Adaptive Anytime Multi-Agent Path Finding using Bandit-Based Large Neighborhood Search

Thomy Phan[1], **Taoan Huang**[1], **Bistra Dilkina**[1], **Sven Koenig**[1]

[1]University of Southern California, USA
{thomy.phan, taoanhua, dilkina, skoenig}@usc.edu

Abstract

Anytime *multi-agent path finding (MAPF)* is a promising approach to scalable path optimization in large-scale multi-agent systems. State-of-the-art anytime MAPF is based on *Large Neighborhood Search (LNS)*, where a fast initial solution is iteratively optimized by destroying and repairing a fixed number of parts, i.e., the neighborhood, of the solution, using randomized destroy heuristics and prioritized planning. Despite their recent success in various MAPF instances, current LNS-based approaches lack exploration and flexibility due to greedy optimization with a fixed neighborhood size which can lead to low quality solutions in general. So far, these limitations have been addressed with extensive prior effort in tuning or offline machine learning beyond actual planning. In this paper, we focus on online learning in LNS and propose *Bandit-based Adaptive LArge Neighborhood search Combined with Exploration (BALANCE)*. BALANCE uses a bi-level multi-armed bandit scheme to adapt the selection of destroy heuristics and neighborhood sizes on the fly during search. We evaluate BALANCE on multiple maps from the MAPF benchmark set and empirically demonstrate performance improvements of at least 50% compared to state-of-the-art anytime MAPF in large-scale scenarios. We find that Thompson Sampling performs particularly well compared to alternative multi-armed bandit algorithms.

1 Introduction

A wide range of real-world applications like goods transportation in warehouses, search and rescue missions, and traffic management can be formulated as *Multi-Agent Path Finding (MAPF)* problem, where the goal is to find collision-free paths for multiple agents with each having an assigned start and goal location. Finding optimal solutions w.r.t. minimal flowtime or makespan is NP-hard, which limits scalability of most state-of-the-art MAPF solvers (Ratner and Warmuth 1986; Yu and LaValle 2013; Sharon et al. 2012).

Anytime MAPF based on *Large Neighborhood Search (LNS)* is a popular approach to finding fast and near optimal solutions to the MAPF problem within a fixed time budget (Li et al. 2021). Given an initial feasible solution and a set of destroy heuristics, LNS iteratively destroys and replans so-called neighborhoods of the solution, i.e., a fixed number of paths, until the time budget runs out. MAPF-LNS represents the current state-of-the-art in anytime MAPF and has been experimentally shown to scale up to large-scale scenarios with hundreds of agents (Li et al. 2021). Due to its increasing popularity, several extensions have been recently proposed like fast local repairing, integration of primal heuristics, or machine learning guided neighborhood selection (Huang et al. 2022; Li et al. 2022; Lam et al. 2023).

However, MAPF-LNS and its variants currently suffer from two limitations that can lead to low quality solutions in general:

1. The neighborhood size is typically fixed which limits flexibility of the optimization process thus possibly affecting the solution quality, especially for a large number of agents (Li et al. 2021). Therefore, prior tuning is required – in addition to the actual LNS procedure – to obtain good solutions.
2. Roulette wheel selection is commonly used to execute and adapt the destroy heuristic selection to determine the neighborhood (Mara et al. 2022; Li et al. 2021). During optimization, roulette wheel selection could greedily converge to poor choices due to lacking exploration. Offline machine learning can guide the selection with solution score prediction but requires sufficient data acquisition and feature engineering (Huang et al. 2022).

In this paper, we address these limitations by proposing *Bandit-based Adaptive LArge Neighborhood search Combined with Exploration (BALANCE)*. BALANCE uses a bi-level multi-armed bandit scheme to adapt the selection of destroy heuristics and neighborhood sizes on the fly during search. Our contributions are as follows:

- We formulate BALANCE as a simple but effective MAPF-LNS framework with adaptive selection of destroy heuristics and neighborhood sizes during search.
- We propose and discuss three concrete instantiations of BALANCE based on roulette wheel selection, UCB1, and Thompson Sampling respectively.
- We evaluate BALANCE on multiple maps from the MAPF benchmark set and empirically demonstrate cost improvements of at least 50% compared to state-of-the-art anytime MAPF in large-scale scenarios. We find that Thompson Sampling performs particularly well compared to alternative multi-armed bandit algorithms.

2 Background

2.1 Multi-Agent Path Finding (MAPF)

We focus on *maps* as undirected unweighted *graphs* $G = \langle \mathcal{V}, \mathcal{E} \rangle$, where vertex set $\mathcal{V}$ contains all possible locations and edge set $\mathcal{E}$ contains all possible transitions or movements between adjacent locations. An *instance* I consists of a map G and a set of *agents* $\mathcal{A} = \{a_1, ..., a_m\}$ with each agent a_i having a *start location* $s_i \in \mathcal{V}$ and a *goal location* $g_i \in \mathcal{V}$.

MAPF aims to find a collision-free plan for all agents. A *plan* $P = \{p_1, ..., p_m\}$ consists of individual paths $p_i = \langle p_{i,1}, ..., p_{i,l(p_i)} \rangle$ per agent a_i, where $\langle p_{i,t}, p_{i,t+1} \rangle = \langle p_{i,t+1}, p_{i,t} \rangle \in \mathcal{E}$, $p_{i,1} = s_i$, $p_{i,l(p_i)} = g_i$, and $l(p_i)$ is the *length* or *travel distance* of path p_i. The *delay* $del(p_i)$ of path p_i is defined by the difference of path length $l(p_i)$ and the length of the shortest path from s_i to g_i w.r.t. map G.

In this paper, we consider *vertex conlicts* $\langle a_i, a_j, v, t \rangle$ that occur when two agents a_i and a_j occupy the same location $v \in \mathcal{V}$ at time step t and *edge conflicts* $\langle a_i, a_j, u, v, t \rangle$ that occur when two agents a_i and a_j traverse the same edge $\langle u, v \rangle \in \mathcal{E}$ in opposite directions at time step t (Stern et al. 2019). A plan P is a *solution*, i.e., *feasible*, when it does not have any vertex or edge conflicts. Our goal is to find a solution that minimizes the *flowtime* $\sum_{p \in P} l(p)$ which is equivalent to minimizing the *sum of delays* $\sum_{p \in P} del(p)$. We use the sum of delays or *(total) cost* $c(P) = \sum_{p \in P} del(p)$ of plan P as primary performance measure in our evaluations.

