Goal Distribution in Conflict-Based Search for Multi-Agent Pathfinding and its Implications to Monte-Carlo Sampling

Colton Simpson[1], **Sumedh Pendurkar**[1], **Guni Sharon**[1]

[1]Texas A&M University, College Station, TX 77843, USA
{csimpson2018, sumedhpendurkar, guni}@tamu.edu

Abstract

Conflict-Based Search (CBS) is a common two-level search framework for solving Multi-Agent Path Finding (MAPF) problem. At its high-level, CBS searches over a Constraint Tree (CT). A body of publications looked at various approaches to perform the high-level search (over the CT) and find a goal (conflict-free) node efficiently. The performance of these variants is dependent on the goal node distribution in the CT. Consequently, we provide a first-of-its-kind analysis of the goal-node distribution in the CT for common benchmark MAPF maps with respect to (1) solution quality, (2) solution depth (as rollout runtime), and (3) solution suboptimality. The results suggest that samples of solution quality and suboptimality can be reasonably fitted (low Kolmogorov-Smirnov distance) with a log-normal distribution and samples of runtime with a 'F' distribution. Moreover, we show that the percentage of CT branches that do not lead to a goal node is marginal ($\sim$0%) for sparse density and low ($\sim$16%) for medium agent density in common MAPF maps. These results suggest that random rollouts on the CT could be effective in such scenarios. Consequently, we consider Monte-Carlo Conflict-Based Search (MC-CBS), where random rollouts are performed over the CT. MC-CBS has four main advantages, (1) it is asymptotically optimal, (2) it provides probabilistic suboptimality guarantees, (3) it enables full utilization of multicore, parallel computation, and (4) it is intuitive and simple to implement. Next, we examine an enhanced variant MC-CBS+ which incorporates state-of-the-art improvements to the low-level search in CBS. Although MC-CBS+ is not asymptotically optimal, it provides computational speedups in dense MAPF instances. Finally, we provide an empirical evaluation of MC-CBS and MC-CBS+ on 12 common benchmark MAPF maps. The results suggest that MC-CBS+ can outperform state-of-the-art bounded suboptimal solver in sparse scenarios with respect to both runtime and solution quality. These results suggest that sampling-based methods, which were previously overlooked, should be further explored by the MAPF community.

1 Introduction

The goal in the Multi-Agent Path Finding (MAPF) problem is to find conflict-free paths for multiple agents leading from their starting location to their respective target location. This problem has a wide range of applications in areas such as traffic management (Okoso, Otaki, and Nishi 2019), warehouse logistics (Ma et al. 2017), and airport operations (Li et al. 2019b).

Conflict-Based Search (CBS) (Sharon et al. 2015) is a widely used framework to solve MAPF problems. CBS works on two levels: at the high-level, CBS constructs a Constraint Tree (CT). Each node in this tree represents a set of constraints and a solution that satisfies the constraints. AQ solution is a set of paths, one per agent. At the low-level, CBS computes a single-agent path for a given agent, returning a path satisfying the constraints imposed by the high level.

Recent work demonstrated significant speed-ups over the original CBS framework when applying advanced search methods over the CT. For instance, Felner et al. (2018) used the A* algorithm (Hart, Nilsson, and Raphael 1968), Boyarski et al. (2020) used iterative deepening A* (Korf 1985), Barer et al. (2014) used focal search (Pearl and Kim 1982), and Li, Ruml, and Koenig (2021) used explicit estimation search (Thayer and Ruml 2011). These studies suggest that the performance of CBS is sensitive to the choice of the high-level (CT) search algorithm. That is, the order in which one generates and expands CT nodes, in search of a goal CT node, impacts the computational effort and solution quality. Consequently, we provide a first-of-its-kind CT goal-node distribution analysis for a variety of benchmark MAPF scenarios. Specifically, we report empirical CT goal node distributions with respect to (1) solution quality (as sum-of-costs), (2) root-to-goal expansion runtime, and (3) (approximated) suboptimality. The reported distributions are fitted to 10,000 random rollouts on the CT. The results suggest that several distributions could be viewed as well-fitting to the samples obtained per map and agent density. Overall, log-normal distribution could be considered a reasonable fit to solution quality (sum-of-costs) and suboptimality distributions in the majority of reported scenarios. Similarly, runtime distribution can be reasonably fitted with a F distribution in the majority of reported scenarios. The results further suggest that relatively few rollouts fail (not leading to a goal node, i.e., dead-ends or timeouts) in scenarios that are not very dense. Specifically, 100% of the sparse scenarios, 83% of medium density scenarios, and 33% of dense scenarios lead to a goal node in more than 90% of sampled rollouts. This implies that random (rollout-based) search algorithms

over the CT could be effective MAPF solvers when used in the CBS framework.

Following our goal distribution study, we consider a random rollout-based approach, which we refer to as Monte Carlo CBS (MC-CBS). MC-CBS has four main advantages (1) it is asymptotically optimal; (2) it provides probabilistic suboptimality guarantees (suboptimality $< x$ with probability y); (3) it enables parallel computation, resulting in full utilization of multicore systems; and (4) it is intuitive and simple to implement on top of the widely accepted CBS algorithm. Moreover, we present an enhanced variant, denoted MC-CBS+, which incorporates state-of-the-art low-level search improvements for CBS. Although MC-CBS+ is not asymptotically optimal, experimental results present reduced runtime over MC-CBS making it more practical in denser scenarios. Finally, we provide a comparative study on 12 common benchmark maps and 3 agent densities. The results suggest that MC-CBS+ can outperform a state-of-the-art bounded suboptimal solver (EECBS+) in sparse scenarios. These results highlight the potential of sampling based methods for searching over the CT (high-level search) in the CBS framework.

2 Preliminaries

A multi-agent path finding (MAPF) problem is defined over a graph, $G(V, E)$, and m agents $\{a_1, a_2, \cdots a_m\}$. Each agent a_i has a distinct start vertex $s_i \in V$ and a distinct goal vertex $g_i \in V$. We consider the discrete time version of MAPF following *Classical MAPF* in Stern et al. (2019). At each time-step, an agent can move to any adjacent vertex or wait at the current vertex. We focus on MAPF variants with *vertex* and *swapping conflicts*, the *stay at target* assumption, and use *sum-of-individual-costs* (SIC) as the optimization objective as described by Stern et al. (2019). A given graph, $G(V, E)$, together with a fixed number of agents, m, are referred to as a *scenario*. A scenario with a fixed set of start and goal vertices, $\forall i \in m, \{s_i, g_i\}$, is referred to as an *instance* of the MAPF scenario. A valid *path* p_i for agent a_i is a chronological sequence of vertices where the first vertex in p_i is s_i, the last vertex is g_i, and any consecutive vertices have a connecting edge in E. The cost of a path, p_i, is the number of vertices in the path denoted by $|p_i|$. A *solution* of the MAPF instance is a set of paths, $P = \{p_1, p_2, \cdots, p_m\}$. A solution is *valid* if it the individual paths are not conflicting. An *optimal solution* is a valid solution that minimizes the SIC objective given by $c^* = \min_P \sum_{i=1}^{i=m} |p_i|$. A solver is *bounded suboptimal* with a factor of w if it is guaranteed to return a valid solution with cost c such that $c \leq w \cdot c^*$ for all solvable instances.

2.1 Conflict-Based Search (CBS)

CBS (Sharon et al. 2015) is a widely used framework for solving MAPF problems either optimally (Sharon et al. 2015; Boyarski et al. 2015b; Felner et al. 2018) or with bounded suboptimality (Barer et al. 2014; Li, Ruml, and Koenig 2021). CBS is a two-level search framework: The **high-level** searches a *Constraint Tree* (CT), where each node represents (1) a set of constraints, each applying to a specific agent, and (2) a solution (not necessarily valid) which satisfies the set of constraints. If one or more conflicts exist in a CT node's (invalid) solution, the CT branches, creating two child nodes with constraints that resolve one of these conflicts. The high-level search in the basic version of CBS (Sharon et al. 2015) performs a best-first search on the CT. A CT node that has a valid solution is denoted a *goal node*. A non-goal CT node with no child nodes is a *dead-end*. These situations can occur when a constraint added for a potential child node disables all possible paths for one of the agents, e.g., by disallowing it to occupy any vertex at a specific timestep. As a result, the potential child node in question is pruned.

Note, a CT is not unique per MAPF instance. CT nodes are dependent on how conflicts are selected at their predecessor node as well the search algorithm used for computing the individual single-agent paths, denoted the *low-level search*. In order to avoid ambiguity in the CT structure, we assume Conflict Prioritization (Boyarski et al. 2015b) for conflict selection. For the low-level search, we consider two variants, (1) single-agent A*, as proposed by Sharon et al. (2015), and (2) focal search with enhancements as proposed by Li, Ruml, and Koenig (2021). We denote the CT obtained with (1), (2) as CT-, CT+ respectively.

2.2 Related Work

MAPF approaches can be broadly divided into three categories (following Okumura (2023)):

(1) Compilation Based: This class of MAPF solvers includes approaches like eMDD-SAT (Surynek et al. 2018), BCP-7 (Lam et al. 2019). eMDD-SAT is a bounded suboptimal solver that reduces a MAPF problem to a Boolean satisfiability (SAT) problem and solves the SAT problem to produce a valid solution. BCP-7 is an optimal solver that uses branch-and-cut-and-price to produce the optimal solution.

(2) Prioritized Planning Based: These approaches sequentially plan individual paths conditioned on previously planed paths. The agents' planning order is based on a specific priority assignment (Erdmann and Lozano-Perez 1987; Silver 2005).

(3) Search Based: Hierarchical Cooperative A* for MAPF (Silver 2005), M* (Wagner and Choset 2015), CBS (Sharon et al. 2015) and LaCAM (Okumura 2023) are examples of search-based MAPF solvers. LaCAM, like CBS, uses a two-level search framework where at the low-level the search is performed over constraints regarding the agent locations, and at the high-level it searches for a sequence of all agents' locations, following the constraints from low-level search. LaCAM can solve challenging scenarios significantly faster as compared to previous baselines, but does not provide any guarantee on the solution quality. An improved version of LaCAM, called LaCAM* (Okumura 2024) is an anytime variant of LaCAM with various improvements, such as Monte-Carlo configuration generation, space utilization optimization, and recursive improvement of discovered solutions, with a guarantee of eventual optimality.

Map	CT-						CT+					
	Sparse		Medium		Dense		Sparse		Medium		Dense	
	TO	DE	TO	DE	TO	DE	TO	DE	TO	DE	TO	DE
brc202d	0	0	0	0	0.24	0	0	0.07	0.01	0.08	0.11	0.02
gallows_templar	0	0	1.38	0.01	46.78	0	0	0.01	0.23	0.25	0.95	0.58
maze	0.02	0.09	46.02	1.92	88.08	11.09	0	37.29	0.32	73.60	5.55	**93.58**
orz900	0	0	0	0	0	0	0.02	0.02	0.01	0	0	0
ost003	0	0	0	0.01	12.93	0.05	0	0.03	0.06	0.79	1.31	1.64
random_64	0	0	0	1.19	51.85	15.85	0	0.80	0	11.10	0	23.50
random	0	0.15	0	10.21	0.11	**91.94**	0	2.58	0	10.62	0	23.28
room32	0	0.43	0	8.92	3.57	**96.40**	0	10.85	0	24.89	0.01	82.11
room_64_8	0	0	0.22	0.10	**98.81**	1.19	0	10.76	0.39	38.90	14.54	52.52
room	0	0	8.88	0.01	**99.87**	0.08	0	4.53	1.52	12.25	19.10	16.46
warehouse_1	0	0	0.03	0.25	3.21	1.77	0	0	0.02	0.17	0.36	0.53
warehouse	0	0	0	0.01	1.29	0	0	0	0	0	0	0

Table 1: Percentage of timeouts and dead-ends per MAPF scenario. TO is the % of rollouts that results in timeouts. DE is the % of rollouts that resulted in dead-ends. Bold values represent scenarios where the TO or DE value is greater than 90%, deeming them less appropriate for solvers based on random sampling.

(3a) CBS Based: Recent optimal CBS variants presented considerable speedups over basic CBS by considering heuristic guidance for searching the CT (high-level search) and advanced search algorithms, e.g., A* (Felner et al. 2018; Li et al. 2019a; Boyarski et al. 2020) or IDA* (Boyarski et al. 2020). Moreover, a line of bounded suboptimal CBS variants presented even greater speedups by relaxing the original optimality guarantee. Enhanced CBS (ECBS) (Barer et al. 2014) uses a focal search (Pearl and Kim 1982) for both the high-level and low-level searches. Explicit Estimation CBS (EECBS) (Li, Ruml, and Koenig 2021) further improves the performance of ECBS by replacing the high-level focal search with the explicit estimation search (Thayer and Ruml 2011), while retaining the guarantees of being a bounded suboptimal framework. Li, Ruml, and Koenig (2021) also propose EECBS+, an EECBS variant that incorporates enhancements like bypassing conflicts (Boyarski et al. 2015a), prioritizing conflicts (Boyarski et al. 2015b), symmetry reasoning (Li et al. 2019c), weighted dependency graph heuristic (Li et al. 2019a), and mutex propagation (Zhang et al. 2020).

(4) Sampling Based: Previous work considered using Monte-Carlo based techniques for MAPF in different settings than the one we consider (as presented later in Sec 2). For example, Atzmon et al. (2020) proposed a Monte-Carlo based verifier for robust MAPF, Skrynnik et al. (2024) use Monte-Carlo tree search for anytime, decentralized, and partially observable settings. In contrast to these previous works, we provide a novel analysis of the goal-node distribution in CTs using Monte-Carlo rollouts followed by a presentation and analysis of Monte-Carlo based methods for the CT (high-level) search.

