A Appendix

A.1 Computing the Longest Path Lengths

This section describes the algorithms for computing the forward longest path lengths (FLPL) and backward longest path lengths (BLPL) on the reduced TPG. They are simple, so we mainly present the pseudocode.

We first describe the non-incremental version of these two algorithms in Algorithm 5 and Algorithm 6. The input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is the reduced TPG of the current STPG, where $\mathcal{E}$ contains both Type-1 edges and non-switchable Type-2 edges. The topological ordering, $Ordered(\mathcal{V})$, obtained in the computation of FLPL will also be input to the calculation of BLPL. The returned $L(v)$ and $L(v, g)$ represent the functions of FLPL and BLPL, respectively.

When computing the FLPL in Algorithm 5, we update $L(v)$ by the predecessors of v (Line 6) in the topological ordering. Similarly, when computing the BLPL in Algorithm 6, we update $L(v, g^i)$ by the successors of v that are connected to g^i (Line 8) in the reverse topological ordering.

Algorithm 5: Forward Longest Path Lengths

1: **function** FLPL($\mathcal{G} = (\mathcal{V}, \mathcal{E})$)
2: Obtain a topological ordering of G, denoted as $Ordered(\mathcal{V})$
3: $L(v) \leftarrow 0$ for $\forall v \in \mathcal{V}$.
4: **for** $v \in Ordered(\mathcal{V})$ **do**
5: Get the predecessors of v, denoted as $Pred(v)$
6: $L(v) \leftarrow \max\{L(u) + 1 \mid u \in Pred(v)\}$
7: **return** $L(v)$,$Ordered(\mathcal{V})$

Algorithm 6: Backward Longest Path Lengths

1: **function** BLPL($\mathcal{G} = (\mathcal{V}, \mathcal{E})$,$Ordered(\mathcal{V})$)
2: Obtain the reverse ordering of $Ordered(\mathcal{V})$, denoted as $ROrdered(\mathcal{V})$
3: $L(v, g^i) \leftarrow \infty$ for $\forall v \in \mathcal{V}, i \in I$
4: $L(g^i, g^i) \leftarrow 0$ for $\forall i \in I$
5: **for** $v \in ROrdred(\mathcal{V})$ **do**
6: Get the successors of v, denoted as $Succ(v)$
7: **for** $i \in I$ **do**
8: $L(v, g^i) \leftarrow \max\{L(u, g^i) + 1 \mid L(u, g^i) \neq \infty, u \in S(v)\}$
9: **return** $L(v, g)$

Here, we present the incremental versions of Algorithm 5 and Algorithm 6, shown in Algorithm 7 and Algorithm 8 respectively, adapted from the paper (Katriel, Michel, and Van Hentenryck 2005).

For both algorithms, in addition to the graph G, we also input $NewEdges$, the set of new edges added to the graph, and $L(v)$, the old FLPL function, which specifies an update order. Notably, L actually specifies a topological order on the original graph G if we sort all the vertices by $L(v)$ in the ascending order. This order or its reverse is maintained by a heap in both algorithms. The returned $L'(v)$ and $L'(v, g)$ represent the functions of new FLPL and BLPL, respectively.

When computing the FLPL in Algorithm 7, we update the $L'(v)$ by the predecessors of v (Line 11). If the length changes, we push into the heap all the successors of v that have not been visited for future updating (Line 17). Similarly, when computing the BLPL in Algorithm 8, we update the $L'(v, g_i)$ by the successors of v that are connected to g^i (Line 12). If any length changes, we push into the heap all the predecessors of v that have not been visited for future updating (Line 18).

Since the incremental implementation only updates the longest path lengths for a small portion of vertices affected by the newly added edge (group) in the reduced TPG, the computational complexity is significantly reduced.

Algorithm 7: Forward Longest Path Lengths (Incremental)

1: **function** FLPL_INC($\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $L(v)$, $NewEdges$)
2: Initialize a min-heap H
3: $L'(v) \leftarrow L(v), \forall v \in \mathcal{V}$
4: $visited(v) \leftarrow False, \forall v \in \mathcal{V}$
5: **for** $e = (v_1, v_2) \in NewEdges$ **do**
6: Push $(L(v_2), v_2)$ into H
7: $visited(v_2) \leftarrow True$
8: **while** H not empty **do**
9: Pop $(L(v), v)$ from H
10: Get the predecessors of v, denoted as $Pred(v)$
11: $L'(v) \leftarrow \max\{L'(u) + 1 \mid u \in Pred(v)\}$
12: $visited(v) \leftarrow True$
13: **if** $L'(v) \neq L(v)$ **then**
14: Get the successors of v, denoted as $Succ(v)$
15: **for** $u \in Succ(v)$ **do**
16: **if** not $visited(u)$ **then**
17: Push $(L(u), u)$ into H
18: **return** $L'(v)$

A.2 The Termination Condition of GSES

This section proves the termination condition of GSES.

