Directed Reachability for Infinite-State Systems

Michael Blondin[1], Christoph Haase[2], Philip Offtermatt[1,3]

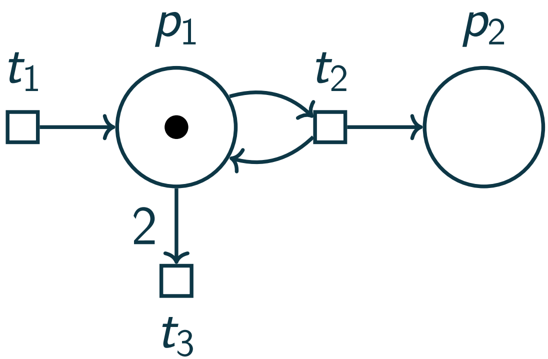
[1] Université de Sherbrooke
[2] University of Oxford
[3] Max Planck Institute for Software Systems

Directed Reachability for Infinite-State Systems

Michael Blondin[1], Christoph Haase[2], Philip Offtermatt[1,3]

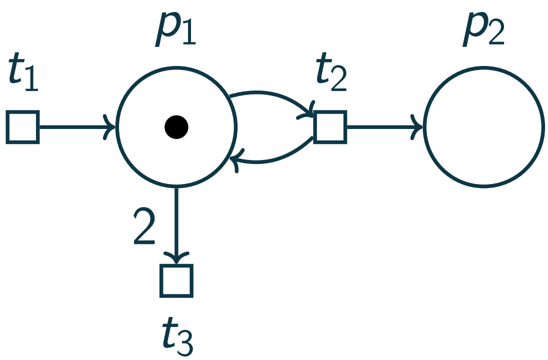
[1] Université de Sherbrooke
[2] University of Oxford
[3] Max Planck Institute for Software Systems

Petri Nets



Petri Nets

Places: $P = \{p_1, p_2\}$



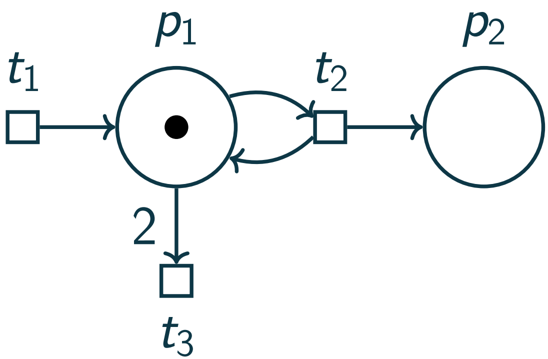
Petri Nets

Places: $P = \{p_1, p_2\}$

Marking: $P \to \mathbb{N}$

$(1, 0)$

p_1, p_2



Petri Nets

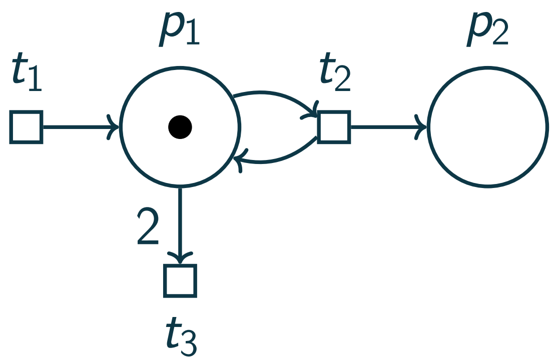
Places: $P = \{p_1, p_2\}$ Pre Post

Transitions: $T = \{t_1, t_2, t_3\} \subseteq \mathbb{N}^P \times \mathbb{N}^P$ e.g. $t_2 = ((1,0),(1,1))$

Marking: $P \to \mathbb{N}$

$(1, 0)$

p_1, p_2



Petri Nets

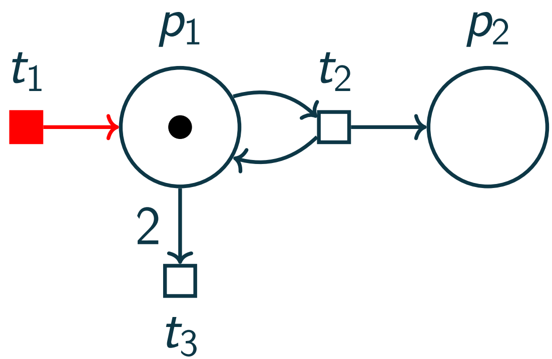
Places: $P = \{p_1, p_2\}$ Pre Post

Transitions: $T = \{t_1, t_2, t_3\} \subseteq \mathbb{N}^P \times \mathbb{N}^P$ e.g. $t_2 = ((1,0),(1,1))$