3 Goal Node Distribution in CTs

The performance of a random sampling approach is sensitive to the goal-node distribution in the underlying state space. For example, if goal nodes are relatively rare, then the probability of encountering a goal node in a single random rollout is low and, as a result, a Monte-Carlo approach is expected to be ineffective. Consequently, we begin our study by empirically examining the goal node distribution in CTs constructed from various common MAPF scenarios.

3.1 Experimental Setup

MAPF Scenarios: We evaluate 12 benchmark MAPF maps[1] following Okumura (2023) and 3 agent densities, namely, sparse, medium and dense. The details of the number of agents for each category are provided in Appendix A.1.

CT Parsing Details: We collect samples by performing random Monte-Carlo rollouts on the CT. That is, starting from the root CT node, one child node is randomly followed until either (1) a goal node (having a valid solution) is reached, (2) a dead-end node is reached, or (3) a timeout limit (600 seconds) is exceeded. we sampled 10,000 independent Monte-Carlo rollouts per scenario by randomizing the agents' start and goal vertices. We present the results on the two types of CT defined in section 2.1, CT- and CT+.

Metrics: The rollout samples are reported and analyzed for the following metrics: (1) solution cost (SIC), (2) rollout runtime,[2] and (3) (approximated) suboptimality. The suboptimality is approximated, following (Okumura 2023), as the

[1]Following common notation in the literature, we use 'map' to describe an undirected graph, $G(V, E)$, per scenario.

[2]note that rollout time is correlated to rollout depth. Consequently, we omit reporting the rollout depth.

ratio between the observed solution cost and the total distance of start-goal pairs (ignoring conflicts), which is defined as $\sum_{i\in\{1,2,\cdots,m\}}\text{dist}(s_i, g_i)$ where $\text{dist}(u, v)$ is the shortest path from vertex u to v. This approximated suboptimality is an upper bound on the true optimality (Okumura 2023). Further, the number of rollouts that result in dead-end nodes, and those that result in timeouts are also reported.

3.2 Results: Dead-ends and Timeouts

Table 1 presents the percentage of unsuccessful rollouts, i.e., rollouts that resulted in either dead-ends or timeouts. The results suggest a trend where, as density increases for a given map, rollouts are less likely to be successful (i.e., they tend to result in dead-ends or timeouts). For instance, consider the map 'gallows_templar'. In sparse scenarios all rollouts on CT- were successful. However, as the agent density increases, the number of unsuccessful rollouts also increases (0% for sparse to 1.39% for medium to 46.78% for dense instances). Furthermore, the results suggest that adding the low-level enhancements (CT+) generally reduces the number of timeouts, but result in more frequent dead-ends (e.g., "Dense - maze" , "Dense - random_64"). We speculate that this is because the enhanced low-level search introduces constraints that block valid solutions, as apposed to the basic (optimal) low-level search (CT-). Overall, we observe a promising trend where 0% of the sparse scenarios, 16.66% of the medium-density scenarios and 66.66% of the dense scenarios result in more than 90% unsuccessful rollouts on CT-. However, for 5 dense scenarios, namely "maze", "random", "room32", "room_64_8", "room", we observe limiting results where more than 90% of the rollouts are unsuccessful (considering TO + DE). Such low success rates limit the significances of the statistical analysis over the successful samples. Consequently, such analysis for the 5 aforementioned dense scenarios is omitted.

3.3 Results: Goal Node Distribution

The reported dead-ends and timeouts analysis suggests that, in some scenarios, the probability of sampling a valid solution through a random rollout is fairly high. However, it does not inform us regarding how good that solution is (with respect to runtime or solution cost). Consequently, we turn to provide a goal-node distribution analysis based on three measurements, namely, (1) solution costs, (2) runtimes, and (3) approximated suboptimality.

Fitting a parametric distribution. In order to fit and report a parametric distribution we follow: (1) collect 10,000 independent, successful, rollout samples (goal nodes), (2) define the empirical distribution by aggregating the samples by frequencies (histogram), (3) fit a parametric distribution to the empirical distribution using maximum likelihood estimation.[3]

Fitness measure. We use the Kolmogorov-Smirnov (K-S) statistic (Massey 1951) to measure the goodness of fit for each resultant distribution. The choice is justified by the non-parametric nature, ease of interpretation, and independence from distributional assumptions of the K-S statistic (Young 1977; Olea and Pawlowsky-Glahn 2008). To determine the statistical cutoff threshold for a well-fit distribution, we use the critical value table sourced from Massey (1951) with $N = 10,000$ and a common significance level of $\alpha = .01$, giving a statistic threshold value of $d = 0.0163$. In this case, the threshold means that in "$[\alpha * 100]$ percent of random samples of size $[N]$, the maximum absolute deviation between the sample cumulative distribution and the population cumulative distribution will be at least $[d]$" (Massey 1951). If d is above this value, we reject the null hypothesis that the sample distribution is derived from the population distribution.

Results. Table 2 reports aggregated K-S statistic value (d) over all the relevant scenarios (maps$\times$densities with success rate > 0.1) for 5 (of 19) best fitting distributions. Results for the other 14 distributions are provided in Appendix B Table 3. For each distribution, we report the mean (average fit) and maximum (worst fit), K-S statistic (d) value over all the scenarios. On top of that, we report the fraction of scenarios within the K-S statistic threshold (as 'FQ').

Discussion. We see several well-fit distributions for solution costs and approximated suboptimality. For example, log-normal and beta distributions have the lowest mean values and highest fraction of scenarios above the K-S threshold for CT-.

However, for CT+ we see some outliers for the beta distribution, thus we consider log-normal distribution for goal costs to be most reliable. Similarly, we consider log-normal distribution to be well-fitting for the suboptimality distributions. We see more outliers for the runtime distributions, resulting in poor fitting in most scenarios (best maximum K-S Statistic of .105 and maximum of 8/31 across scenarios reported). Although none of the theoretical distributions seem to fit any runtime histogram very well, the F distribution could potentially be used to fit those sample distributions.

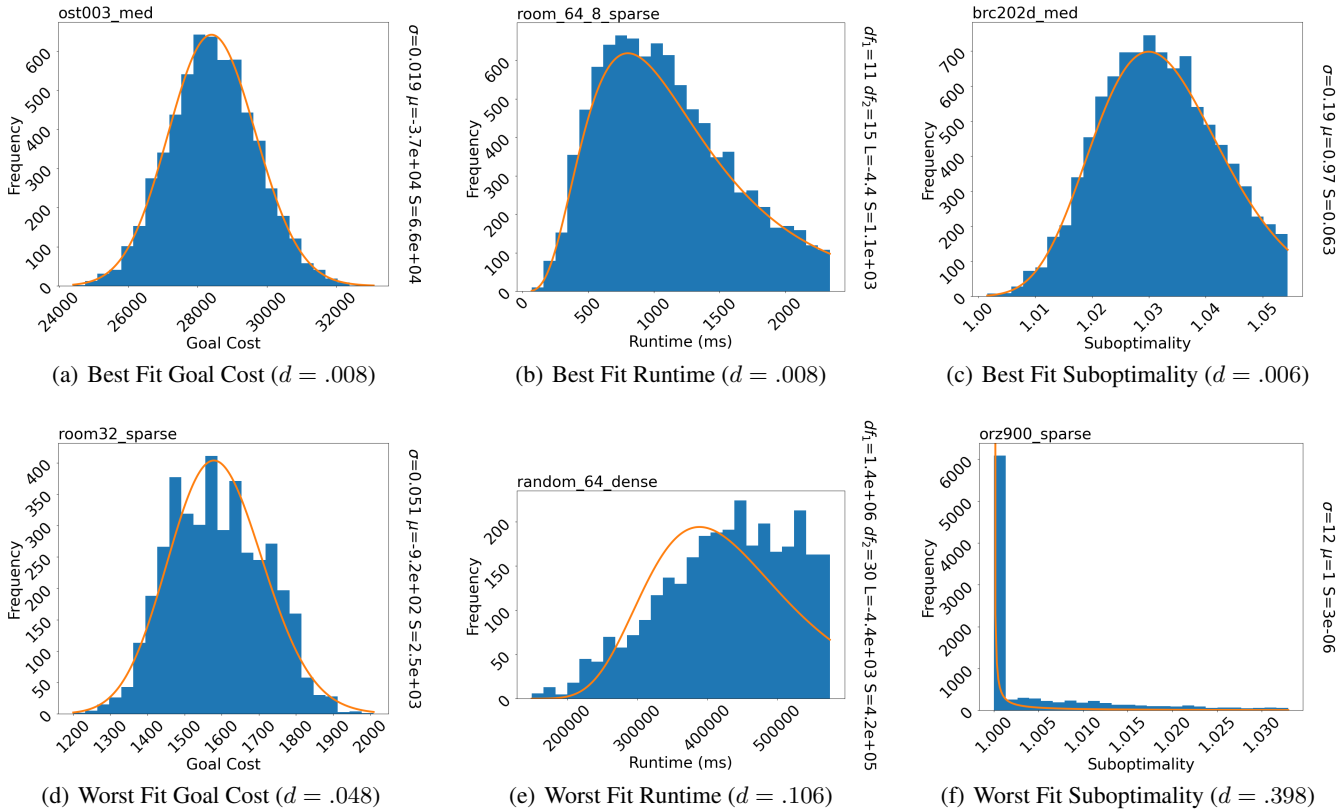
Figure 1 shows the best and worst distributions measured by the K-S Statistic (per histogram as discussed above) for CT-.[4] These results further support that log-normal distribution can be a good fit for the solution cost and approximated suboptimality and F distribution can be a reasonable fit for runtime distribution.

3.4 Benefits of Fitting a Distribution

Assuming that a particular distribution, f, fits any of the true goal node distribution histograms, enables theoretical statements on the number of rollouts required to achieve a desired level of performance. Specifically, assuming the cumulative distribution function, Φ, for f is well defined, we can compute the probability that a single sample is not larger than a given value, T, as $\Phi(T)$. This value can be used to define the number of samples needed to guarantee at least one sample with value of T or lower with probability, p, as

[3] For fitting a distribution parameters we use the Python library implementation by Cokelaer (2024).

[4] Results on CT+ are presented in Appendix B Figure 4.



(a) Best Fit Goal Cost ($d = .008$) (b) Best Fit Runtime ($d = .008$) (c) Best Fit Suboptimality ($d = .006$)

(d) Worst Fit Goal Cost ($d = .048$) (e) Worst Fit Runtime ($d = .106$) (f) Worst Fit Suboptimality ($d = .398$)

Figure 1: MC-CBS best and worst fitting MAPF scenarios for solution costs, runtime, and approximated suboptimality. d denotes the K-S statistic value.

Method	Dist.	Goal Costs			Runtimes			Suboptimality		
		Max	Mean	SAT	Max	Mean	SAT	Max	Mean	SAT
CT-	Gamma	0.049	0.018	16/31	1.000	0.650	1/31	0.518	0.080	19/31
	Log-Norm	0.048	0.018	16/31	0.138	0.045	8/31	0.398	0.063	20/31
	Normal	0.083	0.022	13/31	0.472	0.250	0/31	0.290	0.075	1/31
	F	0.050	0.021	14/31	0.106	0.041	8/31	0.252	0.039	20/31
	Beta	0.040	0.015	19/31	0.331	0.112	1/31	1.000	0.077	20/31
CT+	Gamma	1.000	0.072	20/35	0.998	0.453	1/35	0.950	0.276	9/35
	Log-Norm	0.047	0.016	22/35	0.281	0.070	7/35	0.420	0.038	20/35
	Normal	0.075	0.027	14/35	0.510	0.275	0/35	0.284	0.085	0/35
	F	0.052	0.022	12/35	0.229	0.051	11/35	0.619	0.074	16/35
	Beta	1.000	0.071	22/35	0.765	0.172	1/35	1.000	0.217	8/35

Table 2: Comparison of Kolmogorov-Smirnov distances aggregated over the scenarios considering 5 different parametric distributions. The reported values include the mean K-S statistic value, the max value (demonstrating the worst fit), and the Fit-Quality, "FQ", which is the fraction of scenarios achieving a K-S statistic below 0.0163. Higher FQ values represent better agreement between the empirical and parametric (fitted) distributions.

$$N \geq \frac{\log(1-p)}{\log(1-\Phi(T))} \tag{1}$$

Consequently, we say that a random sampling approach with sufficient samples is *probably-bounded suboptimal* when assuming an underlying suboptimality distribution.

For example, consider the "random_64-medium" scenario from Figure 1. Assuming the suboptimality in this scenario follows a log-normal distribution with $\mu = 0.87, \sigma = 0.071$, and Scale$= 0.27$, we can compute the number of samples, N, required to guarantee a suboptimality of 11% with 0.99 probability. Solving Equation 1 for log-normal distribution results in N = 92.50. Thus, if we sample 93 rollouts for the given scenario, assuming the log-normal distribution fit with the stated parameters, we will attain a solution within a suboptimality of 11% with 0.99 probability.