Proposition 1. *If there is no conflicting switchable edge found in Line 8 of Algorithm 1, GSES can be terminated by fixing all the switchable edges. It returns an acyclic TPG with the same execution cost as the current reduced TPG.*

If all the switchable edges are not conflicting, then their edge slacks $Sl(e) \geq 0, \forall e \in \mathcal{S}$, where $\mathcal{S}$ is the set of switchable edges in the current STPG $\mathcal{G}^S$. To prove that fixing all the switchable edges will not introduce any cycle and extra cost, we prove Lemma 2, which considers the case of adding only one edge to an acyclic TPG.

Lemma 2. *Given an acyclic TPG $\mathcal{G} = (\mathcal{V}, \mathcal{E}_1, \mathcal{E}_2)$ with the forward longest path lengths as $L(v), v \in V$. If we add to $\mathcal{G}$ a new Type-2 edge $e' = (v_1, v_2)$ with its slack $Sl(e') = L(v_2) - L(v_1) - 1 \geq 0$, then the new graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}_1, \mathcal{E}_2')$ remains acyclic, where $\mathcal{E}_2' = \mathcal{E}_2 \bigcup \{e'\}$. Further, it has the same forward longest path lengths as G at each vertex v. Namely, $L'(v) = L(v), v \in V$.*

Proof. By definition, $L(e) \geq 0$ for all $e \in \mathcal{E}_2$. Since $L(e') \geq 0$ as well, so we can state that $L(e) \geq 0$ for all $e \in \mathcal{E}_2'$.

Algorithm 8: Backward Longest Path Lengths (Incremental)

```
1: function BLPL_INC(G = (V, E), L(v), NewEdges)
2:     Initialize a max-heap H
3:     L'(v, g_i) ← L(v, g_i), ∀v ∈ V, i ∈ I
4:     visited(v) ← False, ∀v ∈ V
5:     for e = (v_1, v_2) ∈ NewEdges do
6:         Push (L(v_1), v_1) into H
7:         visited(v_1) ← True
8:     while H not empty do
9:         Pop (L(v), v) from H
10:        Get the successors of v, denoted as Succ(v)
11:        for i ∈ I do
12:            L'(v, g^i) ← max {L'(u, g^i) + 1 |
               L'(u, g^i) ≠ ∞, u ∈ S(v)}
13:        visited(v) ← True
14:        if ∃i ∈ I, L'(v, g_i) ≠ L(v, g_i) then
15:            Get the predecessors of v, denoted as
               Pred(v)
16:            for u ∈ Pred(v) do
17:                if not visited(u) then
18:                    Push (L(u), u) into H
19:    return L'(v, g)
```

First, we prove that $\mathcal{G}'$ is acyclic by contradiction. We assume that there is a cycle $v_1 \to v_2 \to \cdots \to v_n \to v_1$. Since for $e_i = (v_i, v_{i+1}), Sl(e) = L(v_{i+1}) - L(v_i) - 1 \geq 0$, we have $L(v_{i+1}) > L(v_i)$. So, $L(v_n) > L(v_1)$. But there is also an edge from v_n to v_1, so we can also get $L(v_1) > L(v_n)$, leading to the contradiction. Therefore, $\mathcal{G}'$ must be acyclic.

Since $\mathcal{G}'$ is also a direct acyclic graph, we assume we obtain a topological ordering of it in the Algorithm 5, as $Ordered(\mathcal{G}')$. Since $\mathcal{G}'$ has one more edge than $\mathcal{G}$, $Ordered(\mathcal{G}')$ is also a topological ordering of $\mathcal{G}$ when this edge is removed.

Now, if we run Algorithm 5 for both $\mathcal{G}'$ and $\mathcal{G}$ with this ordering, every vertex v ordered before v_2 in $Ordered(\mathcal{G}')$ must have $L(v) = L'(v)$ because its predecessors and its predecessors' forward longest path lengths are not changed.

Then, we validate $L'(v_2) = L(v_2)$. First, there exists some predecessor u of v_2 such that $L(v_2) = L(u) + 1$. Notably, u and v_1 must be vertices ordered before v_2 in $Ordered(\mathcal{G}')$ because they both have an edge pointing to v_2. Thus, $L'(u) = L(u)$ and $L'(v_1) = L(v_1)$. Since $Sl(e') = L(v_2) - L(v_1) - 1 \geq 0$, $L'(u) + 1 = L(u) + 1 = L(v_2) \geq L(v_1) + 1 = L'(v_1) + 1$. Therefore, $L'(u) + 1$ is still the maximum in Line 6 of Algorithm 5, even though a new edge e' is added. So, $L'(v_2) = L'(u) + 1 = L(u) + 1 = L(v_2)$.

