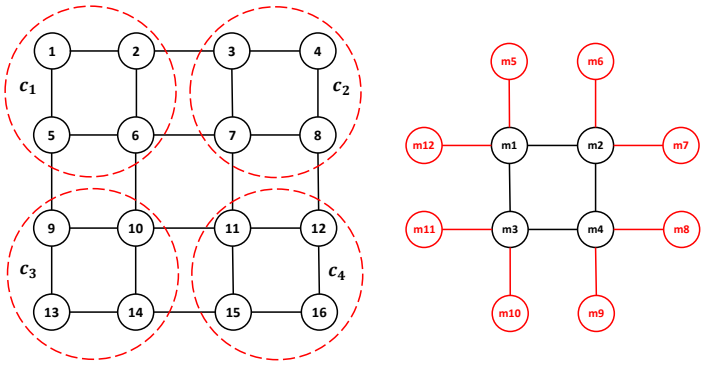
Integrating Knowledge Compilation with Reinforcement Learning for Routes

Appendix A: Extensions

Route distribution and map partitioning

To partition a map represented as an undirected graph $G = (V, E)$, we partition its nodes V into regions or clusters $c_1, ..., c_m$, with each cluster c_i having *internal* and *external* (that cross into c_i) edges. On these clusters, we induce a graph G_p with $c_1, ..., c_m$ as nodes. We then define constraints on $\mathbf{X}$ using G and G_p that paths that are simple in G_p are also simple w.r.t G and induce a distribution $Pr(\mathbf{X})$ over them. More concretely, paths cannot enter a region twice and they also cannot not visit any nodes inside the clusters twice. We represent all the simple paths *inside* the clusters $c_1, ..., c_m$ and also *across* the clusters as psdds. This is a hierarchical representation of paths in which we have two levels of hierarchy, one for across the clusters and another for inside the clusters.

Example: Consider Figure 1(a), where a 4x4 grid map is partitioned into clusters $c_1, ..., c_4$ and the graph G_p is formed from these clusters as nodes. Figure 1(b) represents inside of a cluster which is a 2x2 grid map. The black edges are the internal edges and the red edges are the external ones. We construct psdds between all the red nodes inside the cluster and also a psdd for the 2x2 grid map formed by $c_1, ..., c_4$. Let's say Figure 1(b) represents cluster c_1, i.e., node 1 is mapped to $m1$, 2 is mapped to $m2$ and so on. Similarly, edge $(m2, m7)$ is mapped to the edge $(2, 3)$, $(m4, m8)$ to $(6, 7)$ and so on. Now, to sample a path that starts from node 1, we start from $m12$ (or $m5$) to enter c_1. We keep sampling until we encounter an external edge. If, for example, we encounter the edge $(m4, m8)$, we traverse the edgeg $(6, 7)$ in the 4x4 grid and move to the cluster c_2 and keep sampling until we reach the destination. (We discard $(m2, m6)$ for c_1 because it is not mapped to any of the edge in the 4x4 grid).



(a) (b)

Figure 1: (a)A 4x4 grid map partitioned into regions (or clusters) $c_1, ..., c_4$. $c_1, ..., c_4$ form a 2x2 grid map G_p (b) Inside of a cluster, e.g., c_1. The black edges are the inernal edges and the red edges are the external edges.

Appendix B: Coverage Problem

Cooverage constraints can be used to simulate patrolling by agents in Singapore train network, Mass Rapid Transit (MRT). We test these constraints on the longest rail line (green) of the MRT infrastructure. We divide this rail line with 35 stations in multiple segments. Each segment is patrolled by an agent and has N important stations to visit (like junctions with other rail lines). We focus on the busiest Central Business District (CBD) area between Outram Park and Kallang stations as shown in Figure 2 with $N = 3$. The agent is required to inspect Outram Park, Raffles Place and Bugis every $K = 15$ time steps. We choose these stations as they are more important because of their locations at the junctions with purple, red, and blue lines respectively. Table 5 in the main text gives results for these simulations.