4 Monte Carlo CBS

The outcomes from our goal distribution analysis can be seen as a justification for a simple yet efficient variant of CBS, one that is based on random sampling. Specifically, we propose a Monte Carlo CBS (MC-CBS) algorithm that performs Monte Carlo rollouts on the CT. Note that since these rollouts are independent of each other, MC-CBS can utilize parallel computation hardware that is common in contemporary multicore processors. The output of MC-CBS is aggregated over all rollouts by selecting the best solution encountered among all rollouts. The performance of the rollouts can be based on either solution cost or runtime, depending on the user's preference. We further propose an enhanced variant of MC-CBS, referred to as MC-CBS+, which performs rollouts on CT+. That is, MC-CBS+ incorporates state-of-the-art enhancements to CBS as presented by Li, Ruml, and Koenig (2021). Specifically, the following improvements are incorporated: bypassing conflicts (Boyarski et al. 2015a), target reasoning (Li et al. 2021), corridor reasoning (Li et al. 2021), rectangle reasoning (Li et al. 2019c), and mutex propagation (Zhang et al. 2020). Additionally, MC-CBS+ uses focal search at the low level, following (Barer et al. 2014).

Next, we discuss the theoretical properties of MC-CBS and MC-CBS+. First, we provide formal definitions of probabilistic completeness (PC) and asymptotic optimality (AO) for sampling-based methods. These definitions follow from the path planning literature (Kleinbort et al. 2018).

Definition 1 (Probabilistic Completeness (PC))**.** *A sampling-based algorithm ALG(p, N) is probabilistically complete if, for any solvable problem p, the following holds:*

$$\lim_{N\to\infty} Pr(\textit{ALG(p, N) returns a valid solution}) = 1,$$

where N is the number of samples used by ALG.

Definition 2 (Asymptotic Optimality (AO))**.** *A sampling-based algorithm ALG(p, N) is asymptotically optimal if, for any solvable problem p, the probability that the cost $C(\textit{ALG}(p, N))$ of the solution returned by the algorithm is optimal (c^*) approaches 1 as the number of samples, N, approaches infinity, i.e., the following holds:*

$$Pr\left(\lim_{N\to\infty} C(\textit{ALG}(p, N)) = c^*\right) = 1,$$

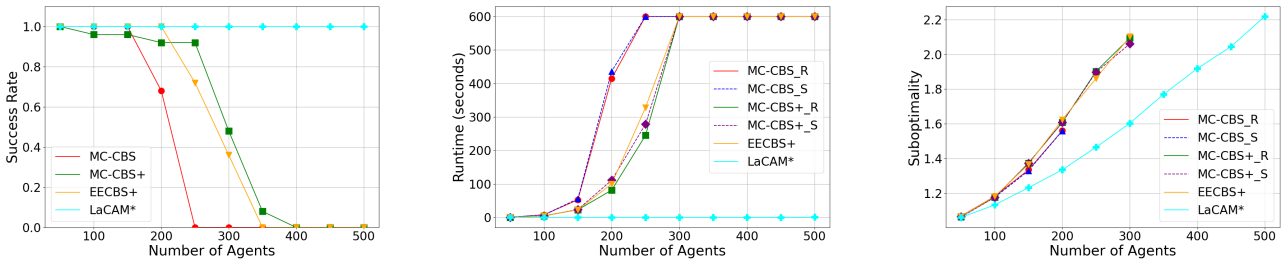
Note, it is trivial that if an algorithm is AO then it is also PC. We show that MC-CBS is AO and MC-CBS+ is PC.

Proposition 1. *MC-CBS is asymptotically optimal.*

Proof. Choosing CT- nodes in best-first order produces an optimal solution to any solvable MAPF problem (Theorem 1 in Sharon et al. (2015)). That is, a best-first search is guaranteed to expand an optimal goal node, n_g. Let τ be the root-to-leaf CT- trajectory leading to n_g. When performing random rollouts on the CT-, every root-to-leaf trajectory has a non-zero probability to be sampled. Hence, as $N \to \infty$, τ will be sampled with probability = 1.

□

Proposition 2. *MC-CBS+ is probabilistically complete.*



(a) Success Rate (b) Runtime (c) Suboptimality

Figure 2: Comparison between MC-CBS, MC-CBS+, EECBS+, and LaCAM* with increasing number of agents on room-64-64-8. 96 rollouts are used for MC-CBS and MC-CBS+. Runtime plot is on logarithmic scale to accommodate higher range of runtime values on single plot.

Proof. MC-CBS+ is motivated by EECBS+ and use the same enhancements, thus both generate similar CT+. As EECBS+ is a complete algorithm (Li, Ruml, and Koenig 2021), EECBS+ produces a valid solution for any solvable MAPF problem. That is, EECBS+ is guaranteed to expand a goal node, n_g. Let τ be the root-to-leaf CT+ trajectory leading to n_g. When performing random rollouts on the CT+, every root-to-leaf trajectory has a non-zero probability to be sampled. Hence, as $N \to \infty$, τ will be sampled with probability = 1. □

5 Empirical Evaluation

MAPF solvers were allowed a timeout limit of 600 seconds. **MAPF Scenarios:** The same scenarios discussed in Section 3.1 are used in this section.

MC-CBS and MC-CBS+ Details: All of the rollouts for both, MC-CBS and MC-CBS+, are executed in parallel. That is, each rollout has a independent timeout limit of 600 seconds and the best rollouts are used for comparison. We set the number of rollouts for both variants to 96 ($N = 96$) following state-of-the-art multicore processors having 96 cores (e.g., AMD Ryzen™ Threadripper™ PRO 7995WX) unless stated otherwise. We report two runtime and suboptimality values for each of the variants: (1) values corresponding to the fastest (CPU time) rollout of the N rollouts (referred to as MC-CBS_R, MC-CBS+_R), and (2) values corresponding to the rollout with the best solution cost (referred to as MC-CBS_S, MC-CBS+_S). While (1) is relevant for time-sensitive applications (returning a solution as soon as the first rollout successfully terminates), (2) is more relevant for quality-sensitive applications (returning the best sampled solution).

Baseline Details: A brief comparison between EECBS+, MC-CBS, LaCAM* (Okumura 2024) and MC-CBS+ is provided in this section. EECBS+ and MC-CBS+ use a suboptimality factor of $w = 5$ following Okumura (2023) for the low-level search. For LaCAM* we pick the first solution generated. We omit results for eMDD-SAT (Surynek et al. 2018), ECBS (Barer et al. 2014), BCP-7 (Lam et al. 2019) as EECBS+ was shown to outperform (in terms of either success rate, solution quality, runtime) all of them by Li, Ruml, and Koenig (2021). Similarly, as LaCAM* (Okumura 2024) outperforms other MAPF solvers like PIBT and PIBT+ (Okumura et al. 2022) they are excluded. Note, EECBS+ has stronger theoretical guarantees than MC-CBS and MC-CBS+, while LaCAM* has an eventual optimal solution cost guarantee. We are unaware of any baseline MAPF solvers with probabilistic (Section 3.4) and asymptotic optimality (Proposition 1) guarantees for a rigorous comparison to MC-CBS and MC-CBS+.

Metrics: We use the same metrics as Okumura (2023). Namely, **(1) Success rate**: Success rate is aggregated across the instances for a given MAPF scenario. For MC-CBS and MC-CBS+ an instance is said to be solved if any of the rollouts (out of N) returns a solution. **(2) Runtime:** The median of runtimes recorded over instances. If a solver fails to solve an instance, the timeout is recorded as the runtime for that instance. **(3) Suboptimality:** A subset of problems is generated which could be solved by all of the MAPF solvers (intersection over all solved problems). The median of the approximated suboptimality over this subset of problems is reported. If a solver is not able to solve any instances it was excluded from the list of solvers used to find the intersection of solved problems.

5.1 Comparison with Baselines

Figure 2 shows the comparison of all the solvers on a representative map, "room-64-64-8", with increasing numbers of agents. The results show that LaCAM* is the best performing algorithm as it can solve all instances of 500 agents on the map while being faster and having better upper bound on suboptimality. Amongst others, we can see that MC-CBS+ has higher success rate and is faster than MC-CBS and EECBS+ while having comparable suboptimality to EECBS+. On the other hand, MC-CBS has better suboptimality values, but has the lowest success rate and slowest runtimes. We also see that both variants (_S and _R) have similar performance for runtime as well as suboptimality (for MC-CBS and MC-CBS+).

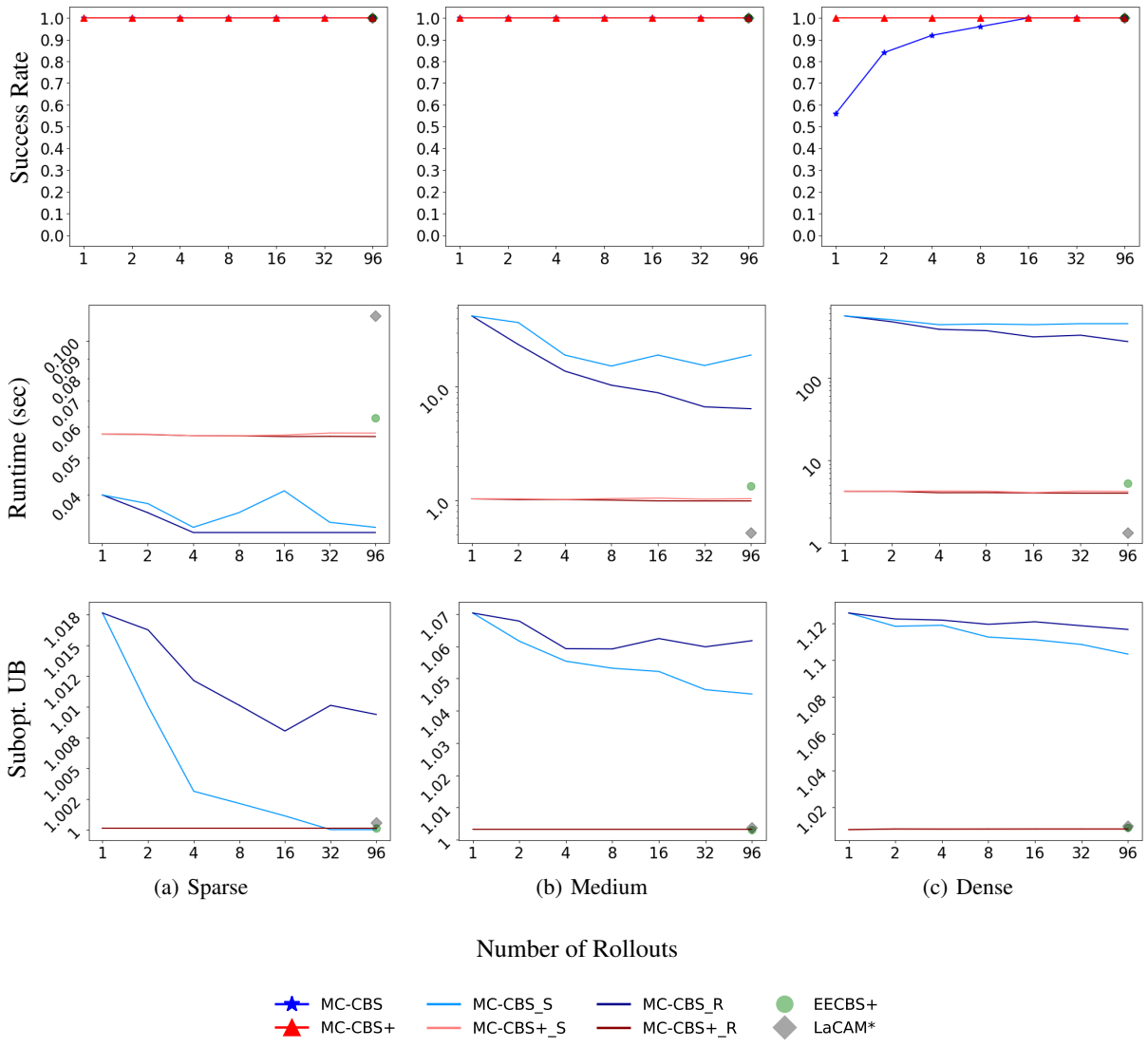


Figure 3: Performance of MC-CBS and MC-CBS+ with an increasing number of rollouts on map "warehouse-20-40-10-2-1". Runtime plot is on logarithmic scale to accommodate higher range of runtime values on single plot.

Results for all of the other MAPF scenarios are provided in Appendix C (Table 4 and Table 5). When comparing to EECBS+, MC-CBS has the best performance in 4/36 scenarios. The results further indicate that MC-CBS+ is the best performing method in 16/36 scenarios, while remaining comparable with MC-CBS and EECBS+ in another 6. In some maps like "room-64-64-16", "gallowstemplar_n", "maze-32-32-2" and "room-64-64-8". EECBS+ seems to perform best, but only for medium/dense agent densities.

5.2 Rollout Count vs. Performance

Figure 3 shows the performance of MC-CBS and MC-CBS+ with increasing number of rollouts. As expected, the results generally improve with the increasing the number of rollouts. MC-CBS has 100% success rate for sparse even with a single rollout. However, the number increases to 16 for dense scenarios.