2.2 Anytime MAPF with LNS

Anytime MAPF searches for solutions within a given time budget. The solution quality expectedly improves with increasing time budget (Cohen et al. 2018; Li et al. 2021).

MAPF-LNS based on *Large Neighborhood Search (LNS)* is the current state-of-the-art approach to anytime MAPF and is shown to scale up to large-scale scenarios with hundreds of agents (Huang et al. 2022; Li et al. 2021). Starting with an initial feasible plan P, e.g., found via *prioritized planning (PP)* from (Silver 2005), MAPF-LNS iteratively modifies P by destroying $N < m$ paths, i.e., the *neighborhood* $P^- \subset P$. The destroyed neighborhood is then repaired or replanned using PP to quickly generate a new solution P^+. If the new cost $c(P^+)$ is lower than the previous cost $c(P)$, then P is replaced by P^+ and the search continues until the time budget runs out. The result of MAPF-LNS is the last accepted solution P with the lowest cost so far.

MAPF-LNS uses a set $\mathcal{H}$ of three *destroy heuristics* $H \in \mathcal{H}$, namely a *random uniform selection* of N paths, an *agent-based heuristic*, and a *map-based heuristic* (Li et al. 2021). The agent-based heuristic generates the neighborhood including the path of agent a_i with the current maximum delay and other paths (determined via random walks) that prevent a_i from achieving a lower delay. The map-based heuristic randomly chooses a vertex $v \in \mathcal{V}$ with a degree greater than 2 and generates a neighborhood of paths containing v.

MAPF-LNS uses a *selection algorithm* π like roulette wheel selection to choose destroy heuristics $H \in \mathcal{H}$ by maintaining updatable *weights* or some *statistics* for all destroy heuristics (Ropke and Pisinger 2006; Li et al. 2021). All weights or statistics used by π to select a destroy heuristic H are denoted by Δ which could represent, e.g., the average cost improvement or the selection count per destroy heuristic H. The statistics Δ will be further explained in Section 4.2 as the concrete definition depends on π.

2.3 Multi-Armed Bandits

Multi-armed bandits (MABs) or simply bandits are fundamental decision making problems, where a *MAB or selection algorithm* π repeatedly chooses an *arm* k among a given set of arms or *options* $\{1, ..., K\}$ to maximize an expected *reward* of a stochastic reward function $\mathcal{R}(k) := X_k$, where X_k is a random variable with an unknown distribution f_{X_k}. To solve a MAB, one has to determine an *optimal arm* k^*, which maximizes the expected reward $\mathbb{E}\big[X_k\big]$. The MAB algorithm π has to balance between sufficiently exploring all arms k to accurately estimate $\mathbb{E}\big[X_k\big]$ via statistics Δ and to exploit its current estimates by greedily selecting the arm k with the currently highest estimate of $\mathbb{E}\big[X_k\big]$. This is known as the *exploration-exploitation dilemma*, where exploration can lead to arms with high expected rewards but requires more time for trying them out, while exploitation can lead to fast convergence but possibly gets stuck in a poor local optimum. In this paper, we will cover *roulette wheel selection*, *UCB1*, and *Thompson Sampling* as common MAB algorithms and further explain them in Section 4.2.

3 Related Work

3.1 Multi-Armed Bandits for LNS

In recent years, MABs have been used as adaptive meta-controllers to tune learning and optimization algorithms on the fly (Schaul et al. 2019; Badia et al. 2020; Hendel 2022). Beside roulette wheel selection, UCB1 and ϵ-greedy are commonly used for destroy heuristic selection in LNS in the context of mixed integer programming, vehicle routing, and scheduling problems with fixed neighborhood sizes (Chen et al. 2016; Chen and Bai 2018; Chmiela et al. 2023). (Hendel 2022) adapts the neighborhood size for mixed integer programming using a mutation-based approach inspired by evolutionary algorithms (Rothberg 2007). Most works use rather complex rewards that are composed of multiple weighted terms with several tunable hyperparameters. We focus on *MAPF problems* and propose a *bi-level* MAB scheme to adapt the selection of destroy heuristics and neighborhood sizes, which is simple to use without requiring any additional mechanisms like mutation. Our approach uses the *cost improvement* as reward, which simply represents the cost difference between two solutions w.r.t. the original objective of MAPF without depending on any additional weighted term that requires prior tuning. To the best of our knowledge, our work first effectively applies *Thompson Sampling* to anytime MAPF in addition to more common MAB algorithms like UCB1 and roulette wheel selection.

3.2 Multi-Armed Bandits in Anytime Planning

MABs are popular in anytime planning algorithms, especially in single-agent Monte Carlo planning (Kocsis and Szepesvári 2006; Silver and Veness 2010). *Monte-Carlo*

Tree Search (MCTS) is the state-of-the-art framework of current Monte Carlo planning algorithms which uses MABs to traverse a search tree within a limited time budget (Kocsis and Szepesvári 2006; Silver and Veness 2010). UCB1 is most commonly used but Thompson Sampling has also gained attention in the last few years due to its effectiveness in domains of high uncertainty (Bai, Wu, and Chen 2013; Bai et al. 2014; Phan et al. 2019a,b). As MABs have been shown to converge to good decisions within short time budgets, we use MABs in our adaptive *multi-agent path finding* setting. Inspired by latest progress in Monte Carlo planning (Świechowski et al. 2023), we intend to employ more sophisticated MAB algorithms like Thompson Sampling to anytime MAPF to improve exploration and performance.

