulated annealing seeks to manipulate the transition matrix such that the expected states of the stationary distribution are in the set of optimal states. In order to ensure that this is possible from any initial state, it is important to maintain irreducibility in the induced Markov chain, and therefore a positive temperature is required. In practice, large temperatures help the induced Markov chain escape local maxima. Cooling schedules gradually have the temperature of the process decrease over time. Simulated annealing converges to the global optimal if the temperature is decreased sufficiently slowly. The cooling rate

$$T_k = \frac{c}{\log(k)}, \qquad \forall k \in \{1, \ldots, K\}, \tag{5}$$

where $c \in \mathbb{R}$ corresponds to the difference between maximum and minimum values of solutions. This cooling schedule is shown [32] to be a necessary and sufficient condition for the algorithm to converge in probability to a set of states that globally optimize V when the underlying Markov chain has both strong irreducible and weak reversible properties. We are also interested in linear cooling schedules of the form

$$\mathcal{T}_k = \max(\mathcal{T}_0 - ck, 0), \qquad \forall k \in \{1, \ldots, K\}, \tag{6}$$

which generally yield strong performance and constant temperature schedules

$$\mathcal{T}_k = \mathcal{T}_0, \qquad \forall k \in \{1, \ldots, K\}, \tag{7}$$

which is relevant for theoretical analysis of our sampling schemes.

2.3. Potential games

In a game-theoretic framework agents select actions in order to maximize their utility. For agent $\alpha \in \mathcal{A}$, a function $u_\alpha : A_\alpha \times A_{-\alpha} \to \mathbb{R}$ defines its *utility* as $u_\alpha(a_\alpha, a_{-\alpha})$, where $a_\alpha \in A_\alpha$ denotes its action and $a_{-\alpha} \in A_{-\alpha}$ denotes the actions of all other agents. Agents have the option of selecting the *null action*, denoted $\emptyset \in A_\alpha$, in which α does not contribute to the completion of any objective in the game. Let $a = \{a_1, \ldots, a_{|\mathcal{A}|}\}$ be the collection of actions selected by all the agents. The collection of actions selected by the agents is a *pure Nash equilibrium* when no agent can select an action that unilaterally improves their utility given the actions of other agents. Formally,

$$u(a_\alpha, a_{-\alpha}) \geq u(a'_\alpha, a_{-\alpha}), \quad \forall \alpha \in \mathcal{A},\ a'_\alpha \in A_\alpha.$$

In our treatment, we rely on the notion of potential game. A game is an *exact potential game* if there exists a function $\mathcal{V} : A_\alpha \times A_{-\alpha} \to \mathbb{R}$, called *potential function*, such that

$$u_\alpha(a_\alpha, a_{-\alpha}) - u_\alpha(a'_\alpha, a_{-\alpha}) = \mathcal{V}(a_\alpha, a_{-\alpha}) - \mathcal{V}(a'_\alpha, a_{-\alpha}), \tag{8}$$

for all $\alpha \in \mathcal{A}$. In an exact potential game, the utilities of all agents are all aligned, in the sense that the strategic improvement of its own utility function by one agent actually contributes to the improvement of a common global utility function $\mathcal{V}$.

In this paper we take advantage of the fact that, given an arbitrary function, one can define utility functions so that the resulting game is an exact potential game. Formally, given $\mathcal{V}$, we define the *wonderful life utility*

$$u_\alpha(a_\alpha, a_{-\alpha}) = \mathcal{V}(a_\alpha, a_{-\alpha}) - \mathcal{V}(\emptyset, a_{-\alpha}), \quad \text{for } \alpha \in \mathcal{A}. \tag{9}$$

The wonderful life utility is a measure of the marginal gain that an agent contributes when selecting a given action. It is easy to see that the wonderful life utility ensures (8) holds.

Generally, potential games have at least one pure Nash equilibrium. An *improvement path* is any sequence of strategies $\{a^t\}_{t=0,1,\ldots}$ such that $u(a^{t+1}) \geq u(a^t)$ wherever a^{t+1} is defined. A *finite improvement path* is an improvement path that terminates at a Nash equilibrium. Exact potential games have finite improvement paths so if agents select actions that improve their utility, they eventually arrive at a pure Nash equilibrium that is a local optimum of the potential function $\mathcal{V}$. The local optimum is generally not global because, when the potential function is not convex, reaching the global optimum from an arbitrary initial condition requires agents to select actions which yield a lower value, i.e., the agents do not follow a finite improvement path.

3. Problem statement

Consider a scenario where a team of N agents seek to jointly determine a schedule of future actions. Agents must select actions that alter the *state* $s = (s_1, \ldots, s_N, s^e) \in S$, where s_1 through s_N represent the states of agents 1 through N, and s_e represents the state of the environment. Agent α is able to select action a_α according to its individual action space A_α. We use $\bar{a}^t = \{a_1^t, \ldots, a_N^t\}$ to represent the set of actions that agents plan to select at exactly time t. The execution of these actions by the agents gives rise to the state s^t. When a state is reached, agents receive a reward determined by $R : S \to \mathbb{R}$. Agents seek to collectively maximize the sum of future rewards

$$\sum_{t=1}^{\infty} R_{s^t}, \tag{10}$$

representing the sum of future rewards that are gathered by the agents over an infinite time horizon. Determining the optimal sequence of agents' actions that optimize (10) is in general difficult due to the infinite nature of the time horizon and the computational effort required to determine a solution under time constraints when considering real-world deployments.

Therefore we solve the problem in a receding horizon fashion. Given a time horizon T, let $\bar{a}_\alpha = [a_\alpha^1, \ldots, a_\alpha^T] \in \bar{A}_\alpha$ denote the *action schedule* that agent α plans to execute over the next T timesteps. The *joint action schedule* $\bar{a} = [\bar{a}_1, \ldots, \bar{a}_N] \in \bar{A}$ is the set of all agents action schedules and, for convenience, we let $\bar{a}_{-\alpha} = \bar{a} \setminus \bar{a}_\alpha$. Define

$$\mathcal{V}_t(\bar{a}) = \sum_{\ell=1}^{T} R_{s^{t+\ell}}, \tag{11}$$

representing the sum of future rewards that are gathered by the agents over the time horizon T. We treat the scheduling problem as a receding time horizon where agents strive to determine

$$\operatorname*{argmax}_{\bar{a} \in \bar{A}} \mathcal{V}_t(\bar{a}),$$

at every time step t during deployment. To achieve this, we set up a potential game among the agents where each one is endowed with the 'wonderful life utility' defined in (9). By selecting action schedules that improve their utility, agents follow a finite improvement path that eventually leads them to reach a Nash equilibrium. However, doing this only guarantees that the agents reach a local optimum of the potential function. Furthermore, because we are solving the problem in a receding horizon fashion, there is nothing precluding agents from deviating from previously agreed upon joint plans from one timestep to the next, leading to poor performance. Because the agents can deviate from their previously agreed upon joint plans, we assume that changes in action schedules by an agent are communicated to all other agents.

We strive to improve upon this process through the use of decentralized simulated annealing and recycling solutions from previous time steps. The use of stochastically selected actions in simulated annealing improves the chances of reaching a Nash equilibrium that belongs to the set of globally optimal states. In addition, we seek to tune the sampling strategy employed during the annealing process to discourage agents from breaking promises in the near future and instead allow them to change plans in the distant future. We explain our strategy in detail in the next section. In what follows, we interchangably use the notation $\bar{a}_\alpha$ and i to refer to an arbitrary action schedule. In general, we use $\bar{a}_\alpha$ when we want to stress the game-theoretic context and i when we want to stress the Markov-chain contfext.

