Report Multi-agent Systems Module II

Tom Jacobs (s0214835), Thomas Uyttendaele (s0215028)

March 29, 2013

Part I

1 Value iteration in MDPs

The different implementations of Value Iteration have been verified using the Q^* provided for the *loadunload* problem. They match exactly within the accuracy of the reference matrix.

Rewards

The average sums of discounted rewards and relevant standard deviations (between parenthesis) have been visualised in 2. They are the result of 100 runs with a horizon of 100 steps. The results for *MLS* and Q_{MDP} will be further discussed in the next section.

A brief comparison between the results for the *Initial Start Distribution* and the *Fully Observable* implementations shows that the latter produces slightly smaller reward values. The main reason is that a horizon chosen at 100 steps still somewhat limits the maximal achievable reward.

Performance statistics

The algorithm for *Value Iteration* presented in the slides was implemented with a few minor modifications for performance reasons. Looping is first done over actions and then nested the different states. To exploit the sparsity of some of the matrices the Matlab *find()* function is used.

To optimize the calculation of the Q^* matrices a second implementation was done where the fetching of the non-zero elements from sparse matrices is done up front. The average running time and standard deviation of running times in seconds for *Value Iteration* can be found in table 1. Here the new algorithm is the modified version with fetching before the actual loops.

note: for all timing results presented in this report the first value is removed before processing, as it is usually an outlier due to initialization and caching delays.

Problem	μ_{Old}	μ_{New}	σ_{Old}	σ_{New}
loadunload	3.744e-02	3.213e-02	1.035e-03	1.305e-03
hallway	7.463e-01	7.167e-01	1.0582e-02	9.933e-03
hallway2	1.167e+00	1.115e+00	1.024e-02	9.640e-03
trc	1.396e+01	1.367e+01	3.102e-01	2.993e-01

Table 1: Value Iteration runtime statistics for both algorithms

	Initial Start Distribution	Fully Observable	MLS	Q_{MDP}
hallway	1.536	1.521 (0.5654)	0.8552 (0.3829)	0.3066 (0.3750)
hallway2	1.201	1.179 (0.4823)	0.1896 (0.1688)	0.09883 (0.2166)
trc	10.89	10.54 (2.808)	4.634 (2.524)	0 (0)

Table 2: Reward values for the different algorithms

Paths followed

In figures 5 and 6 a path followed in the hallway and trc problem is visualized. This is done by first displaying the respective state and using the height of the bars or darkness of the circles to show the continuation in time. If a state is visited multiple times the last visit is drawn.

The figures marked (b) show which states are visited throughout the run itself.

