Reachability properties for uncertain MDPs

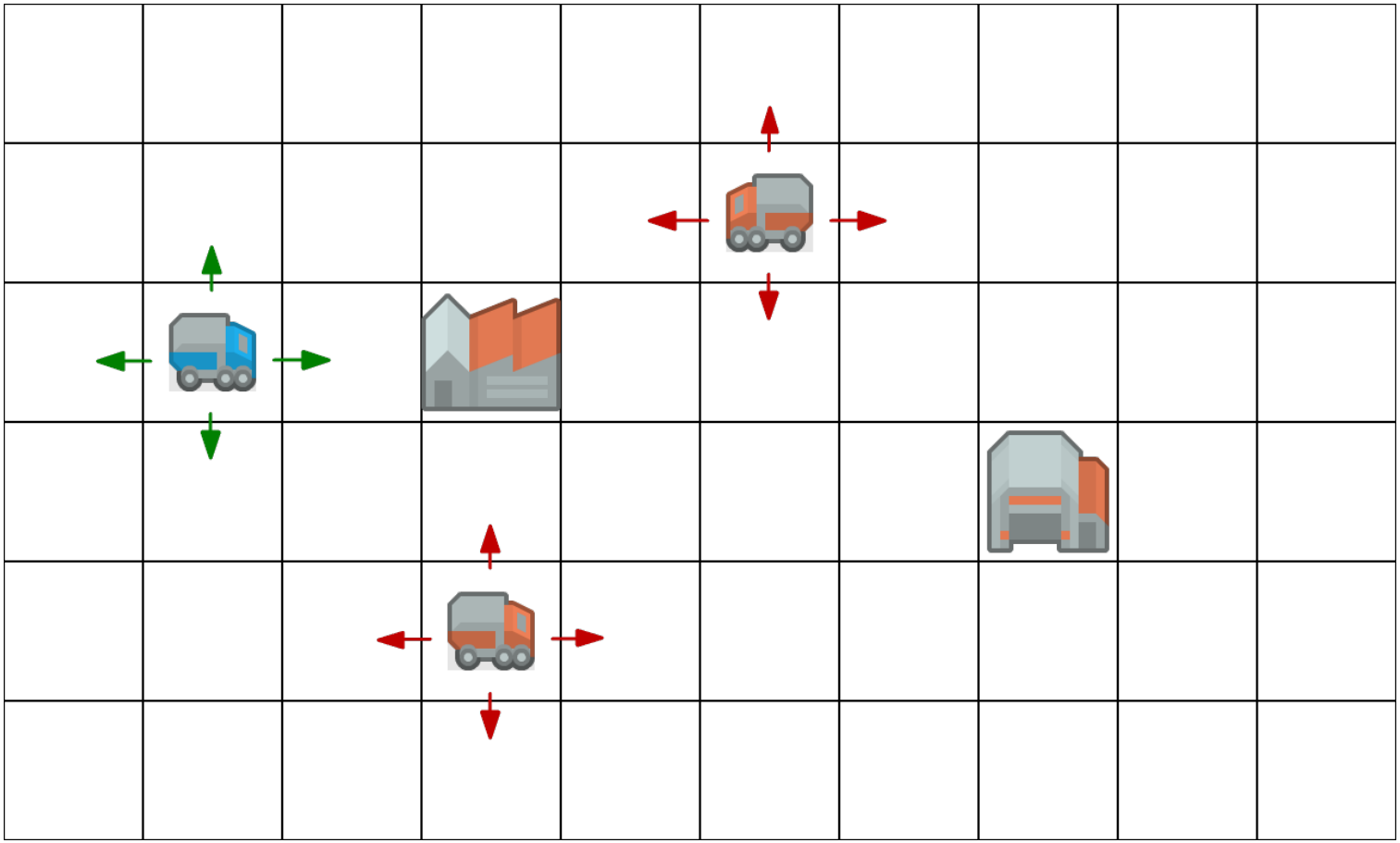
Marnix Suilen

Supervisor: Dr. Nils Jansen

April 12, 2018

Illustrated problem[1]

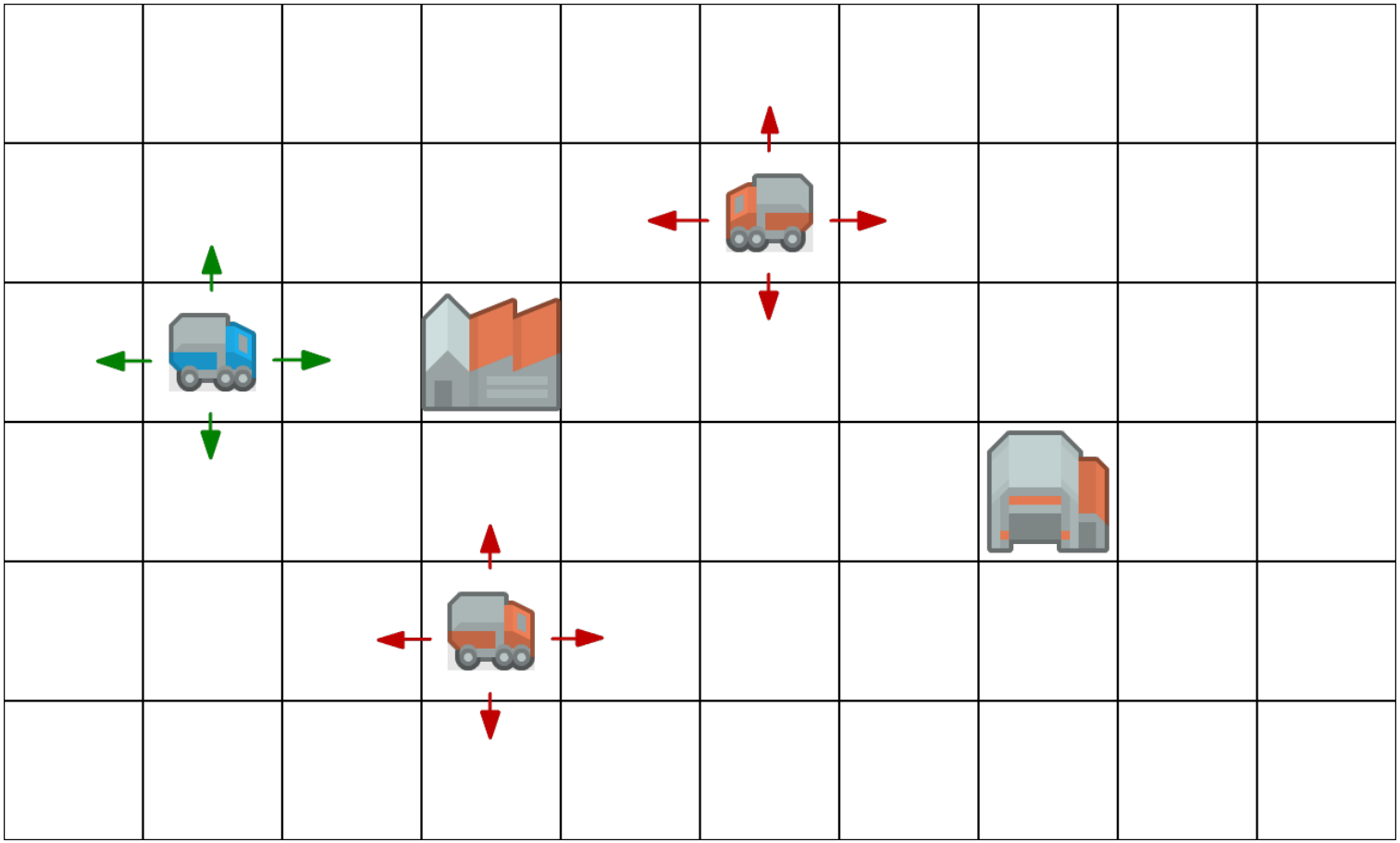
A robot trying to move through the area without bumping into anything.



[1] Images under CC0 1.0 license from `https://kenney.nl/assets/sci-fi-rts`

Illustrated problem[1]

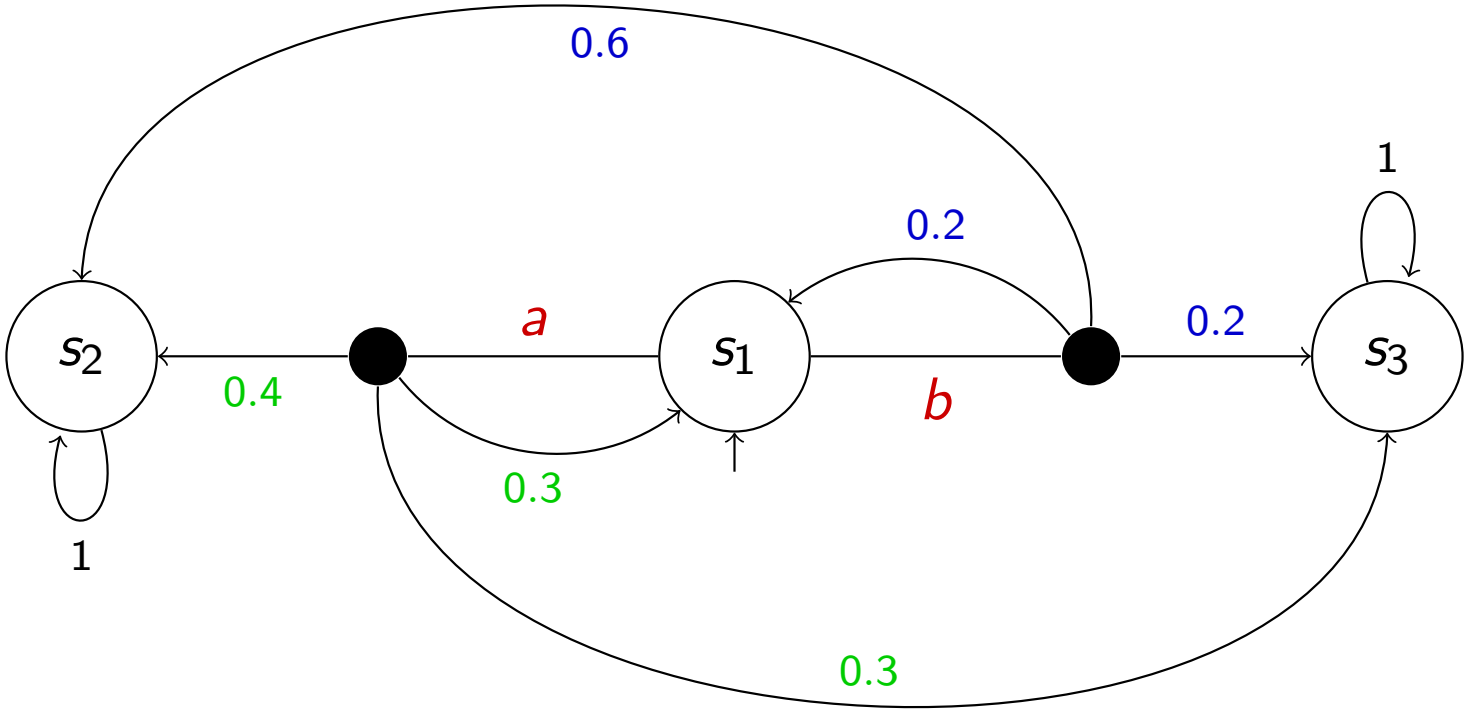
Assumption: probabilistic movement of other objects is exactly known.