Finally, we can deduce every vertex v ordered after v_2 in $Ordered(\mathcal{G}')$ must also have $L(v) = L'(v)$ because its predecessors and its predecessors' forward longest path lengths remain the same. Therefore, $L'(v) = L(v), \forall v \in V$. □

Corollary 1. *The execution cost of $\mathcal{G}'$ is the same as $\mathcal{G}$.*

Proof. Based on Theorem 2, the execution cost of a TPG is the sum of the longest path lengths at all agents' goals. Since the longest path lengths remain the same, the execution costs are also the same. □

Then, we can prove Proposition 1 easily by fixing switchable edges one by one. Each fixing adds a new edge to the acyclic TPG but does not introduce any cycle or extra cost.

A.3 Greedy Matching Algorithm For EWMVC

This section describes the greedy matching algorithm for the Edge-Weighted Minimum Vertex Cover (EWMVC) problem in Section 4.1. We have a weighted fully-connected undirected graph $G_D = (V_D, E_D, W_D)$, where each vertex $u_i \in V_D$ represents an agent, E_D are edges and W_D are edges' weights. Specifically, we set the weight of each edge $(u_i, u_j) \in E_D$ to be the pairwise cost increase, $\Delta cost(g^i, g^j)$ obtained from our stronger heuristic reasoning. Our target is to assign a cost increase x_i to each vertex u_i such that $x_i + x_j \geq \Delta cost(g^i, g^j)$ so that the overall cost increase $\sum_i x_i$ is minimized.

The pseudocode is illustrated in Algorithm 9. Briefly speaking, the algorithm always selects the max-weighted edge at each iteration, whose two endpoints have never been matched. Then, the weight is added to the overall weight, and the two endpoints are marked as matched. Since each iteration checks all the edge weights and we assume it is a fully connected graph, each iteration takes $O(|V_D|^2)$ time. There could be, at most, $|V_D|$ iterations. The worst-case complexity of this algorithm is $O(|V_D|^3)$.

Algorithm 9: Greedy Matching for EWMVC

```
1: function GREEDYMATCHING(G_D = (V_D, E_D, W_D))
2:     matched[i] ← False for ∀i ∈ I
3:     sumW ← 0
4:     while True do
5:         maxW ← 0, p ← −1, q ← −1
6:         for (u_i, u_j) ∈ E_D do
7:             if matched[i] ∨ matched[j] then continue
8:             if W_D[i, j] > maxW then
9:                 maxW ← W[i, j], p ← i, q ← j
10:        if maxW = 0 then break
11:        sumW ← sumW + maxW
12:        matched[p] ← True, matched[q] ← True
13:    return sumW
```

A.4 Experiments

This section explains more details about our benchmark, code implementation, and how to reproduce the experiments. We also illustrate experiment results with different delay probabilities. To be more self-contained, this section may overlap with the main text.

Benchmark The benchmark generation process is described in the main text. Notably, our benchmark evaluates with as twice many agents as the benchmark used in the GSES paper (Feng et al. 2024). Therefore, We use k-robust PBS rather than k-robust CBS to obtain the initial MAPF plan since the former is much faster than the latter when solving instances with more agents. We run k-robust PBS on 25 instances with evenly distributed starts and goals from

Table 2: The number of solved Instances for each map and each agent number.

Random-32-32-10					
#agents	60	70	80	90	100
#solved instances	24	21	25	22	16
Warehouse-10-20-10-2-1					
#agents	110	120	130	140	150
#solved instances	24	23	22	23	20
Lack303d					
#agents	41	49	57	65	73
#solved instances	25	25	25	25	25
Paris_1_256					
#agents	120	140	160	180	200
#solved instances	22	20	20	18	18

the MovingAI benchmark [9] and set the time limit of k-robust PBS to be 6 minutes for each MAPF instance. Each solved instance will generate 6 delay scenarios by simulation. We report the number of solved instances for each map and each agent number in Table 2.

Code Implementation In this paper, we compare with other three optimal STPG optimization algorithms. We incorporate their open-source implementation into our code.

1. GSES: https://github.com/YinggggFeng/Multi-Agent-via-Switchable-ADG
2. MILP: https://github.com/alexberndt/sadg-controller
3. CBS-D: https:/github.com/aria-systems-group/Delay-Robust-MAPF

All codes are implemented in C++ except MILP, which uses a Python interface to an open-sourced brand-and-bound C++ solver, the COIN-OR Branch-and-Cut solver, at https://github.com/coin-or/Cbc.

