Conflict-Based Search for Explainable Multi-Agent Path Finding

Appendix

We now present supplementary material that provides additional insights to the behavior of our proposed algorithms presented in the paper. We begin by outlining the algorithm for the high-level of XG-CBS. Then, we present a more extensive examples section to further prove the efficacy of our proposed explanation scheme and algorithms. We conclude with an extensive table that shows all the experiments performed by our team, further validating the claims made in the paper.

A. XG-CBS Algorithm

Given an Explainable MAPF instance consisting of a graph G, a list of source $(s_i)_{i=1}^n = (s_1, \ldots, s_n)$, a list of goal vertices $(g_i)_{i=1}^n = (g_1, \ldots, g_n)$, an index bound r, and a path length bound B, XG-CBS proceeds as follows.

First, a root node R is initialized with an empty set of constraints $\mathcal{C}$. Then, the low-level planner is called to find a path for each agent. If the graph search fails to generate a root plan P_r then XG-CBS returns no solution. If, however, a full plan is found, then it is saved in R along with all other important information and and added to the priority queue Q.

While the queue is not empty, XG-CBS selects the highest priority node N, removes it from the queue, and evaluates its plan $N.plan$ for a conflict $(a_i, a_j, v_i, v_j, T_i, T_j)$ between Agent i at (v_i, T_i) and Agent j at (v_j, T_j). Note that segmentation conflicts are included in this definition by letting $v_i = v_j$.

If no conflicts exist for a given nodes plan, then it is returned as the solution. Otherwise, for every agent in the conflict, a new node K is added with a new constraint (a_i, v_i, T_i), and the newly constrained agent is re-planned for using low level graph search. If successful, the new plan and all its information is added to K before it is added Q, where it will eventually be evaluated for conflicts.

Algorithm 1 outlines the pseudocode for XG-CBS. Note that the *graphSearch(·)* procedure only utilizes the initial plan P_r or existing P_{-i} as the chosen low level planner specifies. For example, XG-A^* uses the existing plan as outlined in the paper. However, using A^* only segments the plan after π_i is found but before *graphSearch(·)* returns it.

Algorithm 1: XG-CBS$(G, (s_i)_{i=1}^n, (g_i)_{i=1}^n, r, B)$

```
R.C, Q, P_r ← ∅;
for every agent do
    P_r.add( graphSearch(G, s_i, g_i, R.C, P_r, B) )
if P_r.size() < n then
    return no solution
R.plan, R.index, R.cost ← P_r;
Q.add(R);
while Q not empty do
    N ← Q.highestPriority();
    Q.pop(N); c ← conflictCheck(N.plan, r);
    if c is empty then
        return N.plan
    for every agent a_i ∈ c do
        K.C ← N.C ∪ (a_i, v, T_i);
        P_{-i} ← K.plan \ π_i ;
        π_i ← graphSearch(G, s_i, g_i, K.C, P_{-i}, B);
        if π_i exists then
            P_new ← P_{-i} ∪ π_i;
            K.plan, K.index, K.cost ← P_new;
            Q.add(K);
```

B. XG-A^* Algorithm and Speedups.

Algorithms The algorithm for XG-A^* is shown in Algorithm 2. Below, we present some basic observations that help mitigate the limitation presented in Remark 4.

Speeding Up XG-A^* As demonstrated in Remark 4, XG-A^* may spend a lot of time exhausting the plans of a certain index before making any progress. As we now show, we can alleviate some of the computational cost, using simple observations.

Eliminating Cycles Assume that the graph G allows agents to stay in place, i.e., has self-loops. This is typical in, e.g., warehouse robots. Now, consider a plan $P = \{\pi_1, \ldots, \pi_n\}$ such that in its vertex-disjoint decomposition,

Algorithm 2: XG-$A^*(G, s_i, g_i, \mathcal{C}, P_{-i}, B)$

```
Q ← {s_i};
while Q not empty do
    c ← Q.highestPriority();
    if c.loc = g_i then
        return π_i ← c.Path()
    else
        Q.pop(c); N ← expand(c, G, C, B);
        for every n ∈ N do
            n.index ← Segment(n.Path(), P_{-i});
            if n.index ≤ n.parent.index then
                n.gScore ← n.parent.gScore+1;
                Q.add(n);
            else
                if n.parent.gScore+1 ≤n.gScore then
                    n.gScore ← n.parent.gScore+1;
                    Q.add(n);
```

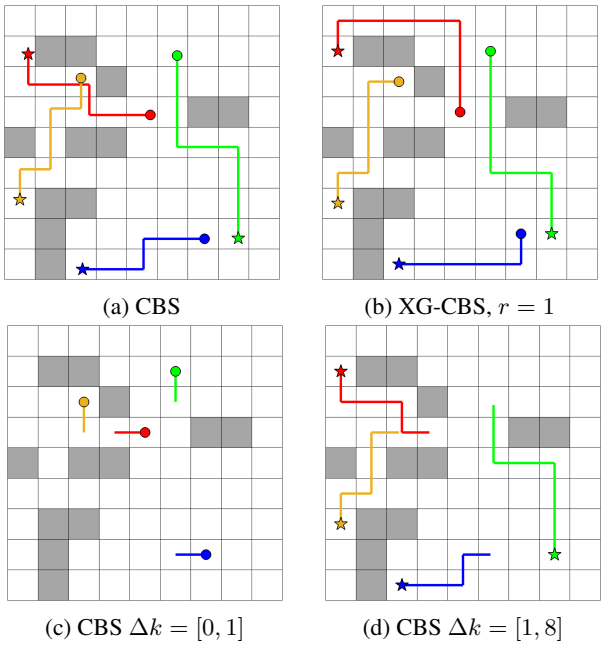
Agent i makes a cycle that is contained entirely within a certain segment. Since paths within a segment are disjoint, we can eliminate this cycle and replace it with Agent i waiting in place for the duration of the cycle. Moreover, we can shift this waiting by a wait in the initial vertex of the segment.

Thus, we observe that any plan can be put in a "normal form" where within each segment, no agent makes a cycle, and staying in place is allowed only on the initial vertex. We use this to speed up XG-A^* by limiting the search space to comply with this condition: it is easy to check whether an agent has a cycle within a segment (except for looping in the initial vertex), as H, the history of the segment, is part of the information of each node.