Marking: $P \to \mathbb{N}$

$(1, 0)$

p_1, p_2



Petri Nets

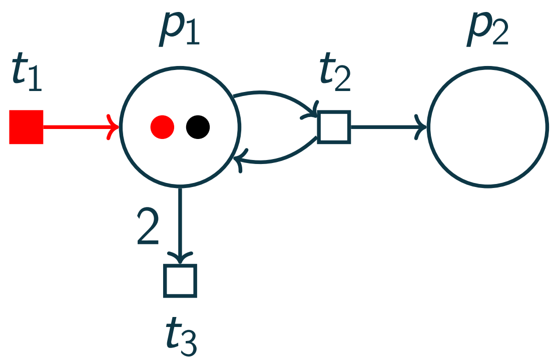
Places: $P = \{p_1, p_2\}$ Pre Post

Transitions: $T = \{t_1, t_2, t_3\} \subseteq \mathbb{N}^P \times \mathbb{N}^P$ e.g. $t_2 = ((1,0),(1,1))$

Marking: $P \to \mathbb{N}$

$(2, 0)$

p_1, p_2



Petri Nets

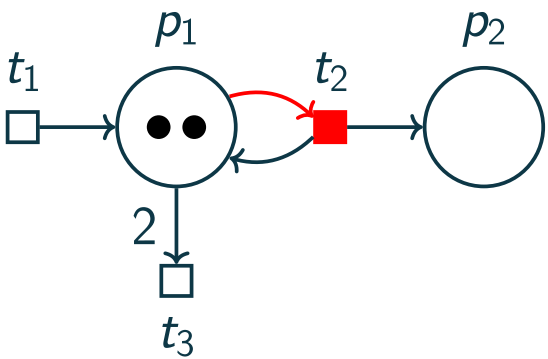
Places: $P = \{p_1, p_2\}$ Pre Post

Transitions: $T = \{t_1, t_2, t_3\} \subseteq \mathbb{N}^P \times \mathbb{N}^P$ e.g. $t_2 = ((1,0),(1,1))$

Marking: $P \to \mathbb{N}$

$(2, 0)$

p_1, p_2



Petri Nets

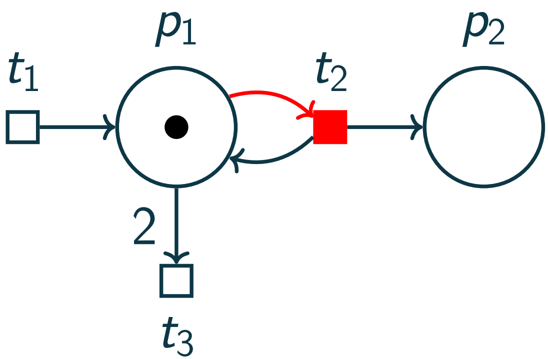
Places: $P = \{p_1, p_2\}$ Pre Post

Transitions: $T = \{t_1, t_2, t_3\} \subseteq \mathbb{N}^P \times \mathbb{N}^P$ e.g. $t_2 = ((1,0),(1,1))$

Marking: $P \to \mathbb{N}$

$(1, 0)$

p_1, p_2



Petri Nets

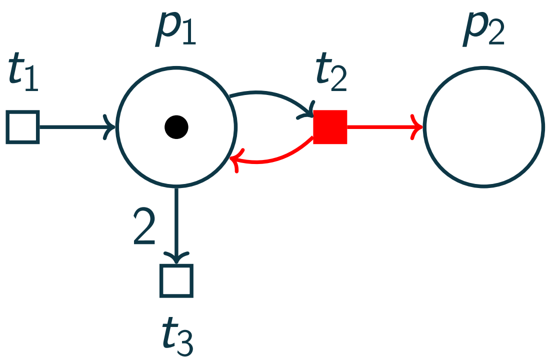
Places: $P = \{p_1, p_2\}$ Pre Post

Transitions: $T = \{t_1, t_2, t_3\} \subseteq \mathbb{N}^P \times \mathbb{N}^P$ e.g. $t_2 = ((1,0),(1,1))$

Marking: $P \to \mathbb{N}$

$(1, 0)$

p_1, p_2



Petri Nets

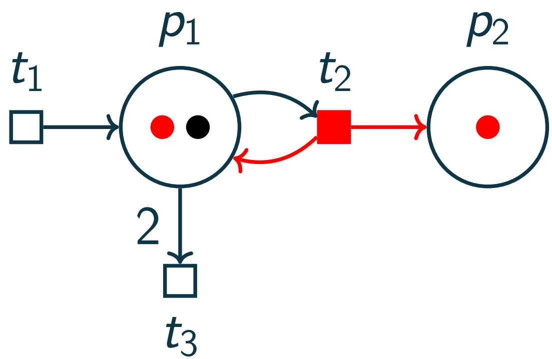
Places: $P = \{p_1, p_2\}$ Pre Post

Transitions: $T = \{t_1, t_2, t_3\} \subseteq \mathbb{N}^P \times \mathbb{N}^P$ e.g. $t_2 = ((1,0),(1,1))$