[1] Images under CC0 1.0 license from `https://kenney.nl/assets/sci-fi-rts`

A Markov decision process (MDP)

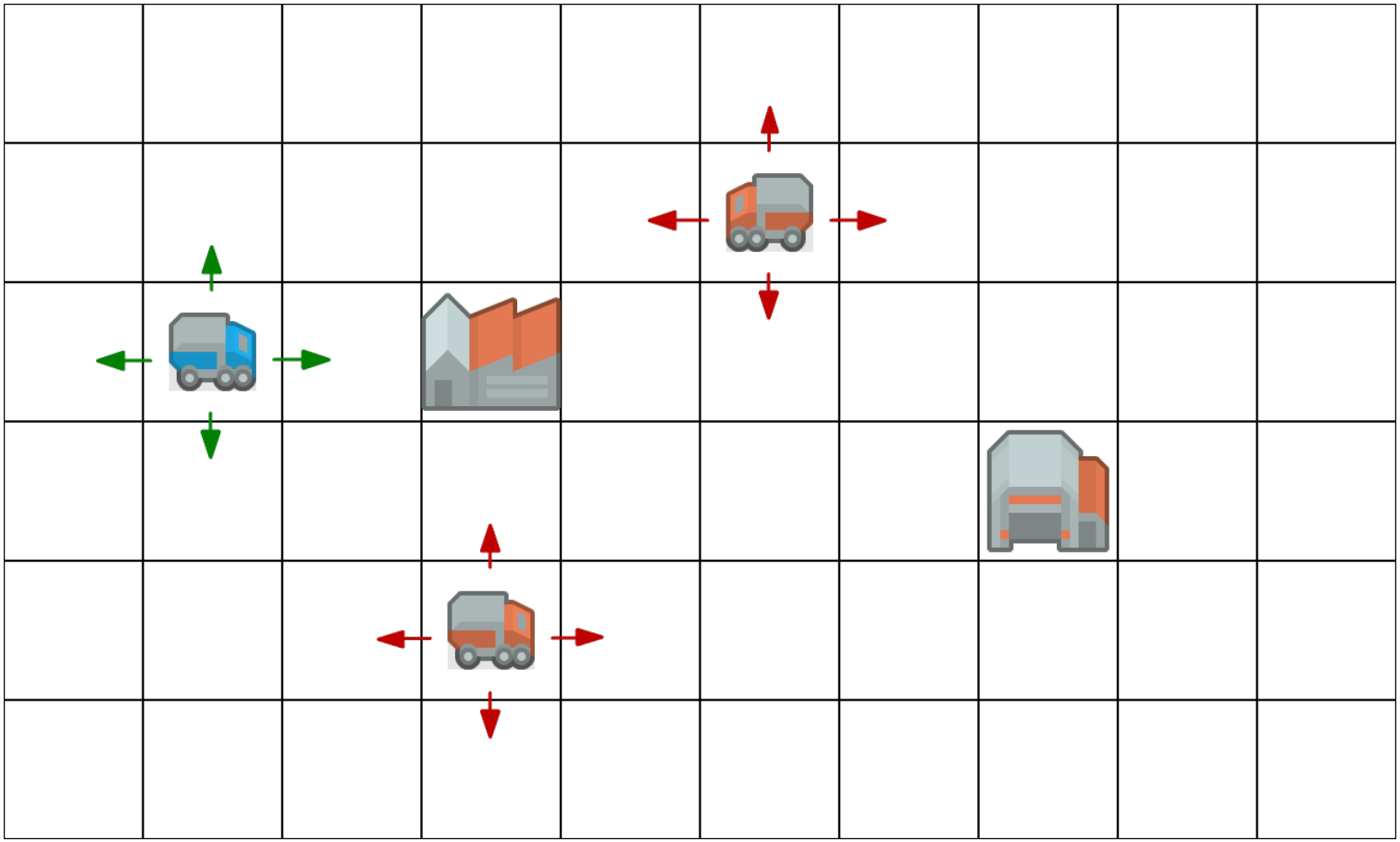
Scenario can be modeled by a Markov decision process.



Self loops of probability 1 for states s_2 and s_3 are over both actions

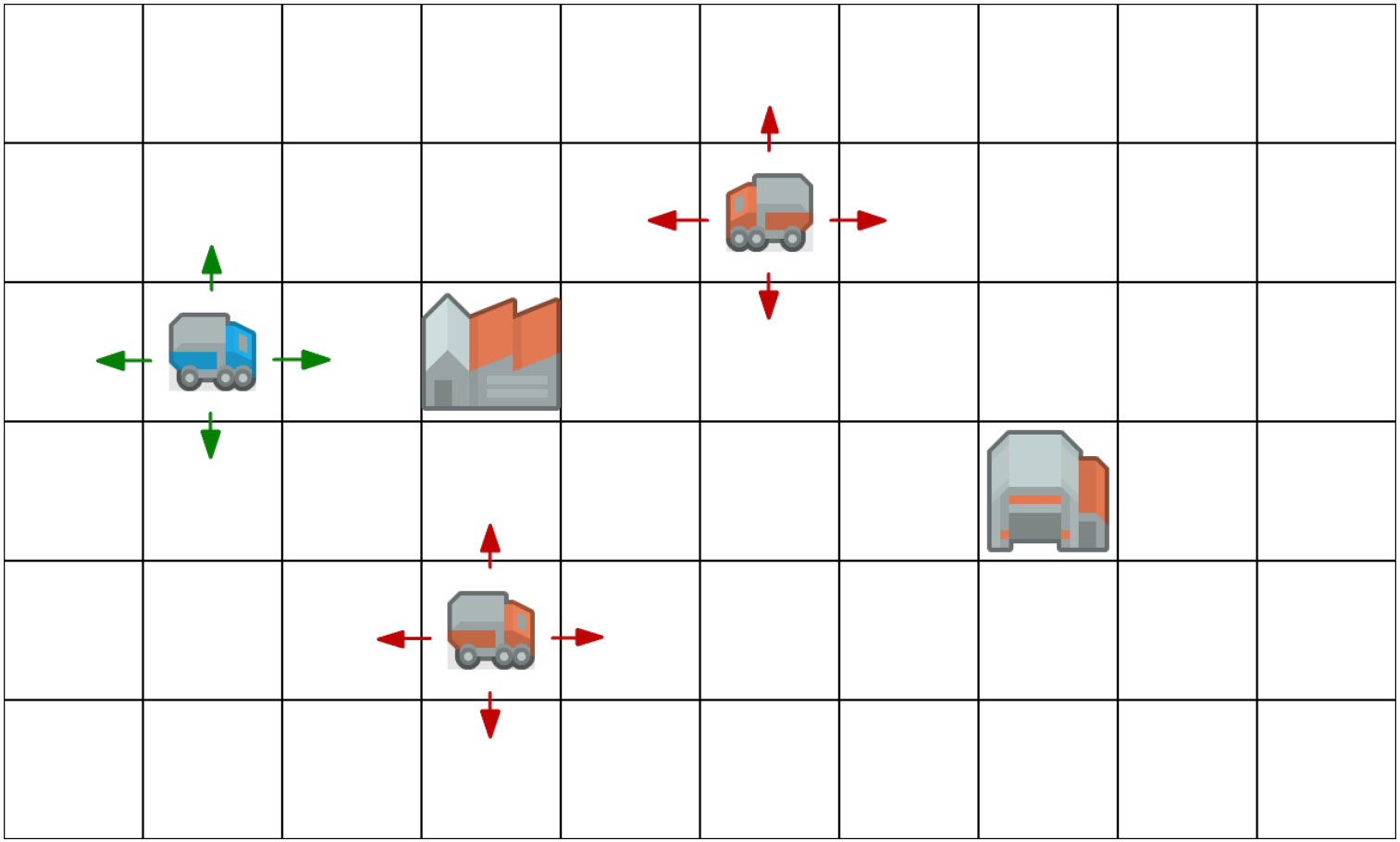
Illustrated problem revisited

A self-learning robot.