Shortest Path after Segment Bound Recall that XG-A^* exhausts all the plans with index i before moving to index $i + 1$. We propose a speed up, whereby when the index reaches the bound $\bar{r}$ (index of P_{-1}), the remaining search is performed using standard A^*, i.e., searching for the shortest path to the target, rather than exhausting the remaining plans. Technically, this modification retains the completeness of the algorithm, and hence Theorem 5 is still valid. Intuitively, this offers a speed up since if we already reached an index beyond the given bound, it is unlikely that planning for the current agent helps to reduce segmentation. Therefore, we terminate the search as quickly as possible and allow for further exploration of the conflict tree. We find that empirically, this heuristic speeds up the algorithm.

C. Extended Case Studies

Illustrative Examples (extended) We now further showcase XG-CBS in many settings. We begin by showing how XG-CBS XG-A^* outperforms XG-CBS with A^* for examples where the optimal explanation requires careful tuning of the plan. Then, we incrementally increase the space size, agent number, and solution complexity to show the capabilities of XG-CBS.



(a) CBS (b) XG-CBS, $r = 1$ (c) CBS $\Delta k = [0, 1]$ (d) CBS $\Delta k = [1, 8]$

Figure 8: Example of 1-segment solution with XG-CBS

Figure 8a shows an example of four agents in a 9×9 grid world. Notice that the shortest plan results in a sub-optimal explanation. XG-CBS with XG-A^* easily returns the optimal explanation shown in Figure 8b in 0.04 seconds. Note that XG-CBS with A^* finds a 2-segment solution almost immediately but fails to return the preferred 1-segment solution given a 15 minute planning time threshold. This is due to the increase in path length when attempting to untangle the red and yellow agents.

We now turn to a more interesting example, shown in Figure 9. The shortest plan produces a 6-segment solution due to the natural leader-follower behavior that appears in the shortest plan. Inspecting the full plan in Figure 9a makes it difficult to validate the plan is collision free. The explanation (Figures 9b-9g) clearly shows a collision free plan. However, there are many segments compared to the small example. XG-CBS is capable of producing a much more explainable plan, shown in Figure 9h. Forcing the green agent to wait and the purple agent to take a longer path enables a 2-segment plan (Figures 9i-9j) that is much easier to explain. We note here that XG-CBS with XG-A^* found the 2-segment plan in 128.1 seconds while the XG-CBS using classical A^* once again failed to return a 2-segment solution after a maximum of 15 minutes of planning.

We see one example where XG-CBS with classical A^* performs significantly better than XG-CBS with XG-A^* in Figure 10. Here, XG-CBS with XG-A^* does not return a 2-segment solution within 15 minutes. However, XG-CBS with A^* returns the preferred solution in 0.44 seconds. Notice that the 6-segment plan returned by CBS (Figure 10a) and the 2-segment plan returned by XG-CBS with A^* (Figure 10h) look very similar. This is different from earlier examples we have seen thus far, where decreasing segments

requires a heavy deviation from the shortest paths for individual agents. Experiments show that in examples like these, where we can greatly simplify the explanation with minor tweaking of the shortest path plan, that XG-CBS with A^* outperforms XG-CBS with XG-A^*. We attribute this to the fact that using A^* quickly makes minor changes to the shortest paths while XG-A^* enables large deviations from shortest paths but suffers from computation time as a result.

We witness an opposite behavior in the example shown in Figure 11. The shortest path plan generated by CBS in Figure 11a shows many paths overlapping each other, suggesting a high number of segments. Indeed, the plan requires five disjoint segments. Overlapping paths also suggest that large deviations are required by the agents to drastically decrease segmentation. XG-CBS with XG-A^* returns the plan shown in Figure 11g in 113.9 seconds. As expected, XG-CBS with A^* did not find a solution in 15 minutes of planning.

The decreasing segments becomes increasingly important as the number of agents rises. For example, the CBS solution to a nine agent MAPF problem in a 26×26 grid world (Figure 12a) requires 8 segments to explain (Figure 12b-12i). More interestingly, many of the segments show tiny intervals of the plan. XG-CBS with XG-A^* returns an alternative solution, shown in Figure 12j, that mitigates the length of the explanation scheme. It presents a plan that only requires two segments (Figure 12k-12l).

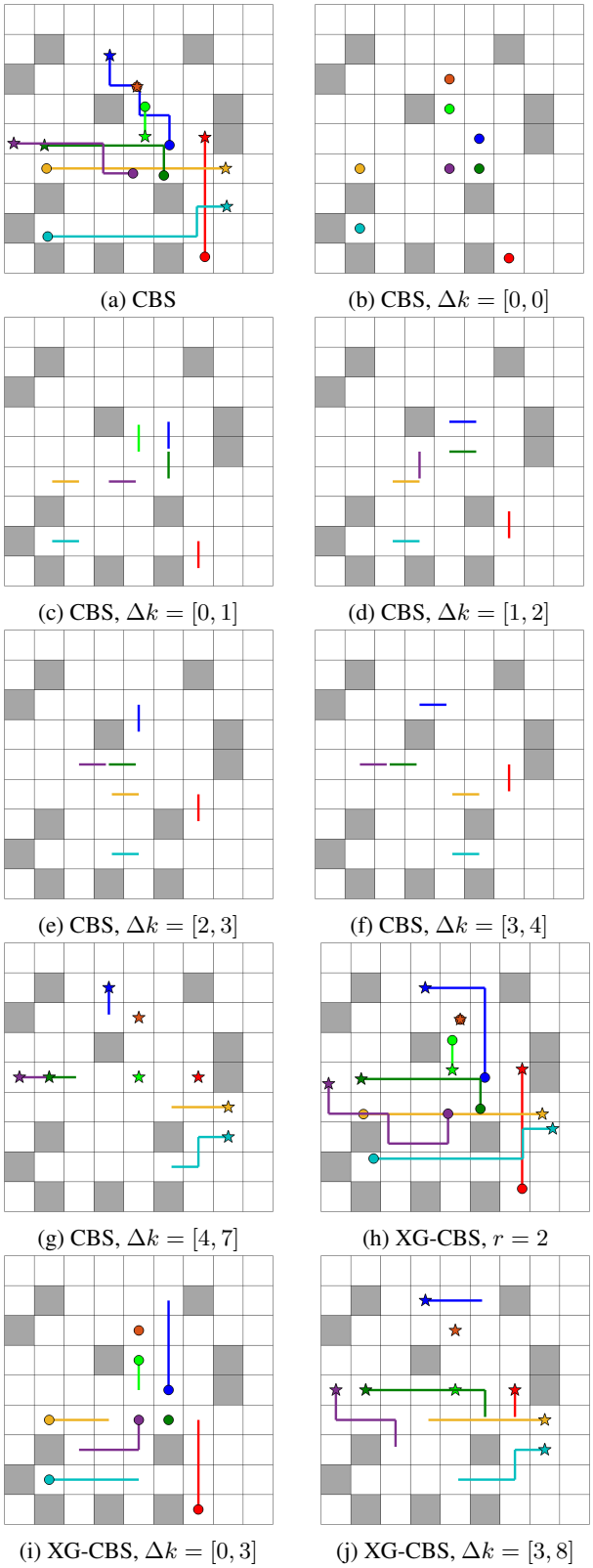
Continuing to scale upwards, we now consider the example in Figure 13 with ten agents in a 33×33 grid world. In 14 seconds, XG-CBS with A^* cuts the explanation by approximately 65%, making it much simpler for a human user to validate the plan.