Notably, the code can handle non-uniform edge costs, even though we assume all the costs are 1 in the main text.

Reproducibility Our code and data are publicly available at https://github.com/DiligentPanda/STPG.git. Results can be reproduced by running the experiment scripts in the Linux system (e.g., Ubuntu). The readme file in the code repository explains the details.

Experiment Results We organize all the experiment data in Table 3. Success Rate, Incremental Analysis, and Ablation Study with delay probability $p = 0.01$ have been discussed in the main text. The conclusions for other delay probabilities are consistent with the ones in the main text.

We further include plots of the mean search time for different maps and agent numbers, which reflects the superiority of IGSES over other optimal algorithms from another view. Of course, with more agents and larger delay probabilities, the search clearly takes more time.

[9] https://www.movingai.com/benchmarks/mapf/index.html

Table 3: Reference to all experiment data.

Delay Probability	$p = 0.002$	$p = 0.01$	$p = 0.03$
Success Rate	Figure 8	Figure 10	Figure 12
Search Time	Figure 9	Figure 11	Figure 13
Incremental Analysis	Table 4	Table 5	Table 6
Ablation Study	Figure 14	Figure 15	Figure 16

Table 4: Incremental analysis on instances solved by all the settings with delay probability $p = 0.002$. Rows 1,2,3 compare the effects of different grouping methods. GSES has no grouping. SG: simple grouping. FG: full grouping. Rows 3,4,5,6 compare the effects of different branching orders. Row 3 applies the default Agent-First branching in GSES. RB: Random branching. EB: Earliest-First branching. SB: Smallest-Edge-Slack-First branching. Rows 6,7 compare the effect of the stronger heuristics. SH: stronger heuristics. Rows 7,8 compare the effect of the incremental implementation. INC: incremental implementation. The last row is also our IGSES.

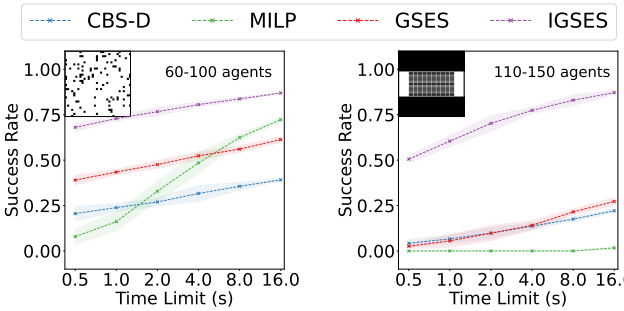
Row	Setting	Random-32-32-10			Warehouse-10-20-10-2-1			Lak303d			Paris_1_256		
		search time (s)	#expanded nodes	#edge groups	search time (s)	#expanded nodes	#edge groups	search time (s)	#expanded nodes	#edge groups	search time (s)	#expanded nodes	#edge groups
1	GSES	1.093	2249.8	1164.0	3.013	757.5	14686.1	1.648	333.0	30230.5	4.516	190.5	44884.0
2	+SG	0.840	1475.6	826.7	1.580	285.7	8668.4	0.848	112.2	18938.0	3.292	104.2	30191.6
3	+FG	0.747	1213.1	518.8	1.355	218.4	2200.0	0.704	80.5	5581.3	2.949	83.5	11873.3
4	+FG +RB	1.259	1996.9	518.8	2.226	358.1	2200.0	2.168	246.3	5581.3	4.393	122.1	11873.3
5	+FG +EB	1.054	1755.0	518.8	1.791	291.9	2200.0	1.120	115.2	5581.3	2.896	79.5	11873.3
6	+FG +SB	0.713	1111.7	518.8	1.312	205.0	2200.0	0.726	83.3	5581.3	3.119	88.4	11873.3
7	+FG +SB +SH	0.127	59.9	518.8	0.743	31.6	2200.0	0.701	25.1	5581.3	1.537	13.9	11873.3
8	+FG +SB +SH +INC	0.035	59.9	518.8	0.116	31.6	2200.0	0.207	25.1	5581.3	0.349	13.9	11873.3

Table 5: Incremental analysis on instances solved by all the settings with delay probability $p = 0.01$. Rows 1,2,3 compare the effects of different grouping methods. GSES has no grouping. SG: simple grouping. FG: full grouping. Rows 3,4,5,6 compare the effects of different branching orders. Row 3 applies the default Agent-First branching in GSES. RB: Random branching. EB: Earliest-First branching. SB: Smallest-Edge-Slack-First branching. Rows 6,7 compare the effect of the stronger heuristics. SH: stronger heuristics. Rows 7,8 compare the effect of the incremental implementation. INC: incremental implementation. The last row is also our IGSES.