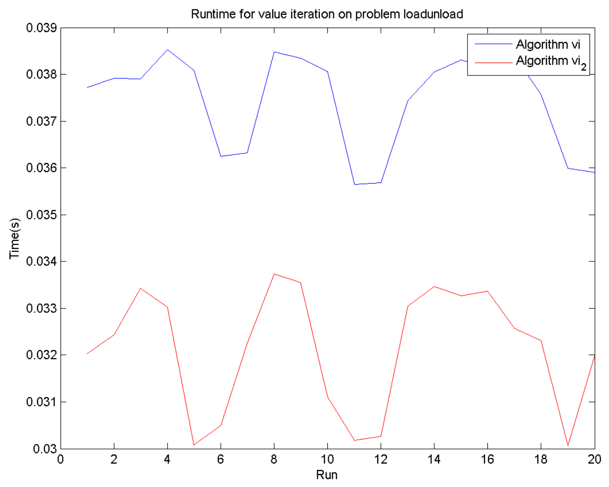


Figure 1: The loadunload problem

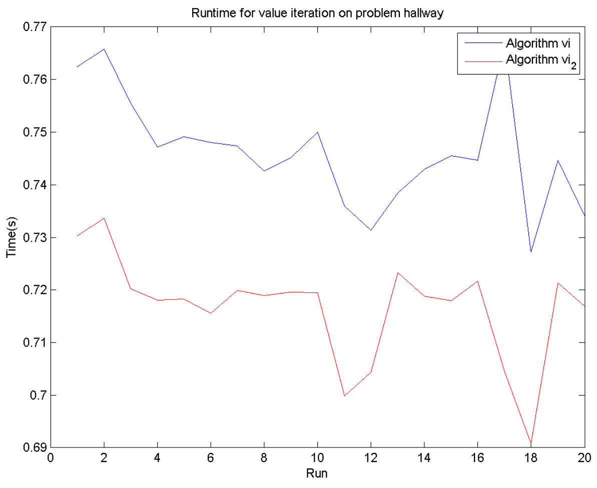


Figure 2: The hallway problem

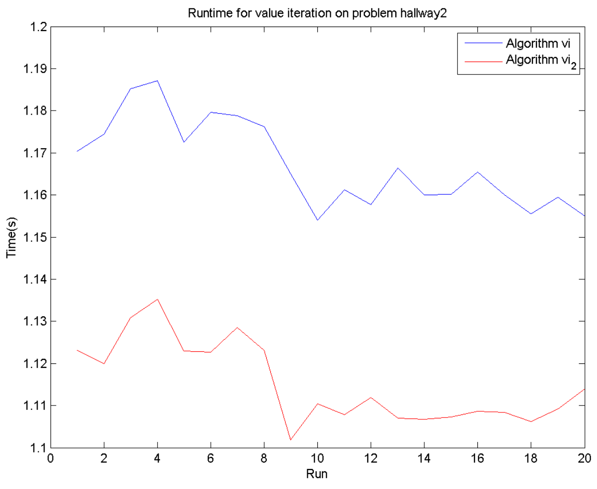


Figure 3: The hallway2 problem

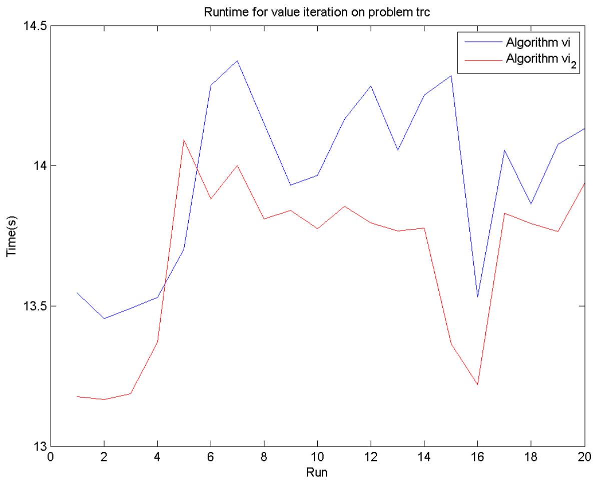
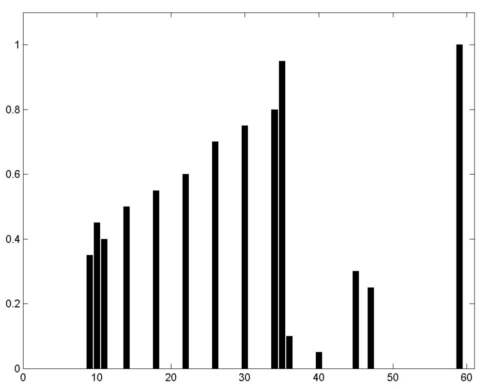
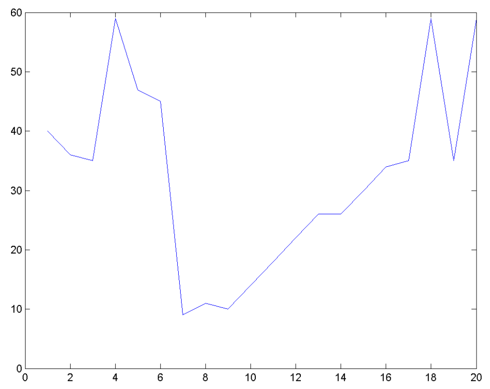