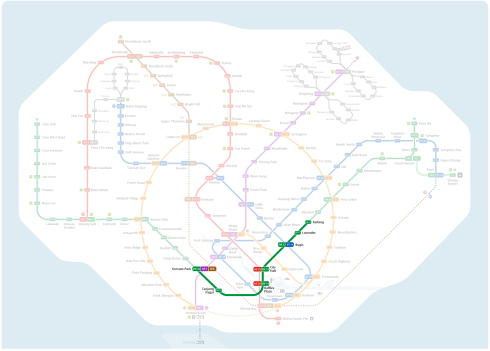
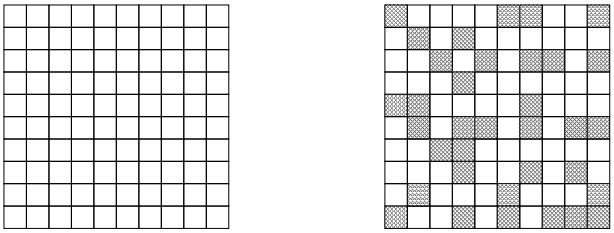


Figure 2: Singapore MRT Green Line

Appendix C: Empirical Evaluation

To represent sdd in our experiments, we use the `GRAPHILLION` [Inoue et al., 2016] package to first construct a ZDD and then convert it to sdd [Nishino et al., 2016, 2017]. We also `PyPSDD`[1] package for constructing the hierarchical clustering map and use the associated sdd.

Experimental Settings: To compare our approach with DCRL, we follow the same settings in [Ling et al., 2020]. For each grid map, sources and destinations are the top and bottom rows. For each agent, we randomly select its source and destination from the top and bottom row. The capacity of each node is sampled uniformly from [1, 2] for 4x4 grid, [1, 3] for 8x8 grid, and [1, 4] for 10x10 grid. For 10x10 grid with obstacles (as shown in Figure 3(b)), the capacity of each node is sampled uniformly from [1, 2] for 2 agents, [1, 3] for 5 agents, and [1, 4] for 10 agents. The t_{min}, t_{max} for moving between two contiguous zones are 1, 5 respectively. We used the same 10x10 grip with obstacle map for evaluating PRIMAL+KCO and PRIMAL. The locations of obstacles are fixed. We generated 10 instances for 2 agents, 5 agents, and 10 agents respectively. For each instance, the source and destination for an agent are randomly selected from the non-blocked nodes. We run each instance for three times, and select the run with the best performance.



(a) (b)

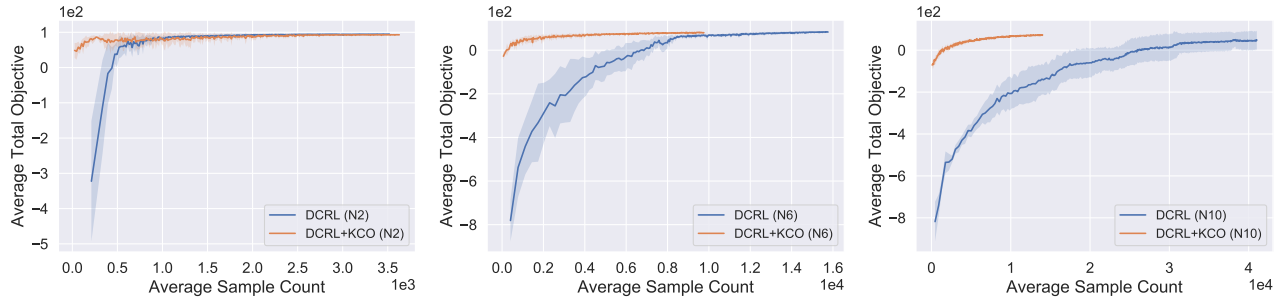
Figure 3: (a) 10x10 open grid map; (b) 10x10 grid with obstacles (0.35 obstacle density, dark nodes are blocked)

[1] https://github.com/art-ai/pypsdd

Hyperparameters and Neural Network Architecture: To compare our DCRL+KCO and MAPQN+KCO with DCRL and MAPQN respectively, we use the same hyperparameters as in [Ling et al., 2020]. The neural network architectures are also the same except for the last $softmax$ layer in DCRL code[2]. Instead, we use a customized layer to generate a probability distribution over all actions according to the policy π defined in the main paper. For comparing the PRIMAL framework with our approach, we use the same neural network architecture as in [Sartoretti et al., 2019] and refer to the code[3]. We also keep all the hyperparameters same when evaluating PRIMAL+KCO. We make a small change to get the final set of valid actions that the agent can take: we take intersection of the set of valid actions given by the PRIMAL environment ($validActions$) with the action set obtained by TD-SAT inference ($sddActions$). Concretely, the final set of valid actions that an agent can take is $validActions \cap sddActions$.

Average Total Objective: We show the plots of average total objective vs average sample count for different settings. Figure 4 shows the results for 4x4 with 2 agents, 8x8 with 6 agents, and 10x10 with 10 agents by DCRL and DCRL+KCO. We clearly observe that DCRL+KCO converges much faster than DCRL especially on the 10x10 grid. Figure 5 shows the comparison of MAPQN and MAPQN+KCO for 4x4 with 2 agents and 8x8 with 6 agents. Again, MAPQN+KCO is more sample efficient than MAPQN. The solution quality is better by MAPQN+KCO as well since all the agents are able to reach their respective destinations. Figure 6 shows the results of different approaches on 10x10 grid with obstacles. It clearly shows that DCRL+KCO and MAPQN+KCO are performing much better.