Row	Setting	Random-32-32-10			Warehouse-10-20-10-2-1			Lak303d			Paris_1_256		
		search time (s)	#expanded nodes	#edge groups	search time (s)	#expanded nodes	#edge groups	search time (s)	#expanded nodes	#edge groups	search time (s)	#expanded nodes	#edge groups
1	GSES	1.421	3051.5	1637.9	3.805	948.3	15441.6	2.608	550.8	30320.6	5.221	253.1	40214.5
2	+SG	1.082	2025.7	1176.4	1.783	315.3	9134.2	1.242	185.7	19012.7	3.131	115.2	27065.5
3	+FG	0.945	1597.9	732.5	1.519	241.4	2367.0	1.021	130.8	5601.8	2.770	88.1	10706.9
4	+FG +RB	1.555	2495.2	732.5	2.685	447.4	2367.0	2.644	324.2	5601.8	3.553	114.5	10706.9
5	+FG +EB	1.185	2036.1	732.5	2.354	381.6	2367.0	1.481	172.9	5601.8	3.118	102.6	10706.9
6	+FG +SB	0.915	1480.5	732.5	1.472	229.5	2367.0	1.027	129.0	5601.8	2.472	78.7	10706.9
7	+FG +SB +SH	0.169	84.6	732.5	0.832	34.6	2367.0	0.871	33.1	5601.8	1.561	15.4	10706.9
8	+FG +SB +SH +INC	0.043	84.6	732.5	0.120	34.6	2367.0	0.221	33.1	5601.8	0.321	15.4	10706.9

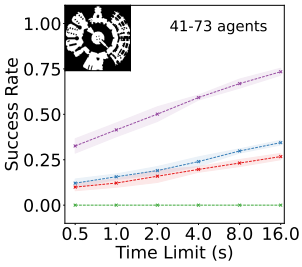
Table 6: Incremental analysis on instances solved by all the settings with delay probability $p = 0.03$. Rows 1,2,3 compare the effects of different grouping methods. GSES has no grouping. SG: simple grouping. FG: full grouping. Rows 3,4,5,6 compare the effects of different branching orders. Row 3 applies the default Agent-First branching in GSES. RB: Random branching. EB: Earliest-First branching. SB: Smallest-Edge-Slack-First branching. Rows 6,7 compare the effect of the stronger heuristics. SH: stronger heuristics. Rows 7,8 compare the effect of the incremental implementation. INC: incremental implementation. The last row is also our IGSES.

Row	Setting	Random-32-32-10			Warehouse-10-20-10-2-1			Lak303d			Paris_1_256		
		search time (s)	#expanded nodes	#edge groups	search time (s)	#expanded nodes	#edge groups	search time (s)	#expanded nodes	#edge groups	search time (s)	#expanded nodes	#edge groups
1	GSES	1.580	3699.3	1564.2	4.210	1080.9	14763.0	2.262	443.3	29775.0	5.457	273.6	39522.9
2	+SG	1.173	2383.1	1124.2	1.959	364.7	8744.3	1.115	153.5	18628.9	3.528	133.5	26525.9
3	+FG	0.996	1824.5	700.0	1.685	277.8	2283.7	0.907	105.7	5492.3	3.011	97.1	10396.3
4	+FG +RB	1.605	2970.8	700.0	2.968	496.8	2283.7	2.397	277.5	5492.3	4.120	133.8	10396.3
5	+FG +EB	1.716	3276.8	700.0	2.534	435.8	2283.7	1.426	170.6	5492.3	3.067	100.2	10396.3
6	+FG +SB	0.736	1317.0	700.0	1.564	253.6	2283.7	0.792	84.2	5492.3	2.680	86.2	10396.3
7	+FG +SB +SH	0.187	96.8	700.0	0.924	40.6	2283.7	0.752	26.3	5492.3	2.080	21.9	10396.3
8	+FG +SB +SH +INC	0.045	96.8	700.0	0.124	40.6	2283.7	0.205	26.3	5492.3	0.337	21.9	10396.3