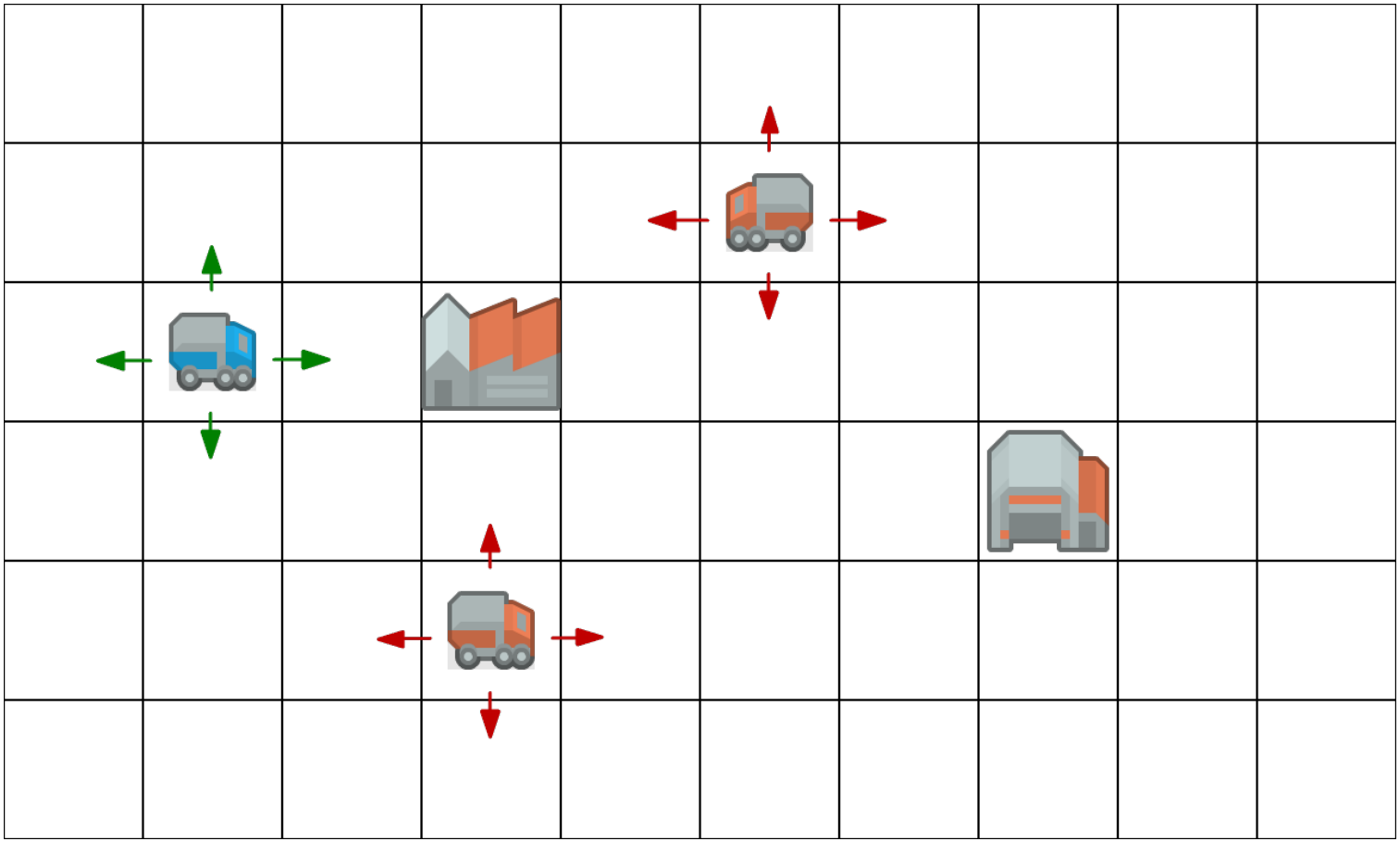
Illustrated problem revisited

Exact probabilities are unrealistic.



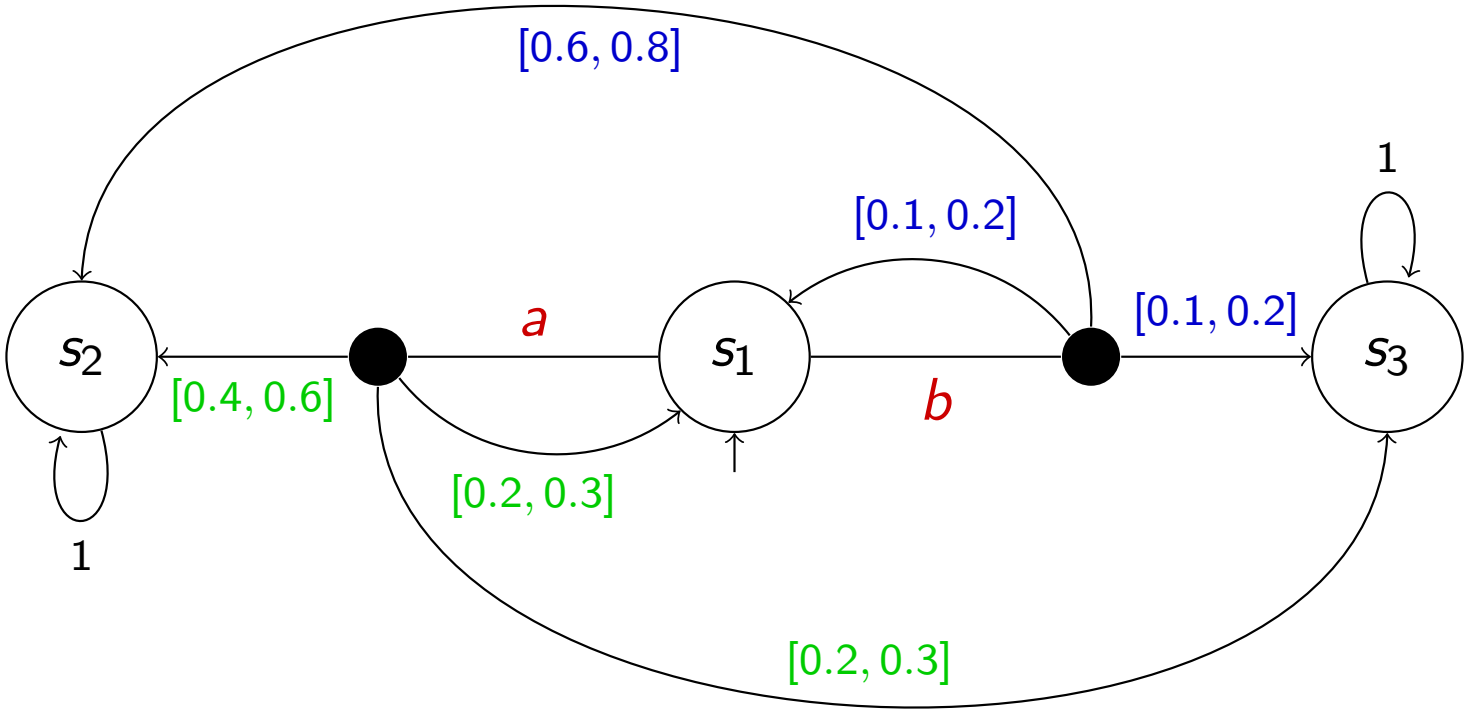
Illustrated problem revisited

Use intervals: probability $\pm$ confidence.



An uncertain Markov decision process (uMDP)

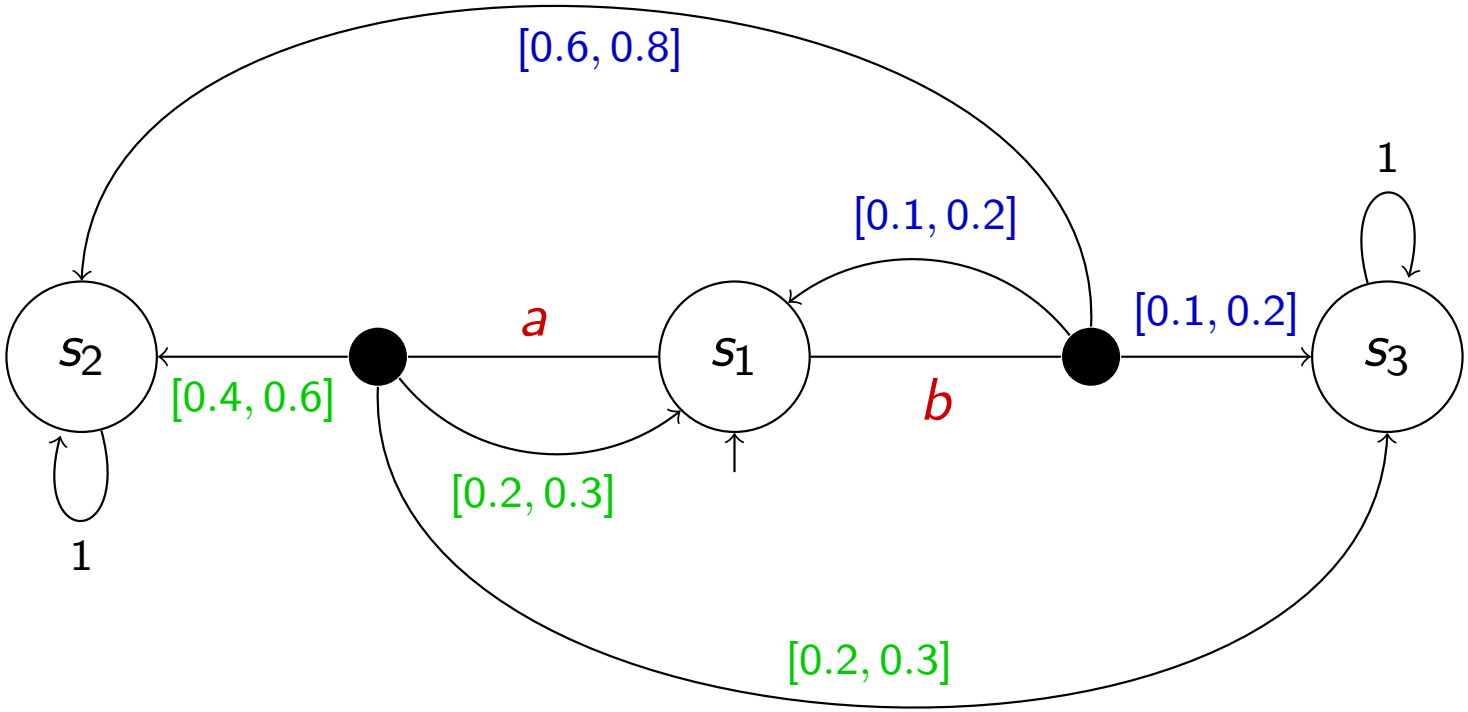
Scenario can be modeled by an uncertain Markov decision process.