We conclude this section by testing our algorithms on a MAPF instance consisting of twelve agents in a 33×33 grid world. The results are shown in Figure 14. Notice that the plan returned by CBS (Figure 14a) requires thirteen segments (Figure 14b-14n). However, planing with XG-CBS using A^*, we get the plan shown in Figure 14o which only requires five segments, shown in Figure 14p-14t.

Benchmark Evaluation (extended)

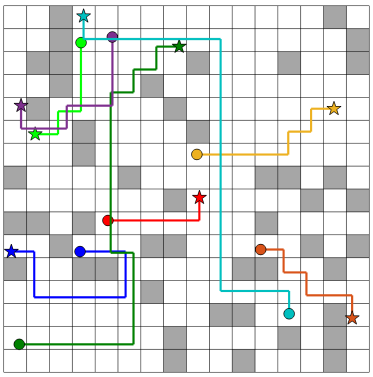
Our benchmarks were on grid worlds with the following sizes and number of agents: 9×9 with 4, 8, 10, and 12 agents, 16×16 with 5, 10, 15, and 20 agents, and 33×33 with 10, 20, and 30 agents. For each grid size and agent number combination, we ran 100 unique experiments, where each experiment consisted of CBS, followed by XG-CBS with each of the low-level algorithms. The timeout for a single algorithm on a single benchmark was 5 minutes (while this may seem like a high threshold, recall that Explainable MAPF is computationally harder than MAPF). The complete results appear in Figures 15, 16, and 17 and contain a comparison of computation time, plan length, segmentation index and success rate.

As can be seen from the trend in computation times, for XG-A^* the computation time is already high for 16×16 environments, with a low success rate. This trend carries on to larger environments, rendering XG-A^* with an extremely low success rate. We therefore do not evaluate XG-A^* and its derivative – WXG-A^* on 33×33 environments.

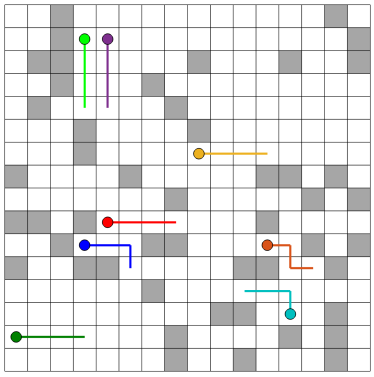


(a) CBS (b) CBS, $\Delta k = [0, 0]$ (c) CBS, $\Delta k = [0, 1]$ (d) CBS, $\Delta k = [1, 2]$ (e) CBS, $\Delta k = [2, 3]$ (f) CBS, $\Delta k = [3, 4]$ (g) CBS, $\Delta k = [4, 7]$ (h) XG-CBS, $r = 2$ (i) XG-CBS, $\Delta k = [0, 3]$ (j) XG-CBS, $\Delta k = [3, 8]$

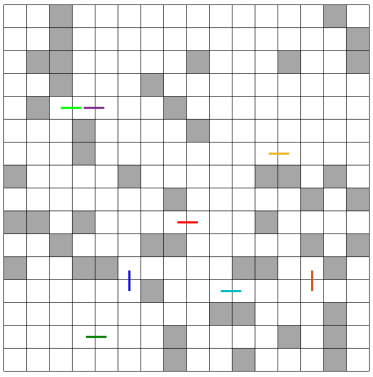
Figure 9: Example of 2-segment solution with XG-CBS