3.3 Machine Learning in Anytime MAPF

Machine learning has been used in MAPF to directly learn collision-free path finding, to guide node selection in search trees, or to select appropriate MAPF algorithms for certain maps (Sartoretti et al. 2019; Kaduri, Boyarski, and Stern 2020; Huang, Dilkina, and Koenig 2021). MAPF-ML-LNS is an anytime MAPF approach that extends MAPF-LNS with a learned score prediction for neighborhood selection as well as a random uniform selection of the neighborhood size N. The predictor is trained offline on pre-collected data from previous MAPF runs (Huang et al. 2022). While the score prediction generalizes to some degree, the predictor is fixed after training therefore not being able to adapt during search, which limits flexibility. MAPF-ML-LNS depends on extensive prior effort like data acquisition, model training, and feature engineering for meaningful score learning. We propose an *online learning* approach to adaptive MAPF-LNS using MABs. The MABs can be trained on the fly with data *directly* obtained from the LNS without any prior data acquisition. Since MABs only learn from *scalar rewards*, there is no need for expensive feature engineering, simplifying our approach and easing application to other domains.

4 Bandit-Based Adaptive MAPF-LNS

We now introduce *Bandit-based Adaptive LArge Neighborhood search Combined with Exploration (BALANCE)* as a simple but effective LNS framework for adaptive MAPF.

4.1 Formulation

BALANCE uses a bi-level MAB scheme to adapt the selection of destroy heuristics and neighborhood sizes on the fly during search. The first level consists of a single MAB, called *$\mathcal{H}$-Bandit* with $K = |\mathcal{H}|$ arms, which selects a destroy heuristic $H \in \mathcal{H}$. The second level consists of $|\mathcal{H}|$ so-called *$\mathcal{N}$-Bandits* with $K = E$ arms. Each $\mathcal{N}$-Bandit conditions on a destroy heuristic choice $H \in \mathcal{H}$ and determines the corresponding neighborhood size $N \in \mathcal{N} = \{2^e | e \in \{1, ..., E\}\}$[1] based on an exponent selection $e \in \{1, ..., E\}$. The bi-level MAB scheme is shown in Figure 1.

BALANCE first selects a destroy heuristic H with the top-level $\mathcal{H}$-Bandit based on its current statistics $\Delta_{\mathcal{H}}$. The selected destroy heuristic H determines the corresponding bottom-level $\mathcal{N}$-Bandit, which is used to select an exponent e based on its current conditional statistics $\Delta_H^{\mathcal{N}}$. The neighborhood size is then determined by $N = 2^e$. After evaluating the total cost $c(P^+)$ of the new solution P^+, i.e. the sum of delays, the statistics of the top-level $\mathcal{H}$-Bandit and corresponding bottom-level $\mathcal{N}$-Bandit are updated incrementally. The *MAB reward* $x_k = x_{\langle H,N \rangle} = \mathit{max}\{0, c(P) - c(P^+)\}$ for the update is defined by the *cost improvement* of the new solution P^+ compared to the previous one P (Li et al. 2021).

[1]The set of neighborhood size options $\mathcal{N}$ can be defined arbitrarily. For simplicity, we focus on sets consisting of powers of two.

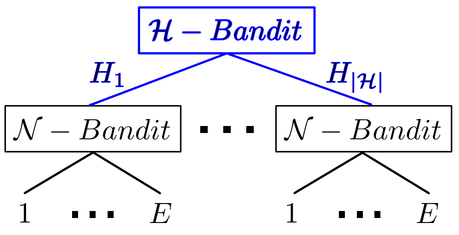

Figure 1: Bi-level multi-armed bandit scheme of BALANCE. The top-level $\mathcal{H}$-Bandit selects a destroy heuristic $H \in \mathcal{H}$. Each bottom-level $\mathcal{N}$-Bandit corresponds to a destroy heuristic choice and selects an exponent $e \in \mathcal{N} = \{1, ..., E\}$ to determine the neighborhood size $N = 2^e$.

The full formulation of BALANCE is provided in Algorithm 1, where I represents the instance to be solved, π represents the MAB algorithm for the bi-level scheme, and E represents the number of neighborhood size options.

Algorithm 1: BALANCE as MAPF-LNS Framework

```
1:  procedure BALANCE($I, \pi, E$)
2:      $P = \{p_1, ..., p_m\} \leftarrow RunInitialSolver(I)$
3:      Initialize statistics $\Delta$ of MAB algorithm $\pi$
4:      while runtime limit not exceeded do
5:          $\langle H, N \rangle \sim BiLevelBanditSelection(\pi, \Delta, E)$
6:          $A \sim SampleNeighborhood(I, H, N)$
7:          $P^- \leftarrow \{p_i | a_i \in A\}$
8:          $P^+ \leftarrow RunRepairSolver(I, A, P \backslash P^-)$
9:          $x_{\langle H,N \rangle} \leftarrow max\{0, c(P) - c(P^+)\}$
10:         if $x_{\langle H,N \rangle} > 0$ then    ▷ check cost improvement
11:             $P \leftarrow (P \backslash P^-) \cup P^+$
12:         $UpdateBandits(\Delta, H, N, x_{\langle H,N \rangle})$
        return $P$
```

4.2 Instantiations

In the following, we describe three concrete MAB algorithms π to implement the bi-level scheme in Figure 1. The definition of the statistics Δ depends on the MAB algorithm.

Roulette Wheel Selection π selects an arm k with a probability of $\frac{w_k}{\sum_{j=1}^{K} w_j}$, where $w_k = \sum_{c=1}^{T_k} x_k^{(c)}$ is the *sum of rewards* or *weight* and T_k is the *selection count* of arm k. Statistics Δ consists of all weights w_k, which can be updated incrementally after each iteration (Goldberg 1988).

UCB1 π selects arms by maximizing the *upper confidence bound* of rewards $\mathit{UCB1}(k) = \overline{x}_k + \xi\sqrt{\frac{\log(T)}{T_k}}$, where $\overline{x}_k$ is the *average reward* of arm k, ξ is an exploration constant, T is the total number of arm selections, and T_k is the selection count of arm k. The second term represents the *exploration bonus*, which becomes smaller with increasing T_k (Auer, Cesa-Bianchi, and Fischer 2002). Statistics Δ consists of all average rewards $\overline{x}_k$ and selection counts T_k.