Figure 4: The trc problem

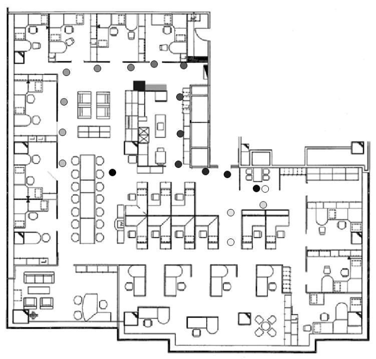


(a) Last time in state, higher means more recent

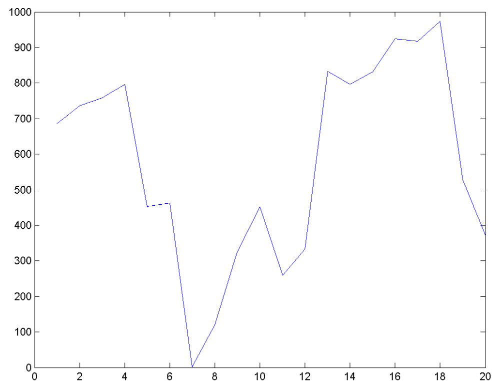


(b) Run through the states

Figure 5: A path through the hallway problem

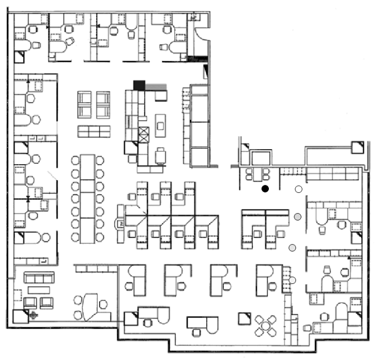


(a) Last time in state, darker means more recent

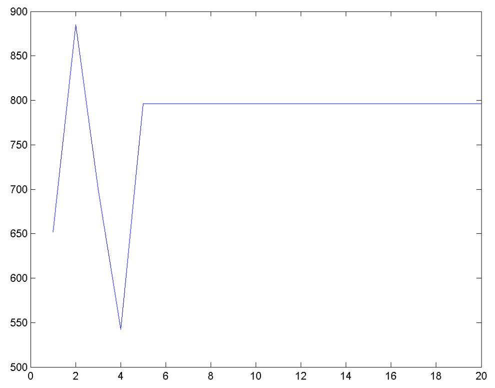


(b) Run through the states

Figure 6: A path through the trc problem

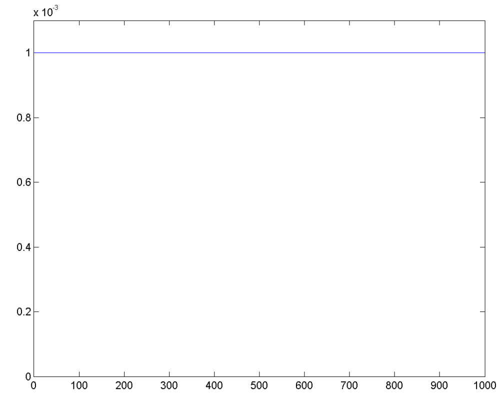


(a) Last time in state, darker means more recent

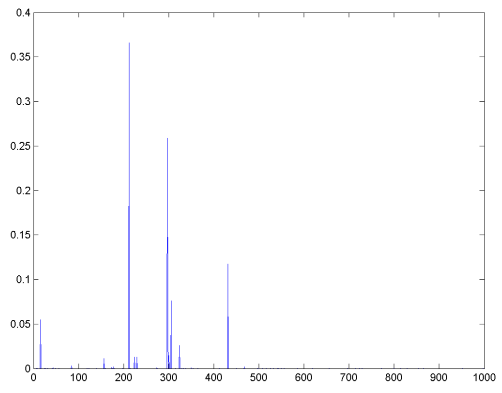


(b) Run through the states

Figure 7: The trc problem during a Q_{MDP} run

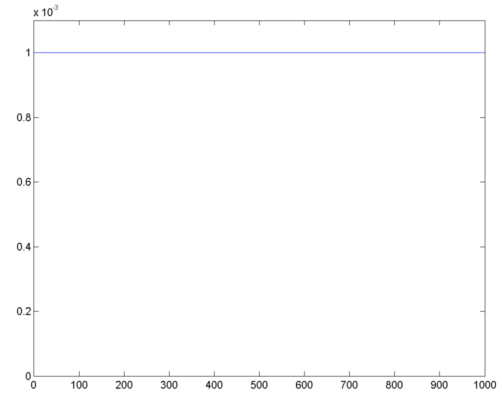


(a) Beliefs in the starting state

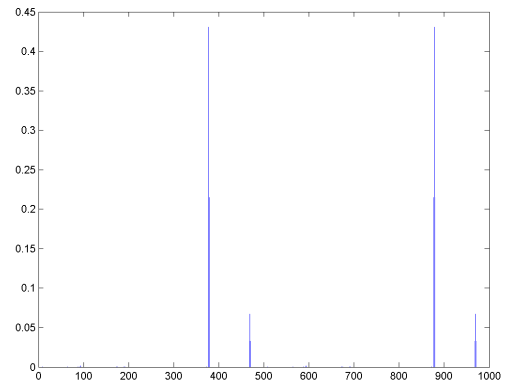


(b) Beliefs in the 20th state

Figure 8: The beliefs in the trc problem during an MLS run

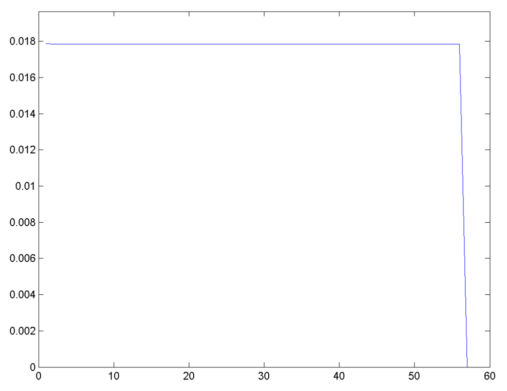


(a) Beliefs in the starting state

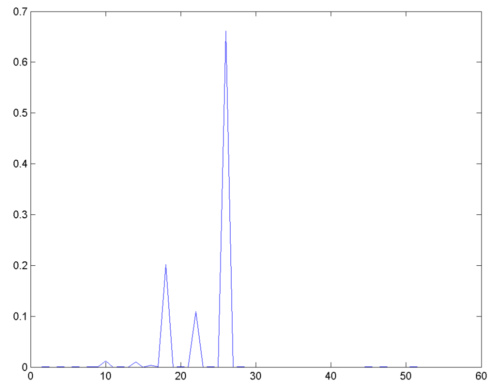


(b) Beliefs in the 20th state

Figure 9: The beliefs in the trc problem during a Q_{MDP} run



(a) Beliefs in the starting state



(b) Beliefs in the 20th state

Figure 10: The beliefs in the hallway problem during a successful Q_{MDP} run

	MLS	Q_{MDP}	$Pers_{10}$	$Pers_{100}$	$Pers_{1000}$	$Pers_{10000}$