4. Action scheduling with recycled solutions

In this section we detail our strategy to employ simulated annealing, taking advantage of the flexibility offered by the sampling stage to recycle action schedules agreed upon by agents at the previous timestep. The inclusion of stochastic sampling of action schedules with a cooling schedule is motivated by the inherent temporal structure of the solutions. Action schedules likely have similar utilities when their actions are similar. Using a cooling schedule means that extra pathways in the Markov chain to the optimal action schedule will exist. The motivation is that by increasing the number of pathways to the optimal solution, we increase the probability to find that solution and it will be found more quickly with simulated annealing than without.

We study three components to the evolution of events that occur during deployment in our algorithms. First, agents execute an action. Then, actions are shifted to represent the step, i.e., $\bar{a}_\alpha^t$ becomes $\bar{a}_\alpha^{t-1}$ for all $t \in (2, \ldots, T)$ and $\alpha \in \mathcal{A}$. The shifted action schedules will become the initial solution for agents in the next step of the finite time horizon. However, these action schedules will be missing $\bar{a}_\alpha^T$, so the third component we consider is the generation of that action. We refer to this process as *recycling solutions*.

Here, we propose a deployment strategy, outlined in Algorithm 2, called *horizon shift planning* (`HSP`). `HSP` uses *horizon shift annealing* (`HSA`), where agents sample action schedules with respect to the sampling matrix P^s (specific schemes to determine this matrix are introduced later in Section 5, giving rise to different instantiations of `HSP`). With probability respective to P^a, the agent accepts the candidate action schedule. The agent then broadcasts the change in action schedule with some probability. This process is repeated for K steps, which captures the limited amount of time allowed for computation during each time step of the receding time horizon. It is important to note that after agents take actions, and the finite time horizon shifts, agents recycle their chosen action schedules $\bar{a}$ to be used as an initial action schedule for the next time step.

Algorithm 2: Horizon shift planning

```
HSP($s, \mathcal{A}$)
    while deploying is **true**:
        for $k = 1 \to K$;
            for all $\alpha \in \mathcal{A}$ in parallel:
                $\bar{a}_\alpha = \texttt{HSA}(\bar{a}_\alpha, k)$
        executeActions($\bar{a}$)
        for $t = 1 \to (T-1)$:
            $s^t = s^{t+1}$
            $\bar{a}^t = \bar{a}^{t+1}$
```

```
HSA($i := \bar{a}_\alpha, k$)
    Select $j$ with probability $\mathrm{P}^s_{ij}$
    $\Delta = u_j - u_i$
    if $\Delta > 0$ or rand() $> e^{\Delta/\mathcal{T}_k}$:
        $i \leftarrow j$
        if rand() $> \tau$:
            updateAgents($i$)
    return $i := \bar{a}_\alpha$
```

Theorem 1. *(Reaching pure Nash equilibria through HSA): Consider the horizon shift annealing HSA described in Algorithm 2 and assume that* $\mathrm{P}^s_{ij} > 0$ *for any i and j, with agents using the wonderful life utility. Under the linear cooling schedule* (6)*, agents reach a pure Nash equilibrium in a finite number of iterations. Furthermore, under the logarithmic cooling schedule, agents reach a pure Nash equilibrium w.p.* 1 *as K tends to infinity.*

Proof. We use the fact that exact potential games exhibit the *finite improvement property*, meaning that every improvement path is finite in length. Because of this property, a Nash equilibrium exists. Under the linear cooling schedule (6), for $k > c$, the temperature becomes 0 and agents will only choose action schedules with positive utility u. Agents then select actions that are along some finite improvement path until they reach a pure Nash equilibrium.

Under the logarithmic cooling schedule, note that there exist a positive probability to sample any of the agents action schedules based on the assumption that $\mathrm{P}^s_{ij} > 0$ for all i and j. The probability of accepting only action schedules that improve the wonderful life utility tends to 1 as the iteration k goes to infinity. It follows that the agent has access to all action schedules which follow a finite improvement path, which are chosen with probability 1 as k goes to infinity. □

Note that the proof of Theorem 1 relies on the fact that, once the temperature is 0, so long as $\mathrm{P}^s_{i,j} > 0$ for all $i := \bar{a}_\alpha$ and $j := \bar{a}'_\alpha$, a locally optimal solution can be found given large enough K. We introduce next strategies for sampling action schedules to ensure that agents are discouraged from breaking promises in the near future (to avoid negatively impacting other agents' plans) while at the same time they are allowed to change plans in the distant future (to improve our chances of finding a globally optimal solution).

5. Sampling schemes for action scheduling

In this section we design sampling schemes for action scheduling. In general, the probability P^s is not viewed as a design choice in annealing approaches, and is often a result of existing state transition probabilities in a Markov chain. In our case, since agents can choose to take an arbitrary action schedule, we design P^s to make sampling more efficient with respect to the finite time horizon and to confront the issue of agents breaking promises in the near future, which may cause other agents to lose utility.

5.1. Matrix structure with respect to action schedule indices

In this section we describe some notational conventions regarding the sampling matrix that will be useful later when introducing our novel sampling schemes. We choose to structure and position the indices of P^s with the following conventions. For agent α, $\mathrm{P}^s \in \mathbb{R}^{|A_\alpha^T| \times |A_\alpha^T|}$ is a block matrix with $|A_\alpha| \times |A_\alpha|$ blocks corresponding to the number of actions the agent can choose for the first time step of the horizon. A block $\mathrm{P}^s_{i^1}$ of P^s is a matrix of probabilities of transition for action schedules with first action i^1 to action schedules with the same first action i^1. $\mathrm{P}^s_{i^1}$ is itself a block matrix with $|A_\alpha| \times |A_\alpha|$ blocks. This organization of P^s has the following properties:

- Diagonal elements are transitions to the same action schedule. Diagonal blocks are transitions to the same action taken at that time step;
- P^s is row-stochastic.

In addition to the sampling matrix, we also structure the acceptance and transition matrices, P^a and P^{tr}, with the same convention for the action schedule indices. Organizing the matrices in this fashion is valuable because it provides intuition for the resulting distribution of action schedules that we get from recycling solutions. We let the *recycled stationary distribution* $\hat{v}_{i^1}$ indicate the conditional probability distribution of action schedules as determined by the transition matrix, given that the selected action schedule has first action i^1. To determine this stationary distribution, we define the *recycled transition matrix* $\hat{\mathrm{P}}^{tr} \in \mathbb{R}^{|A_\alpha^{T-1}| \times |A_\alpha^{T-1}|}$ by

$$\hat{\mathrm{P}}^{tr}_{ij} = \begin{cases} \hat{\mathrm{P}}^s_{ij} + \sum\limits_{k \neq i} \hat{\mathrm{P}}^s_{ik}(1 - (\mathrm{P}^a_{i^1})_{ik}) & \text{if } \ i = j \\ \hat{\mathrm{P}}^s_{ij}(\mathrm{P}^a_{i^1})_{ij} & \text{else,} \end{cases} \tag{12a}$$

where $\mathrm{P}^a_{i^1}$ is a block of P^a that corresponds to the first action i^1 taken, and $\hat{\mathrm{P}}^s \in \mathbb{R}^{|A_\alpha^{T-1}| \times |A_\alpha^{T-1}|}$ is the probability of sampling an action schedule conditioned on the first action taken being i^1,

$$\hat{\mathrm{P}}^s_{ij} = \frac{(\mathrm{P}^s_{i^1})_{ij}}{\sum\limits_j (\mathrm{P}^s_{i^1})_{ij}}. \tag{12b}$$

The recycled stationary distribution $\hat{v}_{i^1}$ is then determined by finding the row-eigenvector that corresponds to the eigenvalue $\lambda_1 = 1$. We are interested in recycled stationary distribution because this is a distribution of initial solutions for our next step in the receding time horizon, which implies that it will require less iterations to reach steady state. Figure 1 illustrates an example sampling matrix with $|A_\alpha| = 2$ and $T = 4$, the corresponding transition matrix P^{tr}, and the recycled transition matrix.