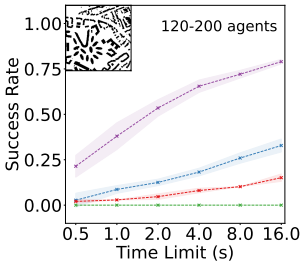


(a) Random-32-32-10

(b) Warehouse-10-20-10-2-1

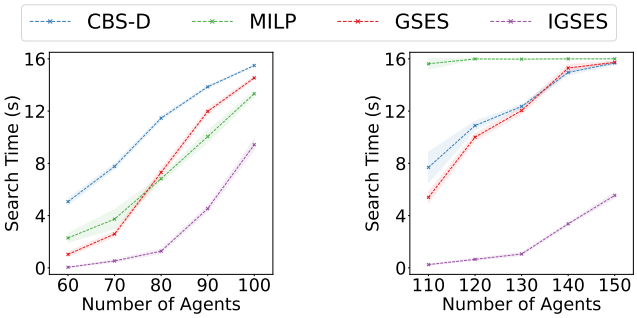


(c) Lak303d



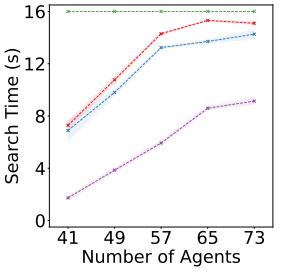
(d) Paris_1_256

Figure 8: Success rates of IGSES and other baselines on four maps with delay probability $p = 0.002$. The shading areas indicate the standard deviations of different runs. They are multiplied by 10 for illustration. In each figure, the top-left shows the corresponding map, and the top-right corner shows the range of agent numbers.

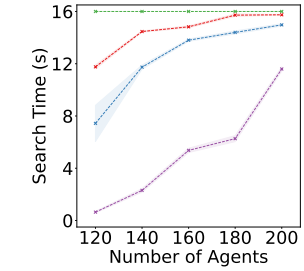


(a) Random-32-32-10

(b) Warehouse-10-20-10-2-1

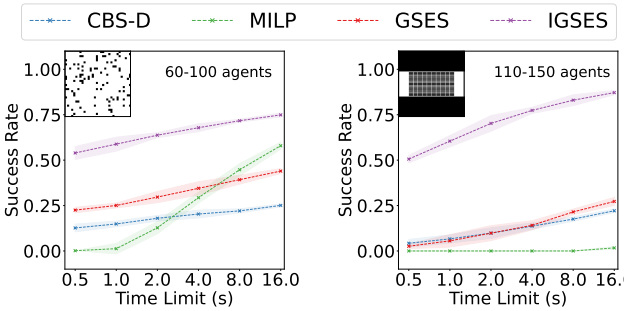


(c) Lak303d



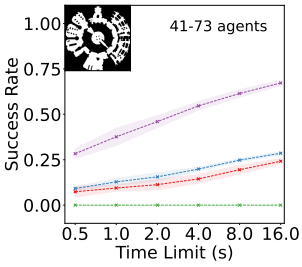
(d) Paris_1_256

Figure 9: Search Time of IGSES and other baselines on four maps with delay probability $p = 0.002$. The search time of unsolved instances is set to the time limit of 16 seconds. The shading areas indicate the standard deviations of different runs. They are multiplied by 10 for illustration.

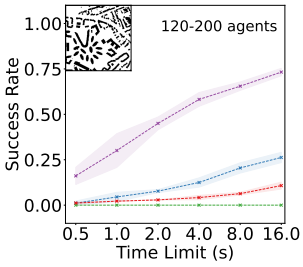


(a) Random-32-32-10

(b) Warehouse-10-20-10-2-1

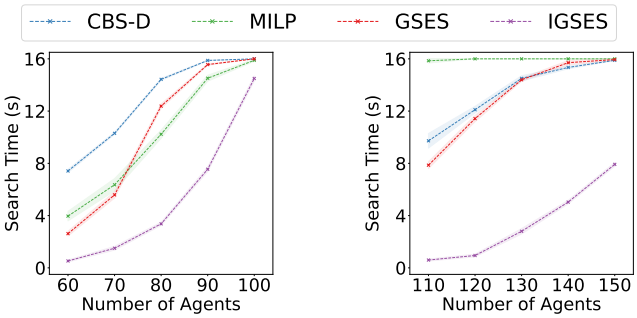


(c) Lak303d



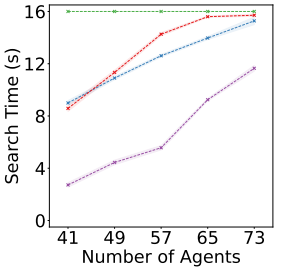
(d) Paris_1_256

Figure 10: Success rates of IGSES and other baselines on four maps with delay probability $p = 0.01$. The shading areas indicate the standard deviations of different runs. They are multiplied by 10 for illustration. In each figure, the top-left shows the corresponding map, and the top-right corner shows the range of agent numbers.