μ_{Reward}	0.1896	0.09883	0.1548	0.2497	0.2473	0.23379
σ_{Reward}	0.1688	0.2166	0	0.007125	0.004098	0.006262
$\mu_{Runtime}$	2.746	2.675	2.1368	55.57	61.99	69.63
$\sigma_{Runtime}$	0.01467	0.014556	0.2102	0.3610	0.9568	2.4624

Table 3: Performance of the different algorithms for the hallway2 problem

2 Partially observable environments: heuristic methods

In table 2 a comparison between the different algorithms can be seen, with averages and standard deviation over 100 runs, each with a horizon of 100 steps. While the fully observable case produces high reward values with a relatively small standard deviation, the MLS and Q_{MDP} implementations output lower and much more varied results.

The Q_{MDP} implementation often stalls during execution when a state is reached where staying in the current state is the best available action. An extreme example here is the result achieved (or clear lack thereof) with the trc problem, where not a single run produced a reward. The evolution of the states in one such run is seen on figure 7 (a) and (b).

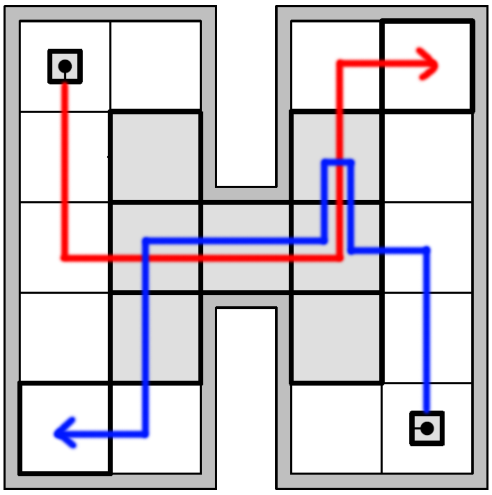
An explanation for the MLS and Q_{MDP} behaviour as compared to the fully observable case is the fact that operation in a partially observable environment lacks total information and thus can't always calculate the best possible course of actions.