Self loops of probability 1 for states s_2 and s_3 are over both actions.

An uncertain Markov decision process (uMDP)

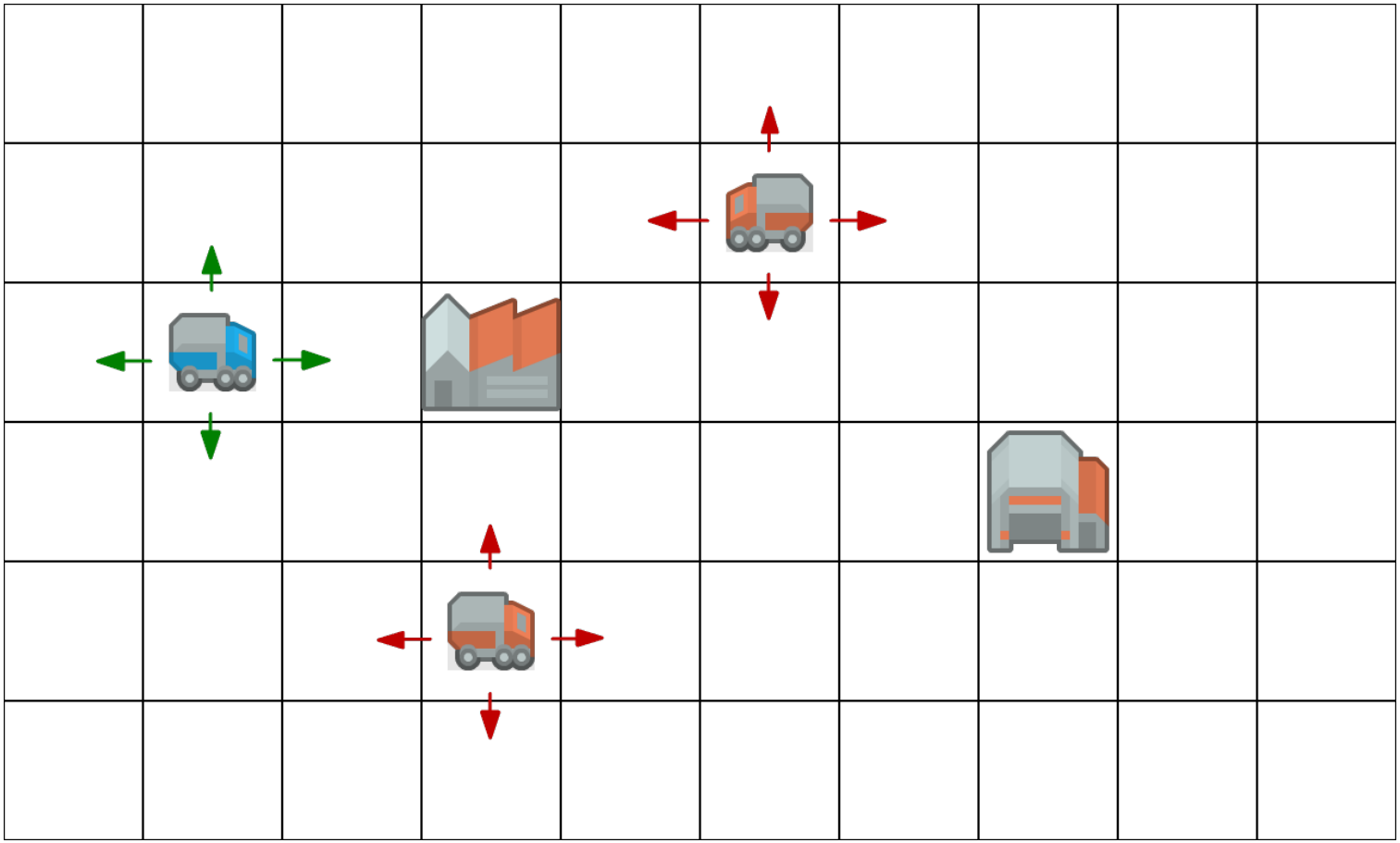
Scenario can be modeled by an uncertain Markov decision process.
Uncertainty given by intervals, other kinds of uncertainty exist too.



Self loops of probability 1 for states s_2 and s_3 are over both actions.