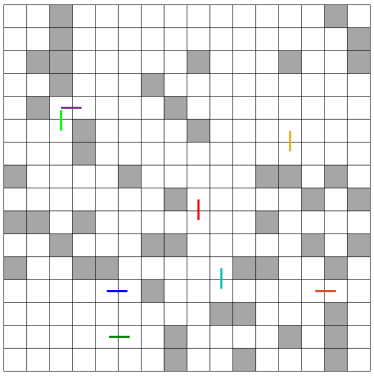
(a) CBS



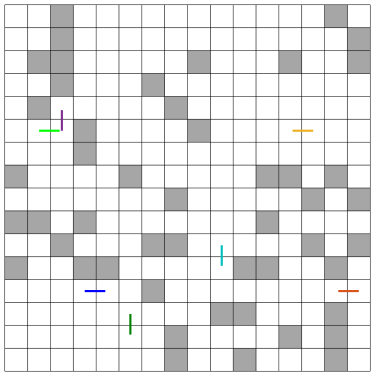
(b) CBS, $\Delta k = [0, 3]$



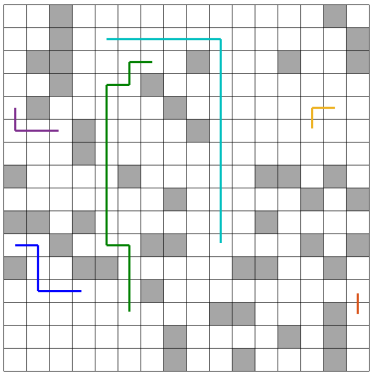
(c) CBS, $\Delta k = [3, 4]$



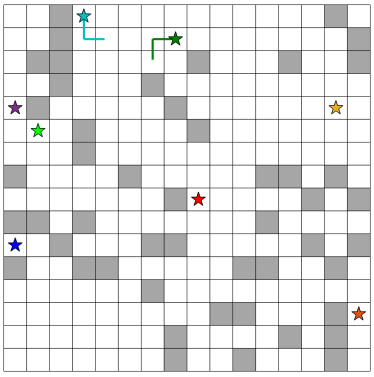
(d) CBS, $\Delta k = [4, 5]$



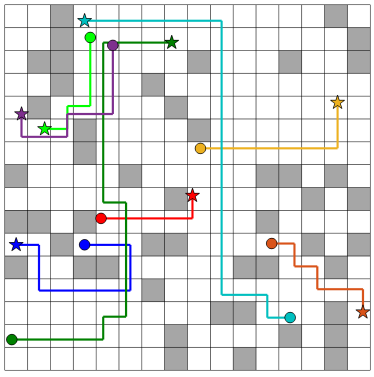
(e) CBS, $\Delta k = [5, 6]$



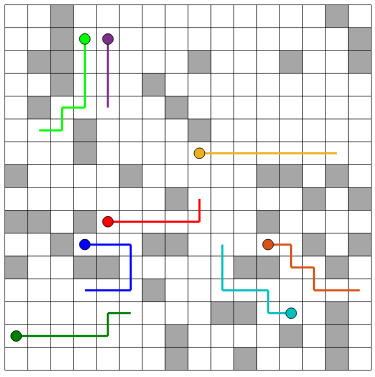
(f) CBS, $\Delta k = [6, 20]$



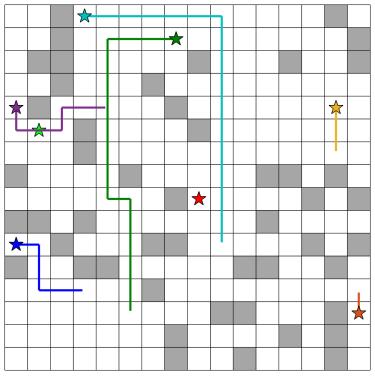
(g) CBS, $\Delta k = [20, 22]$



(h) XG-CBS, $r = 2$

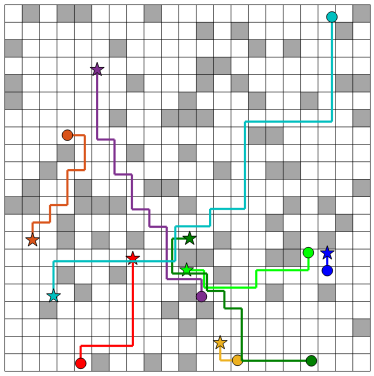


(i) XG-CBS, $\Delta k = [0, 6]$

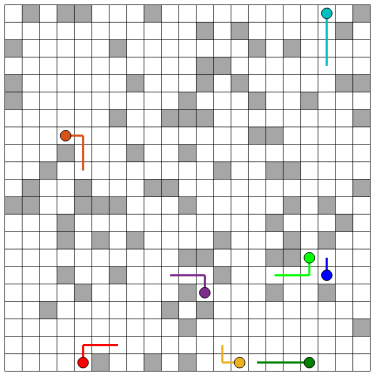


(j) XG-CBS, $\Delta k = [6, 22]$

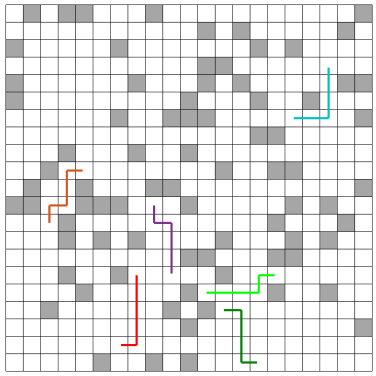
Figure 10: Example of XG-CBS with 2-segment solution vs 6-segment solution of CBS