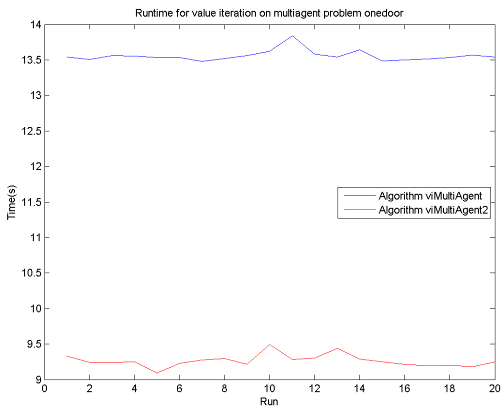
A short visualization of the evolution of beliefs in both the MLS and Q_{MDP} implementation for the trc problem can be seen in figures 8 and 9. The Q_{MDP} case clearly shows to equal sets of peaks, which causes the stalling mentioned earlier. A similar visualization for the hallway problem using the Q_{MDP} implementation and a run that did actually finish can be seen in figure 10

3 Partially observable environments: point-based methods

A performance comparison of the point-based Perseus algorithm and the Q_{MDP} and MLS implementations was done using the hallway2 problem. An overview can be seen in table 3, where $Pers_{\otimes}$ represents the execution of the Perseus algorithm with different sampling parameters

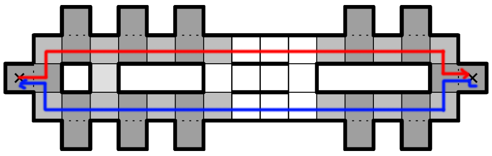
This table demonstrates a clear difference between the relevant algorithms. To keep the total calculation time within reasonable bounds the Perseus algorithm was run 5 times with each parameter setting, and the statistics were calculated on these outputs. It's clearly visible that while both the reward values and calculation times for the Perseus algorithm differ significantly when the number of sampled beliefs is increased from 10 to 100, no real improvement (on the contrary actually) is seen for further increase. The results Perseus achieves at higher sampling parameters are indeed better than MLS or Q_{MDP}, but at a very high cost in computation. Depending on the required accuracy the trade-off can be made to exchange the achieved reward and computational complexity.

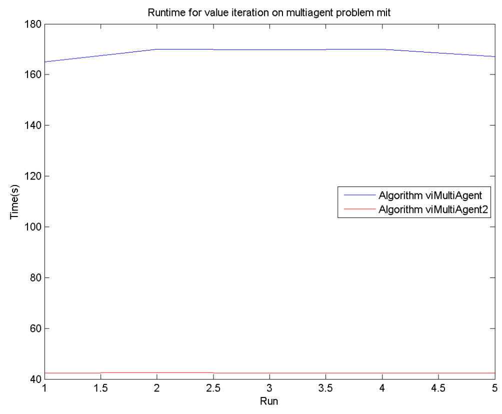

(a) Paths taken by the robots during a run with the multi-agent algorithms



(b) The runtime needed for both the normal and sparse implementations

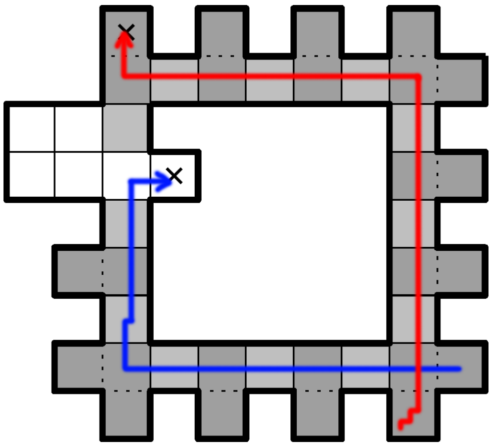
Figure 11: Multi-agent runs on the onedoor problem