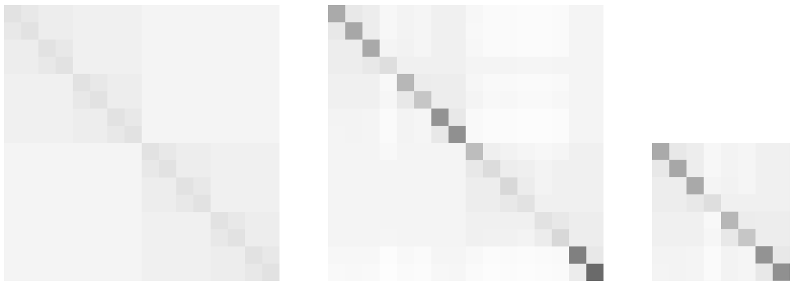

(a) P^{gs} (b) P^{tr} (c) $\hat{\mathrm{P}}^{tr}$

Figure 1: Illustration of the matrix structure with respect to action schedule indices. In (a), we show the probability of action schedule sampling according to P^{gs} as defined in (13). In this case, the agent has an action space $|A_\alpha|$ of 2 and is planning up to $T = 4$ time steps ahead. In the image, the intensity of the pixel i, j corresponds to the probability of the agent sampling $j := \bar{a}'_\alpha$ when its last solution is $i := \bar{a}_\alpha$. Light and dark pixels correspond to low and high probabilities, respectively. In (b), we show the transition matrix P^{tr} for the simulated annealing process using P^{gs} and P^{a} (not shown). In (c), we show $\hat{\mathrm{P}}^{tr}$, which is the block of P^{tr} that corresponds to the first action $i^1 = 1$ being chosen. $\hat{\mathrm{P}}^{tr}$ helps us approximate the transition matrix of the simulated annealing process in the time step after executing action $i^1 = 1$.

5.2. Geometric sampling

Here we introduce the *geometric sampling* scheme. The idea is inspired by simulated annealing where we modify the probability that actions are sampled based on their position in the finite time horizon. Specifically, the generation probability for a sampled action schedule is geometrically reduced as follows

$$\mathrm{P}^{gs}_{ij} = \frac{\rho^{T-t_{ij}}}{\sum\limits_{p \neq j} \rho^{T-t_{ip}}}, \tag{13}$$

where $\rho \in (0, 1)$ and t_{ij} is the minimum time where actions deviate in the two action schedules, i.e., $t_{ij} = \min \{t \in \mathbb{R} \mid i^t \neq j^t\}$. Note that the expression '$T - t_{ij}$' is larger when two action schedules i and j deviate closer to the present time. When the term 'ρ' is small, there is a small probability that the agent samples an action schedule j where t_{ij} is small. We use ρ as a design parameter to influence the agent to sample actions that are further in the finite time horizon, more often. Figure 1 illustrates the above definition.

According to the scheme (13), we decrease the probability that agents sample action schedules that change actions in the near future to avoid 'breaking short-term promises' with other agents. However, agents are not disincentivized from making such changes in the distant future in order to find joint cooperative plans with high reward.

5.2.1. Approximating the next-step transition matrix via horizon shift

For the sake of evaluating the effect of recycling solutions we define the *next-step transition matrix* $\mathrm{P}^{tr\prime} \in \mathbb{R}^{A_\alpha^T \times A_\alpha^T}$, which is the transition matrix after taking a step in the environment. We do not know what the transition matrix will be without evaluating action schedules at the $T+t+1$ time step, so instead we determine the *approximate next-step transition matrix*, $\tilde{\mathrm{P}}^{tr\prime} \in \mathbb{R}^{A_\alpha^{T-1} \times A_\alpha^{T-1}}$, which approximates the transition matrix at the next time step corresponding to a shorter time horizon than usual, $\tilde{T}' = T - 1$.

$$\tilde{\mathrm{P}}^{tr\prime}_{ij} = \begin{cases} \tilde{\mathrm{P}}^{gs\prime}_{ij} + \sum\limits_{k \neq i} \tilde{\mathrm{P}}^{gs\prime}_{ik}(1 - \tilde{\mathrm{P}}^{a\prime}_{ik}) & \text{if } i = j \\ \tilde{\mathrm{P}}^{gs\prime}_{ij}\tilde{\mathrm{P}}^{a\prime}_{ij} & \text{else,} \end{cases} \tag{14}$$

where $\tilde{\mathrm{P}}^{gs\prime} \in \mathbb{R}^{A_\alpha^{T-1} \times A_\alpha^{T-1}}$ and $\tilde{\mathrm{P}}^{a\prime} \in \mathbb{R}^{A_\alpha^{T-1} \times A_\alpha^{T-1}}$ represent approximates, which we define next, of the geometric sampling matrix and acceptance matrix, respectively, at the next time step. These matrices reflect changes that we expect during the next time step. To determine $\tilde{\mathrm{P}}^{a\prime}$, note that the receding time horizon is smaller by 1. We then have the following result regarding evaluating $\tilde{\mathrm{P}}^{a\prime}$ from P^a.

Lemma 2. *(Determining $\tilde{\mathrm{P}}^{a\prime}$ from P^a): Let P^a be the acceptance matrix corresponding to the receding time horizon $[t, T+t]$ and $\tilde{\mathrm{P}}^{a\prime}$ be the acceptance matrix corresponding to the receding time horizon $[t+1, T+t]$. Assume without loss of generality that an action selection with first action i^1 is chosen for the current time step t. Then for a constant cooling schedule with $\mathcal{T} = \mathcal{T}' = \mathcal{T}_0$*

$$\tilde{\mathrm{P}}^{a\prime} = \mathrm{P}^a_{i^1}$$

Proof. Since the temperatures are constant between the transitions, the acceptance probability determined from (3) is determined as a function of evaluations of the potential function

$$\mathcal{V}_t(i) - \mathcal{V}_t(j) = \sum_{l=1}^{T} R_{s^{t+l}} - \sum_{l=1}^{T} R_{s^{t+l}}.$$

We assume that i and j belong to the same block $\mathrm{P}^a_{i^1}$ and that the first action of i and j are the same. This means that the rewards R_{s^t} and R_{s^t} are the same, thus

$$\mathcal{V}_t(i) - \mathcal{V}_t(j) = \sum_{l=2}^{T} R_{s^{t+l}} - \sum_{l=2}^{T} R_{s^{t+l}} = \mathcal{V}_t(i^{2:T}) - \mathcal{V}_t(j^{2:T}).$$

It then follows from (3) that elements of $\mathrm{P}^a_{i^1}$ are equivalent to those in $\tilde{\mathrm{P}}^a$. $\square$

To determine $\tilde{\mathrm{P}}^{gs\prime}$, note that each element of $\tilde{\mathrm{P}}^{gs\prime}$ corresponds to an element of a block of P^{gs} with updated minimum deviation times t_{ij}, cf (13), reduced by 1. To determine $\tilde{\mathrm{P}}^{gs\prime}$ from $\hat{\mathrm{P}}^{gs}$, we apply the *horizon shift operation* defined by

$$\tilde{\mathrm{P}}^{gs\prime}_{ij} = \rho \hat{\mathrm{P}}^{gs}_{ij}, \qquad \forall j \neq i, \tag{15a}$$

$$\tilde{\mathrm{P}}^{gs\prime}_{ij} = \hat{\mathrm{P}}^{gs}_{ij} + (1-\rho)\sum_{j\neq i} \hat{\mathrm{P}}^{gs}_{ij}, \qquad \forall i, \tag{15b}$$

where $\rho \in (0,1)$. When this operation is applied to $\hat{\mathrm{P}}^{gs}$, the probability of all transitions between differing action schedules in P^{tr} is reduced by ρ.