Thompson Sampling π uses a Bayesian approach to balance between exploration and exploitation of arms (Thompson 1933). We focus on a generalized variant of Thompson Sampling, which works for arbitrary reward distributions f_{X_k} by assuming that X_k follows a Normal distribution $\mathcal{N}(\mu_k, \frac{1}{\tau_k})$ with unknown mean μ_k and precision $\tau_k = \frac{1}{\sigma_k^2}$, where σ_k^2 is the variance (Bai, Wu, and Chen 2013; Bai et al. 2014). $\langle \mu_k, \tau_k \rangle$ follows a Normal Gamma distribution $\mathcal{NG}(\mu_0, \lambda_k, \alpha_k, \beta_k)$ with $\lambda_k > 0$, $\alpha_k \geq 1$, and $\beta_k \geq 0$. The distribution over τ_k is a Gamma distribution $\tau_k \sim \mathit{Gamma}(\alpha_k, \beta_k)$ and the conditional distribution over μ_k given τ_k is a Normal distribution $\mu_k \sim \mathcal{N}(\mu_0, \frac{1}{\lambda_k \tau_k})$. Given a *prior distribution* $P(\theta) = \mathcal{NG}(\mu_0, \lambda_0, \alpha_0, \beta_0)$ and n observed rewards $D_k = \{x_1^{(1)}, ..., x_k^{(T_k)}\}$, the *posterior distribution* is defined by $P(\theta|D_k) = \mathcal{NG}(\mu_{k,1}, \lambda_{k,1}, \alpha_{k,1}, \beta_{k,1})$, where $\mu_{k,1} = \frac{\lambda_0 \mu_0 + T_k \overline{x}_k}{\lambda_0 + T_k}$, $\lambda_{k,1} = \lambda_0 + T_k$, $\alpha_{k,1} = \alpha_0 + \frac{T_k}{2}$, and $\beta_{k,1} = \beta_0 + \frac{1}{2}(T_k \sigma_k^2 + \frac{\lambda_0 T_k (\overline{x}_k - \mu_0)^2}{\lambda_0 + T_k})$. $\overline{x}_k$ is the observed average reward in D_k and $\sigma_k^2 = \frac{1}{T_k}\sum_{c=1}^{T_k}(x_k^{(c)} - \overline{x}_k)^2$ is the variance. The posterior is inferred for each arm k to sample an estimate μ_k of the expected reward $\mathbb{E}\big[X_k\big]$. The arm with the highest estimate is selected. Statistics Δ consists of all average rewards $\overline{x}_k$, *average of squared rewards* $\frac{1}{T_k}\sum_{c=1}^{T_k}(x_k^{(c)})^2$, and selection counts T_k.

4.3 Conceptual Discussion

As MAB algorithms balance between exploration and exploitation to quickly find optimal choices, we believe that they are naturally suited to enhance MAPF-LNS with self-adaptive capabilities. According to previous work on MAB-based tree search, BALANCE can provably converge to an optimal destroy heuristic and neighborhood size choice with sufficient exploration, if there is a *stationary optimum* (Kocsis and Szepesvári 2006; Bai et al. 2014). Otherwise, non-stationary MAB techniques are required which we defer to as future work (Garivier and Moulines 2008). Depending on the choice of E, BALANCE maintains $E+1$ MABs in total. Since Δ can be updated incrementally for any quantity like arm selection counts T_k or average rewards $\overline{x}_k$, the bi-level MAB scheme can be updated in constant time thus introducing negligible overhead to the LNS (as replanning of neighborhoods requires significantly more compute in general).

Roulette wheel selection is the simplest method to implement because it only uses the weights w_k as sum of rewards. However, it could lack exploration in the long run, since arms with small weights are likely to be neglected or forgotten over time. UCB1 accommodates for this issue by introducing an exploration bonus that explicitly considers the selection count of arms T_k. Arms that are selected less over time, will have a larger exploration bonus and are therefore more incentivized for selection, depending on the choice of exploration constant ξ. Thompson Sampling is a randomized algorithm whose initial exploration depends on prior parameters, i.e., μ_0, λ_0, α_0, and β_0 thus being more complex than the other MAB approaches. However, previous work reports that using prior distributions that are close to a *uniform distribution* is sufficient in most cases without requiring extensive tuning (Bai, Wu, and Chen 2013; Bai et al. 2014).

Adaptation in MAPF-LNS can be regarded as *stochastic optimization problem*, since all destroy heuristics defined by (Li et al. 2021) are randomized. Therefore, uncertainty-based methods like Thompson Sampling seem promising for this setting as reported in (Chapelle and Li 2011; Kaufmann, Korda, and Munos 2012; Bai et al. 2014).

Alternatively to the bi-level MAB scheme proposed in this work, a single MAB could be employed to directly search the joint arm space of $\mathcal{H} \times \mathcal{N}$. While this approach would solve the same problem from a statistical perspective, the joint arm space scales quadratically which could lead to low quality solutions, if the time budget is very restricted. The bi-level scheme mitigates the scalability issue by first selecting a destroy heuristic H (Section 5.3 indicates that performance is more sensitive to H) before deciding on the neighborhood size N (whose quality depends on the choice of H).

5 Experiments

5.1 Setup

Maps We evaluate BALANCE on five maps from the MAPF benchmark set of (Stern et al. 2019), namely (1) a `random` map (*random-32-32-10*), (2) a `warehouse` map (*warehouse-10-20-10-2-1*), (3) two `game` maps *ost003d* and (4) *den520d* as well as (5) a `city` map (*Paris_1_256*). All maps have different sizes and structures, and are the same as used in (Huang et al. 2022) for comparability with state-of-the-art anytime MAPF as presented below. We conduct all experiments on the available 25 random scenarios per map.

Anytime MAPF Algorithms We implemented[2] different variants of BALANCE using Thompson Sampling (with $\mu_0 = 0$, $\lambda_0 = 0.01$, $\alpha_0 = 1$, $\beta_0 = 100$), UCB1 (with $\xi = 1,000$), and roulette wheel selection. Each BALANCE variant is denoted by *BALANCE (X)*, where *X* is the concrete MAB algorithm (or just random uniform sampling) used for our bi-level scheme in Figure 1. Unless stated otherwise, we always use the $|\mathcal{H}| = 3$ destroy heuristics from (Li et al. 2021) and set $E = 5$ such that the neighborhood size is chosen from $\mathcal{N} = \{2, 4, 8, 16, 32\}$[3]. Our BALANCE implementation is based on the public code of (Li et al. 2022) and uses its default configuration unless stated otherwise. The complete configuration is provided in the Appendix A.2.