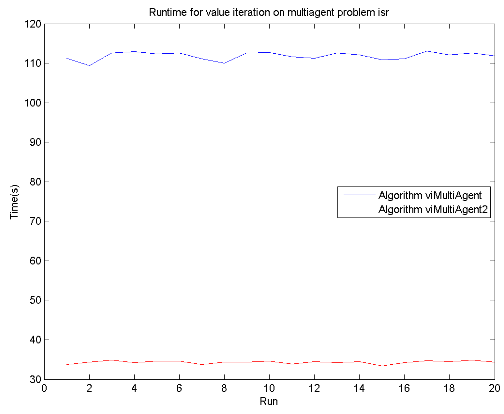

(a) Paths taken by the robots during a run with the multi-agent algorithms



(b) The runtime needed for both the normal and sparse implementations

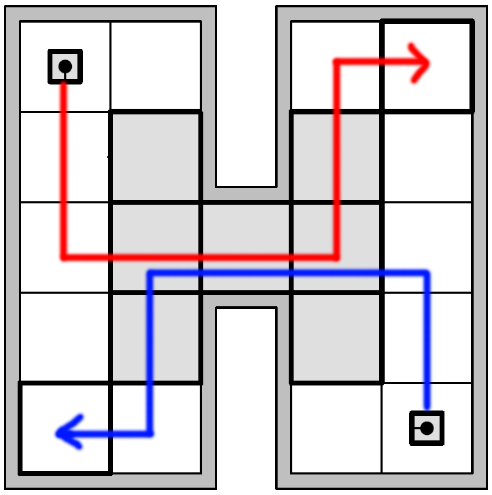
Figure 12: Multi-agent runs on the MIT problem