5.2.2. Relationship between the recycled and next-step transition matrices

Next we describe the relationship between the recycled transition matrix $\hat{\mathrm{P}}^{tr}$ and the approximate next-step transition matrix $\tilde{\mathrm{P}}^{tr\prime}$ given what we know about P^{a}, $\tilde{\mathrm{P}}^{a\prime}$, $\hat{\mathrm{P}}^{gs}$, and $\tilde{\mathrm{P}}^{gs\prime}$.

Theorem 3. *(Relationship between the recycled and next-step transition matrices): Let the recycled transition matrix $\hat{\mathrm{P}}^{tr}$ be determined as shown in* (12a) *using $\hat{\mathrm{P}}^{s}$ from* (12b) *and a block $\mathrm{P}^{a}_{i^1}$ which correspond to a first action i^1. Also, let the approximate next transition matrix $\tilde{\mathrm{P}}^{tr\prime}$ be determined from* (14) *where $\tilde{\mathrm{P}}^{gs\prime}$ be the output of the horizon shift operation on $\hat{\mathrm{P}}^{gs}$ as described in* (15)*, and $\tilde{\mathrm{P}}^{a\prime} = \mathrm{P}^{a}_{i^1}$ be the acceptance matrix as determined in Lemma 2. Then, $\tilde{\mathrm{P}}^{tr\prime}$ is row-stochastic. Also, let $\hat{\lambda}_1, \hat{\lambda}_2, \ldots$ and $\tilde{\lambda}'_1, \tilde{\lambda}'_2, \ldots$ be the eigenvalues of $\hat{\mathrm{P}}^{tr}$ and $\tilde{\mathrm{P}}^{tr\prime}$, respectively, such that*

$$\hat{\lambda} \geq \hat{\lambda} \geq \ldots \geq \hat{\lambda}_{|A^{T-1}_{\alpha}|}$$
$$\tilde{\lambda}' \geq \tilde{\lambda}' \geq \ldots \geq \tilde{\lambda}'_{|A^{T-1}_{\alpha}|},$$

Then, $\tilde{\lambda}'_k = 1 + \rho\hat{\lambda}_k - \rho$. Furthermore, the row-eigenvectors are invariant under the operation.

Proof. The horizon shift operation as presented is applied to $\hat{\mathrm{P}}^{gs}$, to get $\tilde{\mathrm{P}}^{gs\prime}$ in order to determine $\tilde{\mathrm{P}}^{tr\prime}$. We show that the application of the horizon shift operation can be applied directly to the recycled transition matrix $\hat{\mathrm{P}}^{tr}$ to get transition matrix $\tilde{\mathrm{P}}^{tr\prime}$, i.e.

$$\tilde{\mathrm{P}}^{tr\prime}_{ij} = \rho \hat{\mathrm{P}}^{tr}_{ij}, \qquad \forall j \neq i,$$
$$\tilde{\mathrm{P}}^{tr\prime}_{ij} = \hat{\mathrm{P}}^{tr}_{ij} + (1-\rho)\sum_{j\neq i} \hat{\mathrm{P}}^{tr}_{ij}, \qquad \forall i,$$

For all $j \neq i$, we have $\tilde{\mathrm{P}}^{tr\prime}_{ij} = (\rho \hat{\mathrm{P}}^{s}_{ij})\tilde{\mathrm{P}}^{a\prime}_{ij} = \rho \hat{\mathrm{P}}^{tr}_{ij}$. Thus, all off-diagonal elements are the same when the horizon shift operation is applied to either $\hat{\mathrm{P}}^{gs}$ or $\hat{\mathrm{P}}^{tr}$. Then for all diagonal elements $\tilde{\mathrm{P}}^{tr\prime}_{ii}$ from (14)

$$\tilde{\mathrm{P}}^{tr\prime}_{ii} = \tilde{\mathrm{P}}^{gs\prime}_{ii} + \sum_{j\neq i} \tilde{\mathrm{P}}^{gs\prime}_{ij}(1 - \tilde{\mathrm{P}}^{a\prime}_{ij})$$

$$
\begin{aligned}
&= (\mathrm{P}^{gs}_{ii} + (1-\rho)\sum_{j\neq i}\mathrm{P}^{gs}_{ij}) + \rho\sum_{j\neq i}\mathrm{P}^{gs}_{ij}(1-\tilde{\mathrm{P}}^{a\prime}_{ij})\\
&= \mathrm{P}^{gs}_{ii} + \sum_{j\neq i}\mathrm{P}^{gs}_{ij} - \rho\sum_{j\neq i}\mathrm{P}^{gs}_{ij}\tilde{\mathrm{P}}^{a\prime}_{ij}\\
&= \mathrm{P}^{gs}_{ii} + \sum_{j\neq i}\mathrm{P}^{gs}_{ij} - \rho\sum_{j\neq i}\mathrm{P}^{gs}_{ij}\tilde{\mathrm{P}}^{a\prime}_{ij} + \sum_{j\neq i}\mathrm{P}^{gs}_{ij}\tilde{\mathrm{P}}^{a\prime}_{ij} - \sum_{j\neq i}\mathrm{P}^{gs}_{ij}\tilde{\mathrm{P}}^{a\prime}_{ij}\\
&= \mathrm{P}^{gs}_{ii} + \sum_{j\neq i}\mathrm{P}^{gs}_{ij}(1-\tilde{\mathrm{P}}^{a\prime}_{ij}) + (1-\rho)\sum_{j\neq i}\mathrm{P}^{gs}_{ij}\tilde{\mathrm{P}}^{a\prime}_{ij}\\
&= \mathrm{P}^{gs}_{ii} + \sum_{j\neq i}\mathrm{P}^{gs}_{ij}(1-(\mathrm{P}^{a}_{i^1})_{ij}) + (1-\rho)\sum_{j\neq i}\mathrm{P}^{gs}_{ij}(\mathrm{P}^{a}_{i^1})_{ij}\\
&= \hat{\mathrm{P}}^{tr}_{ii} + (1-\rho)\sum_{i\neq j}\hat{\mathrm{P}}^{tr}_{ij}
\end{aligned}
$$

This means that we can apply the horizon shift operation shown in (15) to either $\hat{\mathrm{P}}^{gs}$ or directly to $\hat{\mathrm{P}}^{tr}$ in order to get $\tilde{\mathrm{P}}^{tr\prime}$. It follows from the horizon shift definition that $\tilde{\mathrm{P}}^{gs\prime}$, and therefore $\tilde{\mathrm{P}}^{tr\prime}$, is row-stochastic. Next we show that the row-eigenvectors are invariant under the operation when the horizon shift operation is applied directly to $\hat{\mathrm{P}}^{tr}$, i.e.,

$$v\tilde{\mathrm{P}}^{tr\prime} = v\hat{\mathrm{P}}^{tr} = \lambda v = [v_1, \ldots, v_N]$$

Consider the element-wise calculation for v_i.

$$(v\tilde{\mathrm{P}}^{tr\prime})_i = \sum_{\forall j} v_j\tilde{\mathrm{P}}^{tr\prime}_{ji} = v_i\tilde{\mathrm{P}}^{tr\prime}_{ii} + \sum_{i\neq j} v_j\tilde{\mathrm{P}}^{tr\prime}_{ji}$$