6 Conclusion

This paper examines the distribution of goal nodes within the high-level Constraint Tree (CT) of Conflict-Based Search (CBS), a key framework for Multi-Agent Pathfinding. The study focuses on (1) solution quality (as sum-of-costs), (2) runtime, (3) success rate, and (4) approximated suboptimality. Results show that random rollouts on the CT yield nearly 100% successful rollouts (ending in a goal node) in sparse agent densities and about 84% in medium densities. The reported goal node distributions can be fitted to models that potentially offer probabilistic guarantees on the number of samples needed to achieve a desired solution quality with a desired probability value. Given these un-

derstandings, we propose a simple random rollout-based algorithm, denoted Monte-Carlo Conflict-Based Search (MC-CBS) which might be effective in such scenarios. MC-CBS has two main benefits (1) it is asymptotically optimal, (2) it can utilize parallel computation. Additionally, this paper introduces an enhanced version, MC-CBS+, which incorporates state-of-the-art improvements to the low-level search. While MC-CBS+ lacks asymptotic optimality, it delivers significant computational speedups in dense scenarios. Comparative evaluations with state-of-the-art baselines indicate that MC-CBS+ is competitive or better than EECBS+, a bounded suboptimal algorithm in sparse and medium densities. On the other hand, MC-CBS+ is outperformed by a state-of-the-art eventually optimal solver, LaCAM*, in most cases. Nonetheless, we assert that the findings presented in this paper indicate that sampling-based approaches have the potential to effectively enhance both existing and future CBS variants (e.g., AlphaGo (Silver et al. 2016)-like techniques on CT). Thus we believe that sampling-based approaches should be viewed favorably by the MAPF community.

References

Atzmon, D.; Stern, R.; Felner, A.; Sturtevant, N. R.; and Koenig, S. 2020. Probabilistic robust multi-agent path finding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 29–37.

Barer, M.; Sharon, G.; Stern, R.; and Felner, A. 2014. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Proceedings of the international symposium on combinatorial Search*, volume 5, 19–27.

Boyarski, E.; Felner, A.; Harabor, D.; Stuckey, P. J.; Cohen, L.; Li, J.; and Koenig, S. 2020. Iterative-deepening conflict-based search. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 4084–4090.

Boyarski, E.; Felner, A.; Sharon, G.; and Stern, R. 2015a. Don't split, try to work it out: Bypassing conflicts in multi-agent pathfinding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 25, 47–51.

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Betzalel, O.; Tolpin, D.; and Shimony, E. 2015b. ICBS: The improved conflict-based search algorithm for multi-agent pathfinding. In *Proceedings of the International Symposium on Combinatorial Search*, volume 6, 223–225.

Cokelaer, T. 2024. cokelaer/fitter: v1.7.1.

Erdmann, M.; and Lozano-Perez, T. 1987. On multiple moving objects. *Algorithmica*, 2: 477–521.

Felner, A.; Li, J.; Boyarski, E.; Ma, H.; Cohen, L.; Kumar, T. S.; and Koenig, S. 2018. Adding heuristics to conflict-based search for multi-agent path finding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, 83–87.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Kleinbort, M.; Solovey, K.; Littlefield, Z.; Bekris, K. E.; and Halperin, D. 2018. Probabilistic completeness of RRT for geometric and kinodynamic planning with forward propagation. *IEEE Robotics and Automation Letters*, 4(2): i–vii.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1): 97–109.

Lam, E.; Le Bodic, P.; Harabor, D. D.; and Stuckey, P. J. 2019. Branch-and-cut-and-price for multi-agent pathfinding. In *International Joint Conference on Artificial Intelligence 2019*, 1289–1296. Association for the Advancement of Artificial Intelligence (AAAI).

Li, J.; Felner, A.; Boyarski, E.; Ma, H.; and Koenig, S. 2019a. Improved Heuristics for Multi-Agent Path Finding with Conflict-Based Search. In *IJCAI*, volume 2019, 442–449.

Li, J.; Gong, M.; Liang, Z.; Liu, W.; Tong, Z.; Yi, L.; Morris, R.; Pasearanu, C.; and Koenig, S. 2019b. Departure scheduling and taxiway path planning under uncertainty. In *AIAA Aviation 2019 Forum*, 2930.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; Gange, G.; and Koenig, S. 2021. Pairwise symmetry reasoning for multi-agent path finding search. *Artificial Intelligence*, 301: 103574.

Li, J.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2019c. Symmetry-breaking constraints for grid-based multi-agent path finding. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, 6087–6095.

Li, J.; Ruml, W.; and Koenig, S. 2021. Eecbs: A bounded-suboptimal search for multi-agent path finding. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, 12353–12362.

Ma, H.; Li, J.; Kumar, T. S.; and Koenig, S. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, 837–845.

Massey, F. J. 1951. The Kolmogorov-Smirnov Test for Goodness of Fit. *Journal of the American Statistical Association*, 46(253): 68–78.

Okoso, A.; Otaki, K.; and Nishi, T. 2019. Multi-agent path finding with priority for cooperative automated valet parking. In *2019 IEEE intelligent transportation systems conference (ITSC)*, 2135–2140. IEEE.

Okumura, K. 2023. LaCAM: Search-Based Algorithm for Quick Multi-Agent Pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 11655–11662.

Okumura, K. 2024. Engineering LaCAM*: Towards Real-time, Large-scale, and Near-optimal Multi-agent Pathfinding. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, 1501–1509.

Okumura, K.; Machida, M.; Défago, X.; and Tamura, Y. 2022. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence*, 310: 103752.

Olea, R.; and Pawlowsky-Glahn, V. 2008. Kolmogorov–Smirnov test for spatially correlated data. *Stochastic Environmental Research and Risk Assessment*, 23: 749–757.

Pearl, J.; and Kim, J. H. 1982. Studies in semi-admissible heuristics. *IEEE transactions on pattern analysis and machine intelligence*, (4): 392–399.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. R. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial intelligence*, 219: 40–66.

Silver, D. 2005. Cooperative pathfinding. In *Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment*, volume 1, 117–122.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587): 484–489.

Skrynnik, A.; Andreychuk, A.; Yakovlev, K.; and Panov, A. 2024. Decentralized Monte Carlo Tree Search for Partially Observable Multi-Agent Pathfinding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 17531–17540.

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.

Surynek, P.; Felner, A.; Stern, R.; and Boyarski, E. 2018. Sub-optimal SAT-based approach to multi-agent path-finding problem. In *Proceedings of the International Symposium on Combinatorial Search*, volume 9, 99–105.

Thayer, J. T.; and Ruml, W. 2011. Bounded suboptimal search: a direct approach using inadmissible estimates. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume One*, 674–679.

Wagner, G.; and Choset, H. 2015. Subdimensional expansion for multirobot path planning. *Artificial intelligence*, 219: 1–24.

Young, I. T. 1977. Proof without prejudice: use of the Kolmogorov-Smirnov test for the analysis of histograms from flow systems and other sources. *Journal of Histochemistry & Cytochemistry*, 25(7): 935–941. PMID: 894009.

Zhang, H.; Li, J.; Surynek, P.; Koenig, S.; and Kumar, T. K. S. 2020. Multi-Agent Path Finding with Mutex Propagation. *Proceedings of the International Conference on Automated Planning and Scheduling*, 30(1): 323–332.

A Experimental Setup

A.1 Agent Density Classification

The evaluation is performed on three different agent densities for each map, based on the agent densities observed in the empirical analysis from Okumura (2023). "Sparse" instances contain exactly 50 agents. "Dense" instances contain the maximum amount of agents where EECBS is able to solve at least one instance in Okumura (2023) or 500 agents, whichever is lower.[5] "Medium" density instances contain half the number of agents as in the dense instances, rounded up in 50-agent intervals. One caveat is the Maze map where EECBS was only able to solve up to 100 agents. To keep intervals evenly spaced for analysis, dense maze instances were defined to have 150 agents. All experiments were coded in C++ and run on Linux CentOS 7 compute nodes with 48 total Intel Xeon 6248R processors and a memory limit of 360GB for each node (7.5GB per processor). Parallel computation was handled by an external Python script.

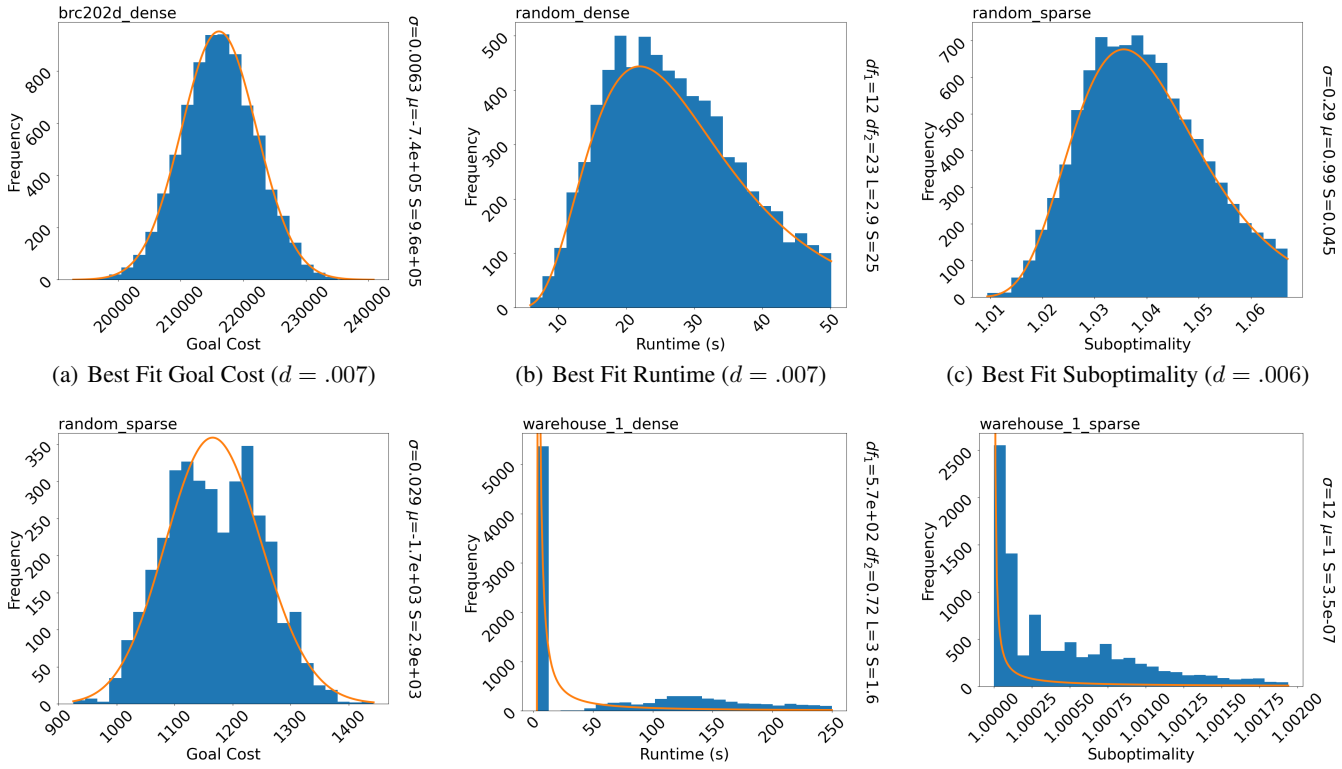
B Goal Node Distribution: Additional Results

This section shows additional results for the goal node distribution of MC-CBS+. Specifically, results over all 19 distributions are presented in Table 3. Figure 4 shows best and worst distributions across scenarios for CT+.

C Empirical Evaluation: Additional Results

This section provides an extended evaluation. Figure 4 to Figure 15 (both numbers inclusive) show plots of success rate, runtime, and suboptimality for each of the scenarios for MC-CBS, MC-CBS+, EECBS+, and LaCAM with increasing rollouts (for MC-CBS and MC-CBS+).

[5]Note that the success rate in our results are not comparable with those reported by Okumura (2023) because we use a larger timeout threshold.



(a) Best Fit Goal Cost ($d = .007$) (b) Best Fit Runtime ($d = .007$) (c) Best Fit Suboptimality ($d = .006$)

(d) Worst Fit Goal Cost ($d = .047$) (e) Worst Fit Runtime ($d = .229$) (f) Worst Fit Suboptimality ($d = .420$)