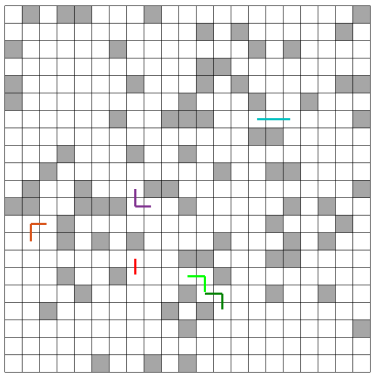
(a) CBS



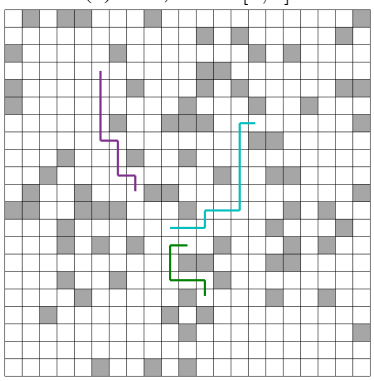
(b) CBS, $\Delta k = [0, 3]$



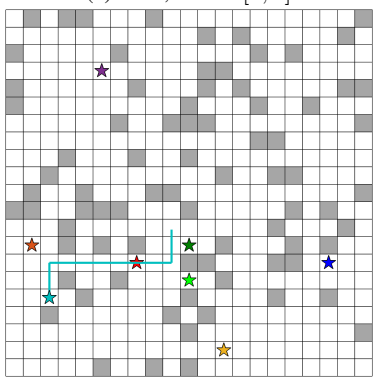
(c) CBS, $\Delta k = [3, 8]$



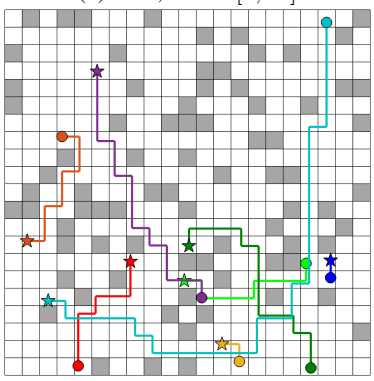
(d) CBS, $\Delta k = [8, 10]$



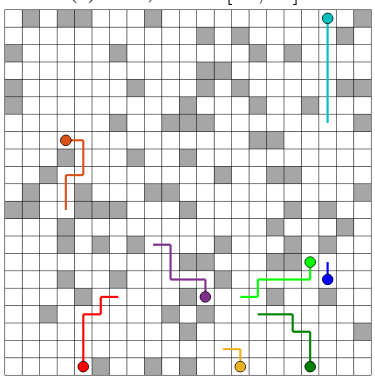
(e) CBS, $\Delta k = [10, 19]$



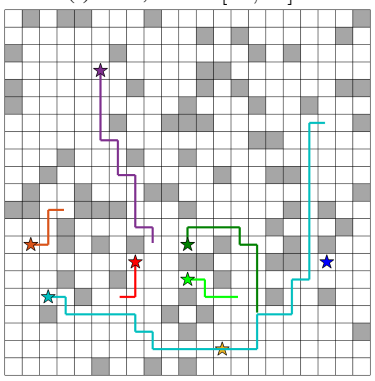
(f) CBS, $\Delta k = [19, 30]$



(g) XG-CBS, $r = 2$

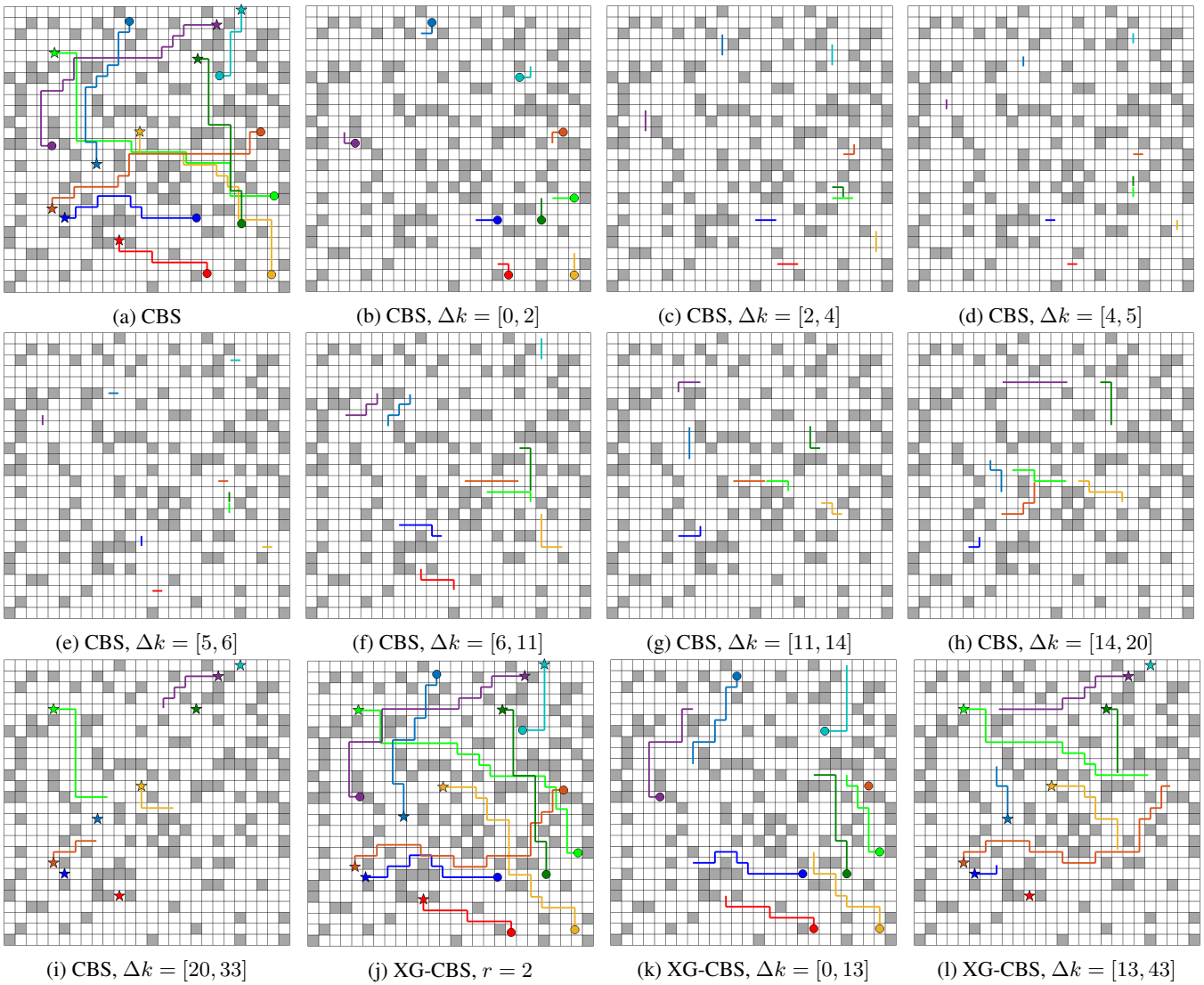


(h) XG-CBS, $\Delta k = [0, 6]$



(i) XG-CBS, $\Delta k = [6, 38]$

Figure 11: Example of XG-CBS reducing a 5-segment solution of CBS to a 2-segment plan.



(a) CBS (b) CBS, $\Delta k = [0, 2]$ (c) CBS, $\Delta k = [2, 4]$ (d) CBS, $\Delta k = [4, 5]$

(e) CBS, $\Delta k = [5, 6]$ (f) CBS, $\Delta k = [6, 11]$ (g) CBS, $\Delta k = [11, 14]$ (h) CBS, $\Delta k = [14, 20]$

(i) CBS, $\Delta k = [20, 33]$ (j) XG-CBS, $r = 2$ (k) XG-CBS, $\Delta k = [0, 13]$ (l) XG-CBS, $\Delta k = [13, 43]$

Figure 12: Example of XG-CBS, reducing 7-segment plan of CBS to a 2-segment plan.