We apply the horizon shift operation to the right-hand side to obtain

$$
\begin{aligned}
&(v_i\hat{\mathrm{P}}^{tr}_{ii} + (1-\rho)\sum_{j\neq i} v_i\hat{\mathrm{P}}^{tr}_{ij}) + \rho\sum_{j\neq i} v_j\hat{\mathrm{P}}^{tr}_{ji}\\
&= (1-\rho)v_i\hat{\mathrm{P}}^{tr}_{ii} + \rho v_i\hat{\mathrm{P}}^{tr}_{ii} + (1-\rho)\sum_{j\neq i} v_i\hat{\mathrm{P}}^{tr}_{ij} + \rho\sum_{j\neq i} v_j\hat{\mathrm{P}}^{tr}_{ji}\\
&= \rho(v_i\hat{\mathrm{P}}^{tr}_{ii} + \sum_{j\neq i} v_j\hat{\mathrm{P}}^{tr}_{ji}) + (1-\rho)(v_i\hat{\mathrm{P}}^{tr}_{ii} + \sum_{j\neq i} v_i\hat{\mathrm{P}}^{tr}_{ij})
\end{aligned}
$$

Then we use the fact that $v_i\hat{\mathrm{P}}^{tr}_{ii} + \sum_{j\neq i} v_j\hat{\mathrm{P}}^{tr}_{ji} = \lambda v_i$ and group the terms by ρ

$$\rho\lambda v_i + (1-\rho)(v_i\hat{\mathrm{P}}^{tr}_{ii} + \sum_{i\neq j} v_i\hat{\mathrm{P}}^{tr}_{ij})$$

Next, since $\hat{\mathrm{P}}^{tr}$ is row-stochastic, we conclude

$$\lambda' v_i = \rho\lambda v_i + (1-\rho)v_i$$

and hence $\lambda' = 1 + \rho\lambda - \rho$, as stated. $\square$

The horizon shift operation helps analyze properties of the stationary distribution as agents execute actions in the environment.

Corollary 4. *(Stationary distribution under horizon shift): Let $\tilde{\mathrm{P}}^{gs\prime}$ be the output of the horizon shift operation on $\hat{\mathrm{P}}^{gs}$ as described in* (15)*, and P^{a} be any acceptance matrix. Then the stationary distributions for the transition matrices $\hat{\mathrm{P}}^{tr}$ and $\tilde{\mathrm{P}}^{tr\prime}$ are the same.*

Recall that the recycled stationary distribution $\hat{v}$ is the left eigenvector of the $\hat{\mathrm{P}}^{tr}$ and the conditional probability distribution of action schedules given that an action schedule with first action i^1 is chosen. In Corollary 4 we find that the recycled stationary distribution and the stationary distribution of the approximate next step transition matrix are equivalent. This implies that reaching a steady state solution for the current time step will result in a recycled solution that approximates the steady state distribution for the next time step in the receding time horizon. The primary motivation behind recycling solutions is to reduce the number of iterations required for the simulated annealing step of `HSA` in order to reach the stationary distribution. The distribution of recycled solutions remain close to the stationary distribution after executing a step in the environment and after the horizon shift operation on $\hat{\mathrm{P}}^{gs}$. Furthermore, by focusing the sampling probability on action schedules that deviate distant in the future, geometric sampling aids in the mixing of the Markov chain for future solutions more efficiently. Another interesting property of the horizon shift operation is its effect on the relaxation time t_{rel}.

Corollary 5. *(Relaxation time under horizon shift): Let $\tilde{\mathrm{P}}^{gs\prime}$ be the output of the horizon shift operation on $\hat{\mathrm{P}}^{gs}$ as described in* (15)*, and P^{a} be any acceptance matrix. Let the relaxation time for the transition matrix $\hat{\mathrm{P}}^{tr}$ and $\tilde{\mathrm{P}}^{tr\prime}$ be denoted $\hat{t}_{rel}$ and $\tilde{t}'_{rel}$, respectively. Then*

$$\tilde{t}'_{rel} = \frac{\hat{t}_{rel}}{\rho}.$$

Proof. From Theorem 3, the second largest eigenvalue of $\tilde{\mathrm{P}}^{tr\prime}$ is

$$\lambda'_2 = 1 + \rho\lambda_2 - \rho.$$

Now, using the definition (1) of the relaxation time, we have

$$\tilde{t}'_{\mathrm{rel}} = \frac{1}{1-\lambda'_2} = \frac{1}{1-(1+\rho\lambda_2-\rho)} = \frac{1}{\rho(1-\lambda_2)} = \frac{\hat{t}_{\mathrm{rel}}}{\rho}.$$

$\square$

This is a useful property because the relaxation time yields bounds found in (2), for the mixing times of the Markov chain induced by the transition matrix.

The trade-off for using a smaller ρ is that the mixing time increases, however it does allow for more mixing to take place in future events, which is beneficial because we recycle solutions for future time steps. If ρ is too small, however, the mixing time becomes large and the agent may be unable iterate enough to reach near-steady state.

5.3. Inference-based sampling

Here we focus on the multi-agent aspect of the algorithm by creating recommendations with machine learning. In this section, we aim to create a sampling matrix that is more efficient than the geometric sampling scheme in terms of number of samples necessary to reach a Nash equilibria. To do this we design a process that generates a dataset $\mathcal{D}$ that contains inputs which correspond to images of the environment, and outputs which correspond to real-number values for selecting action schedules.

5.3.1. Creating a dataset

We train a model on the dataset so that the learned policy can provide recommendations for sampling during deployment. We take advantage of the fact that most robotic deployment scenarios are spatial in nature by training our policy to map a local image, $\mathbb{x}_{\alpha,s}$, of the environment to a vector of values that correspond to action schedules that the agent can select $\mathbb{y} \in \mathbb{R}^{|\bar{A}_\alpha|}$, i.e., $\pi : \mathbb{x}_{\alpha,s} \to \mathbb{y}$. The local image $\mathbb{x}_{\alpha,s}$ is translated and rotated with respect to the pose of agent α. We choose to assign values in $\mathbb{y}$ to action schedules according to

$$\mathbb{y}_{\bar{a}_\alpha} = \max_{\bar{a}_{-\alpha} \in \bar{A}_{-\alpha}} u(\bar{a}_\alpha, \bar{a}_{-\alpha}), \tag{17}$$

for all $\bar{a}_\alpha \in \bar{A}_\alpha$ as an incentive to select a joint action schedule that yield high rewards and cooperates with other agents. Particular high rewarding joint actions that require two or more agents to cooperate may have difficulty being selected because they do not exist in any available ‘finite improvement path’ from the current solution. In this case, an agent may have to choose an action that yields less reward in order to escape local maximums. In order to get a set of inputs and outputs, $\mathbb{x}$ and $\mathbb{y}$, we randomize many states and solve for (17) as outlined in Algorithm 3.