Figure 4: Constraint Tree Analysis MC-CBS+

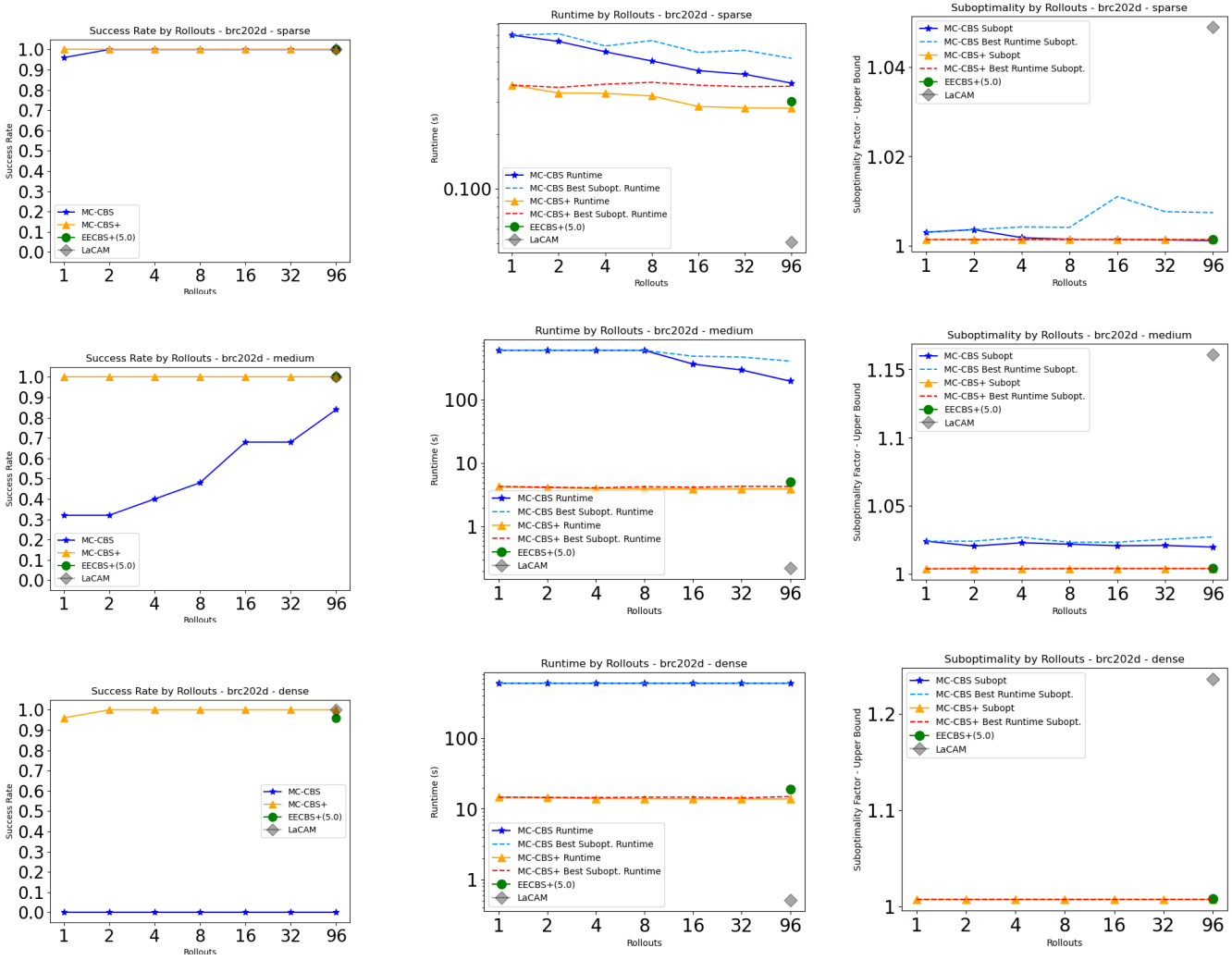


Figure 5: Comparison between MC-CBS, MC-CBS+, EECBS+, and LaCAM with increasing rollouts for MC-CBS and MC-CBS+. 1/11

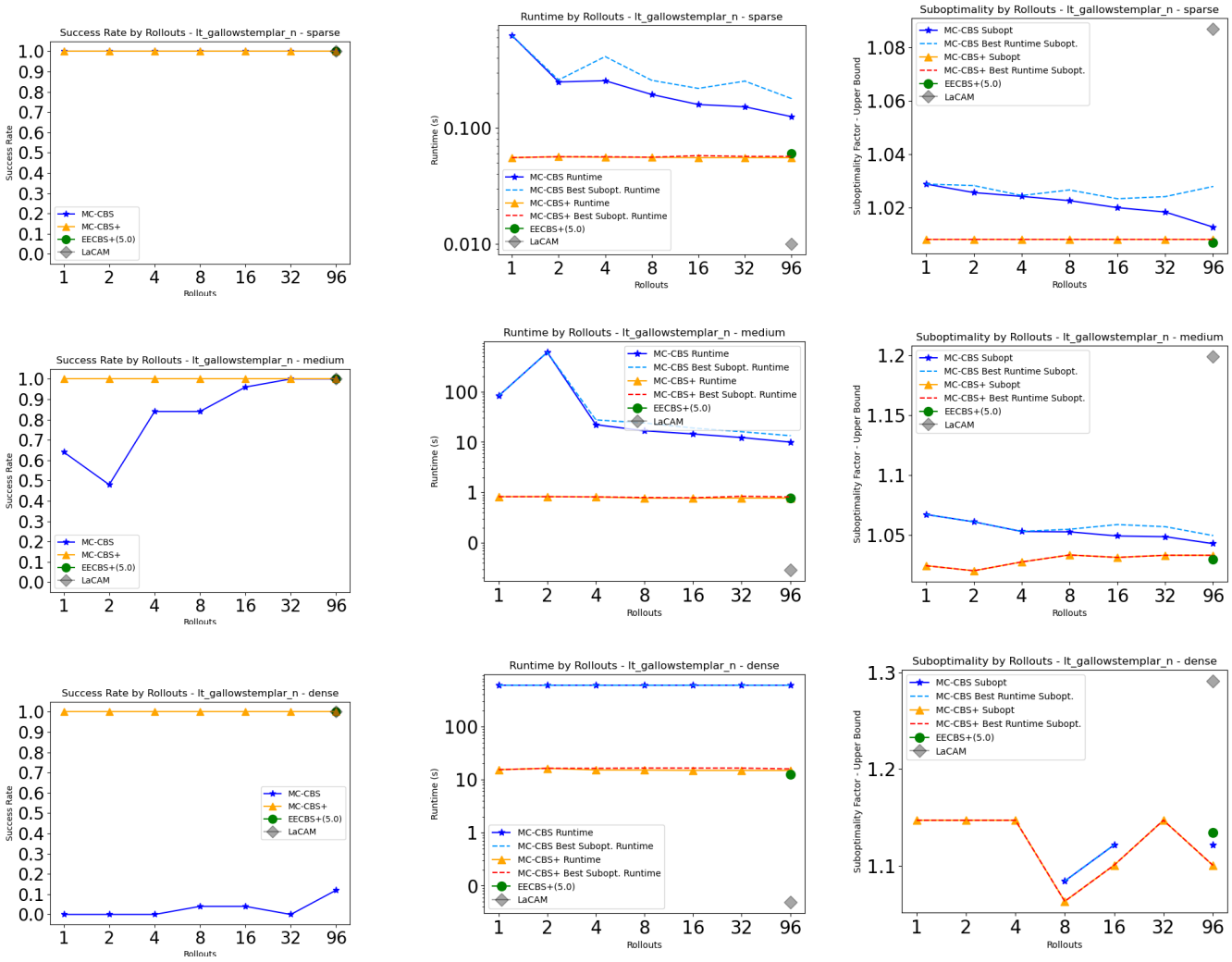


Figure 6: Comparison between MC-CBS, MC-CBS+, EECBS+, and LaCAM with increasing rollouts for MC-CBS and MC-CBS+. 2/11

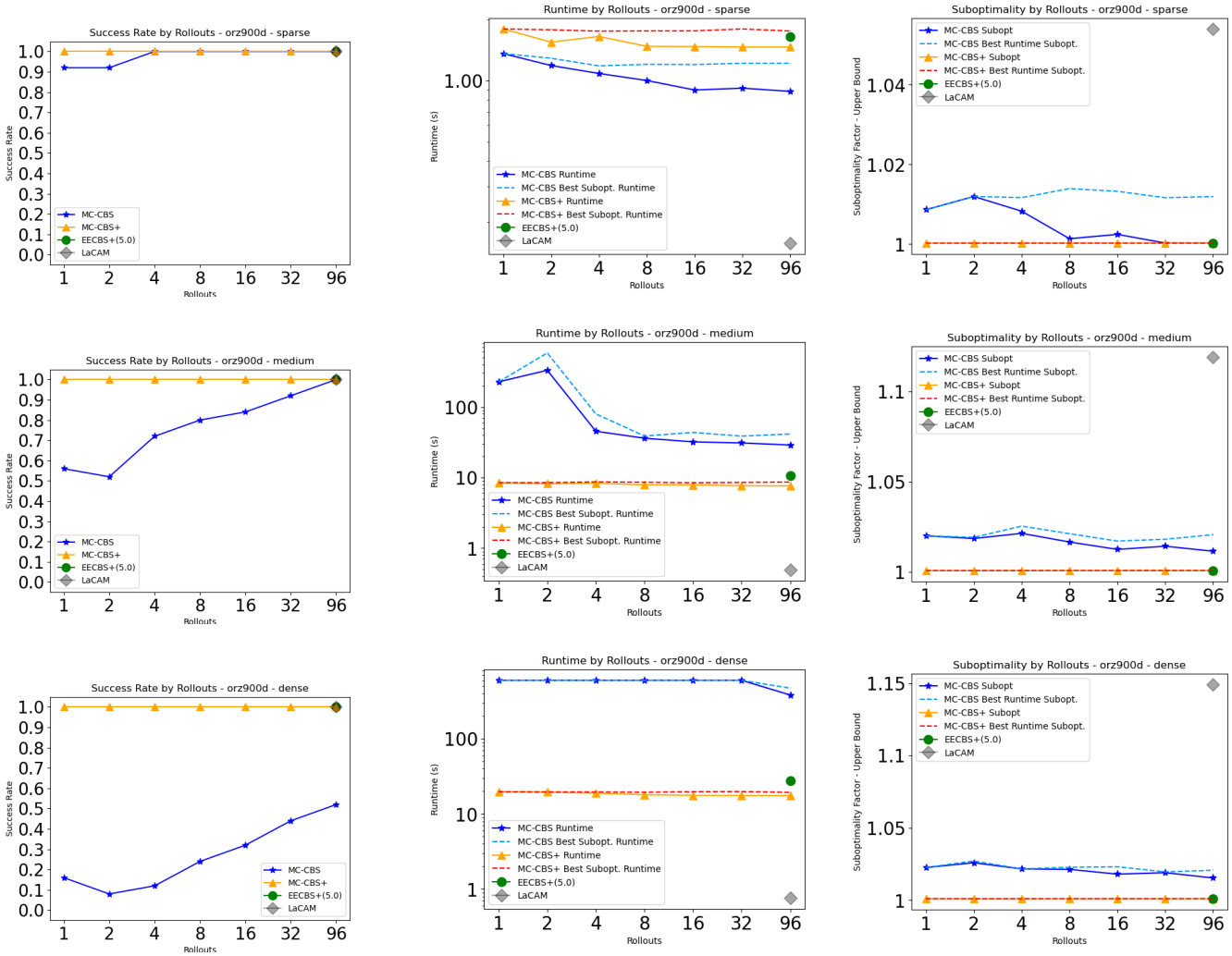


Figure 7: Comparison between MC-CBS, MC-CBS+, EECBS+, and LaCAM with increasing rollouts for MC-CBS and MC-CBS+. 3/11

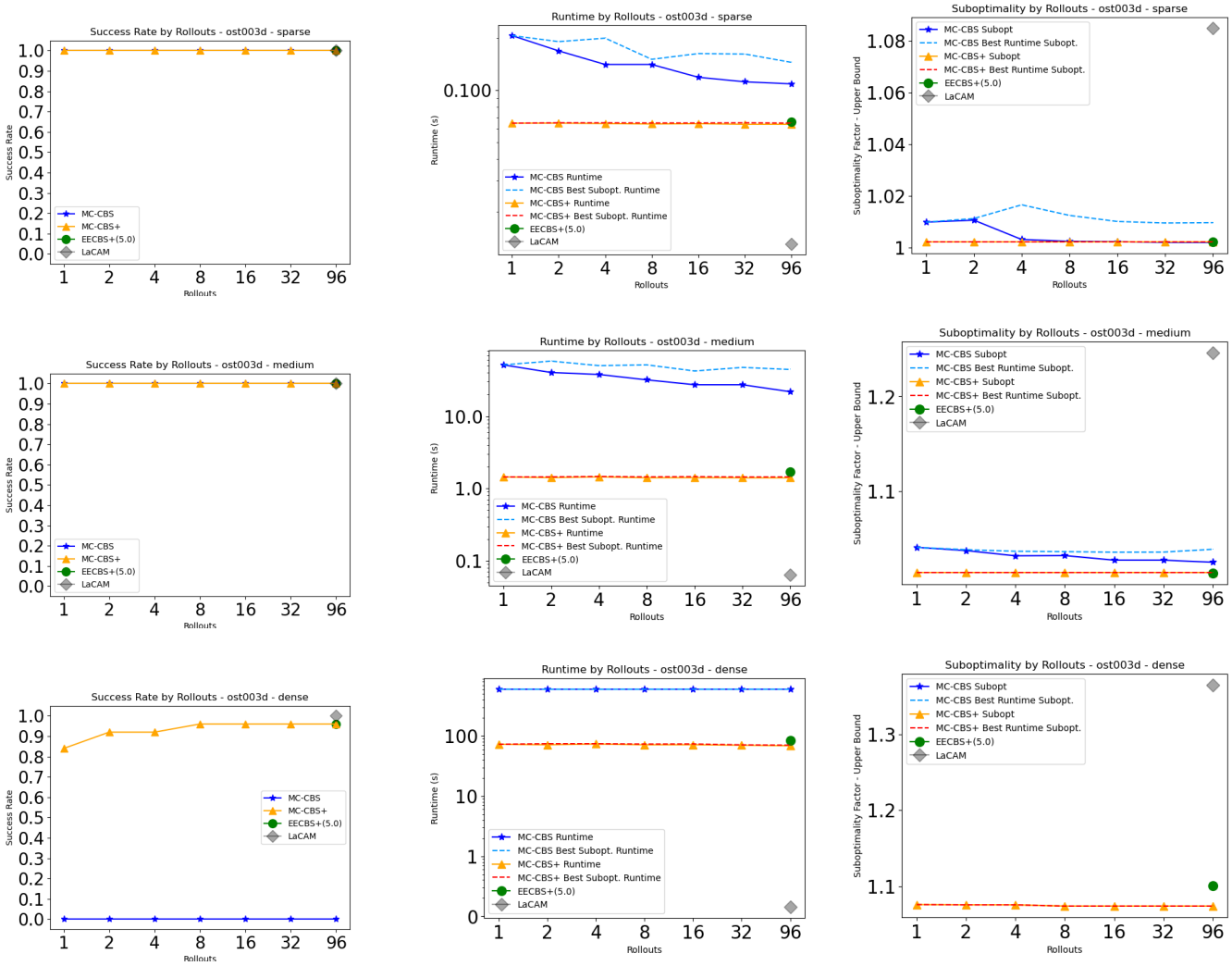


Figure 8: Comparison between MC-CBS, MC-CBS+, EECBS+, and LaCAM with increasing rollouts for MC-CBS and MC-CBS+. 4/11