(a) CBS (b) CBS, $\Delta k = [0, 3]$ (c) CBS, $\Delta k = [3, 4]$ (d) CBS, $\Delta k = [4, 5]$ (e) CBS, $\Delta k = [5, 6]$

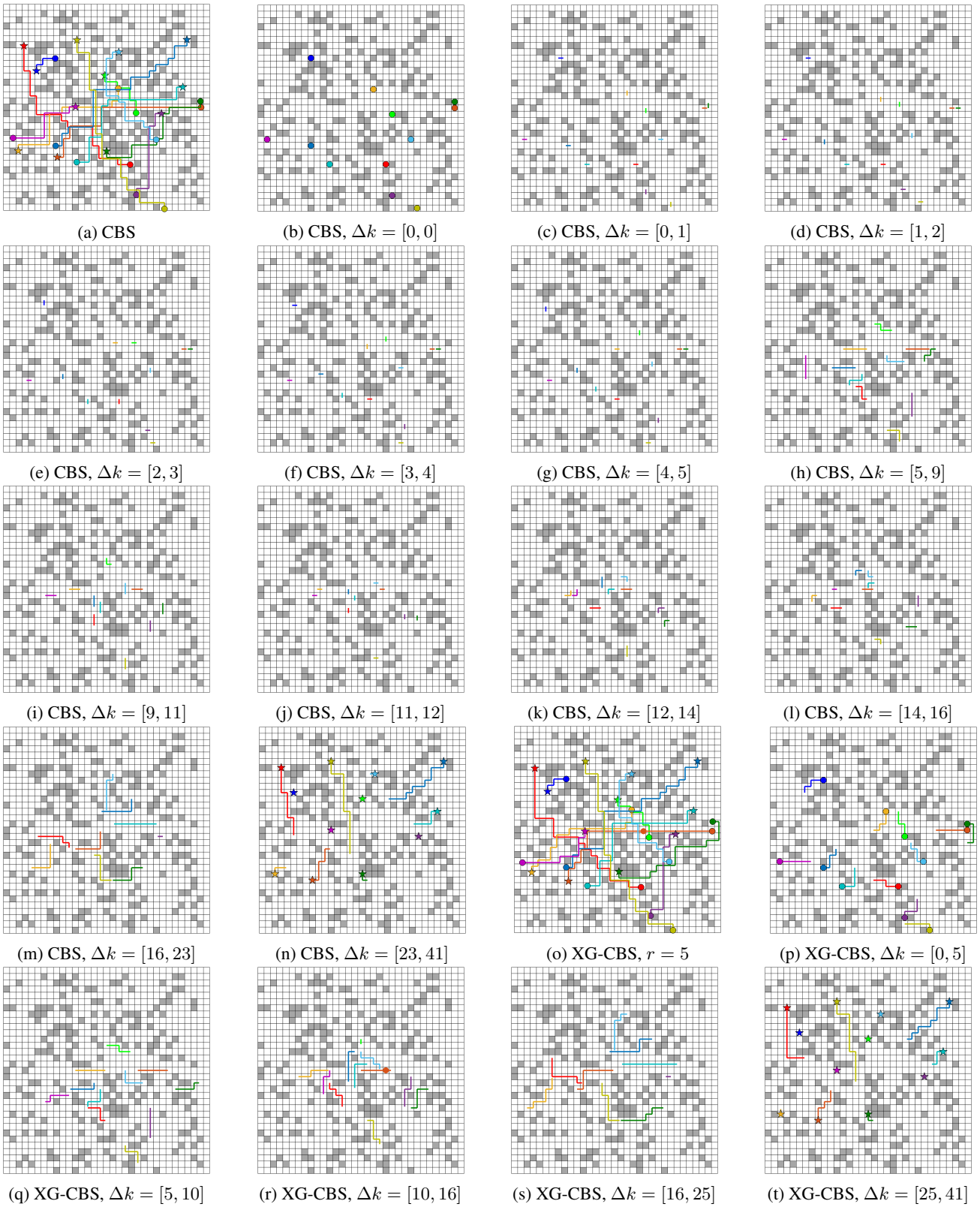
(f) CBS, $\Delta k = [6, 7]$ (g) CBS, $\Delta k = [7, 8]$ (h) CBS, $\Delta k = [8, 11]$ (i) CBS, $\Delta k = [11, 12]$ (j) CBS, $\Delta k = [12, 14]$

(k) CBS, $\Delta k = [14, 15]$ (l) CBS, $\Delta k = [15, 17]$ (m) CBS, $\Delta k = [17, 19]$ (n) CBS, $\Delta k = [19, 20]$ (o) CBS, $\Delta k = [20, 22]$

(p) CBS, $\Delta k = [22, 23]$ (q) CBS, $\Delta k = [23, 26]$ (r) CBS, $\Delta k = [26, 42]$ (s) XG-CBS, $r = 6$ (t) XG-CBS, $\Delta k = [0, 5]$

(u) XG-CBS, $\Delta k = [5, 9]$ (v) XG-CBS, $\Delta k = [9, 16]$ (w) XG-CBS, $\Delta k = [16, 21]$ (x) XG-CBS, $\Delta k = [21, 23]$ (y) XG-CBS, $\Delta k = [23, 43]$

Figure 13: Example of XG-CBS reducing 18-segment plan of CBS to a 6-segment plan.



(a) CBS (b) CBS, $\Delta k = [0, 0]$ (c) CBS, $\Delta k = [0, 1]$ (d) CBS, $\Delta k = [1, 2]$

(e) CBS, $\Delta k = [2, 3]$ (f) CBS, $\Delta k = [3, 4]$ (g) CBS, $\Delta k = [4, 5]$ (h) CBS, $\Delta k = [5, 9]$

(i) CBS, $\Delta k = [9, 11]$ (j) CBS, $\Delta k = [11, 12]$ (k) CBS, $\Delta k = [12, 14]$ (l) CBS, $\Delta k = [14, 16]$

(m) CBS, $\Delta k = [16, 23]$ (n) CBS, $\Delta k = [23, 41]$ (o) XG-CBS, $r = 5$ (p) XG-CBS, $\Delta k = [0, 5]$

(q) XG-CBS, $\Delta k = [5, 10]$ (r) XG-CBS, $\Delta k = [10, 16]$ (s) XG-CBS, $\Delta k = [16, 25]$ (t) XG-CBS, $\Delta k = [25, 41]$

Figure 14: Example of XG-CBS solution with 5-segments vs 13-segments solution of CBS.