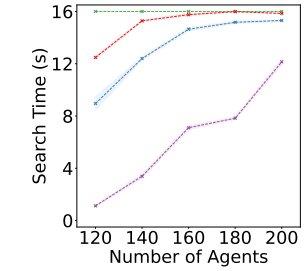


(a) Random-32-32-10

(b) Warehouse-10-20-10-2-1

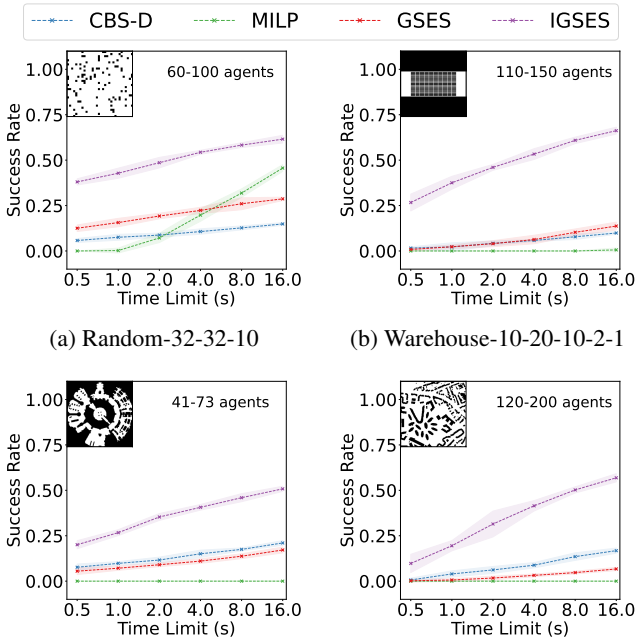


(c) Lak303d



(d) Paris_1_256

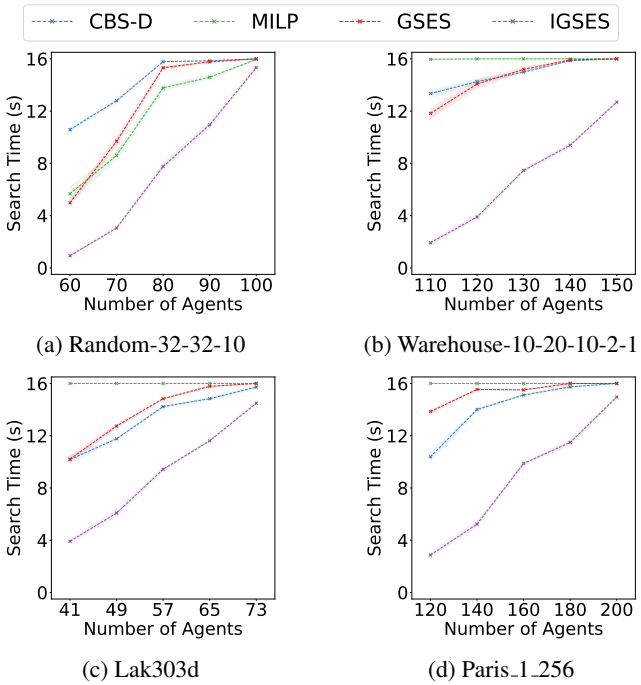
Figure 11: Search Time of IGSES and other baselines on four maps with delay probability $p = 0.01$. The search time of unsolved instances is set to the time limit of 16 seconds. The shading areas indicate the standard deviations of different runs. They are multiplied by 10 for illustration.



(a) Random-32-32-10 (b) Warehouse-10-20-10-2-1

(c) Lak303d (d) Paris_1_256

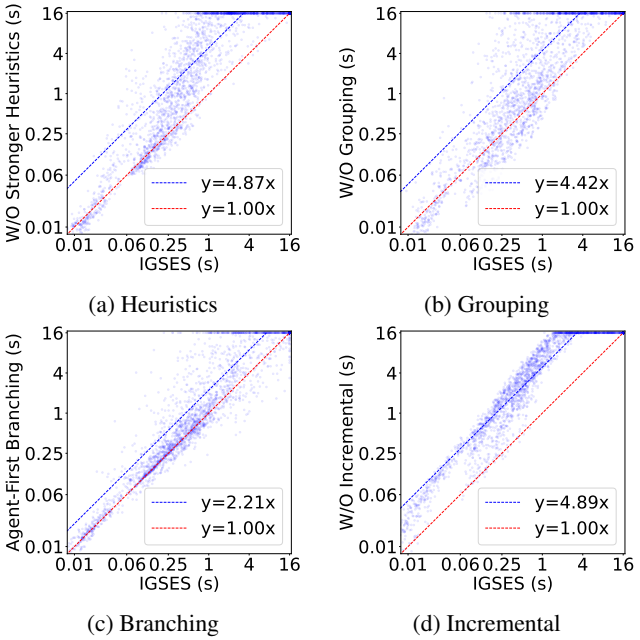
Figure 12: Success rates of IGSES and other baselines on four maps with delay probability $p = 0.03$. The shading areas indicate the standard deviations of different runs. They are multiplied by 10 for illustration. In each figure, the top-left shows the corresponding map, and the top-right corner shows the range of agent numbers.