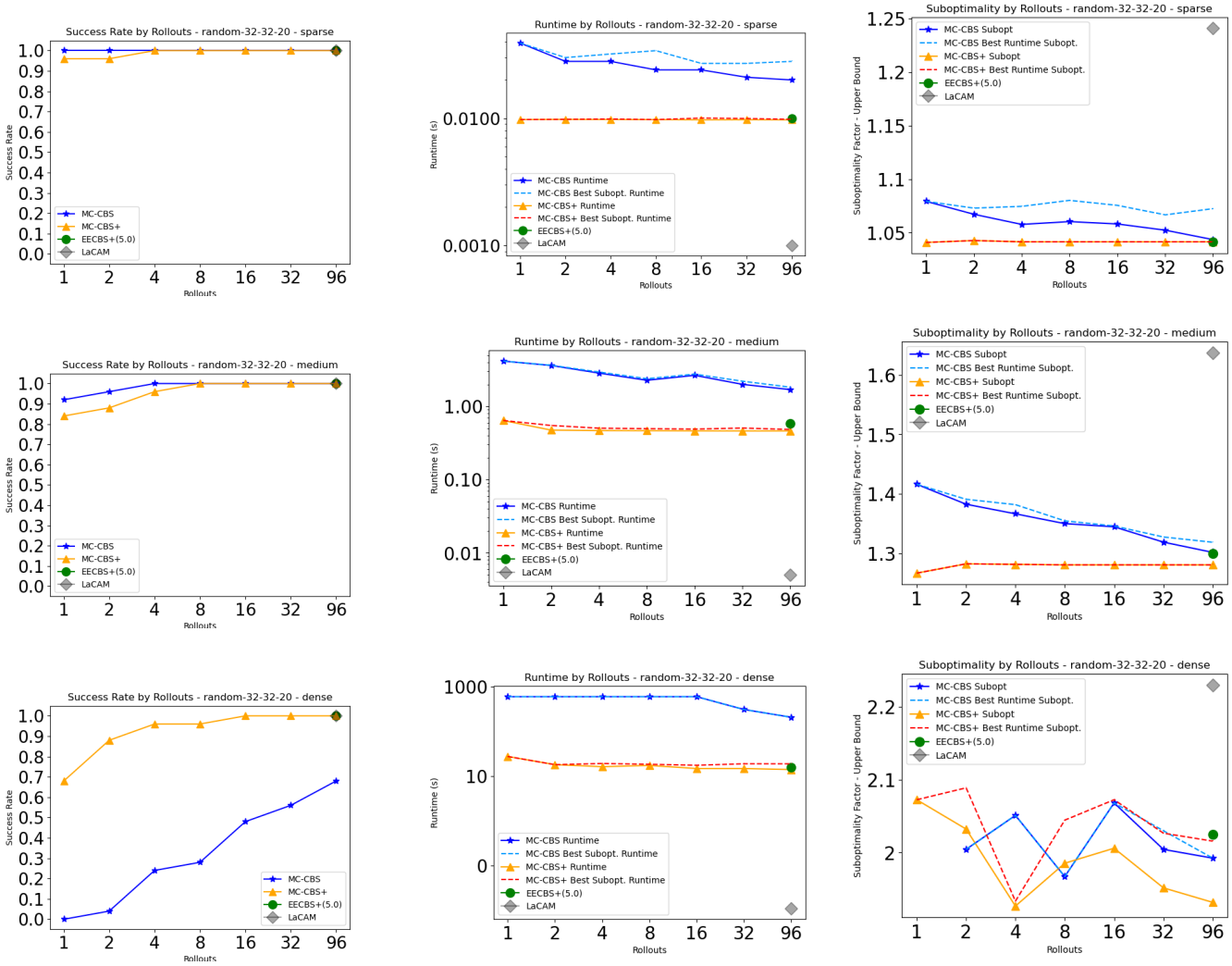


Figure 9: Comparison between MC-CBS, MC-CBS+, EECBS+, and LaCAM with increasing rollouts for MC-CBS and MC-CBS+. 5/11

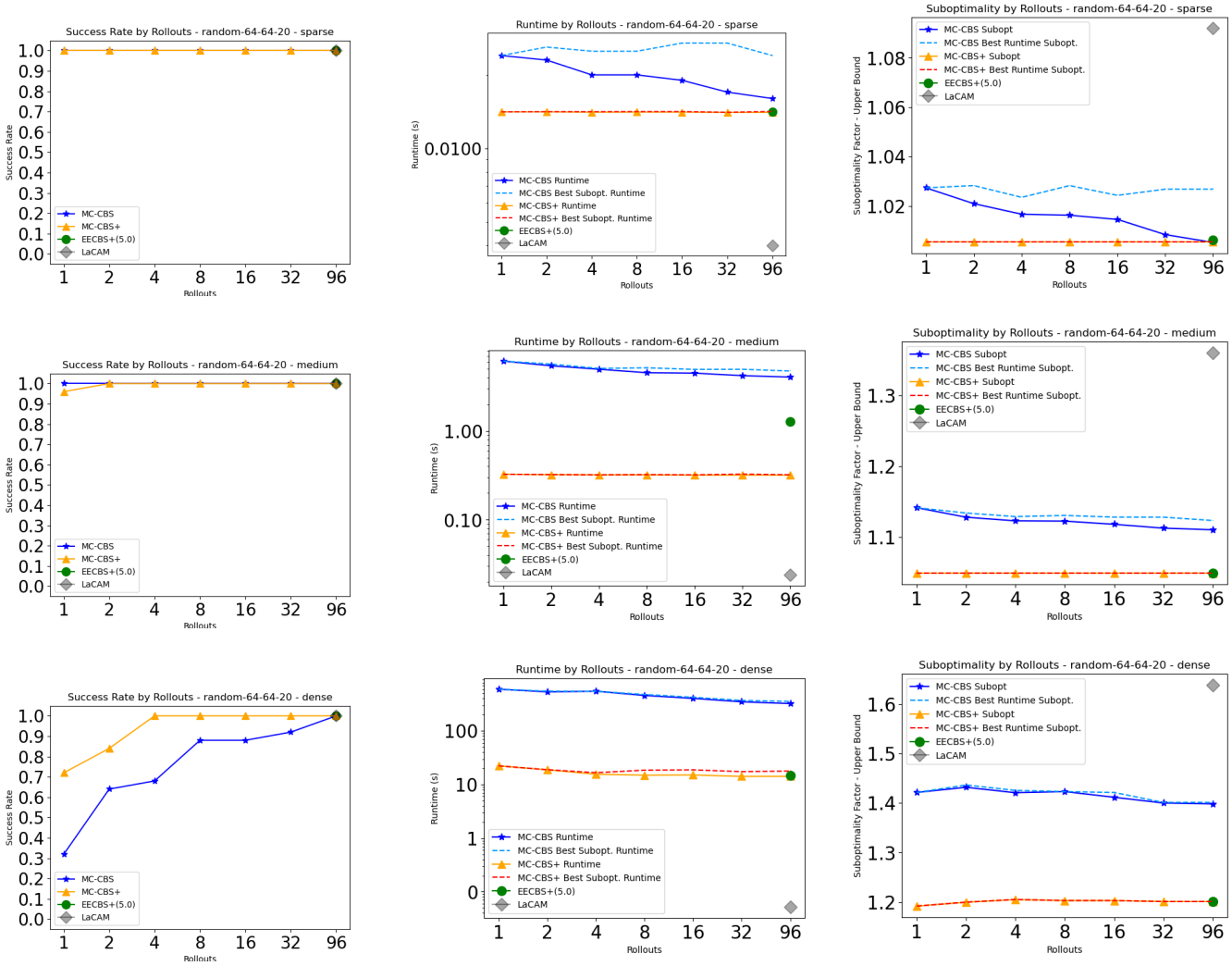


Figure 10: Comparison between MC-CBS, MC-CBS+, EECBS+, and LaCAM with increasing rollouts for MC-CBS and MC-CBS+. 6/11

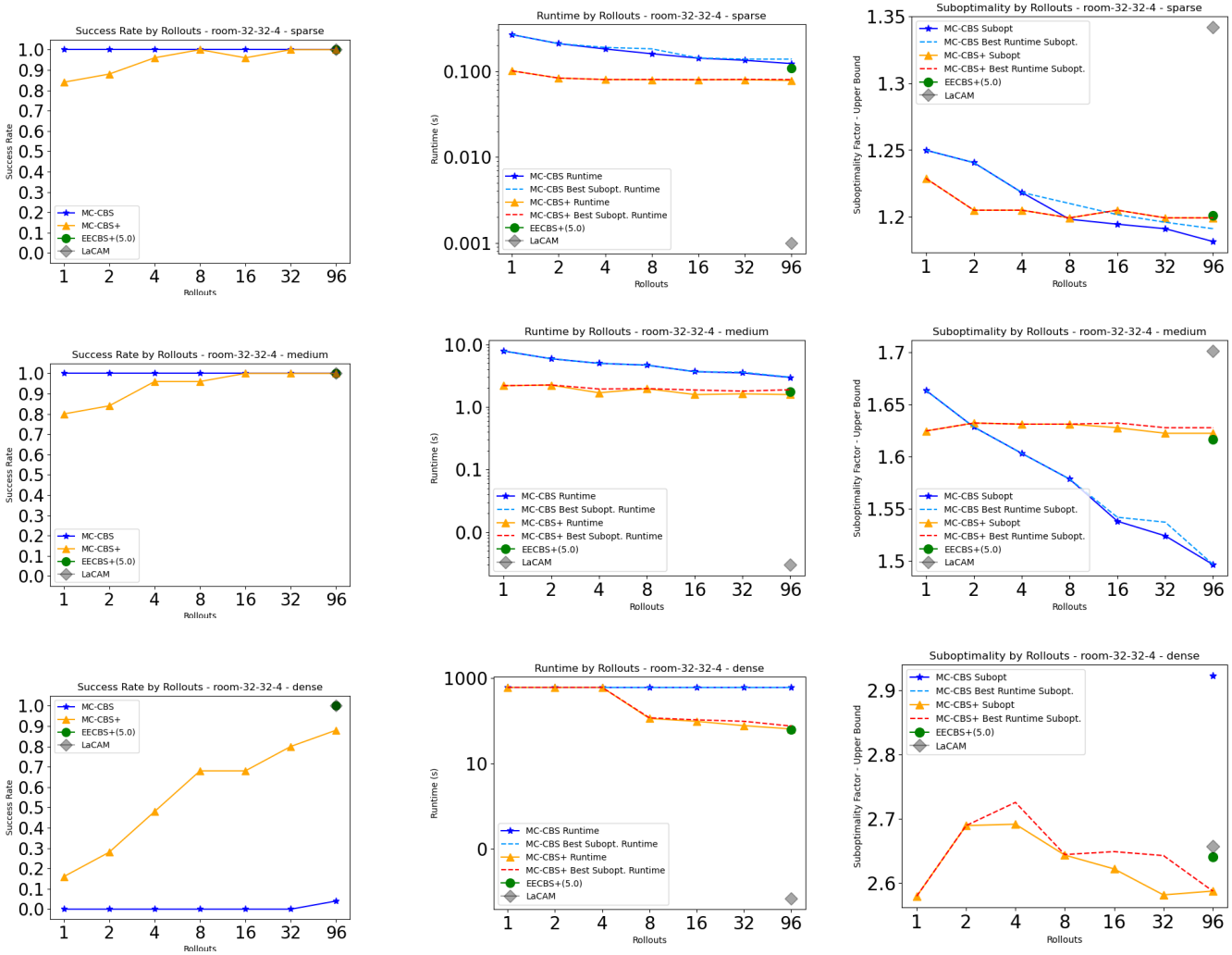


Figure 11: Comparison between MC-CBS, MC-CBS+, EECBS+, and LaCAM with increasing rollouts for MC-CBS and MC-CBS+. 7/11