.

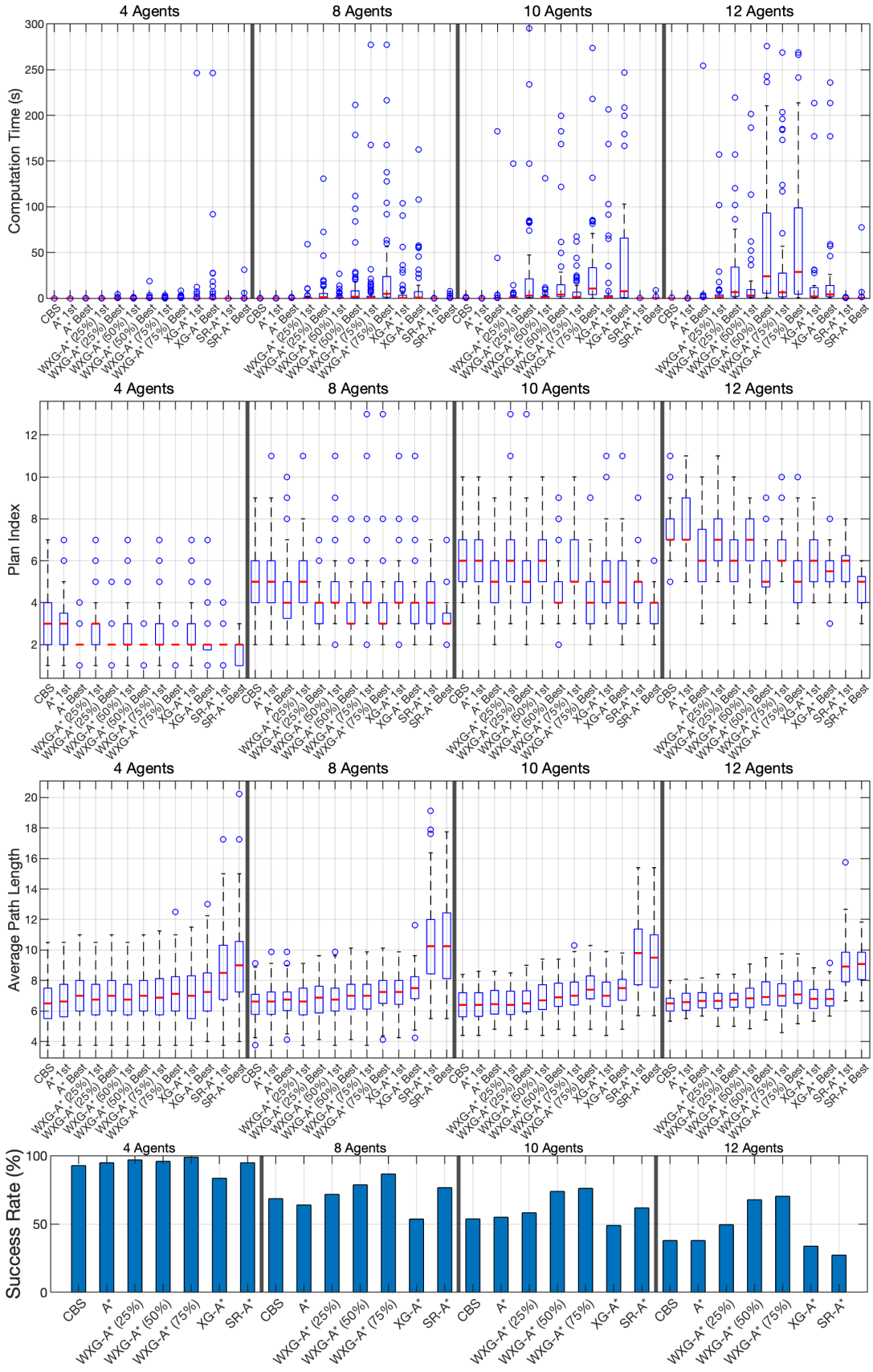


Figure 15: Benchmark results for 9×9 environments.