(a) Random-32-32-10 (b) Warehouse-10-20-10-2-1

(c) Lak303d (d) Paris_1_256

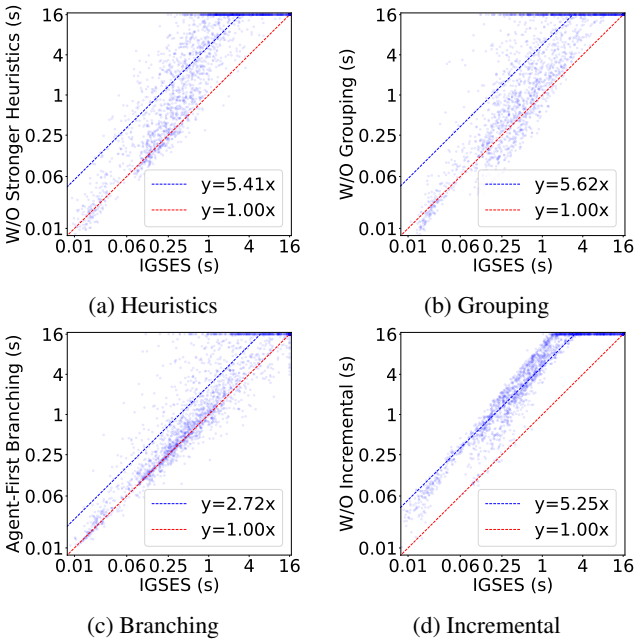
Figure 13: Search Time of IGSES and other baselines on four maps with delay probability $p = 0.03$. The search time of unsolved instances is set to the time limit of 16 seconds. The shading areas indicate the standard deviations of different runs. They are multiplied by 10 for illustration.



(a) Heuristics (b) Grouping

(c) Branching (d) Incremental

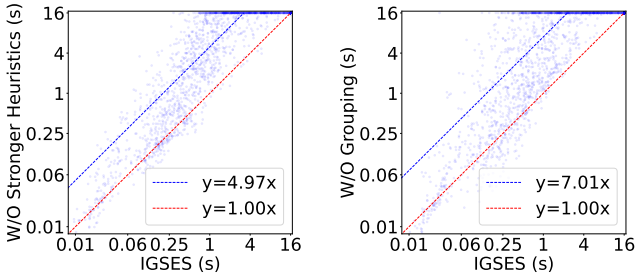
Figure 14: Ablation on four speedup techniques on all instances with delay probability $p = 0.002$. In each figure, we compare IGSES to the setting that replaces one of its techniques by the GSES's choice. Each point in the graph represents an instance. The search time of an unsolved instance is set to 16 seconds. The blue line is the fitted on instances solved by at least one setting. Its sloped indicates the average speedup.



(a) Heuristics (b) Grouping

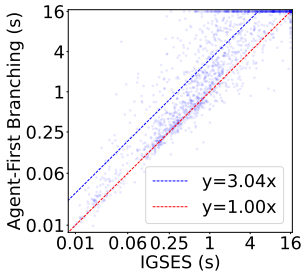
(c) Branching (d) Incremental

Figure 15: Ablation on four speedup techniques on all instances with delay probability $p = 0.01$. In each figure, we compare IGSES to the setting that replaces one of its techniques by the GSES's choice. Each point in the graph represents an instance. The search time of an unsolved instance is set to 16 seconds. The blue line is the fitted on instances solved by at least one setting. Its sloped indicates the average speedup.

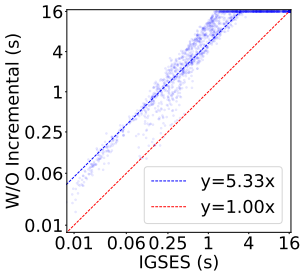


(a) Heuristics

(b) Grouping



(c) Branching



(d) Incremental

Figure 16: Ablation on four speedup techniques on all instances with delay probability $p = 0.03$. In each figure, we compare IGSES to the setting that replaces one of its techniques by the GSES's choice. Each point in the graph represents an instance. The search time of an unsolved instance is set to 16 seconds. The blue line is the fitted on instances solved by at least one setting. Its sloped indicates the average speedup.