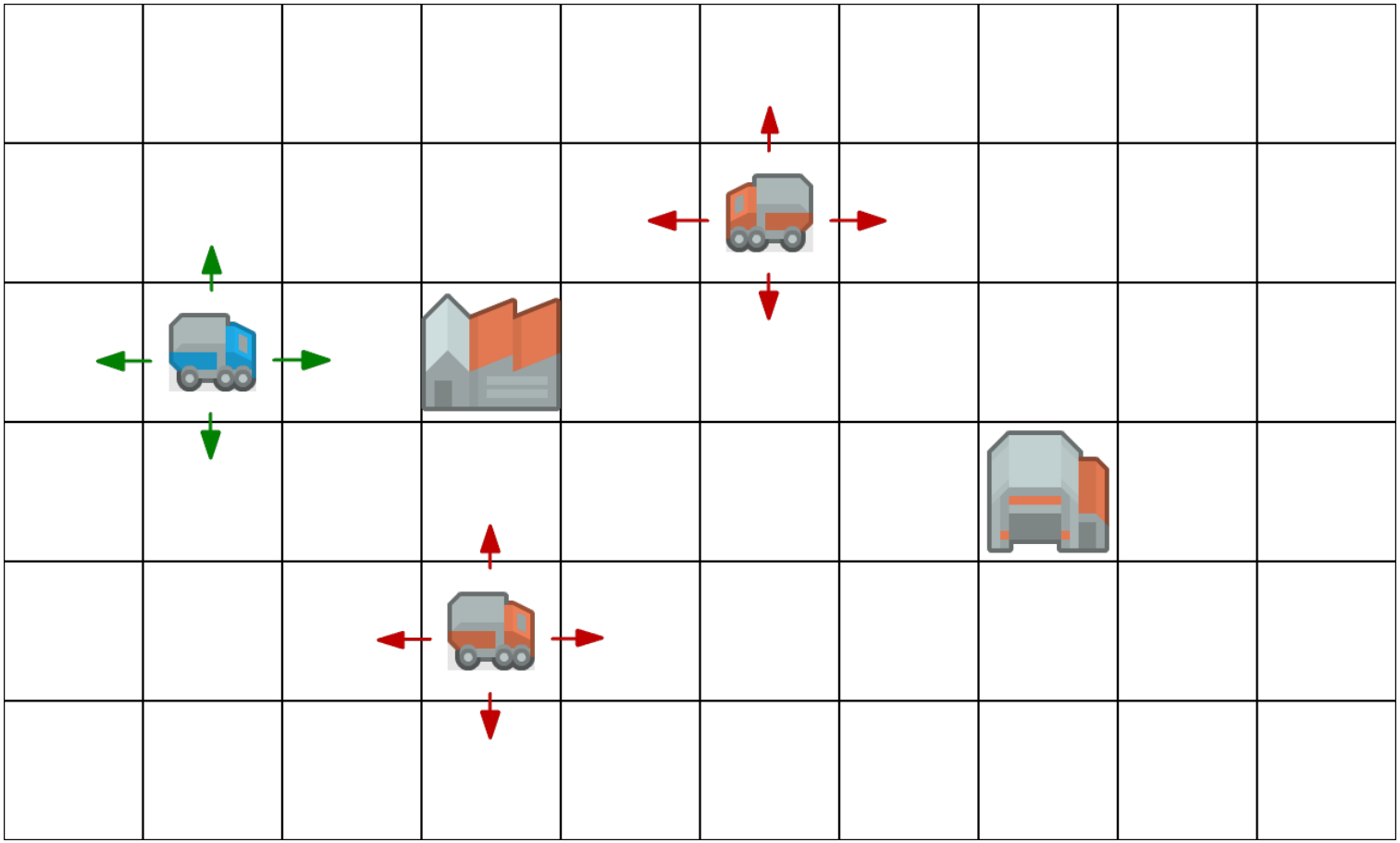
Aim of this thesis

Reach the other side of the area,



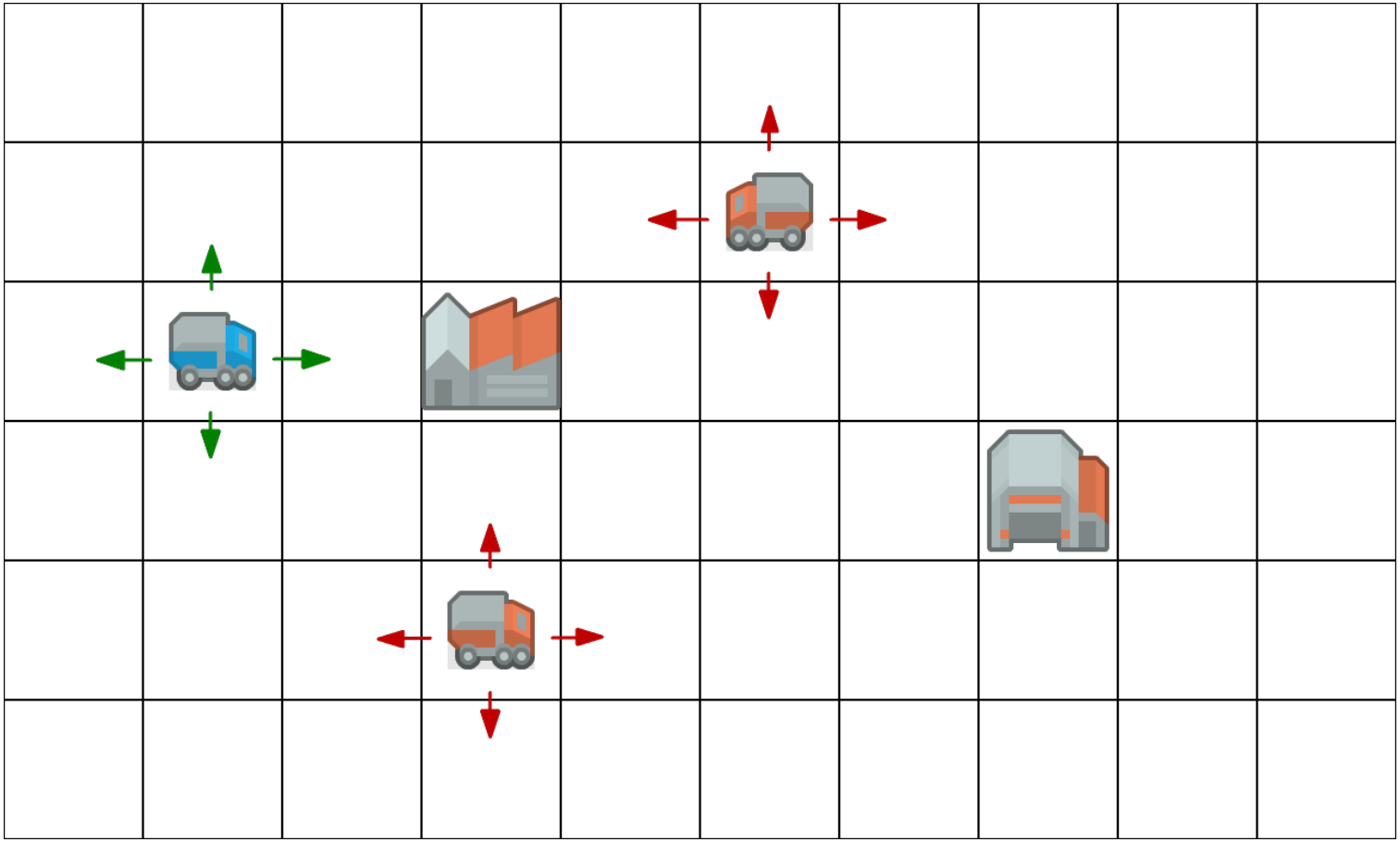
Aim of this thesis

Reach the other side of the area, with probability of bumping into anything lower than some percentage,



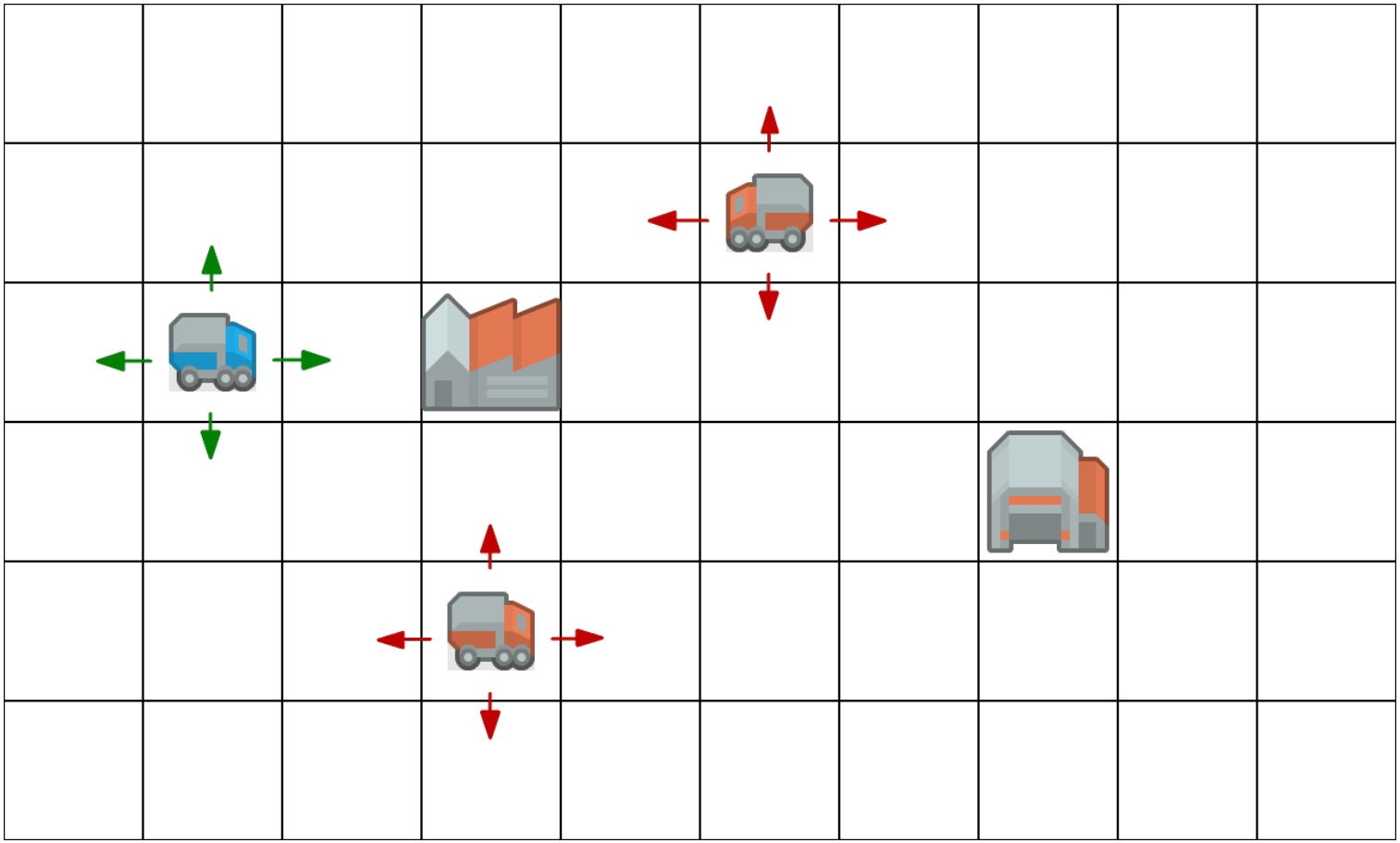
Aim of this thesis

Reach the other side of the area, with probability of bumping into anything lower than some percentage, and within limits of fuel / battery power.



Aim of this thesis

Can we find a schedule of actions that our robot should take at each position?



Definitions: (u)MDPs

Probability distribution: $\mu \colon A \to \mathbb{R}_+$ is a probability distribution over a finite set A if $\sum_{a \in A} \mu(a) = 1$.

Definitions: (u)MDPs

Probability distribution: $\mu \colon A \to \mathbb{R}_+$ is a probability distribution over a finite set A if $\sum_{a \in A} \mu(a) = 1$.

MDP: $\mathcal{M} = (S, Act, s_I, P)$,

S set of states, Act set of actions, s_I initial state,
transition function $P \colon S \times Act \times S \to [0, 1]$ forms a valid probability distribution over the successor states s'.

Definitions: (u)MDPs

Probability distribution: $\mu \colon A \to \mathbb{R}_+$ is a probability distribution over a finite set A if $\sum_{a \in A} \mu(a) = 1$.

MDP: $\mathcal{M} = (S, Act, s_I, P)$,

S set of states, Act set of actions, s_I initial state, transition function $P \colon S \times Act \times S \to [0, 1]$ forms a valid probability distribution over the successor states s'.

uMDP: $\mathcal{M} = (S, Act, s_I, \mathbb{I}, \mathcal{P})$

$\mathcal{P} \colon S \times Act \times S \to I$ uncertain transition function for $I \in \mathbb{I}$.

$P \in \mathcal{P}$ if P takes values in the intervals, and is a valid probability distribution.

Definitions: (u)MDPs

Probability distribution: $\mu\colon A \to \mathbb{R}_+$ is a probability distribution over a finite set A if $\sum_{a \in A} \mu(a) = 1$.

MDP: $\mathcal{M} = (S, Act, s_I, P)$,

S set of states, Act set of actions, s_I initial state,
transition function $P\colon S \times Act \times S \to [0, 1]$ forms a valid probability distribution over the successor states s'.

uMDP: $\mathcal{M} = (S, Act, s_I, \mathbb{I}, \mathcal{P})$

$\mathcal{P}\colon S \times Act \times S \to I$ uncertain transition function for $I \in \mathbb{I}$.
$P \in \mathcal{P}$ if P takes values in the intervals, and is a valid probability distribution.

An uMDP $\mathcal{M}$ can be instantiated by a transition function $P \in \mathcal{P}$.
This gives us the MDP $\mathcal{M}[P]$, which is an instantiation of that uMDP.

Definitions: scheduler

$Distr(X)$ is the set of probability distributions over finite set X.

Definitions: scheduler

$Distr(X)$ is the set of probability distributions over finite set X.

Scheduler: $\sigma : S \times Act \to Distr(Act)$.

Definitions: scheduler

$Distr(X)$ is the set of probability distributions over finite set X.

Scheduler: $\sigma : S \times Act \to Distr(Act)$.

Example of a valid scheduler σ:

$\sigma(s_1)(a) =$	0.3	$\sigma(s_2)(a) =$	1
$\sigma(s_1)(b) =$	0.7	$\sigma(s_2)(b) =$	0
	1		1

Definitions: scheduler

$Distr(X)$ is the set of probability distributions over finite set X.

Scheduler: $\sigma : S \times Act \to Distr(Act)$.