5.3.2. Generating sampling matrix $\mathrm{P}^{\pi s}$

After collecting the data we train our learned policy π. We use the softmax function, indicated by $\Psi(\pi(\mathbb{x}, s), i) : \pi(\mathbb{x}, s) \times i \to \mathbb{R}$ as follows

$$\Psi(\pi(\mathbb{x}, s), i) = \frac{e^{\pi(\mathbb{x},s)_i}}{\sum_{\forall j} e^{\pi(\mathbb{x},s)_j}}$$

Algorithm 3: Creating a dataset

```
1  CreateDataSet()
2      $\mathcal{D} = \emptyset$
3      for $n = 1 \to N$:
4          $\mathbb{x}(s)$
5          $\mathbb{y} \in \mathbb{R}^{|\bar{A}_\alpha|}$
6          for $\bar{a}_\alpha \in \bar{A}_\alpha$:
7              $\mathbb{y}_{\bar{a}_\alpha} = 0$
8              for $\bar{a}_{-\alpha} \in \bar{A}_{-\alpha}$:
9                  if $\mathcal{V}_0(\bar{a}_\alpha, \bar{a}_{-\alpha}) - \mathcal{V}_0(\emptyset, \bar{a}_{-\alpha}) > \mathbb{y}_{\bar{a}_\alpha}$
10                     $\mathbb{y}_{\bar{a}_\alpha} = \mathcal{V}_0(\bar{a}_\alpha, \bar{a}_{-\alpha}) - \mathcal{V}_0(\emptyset, \bar{a}_{-\alpha})$
11         $\mathcal{D}.append$(input: $\mathbb{x}$, label: $\mathbb{y}$)
12     return $\mathcal{D}$
```

in order to convert the output values into a probability distribution to be used in the sampling matrix $\mathrm{P}^{\pi s}$

$$\mathrm{P}^{\pi s}_{\alpha,s} = \begin{bmatrix} \Psi_{\pi(\mathbb{x}_{\alpha,s}),1} & \cdot & \cdot & \Psi_{\pi(\mathbb{x}_{\alpha,s}),|\bar{A}_\alpha|} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \Psi_{\pi(\mathbb{x}_{\alpha,s}),1} & \cdot & \cdot & \Psi_{\pi(\mathbb{x}_{\alpha,s}),|\bar{A}_\alpha|} \end{bmatrix} \tag{18}$$

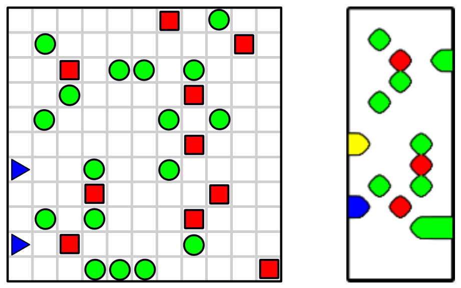
With this sampling matrix, the probability of choosing an action schedule is the same from any other initial action schedule.

Note that, under the inference-based sampling scheme, agents are incentivized to sample action schedules that achieve potentially high 'wonderful life utility'. Thus the recommended distribution of actions is weighted more heavily on action schedules that cooperate with other agents. We argue that this distribution tends to inhibit the probability of breaking promises in the near term, as those action schedules will be sampled less often.

6. Cooperative orienteering

We design an algorithm to test and compare the different generation matrices. In the *cooperative orienteering* 2-D environment, agents are tasked with collecting resources, which may require multiple agents. Resources that require 1 agent yield a reward of 1, and resources that require 2 agents yield a reward of 3. The environment scrolls to the left, but agents always occupy the left-most column where they can choose actions in (up, stay, down). Figure 2 illustrates the cooperative orienteering setup.

We test the following modifications of `HSA` using different generation sampling schemes.



(a) Multi-agent environment (b) $\mathbb{x}_{\alpha,s}$

Figure 2: Cooperative orienteering. (a) shows the cooperative orienteering environment. In this environment, agents, labeled with ▶, are shown on the left column in the environment. Agents move up and down in order to collect resource ●, which requires at least 1 agent on the coordinate that the resource occupies and yields 1 reward. Agents can also collect ■, which requires 2 agents on the coordinate and yields a reward of 3. (b) shows the relative view $\mathbb{x}_{\alpha,s}$ of the agent which is used as an input to the learned policy π. In $\mathbb{x}_{\alpha,s}$, green pixels represent resources which require 1 agent to collect and red pixels represent resources which require 2. The yellow pixel represents the agent α and the blue pixel represents an agent other than α.

$\texttt{HSA}^{f}$: Horizon shift annealing with a 'flat' distribution P^{fs} defined as

$$\mathrm{P}^{fs}_{ij} = \frac{1}{|\bar{A}_\alpha|},$$

for all i and j.

$\texttt{HSA}^{g}$: Horizon shift planning with a geometric distribution P^{gs} as defined in (13). We use $\rho = .25$.

$\texttt{HSA}^{\pi}$: Horizon shift planning with the inferred sampling $\mathrm{P}^{\pi s}$ as defined in (18). We use a random forest classifier with a decision tree depth of 12 with 12 estimators. We convert the environment to local images for each agent as depicted in Figure 2(b). We are able to achieve 82.3% validation accuracy with a validation loss (KL-divergence) of 0.1849 when trained on 10,000 data samples generated from Algorithm 3.

6.1. Single-shift stationary distribution

Here we test the presented algorithms by examining the number of steps that are required in order to reach the stationary distributions defined by the transition probabilities after a shift in the finite time horizon. In this particular test we use one agent. To be more precise, we take a scenario from the cooperative orienteering environment and allow the agent to search for a parameterized number of steps. The agent then executes their active selected action schedule. Once the agent acts, the agent begins searching again. We are particularly interested in this test because it validates our intuition that utilization of recycled action schedules decreases the number of steps required to reach the stationary distribution. Formally, the testing process is outlined in Algorithm 4.

Algorithm 4: Single shift sampling

```
1  $N^k_{\bar{a}_\alpha} = 0$ for all $k \in \{1, \dots, K_2\}$ and $\bar{a}_\alpha \in \bar{A}_\alpha$

2  for $j = 1 \to J$:
3      environment.initialize(seed=j)
4      $\bar{a}_\alpha$ = random.choice($\bar{A}_\alpha$)
5      for $k = 1 \to K_1$:
6          $\bar{a}_\alpha$ = HSA($\bar{a}_\alpha, k$)
7      environment.step($\bar{a}_\alpha$)
8      for $k = 1 \to K_2$
9          $\bar{a}_\alpha$ = HSA($\bar{a}_\alpha, k$)
10         $N^k_{\bar{a}_\alpha} = N^k_{\bar{a}_\alpha} + 1$

11 $\hat{\pi}^k_{\bar{a}_\alpha} = \frac{N^k_{\bar{a}_\alpha}}{J}$ for all $k \in \{1, \dots, K_2\}$ and $\bar{a}_\alpha \in \bar{A}_\alpha$
12 return $\hat{\pi}$
```

The stationary distribution v is determined by calculating the transition probabilities determined with generation and acceptance probabilities. We calculate the KL-divergence between v and the empirically found distribution $\hat{v}$ for every iteration k as follows

$$L^k_{\hat{v}^k, v} = -\sum_{i=1}^{|\bar{A}_\alpha|} v^k_i \log \hat{v}_i + \sum_{i=1}^{|\bar{A}_\alpha|} v^k_i \log v_i. \tag{19}$$

The state of the environment is initially the same for each sample. Agents iterate through HSA for $k \in \{1, \dots, K_1\}$ steps and then select an action. The sub-index k in $\texttt{HSA}^f_k$, $\texttt{HSA}^g_k$, $\texttt{HSA}^\pi_k$ indicates the k number of iterations used on the first time step of the corresponding algorithm. We take samples such that the agent chooses to move straight and discard the rest. After executing the step in the environment, the stationary distribution of the agent's next choice is determined. Figure 3 illustrates this stationary distribution after the shift. The agent then plans for $k \in \{1, \dots, K_2\}$ iterations and the empirical distribution is determined with respect to k. Figure 4(a) shows the resulting KL-divergence between v and $\hat{v}$. We show in Figure 4(b) the KL-divergence vs. values of ρ, 0 through 1. If the user chooses a ρ that is too low (e.g., $\rho < .1$), $\texttt{HSA}^g$ reaches the stationary distribution slower than $\texttt{HSA}^f$, likely because not enough sampling occurs in near time. We leave finding the optimal value of ρ for future work as it is likely determined by characteristics of the environment, K_1, and K_2. We also examine in Figure 5 the average expected reward of the action schedule selected during the second step. .