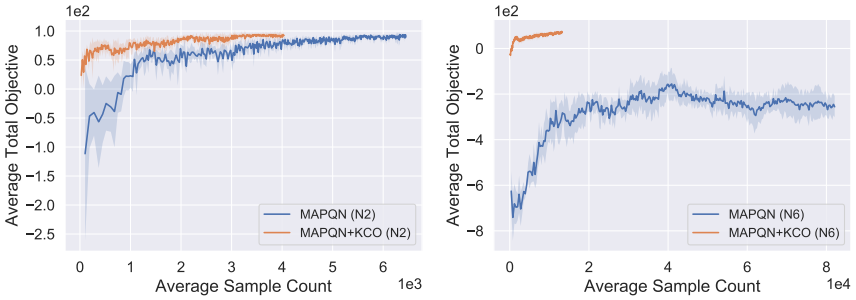
Stranded Agents: Table 1 shows the stranded agents on different settings. All agents can reach their destinations on all experimental settings by DCRL+KCO and MAPQN+KCO. DCRL performs reasonably well on open grids in terms of stranded agents. However, several agents did not reach the destination on 10x10 grid with obstacles by DCRL. MAPQN performs badly especially on 10x10 grid with obstacles.



(a) 4x4 grid (b) 8x8 grid (c) 10x10 grid

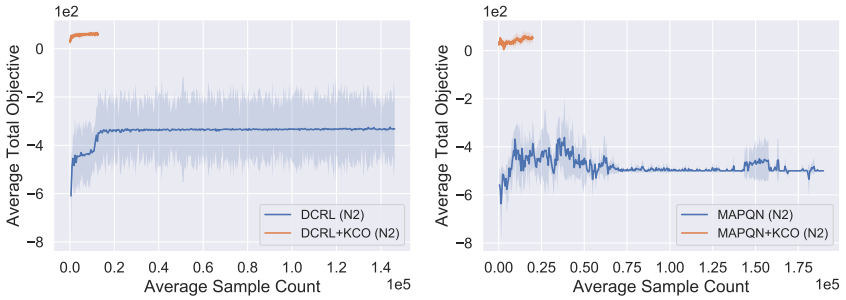
Figure 4: Sample efficiency comparison between DCRL+KCO and DCRL on open grids (N# denotes number of agents)

[2]https://github.com/lingkaching/Zone-Based-Multiagent-Pathfinding
[3]https://github.com/gsartoretti/distributedRL_MAPF



(a) 4x4 grid (b) 8x8 grid

Figure 5: Sample efficiency comparison between MAPQN+KCO and MAPQN on open grids (higher quality better)



(a) DCRL+KCO (b) MAPQN+KCO

Figure 6: Sample efficiency results on 10x10 grid with obstacles

Setting	**DCRL**	**DCRL+KCO**	**MAPQN**	**MAPQN+KCO**
4x4 N2	0	0	0	0
4x4 N6	0	0	0	0
8x8 N6	0	0	3.6	0
10x10 N10	0.2	0	-	-
10x10 N30	0.6	0	-	-
10x10 with obstacles N2	1.4	0	2	0
10x10 with obstacles N5	1	0	5	0
10x10 with obstacles N10	2.2	0	8.8	0

Table 1: Average stranded agents comparisons on different settings (N# denotes number of agents)

References

Takeru Inoue, Hiroaki Iwashita, Jun Kawahara, and Shin-ichi Minato. Graphillion: software library for very large sets of labeled graphs. *International Journal on Software Tools for Technology Transfer*, 18(1):57–66, 2016.

Masaaki Nishino, Norihito Yasuda, Shin-ichi Minato, and Masaaki Nagata. Zero-suppressed sentential decision diagrams. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 1058–1066, 2016.

Masaaki Nishino, Norihito Yasuda, Shin-ichi Minato, and Masaaki Nagata. Compiling graph substructures into sentential decision diagrams. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 1213–1221, 2017.

Jiajing Ling, Tarun Gupta, and Akshat Kumar. Reinforcement learning for zone based multiagent pathfinding under uncertainty. In *International Conference on Automated Planning and Scheduling*, pages 551–559, 2020.

Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, T. K. Satish Kumar, Sven Koenig, and Howie Choset. PRIMAL: Pathfinding via Reinforcement and Imitation Multi-Agent Learning. *IEEE Robotics and Automation Letters*, 4(3):2378–2385, 2019.