Example of a valid scheduler σ:

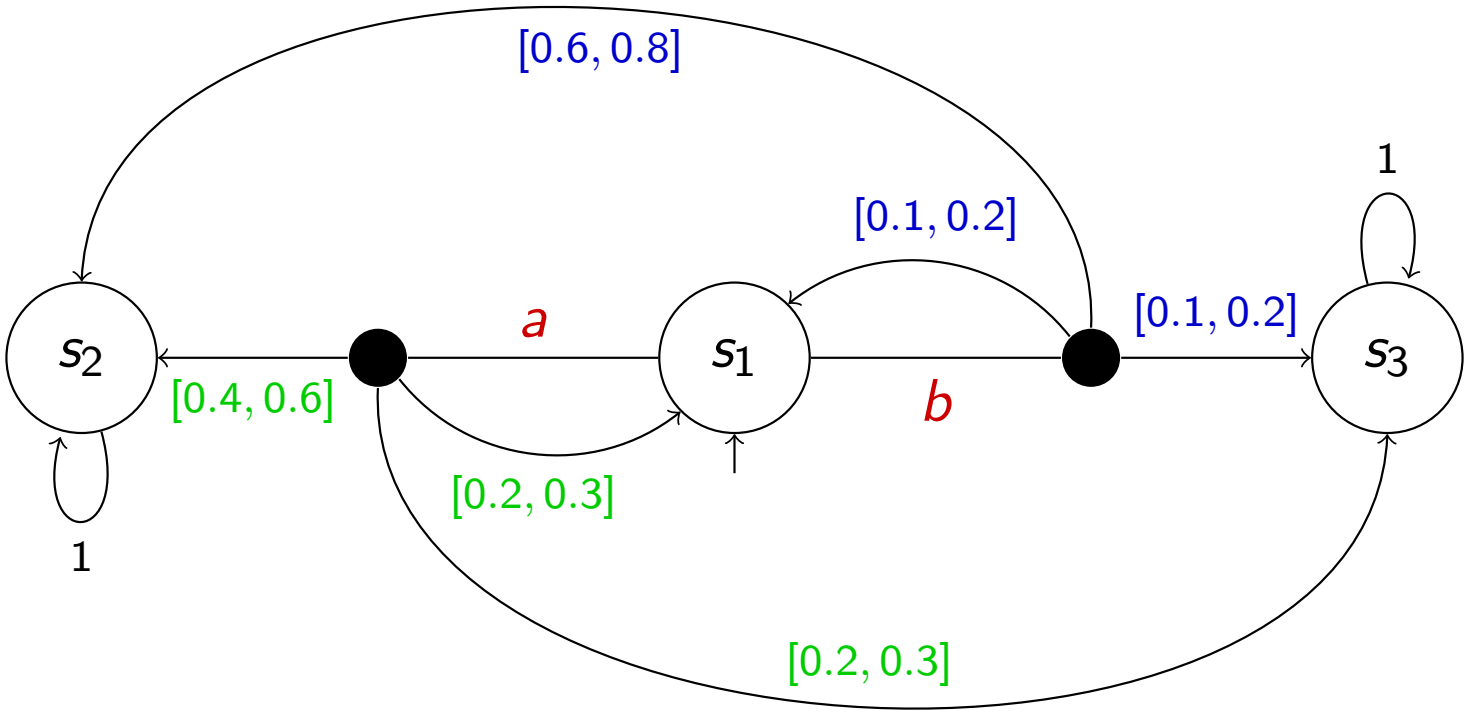
$\sigma(s_1)(a) =$	0.3	$\sigma(s_2)(a) =$	1
$\sigma(s_1)(b) =$	0.7	$\sigma(s_2)(b) =$	0
	1		1

Example of a function σ that is not a scheduler:

$\sigma(s_1)(a) =$	0.3	$\sigma(s_2)(a) =$	0.4
$\sigma(s_1)(b) =$	0.7	$\sigma(s_2)(b) =$	0.2
	1		0.6

Definitions: schedulers and (u)MDPs

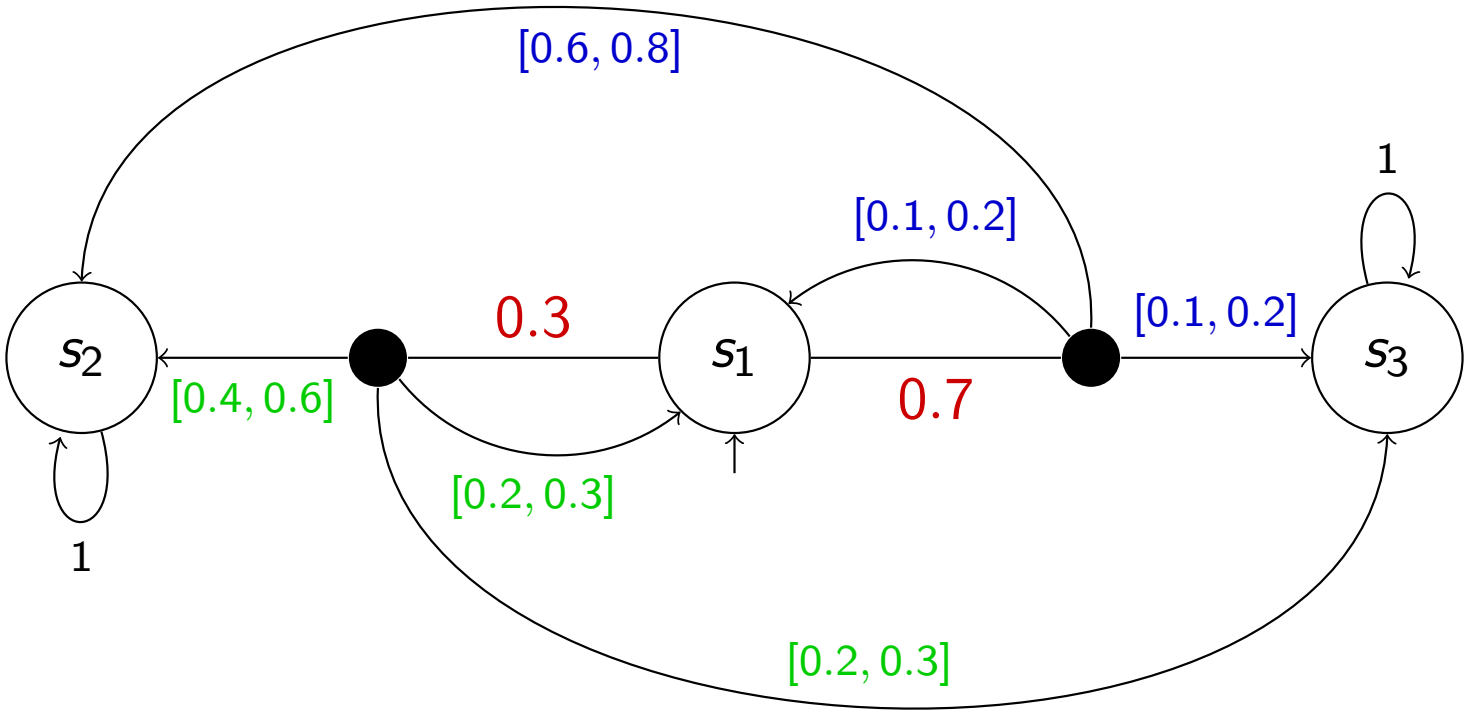
A valid scheduler σ can be applied to a uMDP $\mathcal{M}$.



Self loops of probability 1 for states s_2 and s_3 are over both actions.

Definitions: schedulers and (u)MDPs

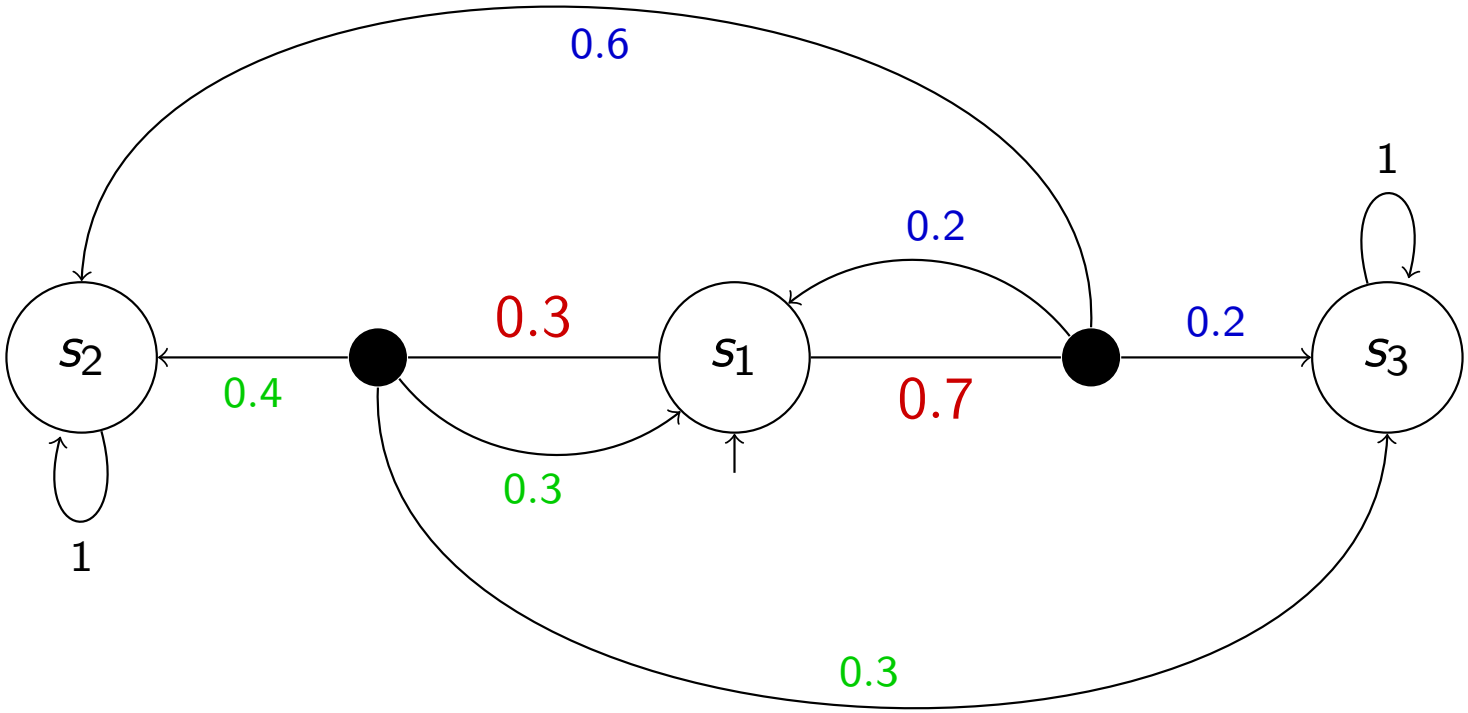
It gives us probabilities for choosing actions at each state. Notation: $\mathcal{M}^\sigma$.