Note that the stationary distributions are slightly different for each of the sampling schemes and that (19) is determined with each of the schemes' stationary distributions, respectively. As expected, the empirical distributions converge

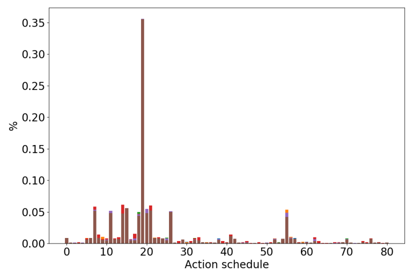
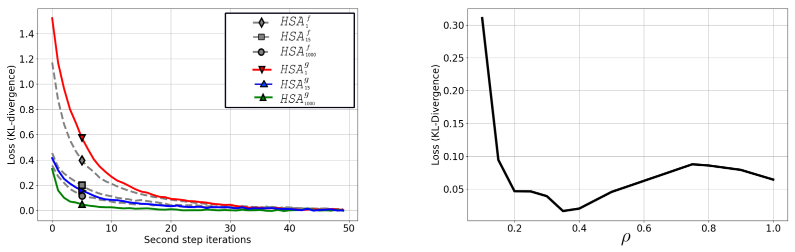


Figure 3: The average reward per step is shown between the stationary distribution and empirical distribution in the single shift experiment. The y-axis indicates probabilities in the stationary distribution for the corresponding action schedules that are indexed 0 through 80 on the x-axis.



(a)

(b)

Figure 4: The KL-divergence is shown for the single shift experiment. On the left (a), $\texttt{HSA}^f$ and $\texttt{HSA}^g$ are both ran initially for 1, 15, and 1000 first step iterations. The x-axis indicates number of iterations during the second step and the y-axis indicates the KL-divergence between the stationary distribution and the empirical distribution during the second step. On the right (b), we show the KL-divergence on the y-axis vs. values of $\rho \in (0, 1)$, where $K_1 = 20$ and $K_2 = 10$.

to the stationary distributions as the number of iterations in the second step increase. Also, as the first step iterations k_1 increase, the number of iterations k_2 required for the KL-divergence to converge in the second step is smaller for both sampling schemes. The faster convergence implies that the Markov chain induced by the transition matrix at the second step is being mixed partially by the first step, which backs up the idea of recycling solutions. When comparing $\texttt{HSA}^f$ and $\texttt{HSA}^g$, it is notable that $\texttt{HSA}^g$ performs preferably as k_1 increases. Intuitively, this is because the agent will dedicate more first step iterations for sampling action schedules in the more distant future. When k_1 is low, the second step will not be mixed well and the mixing time required for $\texttt{HSA}^g$ is greater since its relaxation time is greater.

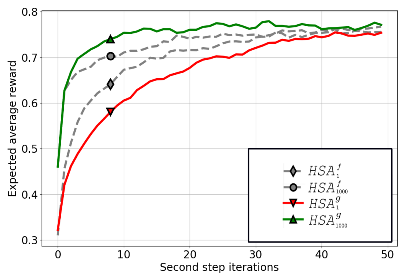


Figure 5: The expected reward is shown for the single shift experiment. The x-axis indicates number of iterations during the second step and the y-axis indicates the expected reward for action schedules chosen during the second step.

6.2. *Probability of finding globally optimal solutions*

For this experiment we are interested in determining how often 2 agents arrive at a joint action schedule $\bar{a}$ that is in the set of globally optimal joint action schedules when considering a single step (no horizon shift) in the environment. Because we are not considering horizon shifts, we omit $\texttt{HSA}^g$, which converges slower than $\texttt{HSA}^f$ if there is no 'pre-shift' mixing.

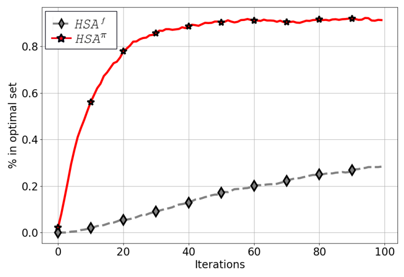
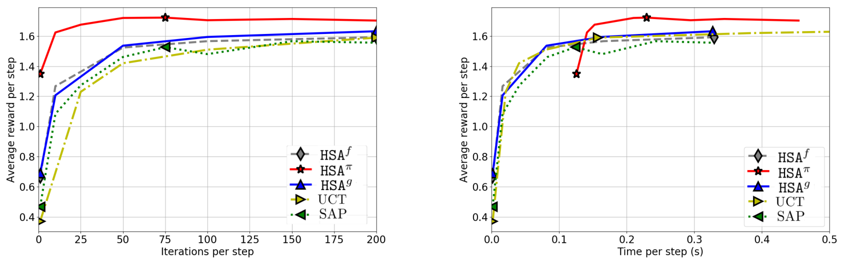


Figure 6: The percentage of joint action schedules that are in the set of optimal Nash equilibriums are shown. The y-axis indicates the percentage of trials where the joint action schedule selected at iteration k, on the x-axis, was in the set of joint Nash equilibriums.

As shown in Figure 6, the probability that the current joint action schedule is in the set of optimal Nash equilibriums increases with the number of iterations. We see that $\texttt{HSA}^\pi$ yields a higher probability. This is because the learned policy is trained to output action schedules with high utility more often as determined in (17) and Algorithm 3.

6.3. *Full trial cooperative reward*

We determine the average reward per step that 2 agents receive when agents use the algorithms on cooperative orienteering for 1000 time steps with a receding time horizon of $T = 4$. In this experiment, we vary the number K of iterations per time steps that agents take during the simulation and plot the results in Figure 7(a). Additionally, we implement centralized UCT and SAP to compare our results against. In centralized UCT, we search over a joint action space up to a finite horizon of 4 time steps. In SAP, agents plan distributively and sample task sequences. Because $\texttt{HSA}^{\pi}$ requires inference from π at every



(a) (b)

Figure 7: Average reward per step for full trials versus (a) allowed iterations per time step and (b) allowed time per time step. The y-axis indicates the average expected reward per step. The x-axis indicates the (a) number of iterations and (b) time in seconds allowed per time step.

time step, we also plot the results with respect to the amount of time that agents use for each step in Figure 7(b).

As shown in Figure 7, the average reward gained per step increases with the number of iterations. As suspected, $\texttt{HSA}^{g}$ performs worse than $\texttt{HSA}^{f}$ for low number of iterations per step. As the number of iterations increase, $\texttt{HSA}^{g}$ outperforms $\texttt{HSA}^{f}$, which may be a consequence of its synergy with recycling solutions and its ability to mix future actions more efficiently. $\texttt{HSA}^{\pi}$ performs the best, likely because it sampled better action schedules more often due to the high categorical accuracy of the model. $\texttt{HSA}^{\pi}$ does take some time for inference however, and initially performs worse than the other algorithms when considering real time per step. In this experiment, UCT requires more iterations, but each iteration generally takes less time. In the cooperative orienteering environment UCT yields similar performance to $\texttt{HSA}^{f}$ and $\texttt{HSA}^{g}$. Given the cost of inference, $\texttt{HSA}^{\pi}$ requires more time per step but is able to overtake UCT in average reward per step in a small number of iterations. As expected, SAP performs most like $\texttt{HSA}^{f}$. SAP performs slightly worse than $\texttt{HSA}^{f}$ because it does not utilize recycling of solutions.

6.4. Keeping and breaking promises

Lastly, we design a metric for the notion of keeping and breaking ‘promises’. Given a deployment with two or more agents, we say that there was a ‘promise’ broken during a time step if the following are all true:

- During iterations of simulated annealing, an agent expects to successfully collect a resource that requires more than one agent;

- Subsequently, another agent who was required for that resource's collection chooses a different action, resulting in the first agent's inability to collect the resource.

In this experiment, we run the algorithms on the cooperative orienteering environment with 2 agents for N time steps and determine the probability of steps where a promise was broken between the two agents.

