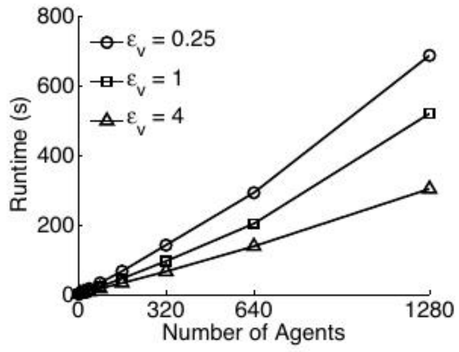
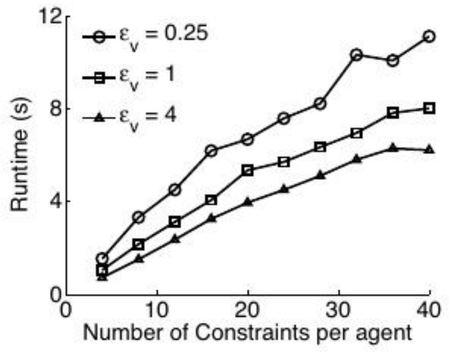
M-DPFP and provides a comparable solution quality. For example, in the problem where each agent has 4 tasks, M-DPFP with $\kappa = 0.2$ finds a solution with quality of 27.7 in 247.6 seconds while SPAC with $\epsilon_v = 0.1$ finds a better solution with quality of 34.7 in 0.6 seconds (412-fold speedup).

#Tks	SPAC $\epsilon_v = 1$	SPAC $\epsilon_v = 0.1$	M-DPFP $\kappa = 0.25$	M-DPFP $\kappa = 0.2$
3	0.1 (24.8)	0.1 (26.0)	0.4 (14.7)	2.7 (16.4)
4	0.4 (34.5)	0.6 (34.7)	21.7 (24.9)	247.6 (27.7)
5	0.4 (34.6)	0.7 (34.9)	63.2 (36.2)	NA
6	0.6 (36.9)	0.8 (38.2)	NA	NA

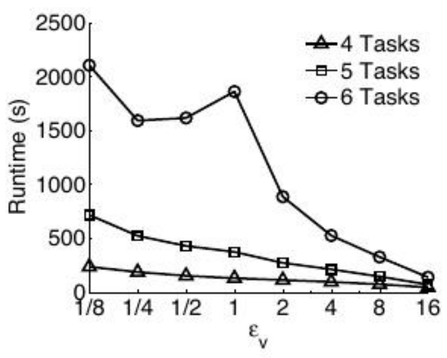
Table 1: Runtime in seconds & (quality): SPAC vs M-DPFP



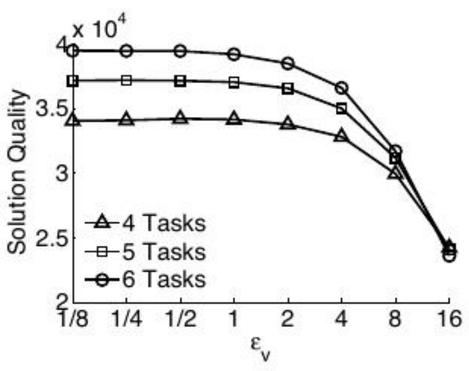
(a) Scale to large number of agents.



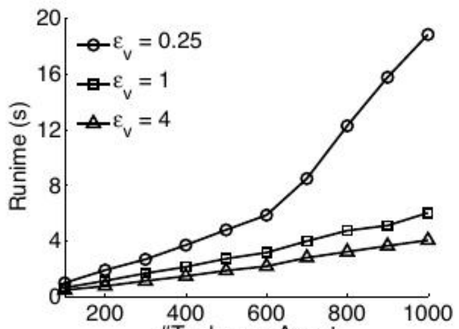
(b) Scale to large number of constraints.



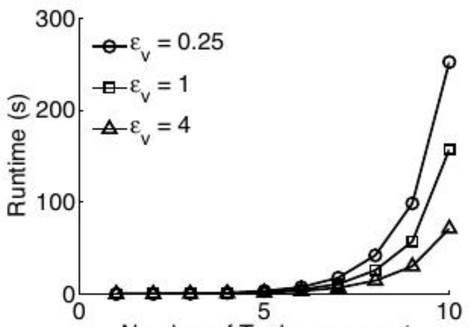
(c) Runtime of varying ϵ_v in 1000-agent problems.