(a) Paths taken by the robots during a run with the multi-agent algorithms

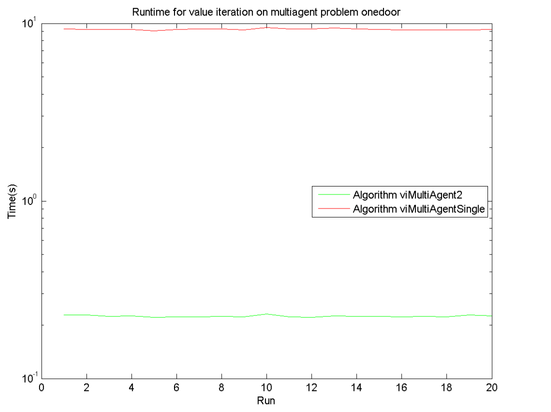


(b) The runtime needed for both the normal and sparse implementations

Figure 13: Multi-agent runs on the ISR problem

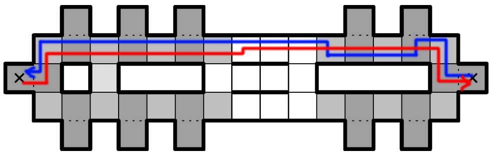


(a) Paths taken during a run with independently acting robots

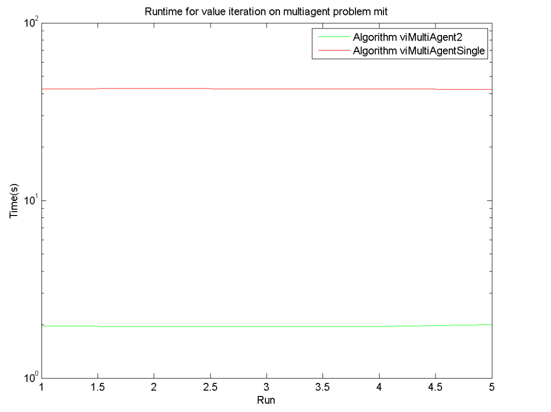


(b) Runtime comparison for the joint and separately acting algorithms

Figure 14: Independently acting multi-agent runs on the onedoor problem

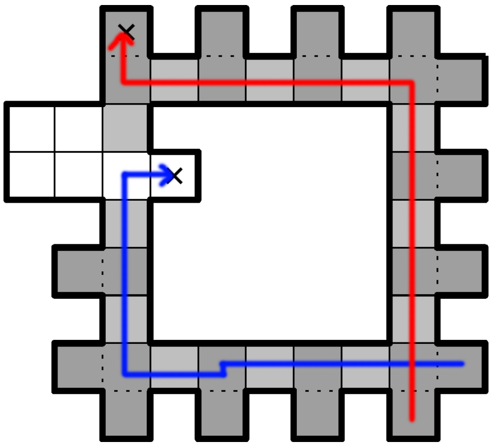


(a) Paths taken during a run with independently acting robots

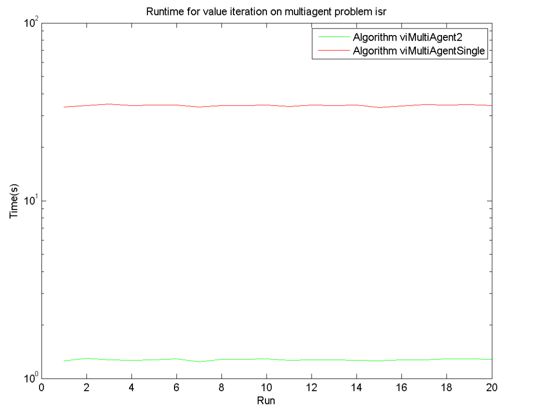


(b) Runtime comparison for the joint and separately acting algorithms

Figure 15: Independently acting multi-agent runs on the MIT problem



(a) Paths taken during a run with independently acting robots



(b) Runtime comparison for the joint and separately acting algorithms

Figure 16: Independently acting multi-agent runs on the ISR problem

Part II

4 Multi-agent planning under uncertainty

Comparison between multi-agent problems solved with MMDP

In this multi-agent case two robots will navigate different problem maps while avoiding collision. As the output generated by these algorithms provides a very extensive visualization of every step taken by both robots a clearer notation is chosen by showing only the path taken by each robot overlaid on the map. Every time a robot stays in place this is shown as a small zigzag in the path. Figures 11,12 and 13 show these runs and a comparison in calculation time between the regular and sparse implementations.

Performance comparison between Single-agent and Multi-agent implementations

Similar to the previous section figures 14, 15 and 16 show the paths taken by robots with separate decision making and a runtime comparison between the joint and this implementations. Computational performance can be seen to be significantly worse, and the MIT problem clearly shows that the lack of coordination produces less than optimal results.

Finally, there are several ways in which collisions between robots could be avoided:

1. A dynamic programming implementation where all states that would have robots arriving on the same cell are pruned. This wouldn't significantly improve performance as there are usually very few states that are pruned this way.
2. Looking around in the neighbouring cells in combination with a predetermined priority between robots would allow both robots to detect the presence of the other, and independently decide which one can go first.
3. If communication is possible querying the other robot for both its location and intentions, again with clear priorities, would allow semi-perfect calculability of mutually optimal paths.

collision avoidance: 3) Query position+intention of other robot, clear priority between robots. 2) Look in 3x3 (for onedoor) + clear priority between robots. 1) Dynamic programming -¿ Alle states waarbij mogelijks zelfde toestand prunen. (hier niet al te veel

Part III

5 Overview of Algorithms

Value Iteration

There scripts can be found in the folder `yourcode_new`.

- *vi.m*: The basic value iteration for single agent problems. It is optimized by using the build-in function `find` so it only iterates over the non-zero elements of the transition matrix.
- *vi_2.m*: A small optimization on vi.m by calculating the non-zero before and using these cached values for further calculations.
- *viMultiAgent.m*: The modification of vi_2.m towards multiple agents because the non-existing of specific field in the problem structure.

- *viMultiAgent2.m*: A significant modification on viMultiAgent so it transforms the transition matrix into a sparse matrix.
- *viMultiAgentSingle.m*: The modification of vi_2.m so the value iteration can calculate the different Q matrix for every single robot in a single script.

Sample trajectory

There scripts can be found in the folder `yourcode_new`.

- *sampleTrajectories.m*: The basic trajectory sampler for single agents, working with full knowledge of the environment. The robot knows it exact location in every step.
- *sampleTrajectoriesWithBeliefsMLS.m*: The trajectory sampler for single agents, working with a certain belief of his position in the environment. The choice of the action is based on the most likely state, which is named MLS.
- *sampleTrajectoriesWithBeliefsQMDP.m*: The trajectory sampler for single agents, working with a certain belief of his position in the environment. The choice of the action is based on the beliefs, which is named Q_{MDP}.
- *sampleTrajectoriesMultiAgents.m*: The trajectory sampler for multiple agents with a combined knowledge of the environment. Also no beliefs are introduced in this problem.
- *sampleTrajectoriesMultiAgentsSingle.m*: The trajectory sampler for multiple agents with no combined knowledge of the environment. Also no beliefs are introduced in this problem.

The reward coming from following a certain path can be afterwards calculated based on the sequence of states and actions. An example can be found in the scripts `calculating_rewards.m` and `calculating_rewards.m` in the folder `yourcode_graphical_scripts`.

Graphical scripts

All the scripts used for generating the images and data in this report can be found in the folder `yourcode_graphical_scripts`.

6 Time Spent

Most of the time spent on this project was done cooperatively and thus the effort is distributed quite evenly. Based on combined agenda's we estimate both having spent between 28 and 35 hours on this project. This does not include the many hours it took to generate some images and statistics over several runs in the more expansive problems.