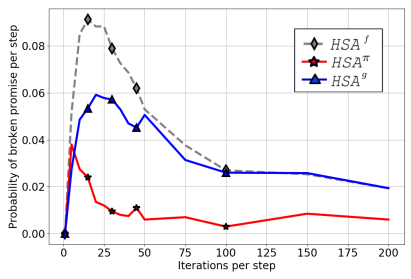


Figure 8: 1000 trials of cooperative orienteering with $N = 1000$ time steps where the probability of steps where 'promises' are broken is depicted by the y-axis and the number of iterations per step is depicted by the x-axis.

Figure 8 shows that the probability of breaking promises decreases as the number of iterations increase for all algorithms. As expected, promises are kept more often under both the geometric and inference-based sampling schemes when compared to the flat sampling scheme.

7. Conclusions

We have considered multi-agent task planning problems where individuals must coordinate their actions to cooperatively solve a set of tasks. We have formulated the selection of joint agents' actions as a potential game and, instead of relying on methods based on improvement paths to reach a pure Nash equilibrium, we have introduced a decentralized simulated annealing process. Agents implement the strategy in a receding horizon fashion, recycling the solutions from one time step to the next, and sample new actions with a probability determined by two novel schemes termed geometric and inference-based sampling. In geometric sampling, actions are sampled based on their position in the finite time horizon, with new actions having larger probability as the considered timestep is farther into the future. In inference-based sampling, we rely on the recommendations provided by a learned model to output sampling probabilities. We show that the proposed algorithm, given enough time, will find at least a local pure Nash equilibrium and analyze the properties of geometric sampling with respect to its stationary distribution as the finite time horizon shifts. Future work will develop methods to optimize the rate of increase with the number of iterations in the sampling probability in the geometric scheme.

We also plan to explore the improvement of the ability of the learned policy to influence agents to reach an optimal Nash equilibrium by training the output conditioned on the current solution of the agents.

Acknowledgments

This work was supported by ONR Award N00014-16-1-2836.

References

[1] F. Bullo, J. Cortés, S. Martinez, Distributed Control of Robotic Networks, Applied Mathematics Series, Princeton University Press, 2009.

[2] M. Mesbahi, M. Egerstedt, Graph Theoretic Methods in Multiagent Networks, Applied Mathematics Series, Princeton University Press, 2010.

[3] M. Dunbabin, L. Marques, Robots for environmental monitoring: Significant advancements and applications, IEEE Robotics & Automation Magazine 19 (1) (2012) 24–39.

[4] J. Das, F. Py, J. B. J. Harvey, J. P. Ryan, A. Gellene, R. Graham, D. A. Caron, K. Rajan, G. S. Sukhatme, Data-driven robotic sampling for marine ecosystem monitoring, The International Journal of Robotics Research 34 (12) (2015) 1435–1452.

[5] J. Cortés, M. Egerstedt, Coordinated control of multi-robot systems: A survey, SICE Journal of Control, Measurement, and System Integration 10 (6) (2017) 495–503.

[6] R. Sutton, D. Precup, S. Singh, Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning, Artificial Intelligence 112 (1-2) (1999) 181–211.

[7] F. Broz, I. Nourbakhsh, R. Simmons, Planning for human-robot interaction using time-state aggregated POMDPs, in: AAAI, Vol. 8, 2008, pp. 1339–1344.

[8] M. L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, Wiley, New York, 2014.

[9] R. A. Howard, Dynamic Programming and Markov Processes, MIT Press, Cambridge, MA, 1960.

[10] R. E. Parr, S. Russell, Hierarchical control and learning for Markov decision processes, University of California, Berkeley Berkeley, CA, 1998.

[11] D. P. Bertsekas, Dynamic Programming and Optimal Control, Athena Scientific, 1995.

[12] L. Kocsis, C. Szepesvári, Bandit based Monte-Carlo planning, in: ECML, Vol. 6, Springer, New York, 2006, pp. 282–293.

[13] J. Co-Reyes, Y. Liu, A. Gupta, B. Eysenbach, P. Abbeel, S. Levine, Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings, arXiv preprint arXiv:1806.02813 (2018).

[14] A. Ma, M. Ouimet, J. Cortés, Hierarchical reinforcement learning via dynamic subspace search for multi-agent planning, Autonomous Robots 44 (3-4) (2020) 485–503.

[15] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. V. D. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of Go with deep neural networks and tree search, Nature 529 (7587) (2016) 484.

[16] X. Guo, S. Singh, H. Lee, R. Lewis, X. Wang, Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning, in: Conference on Neural Information Processing Systems, Montreal, Canada, 2014, pp. 3338–3346.

[17] S. Gelly, D. Silver, Combining online and offline knowledge in UCT, in: Proceedings of the 24th International Conference on Machine Learning, Corvallis, OR, 2007, pp. 273–280.

[18] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, D. Silver, Mastering Atari, Go, Chess and Shogi by planning with a learned model, arXiv preprint arXiv:1911.08265 (2019).

[19] T. Anthony, Z. Tian, D. Barber, Thinking fast and slow with deep learning and tree search, in: Conference on Neural Information Processing Systems, Long Beach, CA, 2017, pp. 5360–5370.

[20] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al., Mastering the game of Go without human knowledge, Nature 550 (7676) (2017) 354.

[21] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al., Mastering atari, go, chess and shogi by planning with a learned model, arXiv preprint arXiv:1911.08265 (2019).

[22] D. Fudenberg, J. Tirole, Game Theory, MIT Press, Cambridge, MA, 1991.

[23] D. Fudenberg, D. K. Levine, The Theory of Learning in Games, MIT Press, Cambridge, MA, 1998.

[24] D. Monderer, L. S. Shapley, Potential games, Games and Economic Behavior 14 (1996) 124–143.

[25] J. R. Marden, G. Arslan, J. S. Shamma, Cooperative control and potential games, IEEE Transactions on Systems, Man & Cybernetics. Part B: Cybernetics 39 (2009) 1393–1407.

[26] A. C. Chapman, R. A. Micillo, R. Kota, N. R. Jennings, Decentralised dynamic task allocation: a practical game: theoretic approach, in: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2, International Foundation for Autonomous Agents and Multiagent Systems, 2009, pp. 915–922.

[27] L. Blume, et al., The statistical mechanics of strategic interaction, Games and Economic Behavior 5 (3) (1993) 387–424.

[28] H. P. Young, Individual Strategy and Social Structure: an Evolutionary Theory of Institutions, Princeton University Press, Princeton, NJ, 1998.

[29] S. Kirkpatrick, C. Gelatt, M. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.

[30] P. J. M. V. Laarhoven, E. H. L. Aarts, Simulated annealing, in: Simulated annealing: Theory and applications, Vol. 37 of Mathematics and Its Applications, Springer, New York, 1987, pp. 7–15.

[31] M. Malek, M. Guruswamy, M. Pandya, H. Owens, Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem, Annals of Operations Research 21 (1) (1989) 59–84.

[32] B. Hajek, Cooling schedules for optimal annealing, Mathematics of Operations Research 13 (2) (1988) 311–329.

[33] B. Suman, P. Kumar, A survey of simulated annealing as a tool for single and multiobjective optimization, Journal of the Operational Research Society 57 (10) (2006) 1143–1160.

[34] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, et al., Alphastar: Mastering the real-time strategy game Starcraft II, DeepMind blog (2019) 2.

[35] D. A. Levin, Y. Peres, Markov Chains and Mixing Times, Vol. 107, American Mathematical Society, Providence, RI, 2017.

[36] D. Henderson, S. H. Jacobson, A. W. Johnson, The theory and practice of simulated annealing, in: Handbook of metaheuristics, Springer, 2003, pp. 287–319.