Self loops of probability 1 for states s_2 and s_3 are over both actions.

Definitions: schedulers and (u)MDPs

If we also instantiate $\mathcal{M}$ with P, we get $\mathcal{M}^\sigma[P]$. This is called an induced Markov chain.



Self loops of probability 1 for states s_2 and s_3 are over both actions.

Definitions: reachability and expected cost properties

Cost: $c : S \times Act \to \mathbb{R}$ assigns a cost to each action at a state.

Definitions: reachability and expected cost properties

Cost: $c : S \times Act \to \mathbb{R}$ assigns a cost to each action at a state.

Reachability property: $\Pr(\mathcal{M}^\sigma[P], \Diamond T) \leq \lambda$, for a target set $T \subseteq S$.
Probability of reaching T from s_I in $\mathcal{M}^\sigma[P]$ is less than λ.

Definitions: reachability and expected cost properties

Cost: $c : S \times Act \to \mathbb{R}$ assigns a cost to each action at a state.

Reachability property: $\Pr(\mathcal{M}^\sigma[P], \Diamond T) \leq \lambda$, for a target set $T \subseteq S$.
Probability of reaching T from s_I in $\mathcal{M}^\sigma[P]$ is less than λ.

Expected cost property: $\mathrm{EC}(\mathcal{M}^\sigma[P], \Diamond G) \leq \kappa$, for a goal set $G \subseteq S$.
Expected cost of reaching G from s_I in $\mathcal{M}^\sigma[P]$ is less than κ.

Problem statement

Find a scheduler for which a given reachability property and a given expected cost property holds, for all instantiations of the uMDP.

Problem statement

This is a very relevant problem.

Problem statement

This is a very relevant problem.

No established solution available for uMDPs.

Nonlinear programming

A nonlinear program (NLP) is an optimization problem of the form

$$\begin{aligned} &\text{minimize } f_0(x), \\ &\text{subject to } f_i(x) \leq 0, \quad i = 1, \ldots, m, \\ &\qquad\qquad g_j(x) = 0, \quad j = 1, \ldots, p. \end{aligned}$$

Nonlinear programming

A nonlinear program (NLP) is an optimization problem of the form

$$\begin{aligned} &\text{minimize } f_0(x), \\ &\text{subject to } f_i(x) \leq 0, \quad i = 1, \ldots, m, \\ &\qquad\qquad g_j(x) = 0, \quad j = 1, \ldots, p. \end{aligned}$$

We can use a NLP to solve our problem for a MDP, but in a uMDP we have to account for uncertainty.

Robust nonlinear programming

A robust NLP is a NLP that accounts for uncertainty in the problem data.

Robust nonlinear programming

A robust NLP is a NLP that accounts for uncertainty in the problem data.

Example: $f(x) = a \cdot \sin x$, for all $a \in \mathcal{U}$ is a robust constraint.

Properties of the robust NLP encoding

The robust NLP encoding of the problem is correct and complete by construction.

Properties of the robust NLP encoding

The robust NLP encoding of the problem is correct and complete by construction.

Correctness: a solution to the robust NLP is also a solution to our problem.

Properties of the robust NLP encoding

The robust NLP encoding of the problem is correct and complete by construction.

Correctness: a solution to the robust NLP is also a solution to our problem.

Completeness: for every instance of our problem that has a solution, the robust NLP will find a solution.

Problem of the robust NLP encoding

However, most nonlinear programs are NP-hard. The complexity of robust NLP is unknown.

Problem of the robust NLP encoding

However, most nonlinear programs are NP-hard. The complexity of robust NLP is unknown.

Idea: transform the robust NLP into an optimization problem that can be solved efficiently.

Problem of the robust NLP encoding

However, most nonlinear programs are NP-hard. The complexity of robust NLP is unknown.

Idea: transform the robust NLP into an optimization problem that can be solved efficiently.

Use robust geometric programming.

Definitions: monomials and posynomials

Let $\mathbb{R}_{++} = \{r \in \mathbb{R} \mid r > 0\}$, and $\mathbf{x} = (x_1, ..., x_n) \in \mathbb{R}^n_{++}$.

Definitions: monomials and posynomials

Let $\mathbb{R}_{++} = \{r \in \mathbb{R} \mid r > 0\}$, and $\mathbf{x} = (x_1, ..., x_n) \in \mathbb{R}^n_{++}$.

A monomial is a function $f : \mathbb{R}^n_{++} \to \mathbb{R}$ with

$$f(\mathbf{x}) = d \prod_{j=1}^{n} x_j^{a_j}$$

where $d \geq 0$ and $a_j \in \mathbb{R}$.

Definitions: monomials and posynomials

Let $\mathbb{R}_{++} = \{r \in \mathbb{R} \mid r > 0\}$, and $\mathbf{x} = (x_1, ..., x_n) \in \mathbb{R}^n_{++}$.

A monomial is a function $f : \mathbb{R}^n_{++} \to \mathbb{R}$ with

$$f(\mathbf{x}) = d \prod_{j=1}^{n} x_j^{a_j}$$

where $d \geq 0$ and $a_j \in \mathbb{R}$.

Example: $12 \cdot x_1 \cdot x_2^7$.

Definitions: monomials and posynomials

A posynomial is a sum of a number of monomials:

$$f(\mathbf{x}) = \sum_{k=1}^{K} d_k \prod_{j=1}^{n} x_j^{a_{jk}}.$$

Definitions: monomials and posynomials

A posynomial is a sum of a number of monomials:

$$f(\mathbf{x}) = \sum_{k=1}^{K} d_k \prod_{j=1}^{n} x_j^{a_{jk}}.$$

Example: $12 \cdot x_1 \cdot x_2^7 + x_2$.

Geometric programming (GP)

A geometric program is an optimization problem of the form

$$\begin{aligned} \text{minimize} \quad & f_0(x), \\ \text{subject to} \quad & f_i(x) \leq 1, && i = 1, \ldots, m, \\ & g_j(x) = 1, && j = 1, \ldots, p, \end{aligned}$$

where the objective function $f_0(x)$ and the inequality constraints $f_i(x)$ are posynomials, and the equality constraints $g_j(x)$ are monomials.

Geometric programming (GP)

A geometric program is an optimization problem of the form

$$\begin{aligned} \text{minimize} \quad & f_0(x), \\ \text{subject to} \quad & f_i(x) \leq 1, && i = 1, \ldots, m, \\ & g_j(x) = 1, && j = 1, \ldots, p, \end{aligned}$$

where the objective function $f_0(x)$ and the inequality constraints $f_i(x)$ are posynomials, and the equality constraints $g_j(x)$ are monomials.

Just as with NLP, if one or more of the constraints accounts for uncertainty in the problem data, this is called a robust geometric program.

Robust GP encoding of the problem

We can transform the robust NLP encoding into a robust GP.

Robust GP encoding of the problem

We can transform the robust NLP encoding into a robust GP.

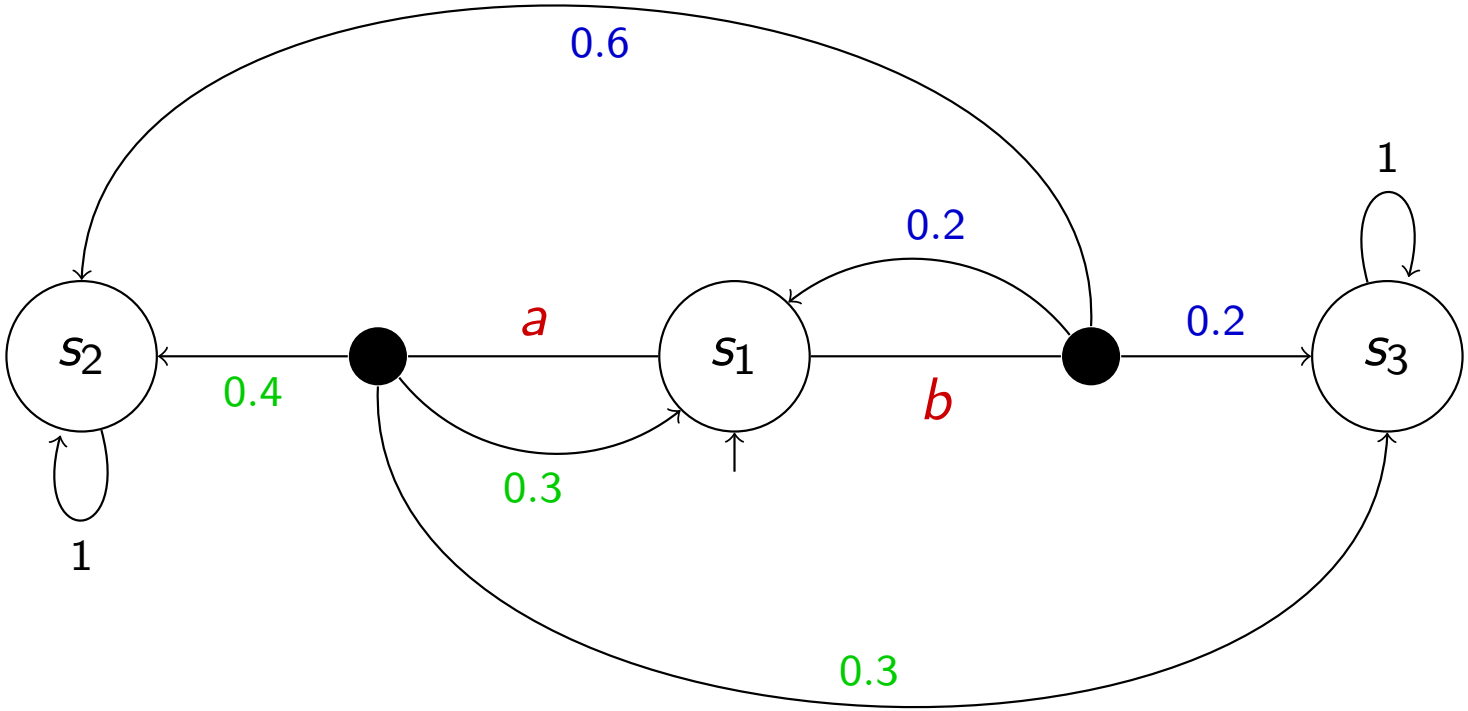
The robust NLP and the robust GP are not equivalent. We can no longer assume the robust GP to be a correct and complete by construction.