[2]Our code is provided in the supplementary and will be published upon acceptance.

[3]Even though previous work (Li et al. 2021, 2022) already indicates good values for fixed neighborhood sizes N, we keep optimizing our MABs on a broader set of options to confirm convergence to adequate choices without assuming any prior knowledge.

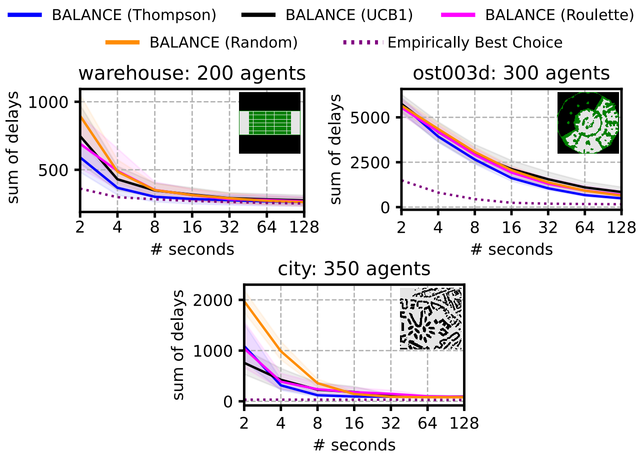

Figure 2: Sum of delays for different BALANCE variants with different time budgets compared with the respective empirically best choices. Shaded areas show the 95% confidence interval. The legend at the top applies across all plots.

To directly compare with MAPF-LNS and MAPF-ML-LNS as state-of-the-art approaches to anytime MAPF, we take the performance values reported in (Huang et al. 2022), running our experiments on the same hardware specification.

Compute Infrastructure All experiments were run on a computing cluster of fifteen x86_64 GNU/Linux (Ubuntu 18.04.5 LTS) machines with i7 @ 2.4 GHz CPU and 16 GB RAM similarly to (Huang et al. 2022).

5.2 Experiment – BALANCE Convergence

Setting To assess convergence w.r.t. time budget, we run *BALANCE (Thompson)*, *BALANCE (UCB1)*, *BALANCE (Roulette)*, and *BALANCE (Random)* on the `random`, `ost003d`, and `city` map with m = 200, 300, and 350 agents respectively. We additionally determine the *empirically best choice*, where we run a grid search over all $|\mathcal{H}| = 3$ destroy heuristics and $|\mathcal{N}| = E = 5$ neighborhood size options $\mathcal{N} = \{2, 4, 8, 16, 32\}$ to compare with a pre-tuned LNS without any online adaptation.

Results The results are shown in Figure 2. With increasing time budget, all BALANCE variants converge to an average sum of delays close to the empirically best choice. In `city`, all MAB enhanced variants converge faster than *BALANCE (Random)*. *BALANCE (Thompson)* performs best in all maps, especially when the time budget is low.

Discussion The results show that any version of BALANCE is able to perform well with an increasing time budget. Given a sufficient time budget, all versions are able to (nearly) match the performance of the empirically best choice through online learning without running a prior grid search that requires roughly $|\mathcal{H}| \cdot |\mathcal{N}| = 3E = 15$ times the compute of any BALANCE variant in total. Thompson Sampling performs particularly well, presumably due to the inherent uncertainty provided by the randomized destroy heuristics in the MAPF-LNS setting.

5.3 Experiment – BALANCE Exploration

Setting Next, we evaluate the explorative behavior of *BALANCE (Thompson)*, *BALANCE (UCB1)*, and *BALANCE (Roulette)* after 128 seconds of LNS runtime. We also analyze the degree of exploration relative to the empirically best choice and compare with random uniform sampling to assess the greediness of the tested MAB algorithms.

Results The relative frequencies of MAB choices are displayed as heatmaps in Figure 3. The empirically best destroy heuristics and neighborhood sizes are highlighted by magenta dashed boxes. *BALANCE (UCB1)* and *BALANCE (Roulette)* strongly prefer the random destroy heuristic, while the preferred neighborhood size depends on the actual map. *BALANCE (Thompson)* also prefers the random destroy heuristic to some degree but still explores other heuristics mainly with neighborhood sizes $N \geq 8$. Compared to the other variants, *BALANCE (Thompson)* explores more regions, where either the destroy heuristic H or the neighborhood size N is empirically best at least.

To quantify the exploration relative to the empirically best choice, we regard H and N as *stochastic events* and classify them either as empirically best (b) or not ($\neg b$). The *normalized joint entropy* $\eta = -\frac{1}{\eta_{uniform}} \sum_{H \in \{b,\neg b\}} \sum_{N \in \{b,\neg b\}} Pr(H,N) log(Pr(H,N))$ of both events (see Appendix A.4) is shown in Figure 4 for each MAB, where Thompson Sampling is shown to explore the event combinations $\{b, \neg b\}$ of H and N the most – even more than random uniform in `ost003d` and `city`.

Discussion None of the BALANCE variants clearly converges to the empirically best choice, which could be due to a short time budget, marginal improvement over time, or potential non-stationarity of the actual optimal choice. Nevertheless, Figure 3 and 4 suggest that Thompson Sampling performs more focused exploration than any other BALANCE variant. UCB1 and Roulette exhibit a lower joint entropy than random uniform, indicating that both algorithms behave more greedily and therefore lack sufficient exploration.

5.4 Experiment – Neighborhood Size Options

Setting We run *BALANCE (Thompson)*, *BALANCE (UCB1)*, *BALANCE (Roulette)*, and *BALANCE (Random)* with different neighborhood size options by varying E, i.e., the number of exponents e to determine the neighborhood size $N = 2^e$. The same maps and number of agents m as above are used with a time budget of 128 seconds. We additionally evaluate with a doubled number of agents per map.

Results The results are shown in Figure 5. All approaches significantly improve when the number of options is increased to $E = 3$ with marginal to no improvement afterwards. *BALANCE (Thompson)* and *BALANCE (Random)* benefit the most from the increase of E except for `city` with 700 agents, where *BALANCE (UCB1)* keeps up with *BALANCE (Thompson)*.