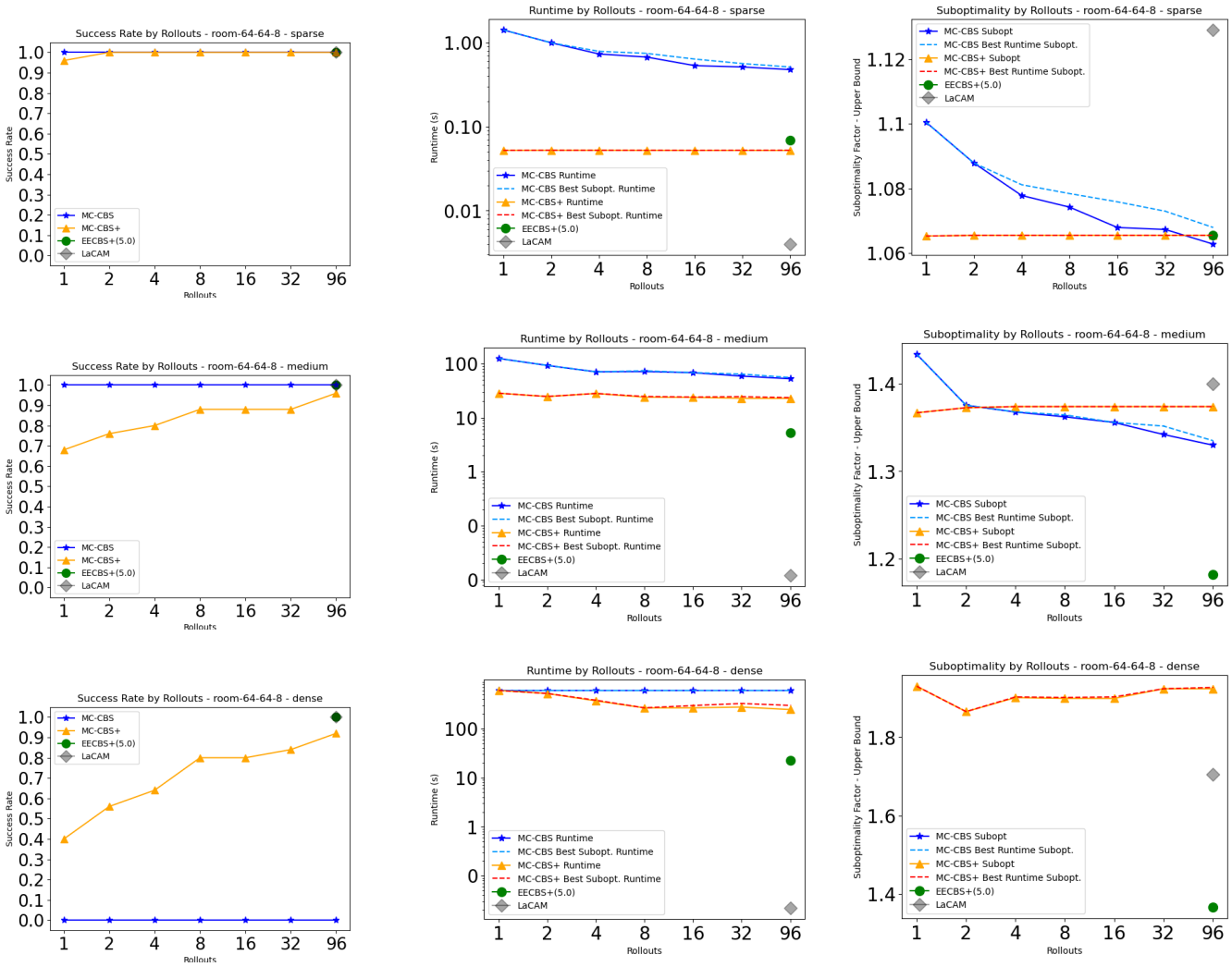


Figure 12: Comparison between MC-CBS, MC-CBS+, EECBS+, and LaCAM with increasing rollouts for MC-CBS and MC-CBS+. 8/11

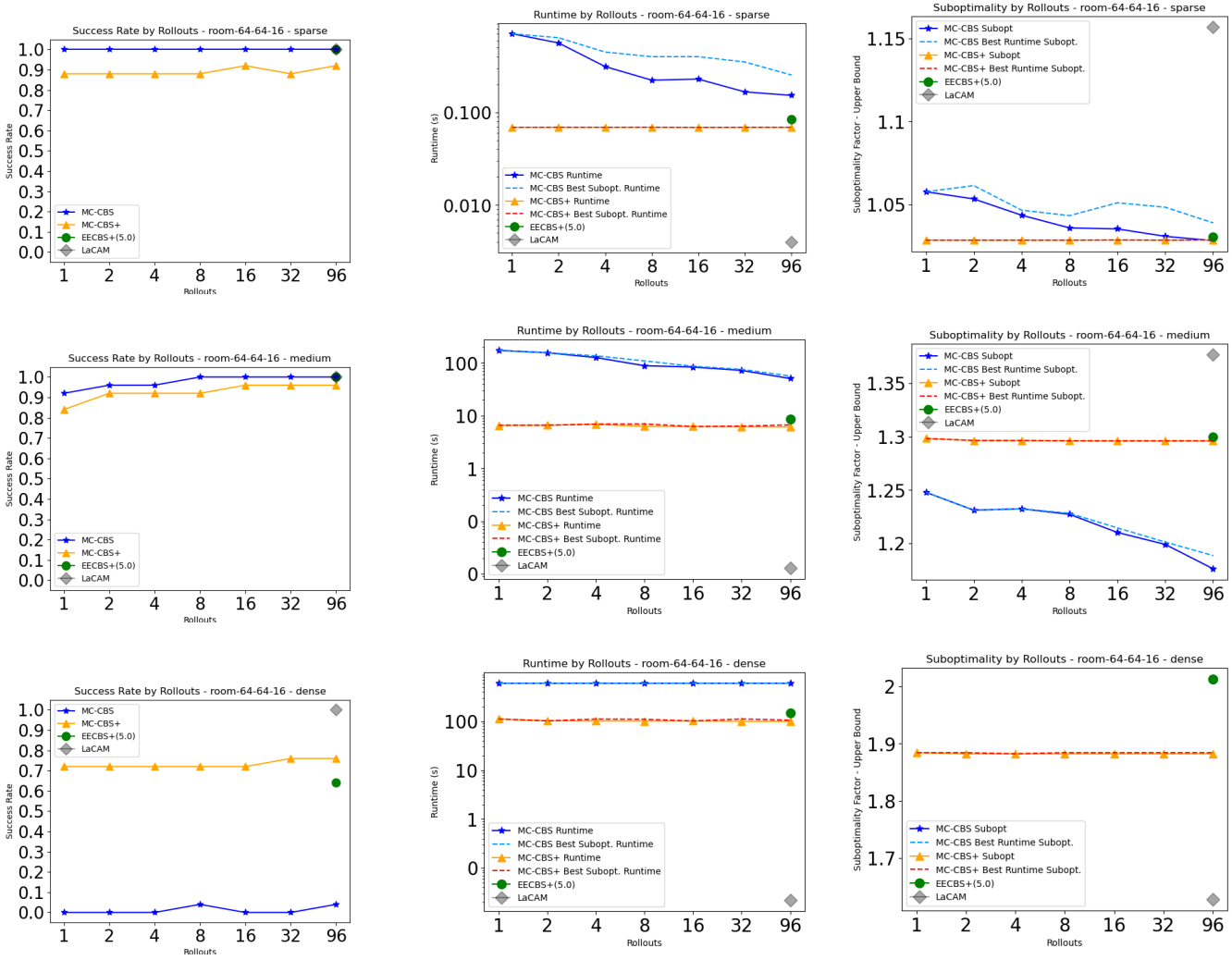


Figure 13: Comparison between MC-CBS, MC-CBS+, EECBS+, and LaCAM with increasing rollouts for MC-CBS and MC-CBS+. 9/11

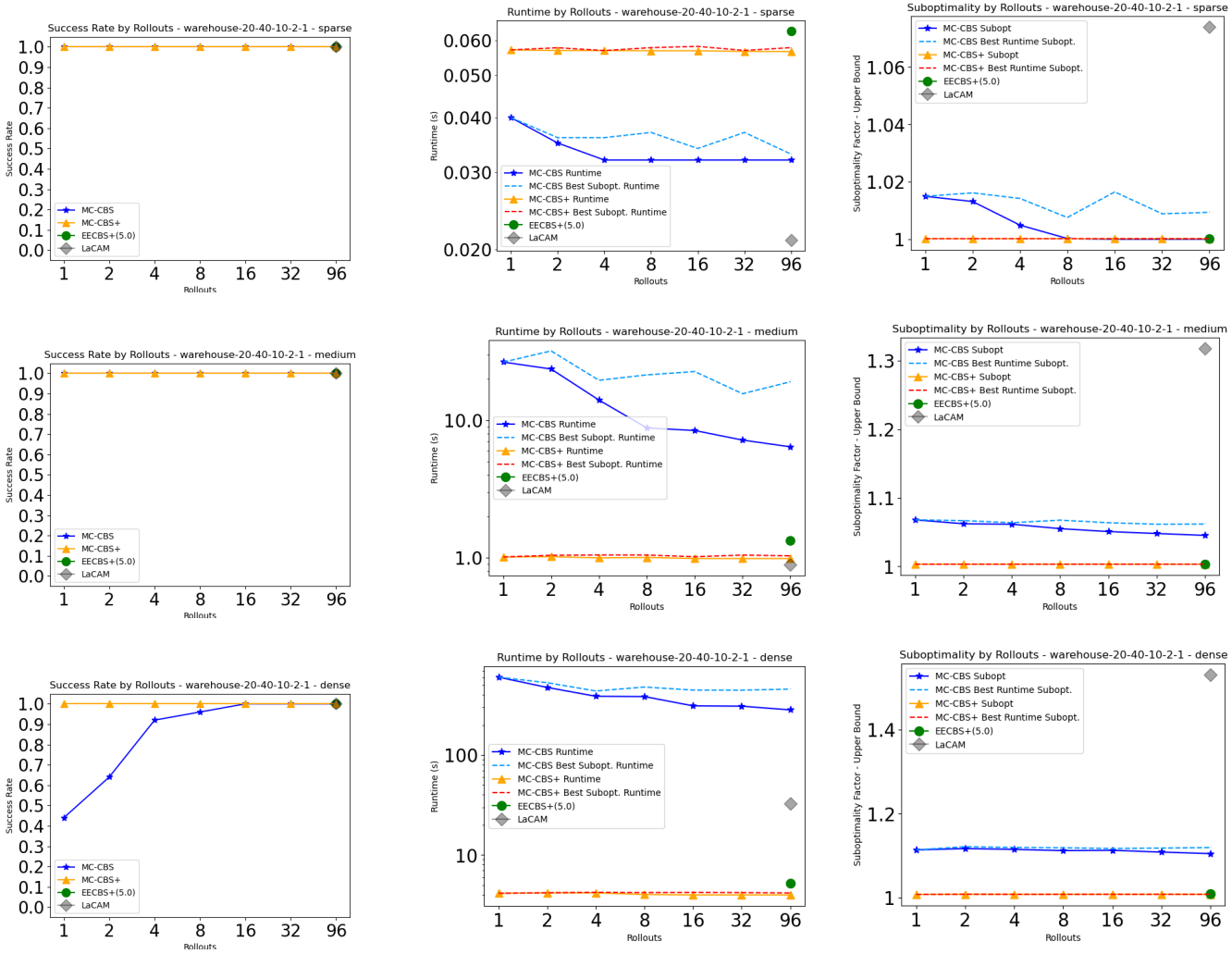


Figure 14: Comparison between MC-CBS, MC-CBS+, EECBS+, and LaCAM with increasing rollouts for MC-CBS and MC-CBS+. 10/11

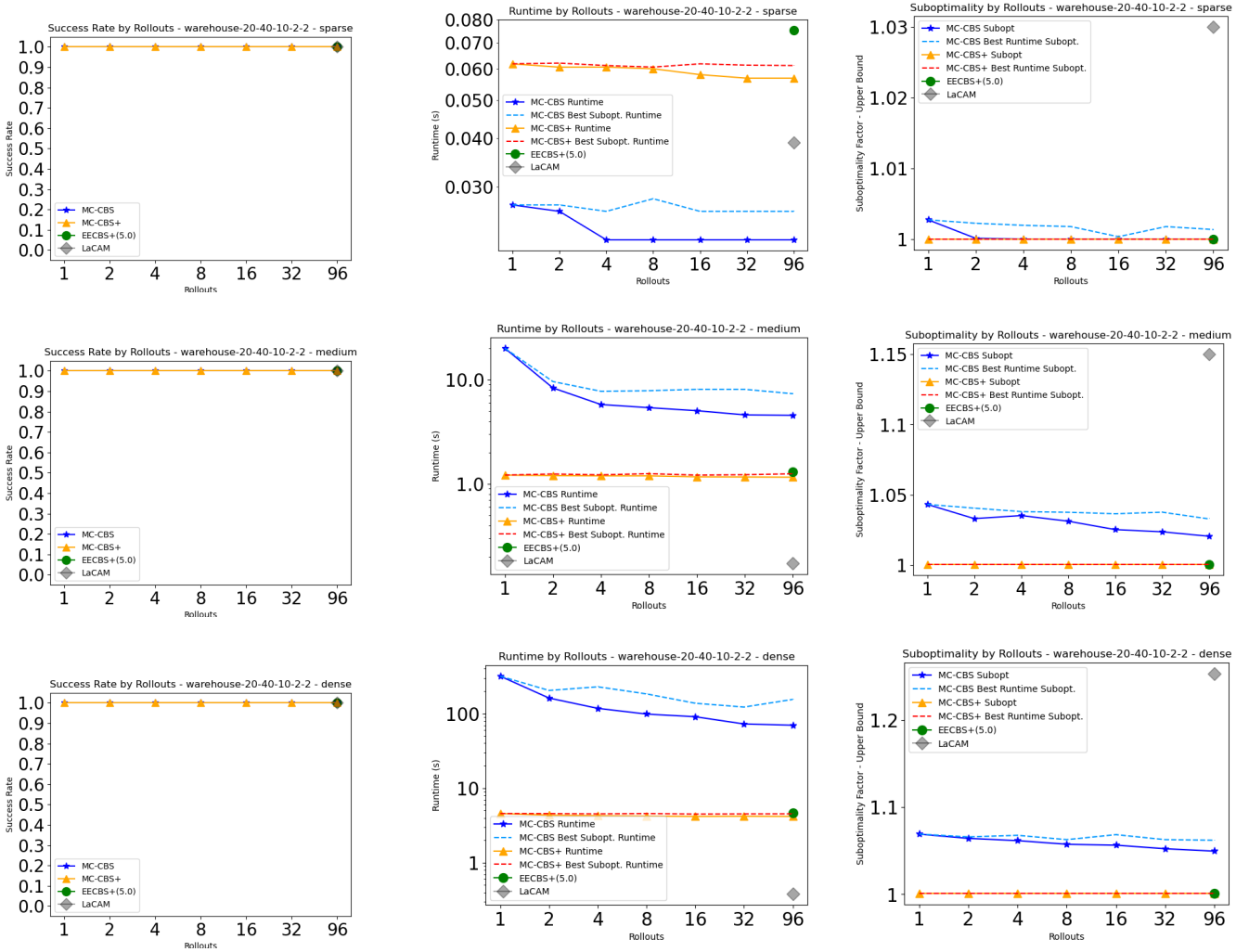


Figure 15: Comparison between MC-CBS, MC-CBS+, EECBS+, and LaCAM with increasing rollouts for MC-CBS and MC-CBS+. 11/11