Correctness

This robust GP encoding is correct.

Incompleteness

The robust GP encoding is unfortunately not complete. Counter example:



Self loops of probability 1 for states s_2 and s_3 are over both actions.

Counter example results

By construction of this counter example, we know that for all $\delta \in [0, 1]$ a valid scheduler exists. The robust GP, however, does not give us a valid scheduler.

Counter example results

By construction of this counter example, we know that for all $\delta \in [0, 1]$ a valid scheduler exists. The robust GP, however, does not give us a valid scheduler.

Table 1: Solutions of the counter example GP

δ	λ	$\sigma^{1,a}$	$\sigma^{1,b}$	$\sigma^{1,a} + \sigma^{1,b}$	Error	p_1
1	0.429	0.5	0.5	1	0%	0.3403
0.5	0.3395	0.5	0.5	1	0%	0.3337
0.4	0.3216	0.4465	0.5469	0.9934	0.66%	0.3216
0.3	0.3037	0.4275	0.5235	0.9510	4.9%	0.3037
0.2	0.2858	0.4079	0.4995	0.9074	9.3%	0.2858

Solving the robust GP

How do we solve a robust GP?

Solving the robust GP

Robust geometric programming is PSPACE-hard.
(Chassein and Goerigk, 2014).

Solving the robust GP

Luckily, an approximation method exists:

Tractable approximate robust geometric programming.
(Hsiun, Kim, and Boyd, 2008)

This article describes how to approximate a robust geometric program by a robust linear program.

Applying the approximation method; a summary

Our robust GP is in posynomial form. We transform it into convex form.

Applying the approximation method; a summary

Our robust GP is in posynomial form. We transform it into convex form.

This replaces the monomials and posynomials by the log-sum-exp-function (lse-function).

Applying the approximation method; a summary

Our robust GP is in posynomial form. We transform it into convex form.

This replaces the monomials and posynomials by the log-sum-exp-function (lse-function).

$$\mathsf{lse}(z_1, ..., z_k) = \log(e^{z_1} + \cdots + e^{z_k}), \quad z_1, ..., z_k \in \mathbb{R}$$

Applying the approximation method; a summary

In convex form, we can simplify the robust GP by some linear algebra. This results in a robust GP with constraints consisting of k-term lse-functions, where each term is a single vector product: $\mathbf{a}^T\mathbf{y}$.

Applying the approximation method; a summary

In convex form, we can simplify the robust GP by some linear algebra. This results in a robust GP with constraints consisting of k-term lse-functions, where each term is a single vector product: $\mathbf{a}^T\mathbf{y}$.

Each k-term lse-constraint can be split in $k-1$ 2-term constraints.

Applying the approximation method; a summary

In convex form, we can simplify the robust GP by some linear algebra. This results in a robust GP with constraints consisting of k-term lse-functions, where each term is a single vector product: $\mathbf{a}^T\mathbf{y}$.

Each k-term lse-constraint can be split in $k-1$ 2-term constraints.

Each 2-term lse-function can be approximated by a piecewise-linear function.

Applying the approximation method; a summary

In convex form, we can simplify the robust GP by some linear algebra. This results in a robust GP with constraints consisting of k-term lse-functions, where each term is a single vector product: $\mathbf{a}^T\mathbf{y}$.

Each k-term lse-constraint can be split in $k-1$ 2-term constraints.

Each 2-term lse-function can be approximated by a piecewise-linear function.

A Python implementation of the approximation algorithm is included with the thesis.

Applying the approximation method; a summary

Replacing each 2-term lse-function by its piecewise-linear approximation gives us a robust linear program.

Applying the approximation method; a summary

Replacing each 2-term lse-function by its piecewise-linear approximation gives us a robust linear program.

Robust LPs can be solved efficiently for a number of different kinds of uncertainty sets, such as polyhedral uncertainty, ellipsoidal uncertainty, or interval uncertainty.

Applying the approximation method; a summary

Replacing each 2-term lse-function by its piecewise-linear approximation gives us a robust linear program.

Robust LPs can be solved efficiently for a number of different kinds of uncertainty sets, such as polyhedral uncertainty, ellipsoidal uncertainty, or interval uncertainty.

So what kind of uncertainty is $\mathcal{P}$?

Open problem: the uncertainty set $\mathcal{P}$

The exact uncertainty set of $\mathcal{P}$ is unknown.

Open problem: the uncertainty set $\mathcal{P}$

The exact uncertainty set of $\mathcal{P}$ is unknown.

The type of uncertainty used in the uMDP is not the type of uncertainty in the robust NLP/GP/LP encodings.

Conclusions

Key results:

Conclusions

Key results:

- How robust GP can be used to solve this problem

Conclusions

Key results:

- How robust GP can be used to solve this problem
- Proven correctness and incompleteness

Conclusions

Key results:

- How robust GP can be used to solve this problem
- Proven correctness and incompleteness
- Convex reformulation of the robust GP

Conclusions

Key results:

- How robust GP can be used to solve this problem
- Proven correctness and incompleteness
- Convex reformulation of the robust GP
- Further reformulations and approximation by a robust LP
  proven that all the steps taken are correct

Conclusions

Key results:

- How robust GP can be used to solve this problem
- Proven correctness and incompleteness
- Convex reformulation of the robust GP
- Further reformulations and approximation by a robust LP
  proven that all the steps taken are correct
- Python implementation of approximation algorithm

Conclusions

Key results:

- How robust GP can be used to solve this problem
- Proven correctness and incompleteness
- Convex reformulation of the robust GP
- Further reformulations and approximation by a robust LP
  proven that all the steps taken are correct
- Python implementation of approximation algorithm
- Clear path how to use robust GPs for this and other problems

Conclusions

Key results:

- How robust GP can be used to solve this problem
- Proven correctness and incompleteness
- Convex reformulation of the robust GP
- Further reformulations and approximation by a robust LP
  proven that all the steps taken are correct
- Python implementation of approximation algorithm
- Clear path how to use robust GPs for this and other problems

Future work:

Conclusions

Key results:

- How robust GP can be used to solve this problem
- Proven correctness and incompleteness
- Convex reformulation of the robust GP
- Further reformulations and approximation by a robust LP
  proven that all the steps taken are correct
- Python implementation of approximation algorithm
- Clear path how to use robust GPs for this and other problems

Future work:

- Find a known uncertainty set that works for $\mathcal{P}$

Conclusions

Key results:

- How robust GP can be used to solve this problem
- Proven correctness and incompleteness
- Convex reformulation of the robust GP
- Further reformulations and approximation by a robust LP
  proven that all the steps taken are correct
- Python implementation of approximation algorithm
- Clear path how to use robust GPs for this and other problems

Future work:

- Find a known uncertainty set that works for $\mathcal{P}$
- Find ways to "increase" completeness

Robust NLP encoding of the problem

Optimization variables:

- $\{\sigma^{s,\alpha} \mid s \in S, \alpha \in Act(s)\}$, which define the schedulers σ given by
- $\{p_s \mid s \in S\}$, where p_s is the probability of reaching the target set $T \subseteq S$ from state s
- $\{c_s \mid s \in S\}$, where c_s is the expected cost to reach $G \subseteq S$ from state s under scheduler

Robust NLP encoding of the problem

$$
\begin{aligned}
\text{minimize} \quad & c_{s_I}, \\
\text{subject to} \quad & p_{s_I} \leq \lambda, \\
& c_{s_I} \leq \kappa, \\
\forall s \in S. \quad & \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} = 1,
\end{aligned}
$$

Robust NLP encoding of the problem (cont'd)

$$\forall s \in T. \qquad p_s = 1,$$

$$\begin{array}{r} \forall s \in S \setminus T. \\ \forall P \in \mathcal{P}. \end{array} \qquad p_s = \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s, \alpha, s') \cdot p_{s'},$$

$$\forall s \in G. \qquad c_s = 0,$$

$$\begin{array}{r} \forall s \in S \setminus G. \\ \forall P \in \mathcal{P}. \end{array} \qquad c_s = \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \Big(c(s, \alpha) + \sum_{s' \in S} P(s, \alpha, s') \cdot c_{s'} \Big).$$

The robust GP encoding

New optimization variable: replace c_s with $c'_s = c_s + 1$.

$$\text{minimize} \quad c'_{s_I} + \sum_{s \in S,\ \alpha \in Act} \frac{1}{\sigma^{s,\alpha}},$$

$$\text{subject to} \quad \frac{p_{s_I}}{\lambda} \leq 1,$$

$$\frac{c'_{s_I}}{\kappa + 1} \leq 1,$$

$$\forall s \in S. \quad \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \leq 1,$$

$$\forall s \in T. \quad p_s = 1,$$

The robust GP encoding (cont'd)

$$\begin{array}{rl} \forall s \in S \setminus T. & \dfrac{\sum\limits_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum\limits_{s' \in S} P(s,\alpha,s') \cdot p_{s'}}{p_s} \leq 1, \\ \forall P \in \mathcal{P}. & \end{array}$$

$$\forall s \in G. \quad c'_s = 1,$$

$$\begin{array}{rl} \forall s \in S \setminus G. & \dfrac{\sum\limits_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \left(c(s,\alpha) + \sum\limits_{s' \in S} P(s,\alpha,s') \cdot c'_{s'} \right)}{c'_s} \leq 1. \\ \forall P \in \mathcal{P}. & \end{array}$$