Discussion Since Thompson Sampling and random uniform explore more than UCB1 and Roulette, they can better leverage the neighborhood size options. The results indicate

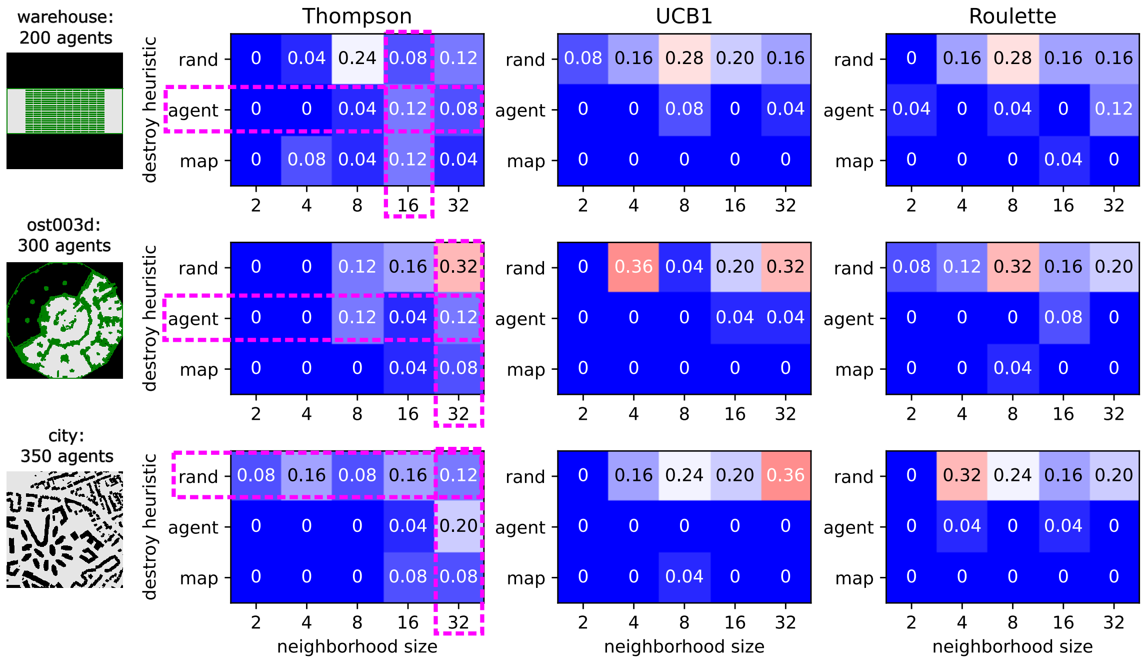

Figure 3: Relative frequencies of selected destroy heuristic and neighborhood size combinations $\langle H, N \rangle$ per BALANCE variant for a time budget of 128 seconds. Magenta dashed boxes indicate the empirically best destroy heuristic and neighborhood size.

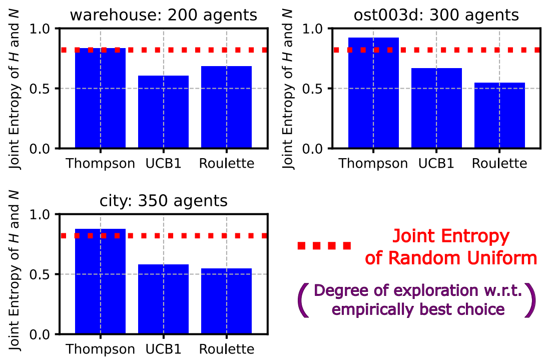

Figure 4: Normalized joint entropy of different BALANCE variants w.r.t. the empirically best destroy heuristic H and neighborhood size N for a time budget of 128 seconds.

that neighborhood size adaptation and the sufficient availability of options can significantly affect performance. However, the neighborhood size also affects the amount of compute for replanning, which explains why *BALANCE (Random)* performs worse in `ost003d` when $E = 5$.

5.5 Experiment – State-of-the-Art Comparison

Setting We run *BALANCE (Thompson)*, *BALANCE (UCB1)*, and *BALANCE (Roulette)* on the `random`, `warehouse`, `ost003d`, `den520d`, and `city` map with different number of agents m. We compare our results with MAPF-LNS and MAPF-ML-LNS whose performance values are taken from (Huang et al. 2022). For direct comparability, we set the time budget to 60 seconds.

Results The results are shown in Figure 6. All BALANCE variants significantly outperform MAPF-LNS and MAPF-ML-LNS by at least 50% when $m \geq 350$. In the `random`, `ost003d`, and `city` map, *BALANCE (Thompson)* slightly outperforms the other BALANCE variants.

Discussion The experiment demonstrates that BALANCE effectively mitigates the limitations of state-of-the-art anytime MAPF regarding fixed neighborhood sizes and the greediness of roulette wheel selection, especially in instances with a large number of agents m. While Thompson Sampling seems to be the best MAB algorithm in most cases, using BALANCE with any MAB algorithm is generally beneficial to significantly improve performance.

6 Conclusion

We presented BALANCE, an LNS framework using a bi-level multi-armed bandit scheme to adapt the selection of destroy heuristics and neighborhood sizes during search.

Our experiments show that BALANCE offers a simple but effective framework for adaptive anytime MAPF, which is able to significantly outperform state-of-the-art anytime MAPF without requiring extensive prior effort like neighborhood size tuning, data acquisition, or feature engineering. Sufficient availability of neighborhood size options is important to provide enough room for adaptation at the potential cost of runtime due to increasing replanning effort. Thompson Sampling is a promising choice for most scenarios due to the inherent uncertainty of the randomized destroy heuristics and its ability to explore promising choices.

Future work includes the investigation of non-stationary MAB approaches and online learnable destroy heuristics.