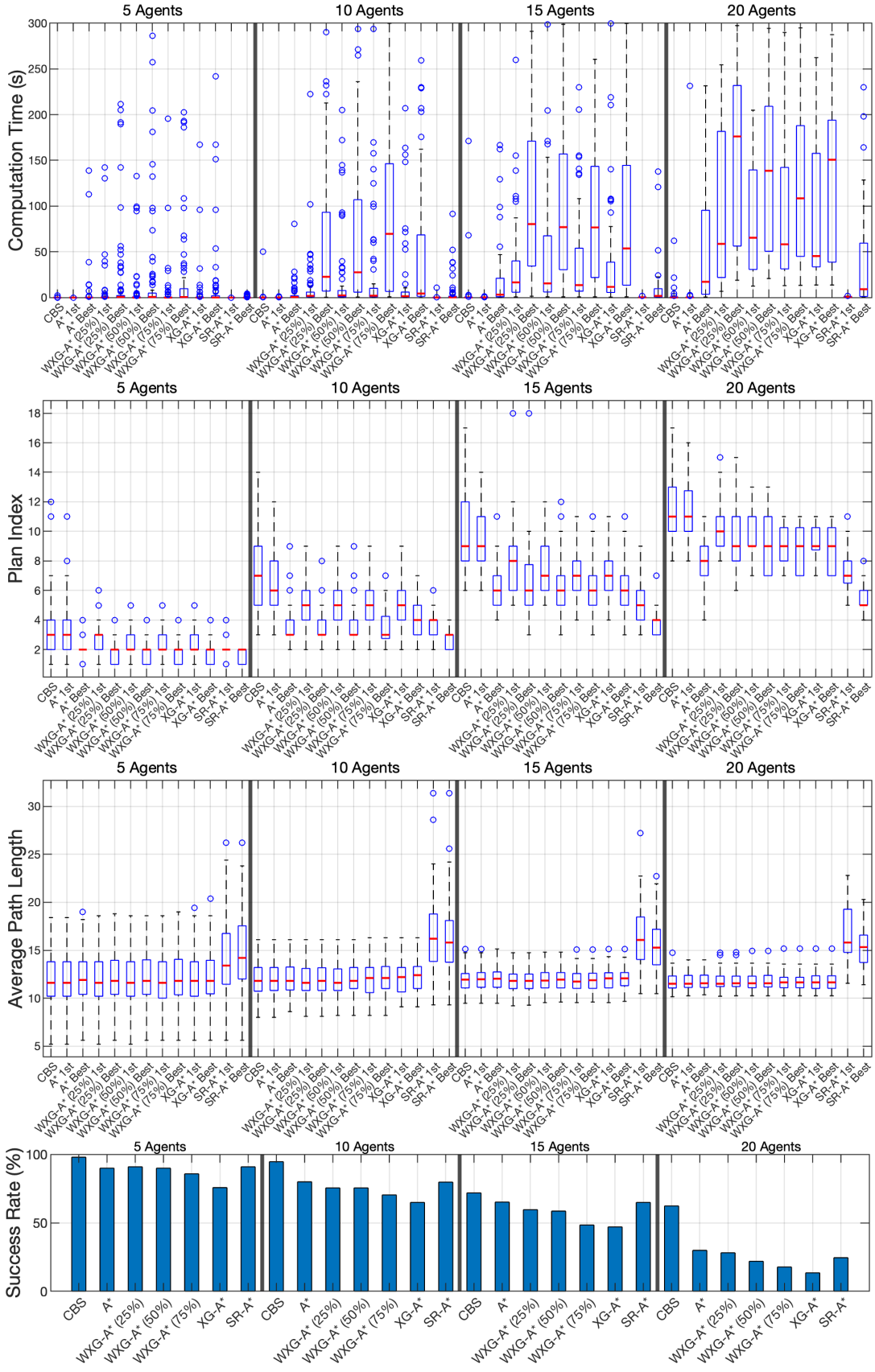


Figure 16: Benchmark results for 16×16 environments.

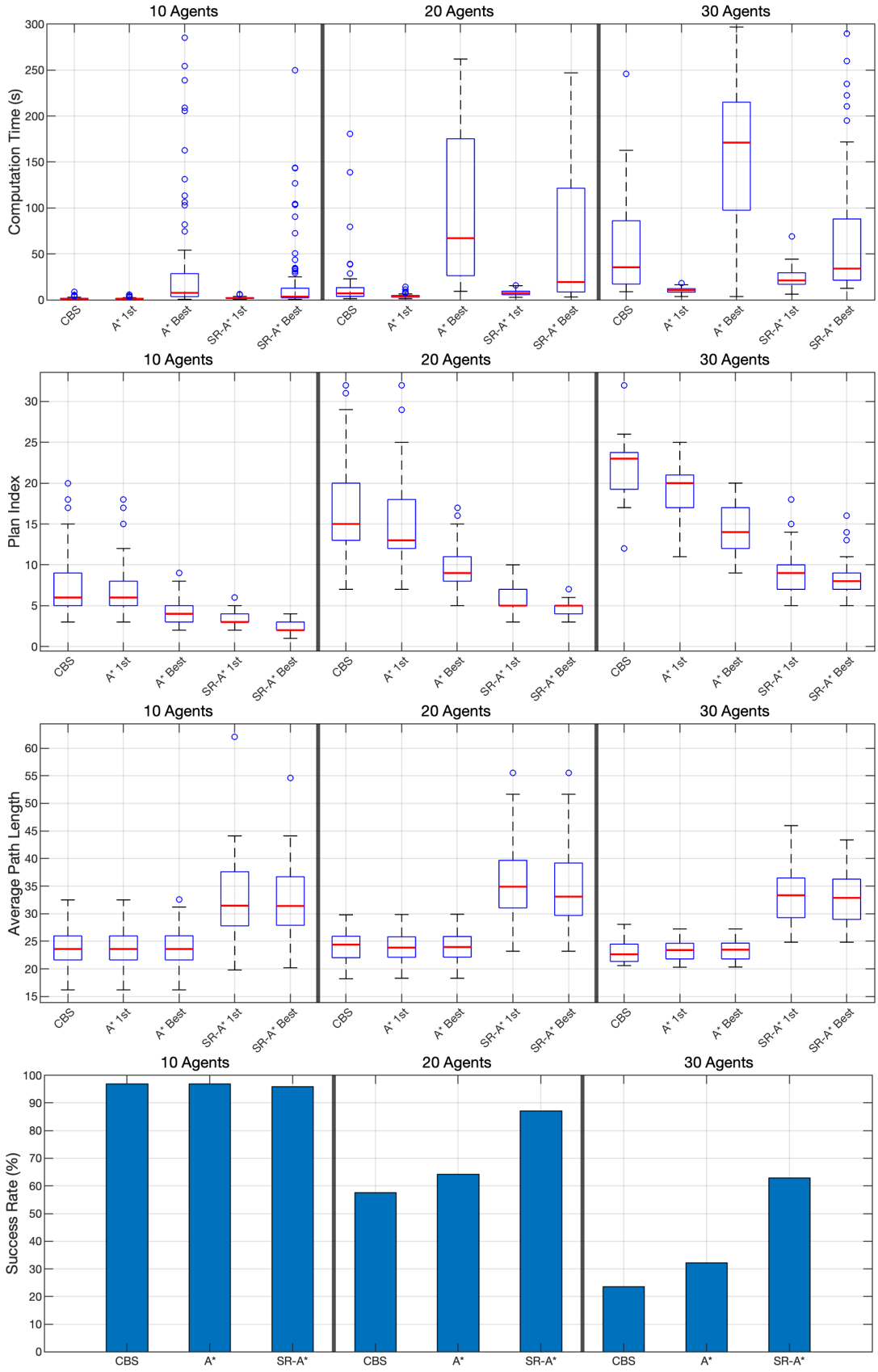


Figure 17: Benchmark results for 33×33 environments.