Method	Dist.	Goal Costs			Runtimes			Suboptimality		
		Max	**Mean**	**SAT**	**Max**	**Mean**	**SAT**	**Max**	**Mean**	**SAT**
CT-	Cauchy	0.121	0.084	0/35	0.230	0.160	0/35	0.367	0.135	0/35
	Chi^2	1.000	0.266	2/35	0.999	0.343	0/35	0.816	0.329	7/35
	Expon.	0.393	0.353	0/35	0.514	0.236	0/35	0.469	0.296	0/35
	Expon.-Pow.	1.000	0.995	0/35	1.000	0.575	0/35	0.540	0.155	0/35
	Gamma	0.049	0.018	16/31	1.000	0.650	1/31	0.518	0.080	19/31
	Log-Norm	0.048	0.018	16/31	0.138	0.045	8/31	0.398	0.063	20/31
	Normal	0.083	0.022	13/31	0.472	0.250	0/31	0.290	0.075	1/31
	Rayleigh	0.245	0.187	0/35	0.554	0.260	0/35	0.352	0.148	0/35
	Uniform	0.507	0.264	0/35	0.988	0.718	0/35	0.715	0.433	0/35
	Alpha	0.060	0.025	3/35	0.135	0.047	5/35	0.289	0.067	10/35
	F	0.050	0.021	14/31	0.106	0.041	8/31	0.252	0.039	20/31
	Beta	0.040	0.015	19/31	0.331	0.112	1/31	1.000	0.077	20/31
	Weibull	0.053	0.021	12/35	0.684	0.200	0/35	0.347	0.077	1/35
	Pareto	0.612	0.409	0/35	0.613	0.252	0/35	0.469	0.288	0/35
	T	0.086	0.022	11/35	0.231	0.143	0/35	0.340	0.077	3/35
	Logistic	0.063	0.030	0/35	0.416	0.193	0/35	0.312	0.071	0/35
	Erlang	0.044	0.018	18/35	0.999	0.216	0/35	0.583	0.083	18/35
	Chi	1.000	0.598	2/35	1.000	0.422	0/35	0.476	0.126	13/35
CT+	Cauchy	0.109	0.085	0/35	0.439	0.157	0/35	0.500	0.117	0/35
	Chi^2	1.000	0.325	4/35	0.548	0.215	1/35	0.918	0.420	2/35
	Expon.	0.411	0.353	0/35	0.796	0.316	0/35	0.509	0.304	0/35
	Expon.-Pow.	1.000	0.997	0/35	0.874	0.396	0/35	0.643	0.360	0/35
	Gamma	1.000	0.072	20/35	0.998	0.453	1/35	0.950	0.276	9/35
	Log-Norm	0.047	0.016	22/35	0.281	0.070	7/35	0.420	0.038	20/35
	Normal	0.075	0.027	14/35	0.510	0.275	0/35	0.284	0.085	0/35
	Powerlaw	0.390	0.322	0/35	0.577	0.409	0/35	0.540	0.382	0/35
	Rayleigh	0.264	0.185	0/35	0.597	0.321	0/35	0.353	0.132	0/35
	Uniform	0.477	0.302	0/35	0.998	0.761	0/35	0.915	0.563	0/35
	Alpha	0.047	0.022	14/35	0.413	0.075	7/35	0.562	0.100	10/35
	F	0.052	0.022	12/35	0.229	0.051	11/35	0.619	0.074	16/35
	Beta	1.000	0.071	22/35	0.765	0.172	1/35	1.000	0.217	8/35
	Weibull	0.056	0.026	3/35	0.819	0.224	0/35	0.820	0.223	0/35
	Pareto	0.617	0.429	0/35	0.389	0.203	0/35	0.509	0.304	0/35
	T	0.819	0.052	12/35	0.347	0.128	1/35	0.262	0.061	1/35
	Logistic	0.055	0.029	1/35	0.454	0.207	0/35	0.269	0.057	0/35
	Erlang	0.342	0.027	23/35	0.939	0.219	1/35	0.781	0.310	9/35
	Chi	1.000	0.776	0/35	0.960	0.270	0/35	0.591	0.281	4/35

Table 3: Comparison of Kolmogorov-Smirnov distances aggregated over the scenarios for 19 distributions. The reported values include the mean K-S statistic value, the max value (demonstrating the worst fit), and "SAT", the fraction of scenarios achieving a K-S statistic value that is below 0.0163, representing a good fit.

Map	Density	Metric	MC-CBS_R	MC-CBS_S	MC-CBS+_R	MC-CBS+_S	EECBS+
orz900d	sparse	rate	**1.000**	**1.000**	1.000	1.000	1.000
		subopt	**1.012**	**1.000**	1.000	1.000	1.000
		runtime	**0.883**	**1.215**	1.460	1.751	1.697
	medium	rate	1.000	1.000	**1.000**	1.000	1.000
		subopt	1.021	1.011	**1.001**	1.001	1.001
		runtime	28.898	41.437	**7.624**	8.646	10.722
	dense	rate	0.520	0.520	**1.000**	1.000	1.000
		subopt	1.021	1.015	**1.001**	1.001	1.001
		runtime	383.338	472.575	**17.584**	19.372	27.292
maze-32-32-2	sparse	rate	**1.000**	1.000	1.000	1.000	1.000
		subopt	**1.247**	1.227	1.225	1.225	1.250
		runtime	**0.777**	0.911	1.715	1.927	1.475
	medium	rate	0.960	0.960	0.920	0.920	**1.000**
		subopt	1.599	1.586	1.990	1.909	**2.094**
		runtime	14.725	16.488	46.235	92.706	**50.446**
	dense	rate	0.200	0.200	0.400	0.400	**0.720**
		subopt	2.494	2.494	2.870	2.870	**3.007**
		runtime	600.000	600.000	600.000	600.000	**433.838**
warehouse-20-40-10-2-2	sparse	rate	**1.000**	**1.000**	1.000	1.000	1.000
		subopt	**1.001**	**1.000**	1.000	1.000	1.000
		runtime	**0.022**	**0.026**	0.057	0.061	0.075
	medium	rate	1.000	1.000	**1.000**	1.000	1.000
		subopt	1.033	1.020	**1.001**	1.001	1.001
		runtime	4.551	7.331	**1.159**	1.250	1.296
	dense	rate	1.000	1.000	**1.000**	1.000	1.000
		subopt	1.062	1.050	**1.001**	1.001	1.001
		runtime	69.865	155.098	**4.170**	4.522	4.688
random-32-32-20	sparse	rate	1.000	1.000	**1.000**	**1.000**	**1.000**
		subopt	1.072	1.043	**1.041**	**1.041**	**1.041**
		runtime	0.020	0.028	**0.010**	**0.010**	**0.010**
	medium	rate	1.000	1.000	**1.000**	1.000	1.000
		subopt	1.319	1.302	**1.281**	1.281	1.300
		runtime	1.697	1.830	**0.463**	0.484	0.588
	dense	rate	0.680	0.680	**1.000**	**1.000**	1.000
		subopt	1.992	1.992	**2.016**	**1.931**	2.024
		runtime	208.277	208.277	**14.045**	**18.898**	16.191
room-64-64-8	sparse	rate	1.000	**1.000**	**1.000**	**1.000**	1.000
		subopt	1.068	**1.063**	**1.066**	**1.066**	1.066
		runtime	0.476	**0.512**	**0.052**	**0.052**	0.069
	medium	rate	1.000	1.000	0.960	0.960	**1.000**
		subopt	1.183	1.177	1.177	1.177	**1.182**
		runtime	7.830	7.863	4.722	4.826	**5.302**
	dense	rate	**1.000**	1.000	0.960	0.960	**1.000**
		subopt	**1.335**	1.33	1.374	1.374	**1.367**
		runtime	**52.573**	55.354	22.733	23.482	**22.212**
random-64-64-20	sparse	rate	1.000	1.000	**1.000**	**1.000**	1.000
		subopt	1.027	1.005	**1.005**	**1.005**	1.006
		runtime	0.016	0.024	**0.014**	**0.014**	0.014
	medium	rate	1.000	1.000	**1.000**	1.000	1.000
		subopt	1.123	1.11	**1.049**	1.049	1.049
		runtime	4.058	4.751	**0.318**	0.322	1.263
	dense	rate	1.000	1.000	**1.000**	1.000	1.000
		subopt	1.401	1.398	**1.201**	1.201	1.201
		runtime	324.062	355.159	**14.171**	17.823	14.866

Table 4: Comparison between MC-CBS, MC-CBS+ and EECBS+. 96 rollouts are used for MC-CBS and MC-CBS+, Table 1/2

Map	Density	Metric	MC-CBS_R	MC-CBS_S	MC-CBS+_R	MC-CBS+_S	EECBS+
room-32-32-4	sparse	rate	1.000	**1.000**	**1.000**	1.000	1.000
		subopt	1.191	**1.181**	**1.199**	1.199	1.201
		runtime	0.122	**0.138**	**0.078**	0.080	0.106
	medium	rate	1.000	1.000	1.000	1.000	1.000
		subopt	1.496	1.496	1.627	1.622	1.616
		runtime	2.956	2.985	1.577	1.882	1.726
	dense	rate	0.040	0.040	0.880	0.880	**1.000**
		subopt	2.923	2.923	2.587	2.587	**2.641**
		runtime	600.000	600.000	64.260	74.990	**62.700**
room-64-64-16	sparse	rate	1.000	1.000	0.920	0.920	**1.000**
		subopt	1.039	1.028	1.028	1.028	**1.030**
		runtime	0.153	0.253	0.069	0.069	**0.085**
	medium	rate	1.000	1.000	0.960	0.960	**1.000**
		subopt	1.189	1.176	1.296	1.296	**1.300**
		runtime	50.708	56.594	6.115	6.702	**8.745**
	dense	rate	0.040	0.040	0.760	0.760	0.640
		subopt	nan	nan	nan	nan	2.013
		runtime	600.000	600.000	99.435	105.582	148.242
ost003d	sparse	rate	1.000	1.000	**1.000**	1.000	1.000
		subopt	1.010	1.002	**1.002**	1.002	1.002
		runtime	0.109	0.145	**0.064**	0.065	0.066
	medium	rate	1.000	1.000	**1.000**	1.000	1.000
		subopt	1.039	1.025	**1.014**	1.014	1.014
		runtime	21.831	44.329	**1.406**	1.448	1.699
	dense	rate	0.000	0.000	0.960	0.960	0.960
		subopt	nan	nan	nan	nan	1.100
		runtime	600.000	600.000	68.783	70.238	84.920
warehouse-20-40-10-2-1	sparse	rate	**1.000**	**1.000**	1.000	1.000	1.000
		subopt	**1.009**	**1.000**	1.000	1.000	1.000
		runtime	**0.032**	**0.033**	0.057	0.058	0.064
	medium	rate	1.000	1.000	**1.000**	1.000	1.000
		subopt	1.062	1.045	**1.003**	1.003	1.003
		runtime	6.411	19.047	**0.988**	1.035	1.334
	dense	rate	1.000	1.000	**1.000**	1.000	1.000
		subopt	1.116	1.103	**1.008**	1.008	1.009
		runtime	283.248	438.333	**3.991**	4.175	5.179
brc202d	sparse	rate	1.000	1.000	**1.000**	1.000	1.000
		subopt	1.007	1.001	**1.001**	1.001	1.001
		runtime	0.381	0.521	**0.277**	0.366	0.295
	medium	rate	0.840	0.840	**1.000**	1.000	1.000
		subopt	1.027	1.020	**1.004**	1.004	1.004
		runtime	196.502	405.75	**3.895**	4.286	5.106
	dense	rate	0.000	0.000	1.000	1.000	0.960
		subopt	nan	nan	nan	nan	1.008
		runtime	600.0	600.0	13.771	14.925	18.923
gallowstemplar_n	sparse	rate	1.000	1.000	**1.000**	1.000	**1.000**
		subopt	1.028	1.013	**1.008**	1.008	**1.007**
		runtime	0.125	0.179	**0.055**	0.057	**0.060**
	medium	rate	0.960	0.960	1.000	1.000	**1.000**
		subopt	1.052	1.044	1.031	1.031	**1.029**
		runtime	7.830	13.311	0.770	0.814	**0.770**
	dense	rate	0.120	0.120	**1.000**	1.000	**1.000**
		subopt	1.122	1.122	**1.101**	1.101	**1.134**
		runtime	600.000	600.000	**14.866**	16.015	**12.860**

Table 5: Comparison between MC-CBS, MC-CBS+ and EECBS+. 96 rollouts are used for MC-CBS and MC-CBS+, Table 2/2