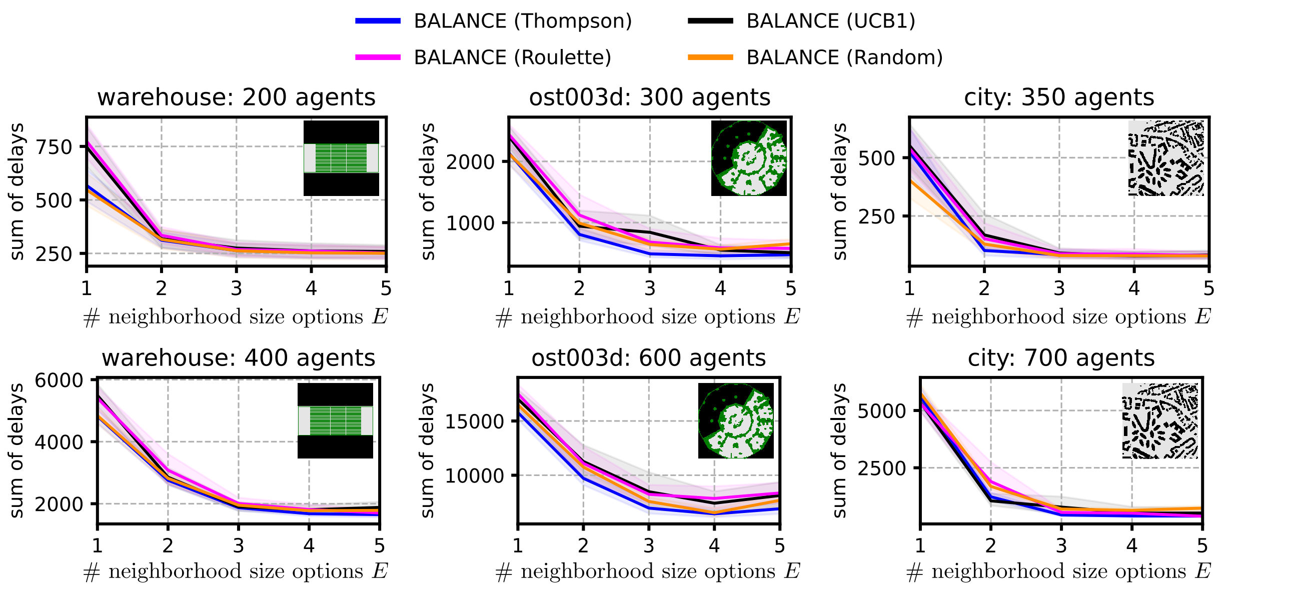

Figure 5: Sum of delays for different BALANCE variants with different neighborhood size options E and number of agents m. The time budget is 128 seconds. Shaded areas show the 95% confidence interval. The legend at the top applies across all plots.

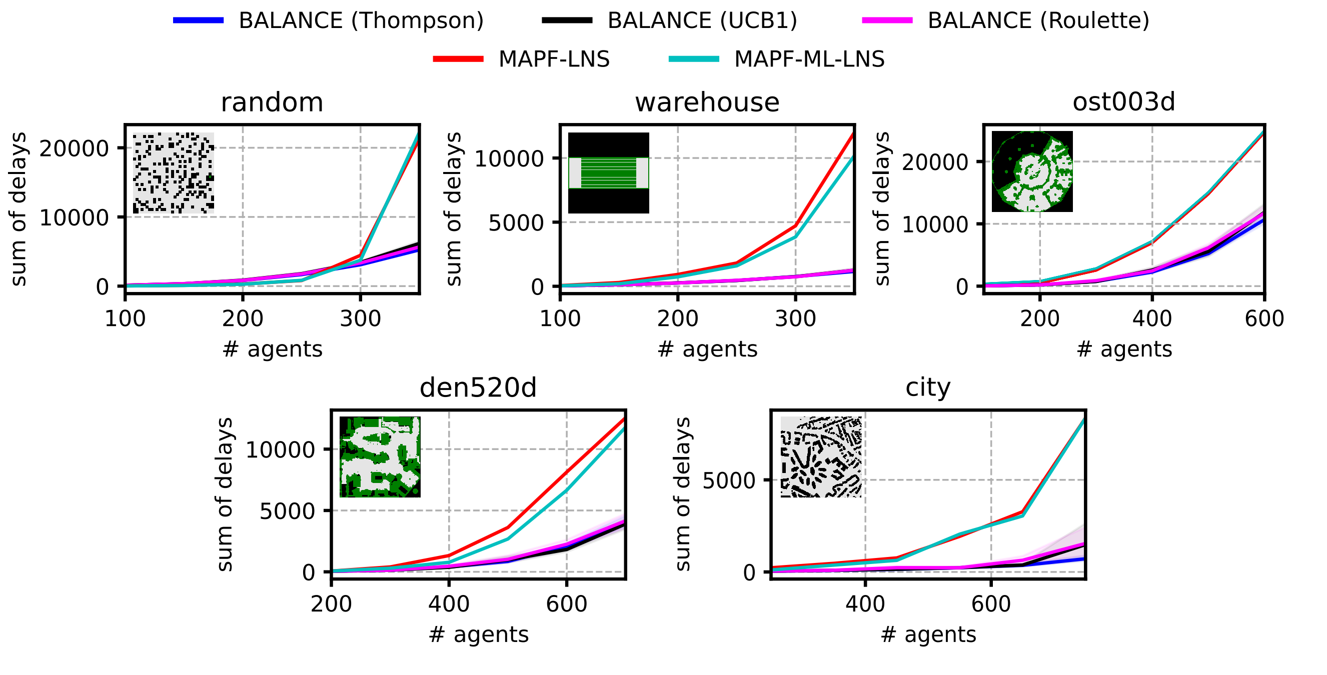

Figure 6: Sum of delays for different variants of BALANCE compared with state-of-the-art anytime MAPF-LNS and MAPF-ML-LNS for different number of agents m. The performance values of MAPF-LNS and MAPF-ML-LNS are taken from (Huang et al. 2022). Our experiments are run on the same hardware specification with a time budget of 60 seconds. Shaded areas show the 95% confidence interval. The legend at the top applies across all plots.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-Time Analysis of the Multiarmed Bandit Problem. *Machine learning*, 47(2-3): 235–256.

Badia, A. P.; Piot, B.; Kapturowski, S.; Sprechmann, P.; Vitvitskyi, A.; Guo, Z. D.; and Blundell, C. 2020. Agent57: Outperforming the Atari Human Benchmark. In *International conference on machine learning*, 507–517. PMLR.