(d) Solution quality of varying ϵ_v in 1000-agent problems.



(e) Scale to large number of ordered tasks.



(f) Scale to large number of unordered tasks.

Figure 4: Results of SPAC.

Next, we conduct experiments to show SPAC's scalability. The task durations are chosen randomly from a set of pre-generated uniform distributions with mean in $[0, 0.5]$ and variance of 0.01. Task values are chosen uniformly randomly between 0 and 10, and joint rewards are chosen uniformly randomly between 0 and 20 (-20 for penalties). The temporal constraints are randomly assigned between tasks.

First we fix the number of tasks per agent to 5, the number of constraints per agent to 8, and vary the number of agents from 5 to 1280 with different error bounds $\epsilon_v = 0.25, 1, 4$. For each setting we create 100 random problems and report the average runtime results in Figure 4(a). SPAC scales almost linearly with respect to the number of agents for every ϵ_v setting. Second, we fix the number of agents to 10, the number of tasks per agent to 5, and vary the number of constraints per agent. Again Figure 4(b) shows the average results over 100 random problems. As we can see, the runtime is roughly linear to the number of constraints per agent.

Next, we test on large scale problems where there are 1000 agents with 8 constraints per agent. We consider 4, 5, and 6 tasks per agent with varying ϵ_v from 0.125 to 16. Figure 4(c) and Figure 4(d) show the runtime and solution quality results of the average over 10 random problems respectively. The x-axes in both figures are in log-scale. These figures show that using smaller value of ϵ_v helps find better solutions at the cost of increasing runtime (however, the benefit of decreasing $\epsilon_v \leq 0.5$ is negligible). SPAC with $\epsilon_v = 0.5$ is seen to solve a 1000 agent problem with 5 tasks per agent in 8 minutes – SPAC scales beyond capabilities of current algorithms.

Finally, we test the scalability with respect to the number of tasks per individual agent. To this end, we create 10-agent problems with 8 constraints per agent and varying number of tasks with task ordering either fixed or not fixed. If the task ordering is fixed, the number of discrete states of each agent is the number of tasks $+1$ and at each discrete state there is only one non-wait action. The problem with fixed task ordering is essentially to find an optimal starting time for the next task at every decision step. When the task ordering is not fixed, the problem is much more difficult – the number of discrete states is exponential to the number of tasks and the agent needs to choose one from the available tasks at each decision step. Figure 4(e) shows SPAC scales linearly with respect to the number of tasks per agent when the task ordering is fixed. It runs under 20 seconds for $1,000$ tasks, highlighting its scalability. However as shown in Figure 4(f), if the task ordering is not fixed, SPAC does not scale well – solving a problem with 10 unordered tasks per agent requires 250 seconds with $\epsilon_v = 0.25$.

To test SPAC's solution quality – in the absence of an efficient global optimal solver – we run SPAC with $\epsilon_v = 0.25$ for 5, 10, and 20 seconds and compare with best solution found in running SPAC with $\epsilon_v = 0.1$ for 2 hours. As Table 2 shows, we can find good solutions with quality of at least 92% of the best by running SPAC for 5 seconds, and of at least 95% of the best by running SPAC for 20 seconds. Thus, SPAC can find reasonable solutions at a cheap computational cost.

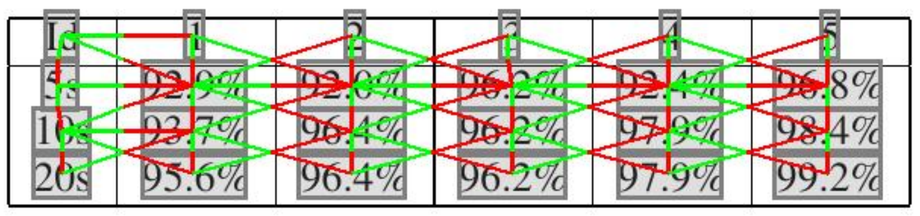


Table 2: SPAC: comparing to best solution found in 2 hours.

6 Conclusion

This paper presents three key contributions. First, it introduces a novel continuous-time model (MCT-MDP) for mul-