Bai, A.; Wu, F.; and Chen, X. 2013. Bayesian Mixture Modelling and Inference based Thompson Sampling in Monte-Carlo Tree Search. In *Advances in Neural Information Processing Systems*, 1646–1654.

Bai, A.; Wu, F.; Zhang, Z.; and Chen, X. 2014. Thompson Sampling based Monte-Carlo Planning in POMDPs. In *Proceedings of the Twenty-Fourth International Conferenc on International Conference on Automated Planning and Scheduling*, 29–37. AAAI Press.

Chapelle, O.; and Li, L. 2011. An Empirical Evaluation of Thompson Sampling. In *Advances in neural information processing systems*, 2249–2257.

Chen, B.; and Bai, R. 2018. A Reinforcement Learning Based Variable Neighborhood Search Algorithm for Open Periodic Vehicle Routing Problem with Time Windows.

Chen, Y.; Cowling, P. I.; Polack, F. A. C.; and Mourdjis, P. 2016. A Multi-Arm Bandit Neighbourhood Search for Routing and Scheduling Problems.

Chmiela, A.; Gleixner, A.; Lichocki, P.; and Pokutta, S. 2023. Online Learning for Scheduling MIP Heuristics. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 114–123. Springer.

Cohen, L.; Greco, M.; Ma, H.; Hernández, C.; Felner, A.; Kumar, T. S.; and Koenig, S. 2018. Anytime Focal Search with Applications. In *IJCAI*, 1434–1441.

Garivier, A.; and Moulines, E. 2008. On Upper-Confidence Bound Policies for Non-Stationary Bandit Problems. *arXiv preprint arXiv:0805.3415*.

Goldberg, D. E. 1988. Genetic Algorithms in Search Optimization and Machine Learning.

Hendel, G. 2022. Adaptive Large Neighborhood Search for Mixed Integer Programming. *Mathematical Programming Computation*, 1–37.

Huang, T.; Dilkina, B.; and Koenig, S. 2021. Learning Node-Selection Strategies in Bounded Suboptimal Conflict-Based Search for Multi-Agent Path Finding. In *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Huang, T.; Li, J.; Koenig, S.; and Dilkina, B. 2022. Anytime Multi-Agent Path Finding via Machine Learning-Guided Large Neighborhood Search. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI)*, 9368–9376.

Kaduri, O.; Boyarski, E.; and Stern, R. 2020. Algorithm Selection for Optimal Multi-Agent Pathfinding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 161–165.

Kaufmann, E.; Korda, N.; and Munos, R. 2012. Thompson Sampling: An Asymptotically Optimal Finite-Time Analysis. In *International Conference on Algorithmic Learning Theory*, 199–213. Springer.

Kocsis, L.; and Szepesvári, C. 2006. Bandit based Monte-Carlo Planning. In *ECML*, volume 6, 282–293. Springer.

Lam, E.; Harabor, D.; Stuckey, P. J.; and Li, J. 2023. Exact Anytime Multi-Agent Path Finding Using Branch-and-Cut-and-Price and Large Neighborhood Search. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2021. Anytime Multi-Agent Path Finding via Large Neighborhood Search. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 4127–4135.

Li, J.; Chen, Z.; Harabor, D.; Stuckey, P. J.; and Koenig, S. 2022. MAPF-LNS2: Fast Repairing for Multi-Agent Path Finding via Large Neighborhood Search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9): 10256–10265.

Mara, S. T. W.; Norcahyo, R.; Jodiawan, P.; Lusiantoro, L.; and Rifai, A. P. 2022. A Survey of Adaptive Large Neighborhood Search Algorithms and Applications. *Computers & Operations Research*, 146: 105903.

Phan, T.; Belzner, L.; Kiermeier, M.; Friedrich, M.; Schmid, K.; and Linnhoff-Popien, C. 2019a. Memory Bounded Open-Loop Planning in Large POMDPs using Thompson Sampling. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01): 7941–7948.

Phan, T.; Gabor, T.; Müller, R.; Roch, C.; and Linnhoff-Popien, C. 2019b. Adaptive Thompson Sampling Stacks for Memory Bounded Open-Loop Planning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI-19*, 5607–5613. International Joint Conferences on Artificial Intelligence Organization.

Ratner, D.; and Warmuth, M. 1986. Finding a Shortest Solution for the NxN Extension of the 15-Puzzle is Intractable. In *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, AAAI'86, 168–172. AAAI Press.

Ropke, S.; and Pisinger, D. 2006. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation science*, 40(4): 455–472.

Rothberg, E. 2007. An Evolutionary Algorithm for Polishing Mixed Integer Programming Solutions. *INFORMS Journal on Computing*, 19(4): 534–541.

Sartoretti, G.; Kerr, J.; Shi, Y.; Wagner, G.; Kumar, T. S.; Koenig, S.; and Choset, H. 2019. PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning. *IEEE Robotics and Automation Letters*, 4(3): 2378–2385.

Schaul, T.; Borsa, D.; Ding, D.; Szepesvari, D.; Ostrovski, G.; Dabney, W.; and Osindero, S. 2019. Adapting Behaviour for Learning Progress. *arXiv preprint arXiv:1912.06910*.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. 2012. Conflict-Based Search For Optimal Multi-Agent Path Finding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 26(1): 563–569.

Silver, D. 2005. Cooperative Pathfinding. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 1(1): 117–122.

Silver, D.; and Veness, J. 2010. Monte-Carlo Planning in Large POMDPs. In *Advances in neural information processing systems*, 2164–2172.

Stern, R.; Sturtevant, N.; Felner, A.; Koenig, S.; Ma, H.; Walker, T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T.; et al. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, 151–158.

Świechowski, M.; Godlewski, K.; Sawicki, B.; and Mańdziuk, J. 2023. Monte Carlo Tree Search: A Review of Recent Modifications and Applications. *Artificial Intelligence Review*, 56(3): 2497–2562.

Thompson, W. R. 1933. On the Likelihood that One Unknown Probability exceeds Another in View of the Evidence of Two Samples. *Biometrika*, 25(3/4): 285–294.

Yu, J.; and LaValle, S. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 27(1): 1443–1449.