Marking: $P \to \mathbb{N}$

$(2,1)$

p_1, p_2



Petri Nets

Places: $P = \{p_1, p_2\}$

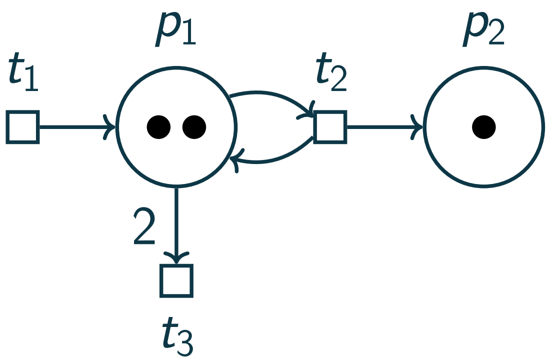
Pre Post

Transitions: $T = \{t_1, t_2, t_3\} \subseteq \mathbb{N}^P \times \mathbb{N}^P$ e.g. $t_2 = ((1,0),(1,1))$

Marking: $P \to \mathbb{N}$

$(2,1)$

p_1, p_2



Petri Nets

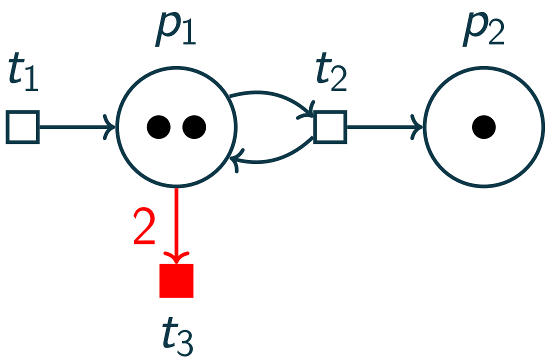
Places: $P = \{p_1, p_2\}$ Pre Post

Transitions: $T = \{t_1, t_2, t_3\} \subseteq \mathbb{N}^P \times \mathbb{N}^P$ e.g. $t_2 = ((1,0),(1,1))$

Marking: $P \to \mathbb{N}$

$(2, 1)$

p_1, p_2



Petri Nets

Places: $P = \{p_1, p_2\}$

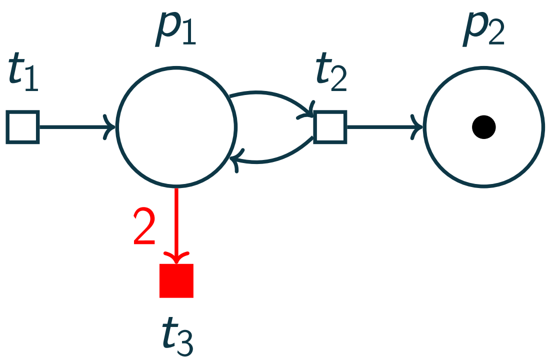
Pre Post

Transitions: $T = \{t_1, t_2, t_3\} \subseteq \mathbb{N}^P \times \mathbb{N}^P$ e.g. $t_2 = ((1, 0), (1, 1))$

Marking: $P \to \mathbb{N}$

$(0, 1)$

p_1, p_2



Petri Nets

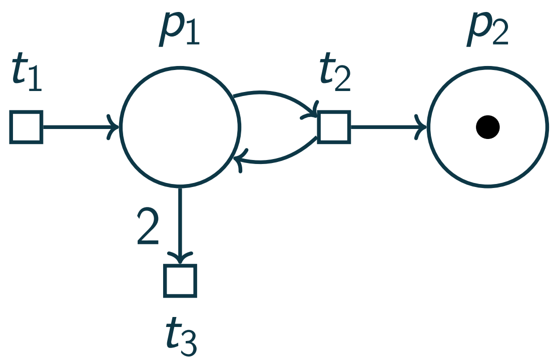
Places: $P = \{p_1, p_2\}$ Pre Post

Transitions: $T = \{t_1, t_2, t_3\} \subseteq \mathbb{N}^P \times \mathbb{N}^P$ e.g. $t_2 = ((1, 0), (1, 1))$

Marking: $P \to \mathbb{N}$

$(0, 1)$

p_1, p_2



Petri Nets

Places: $P = \{p_1, p_2\}$

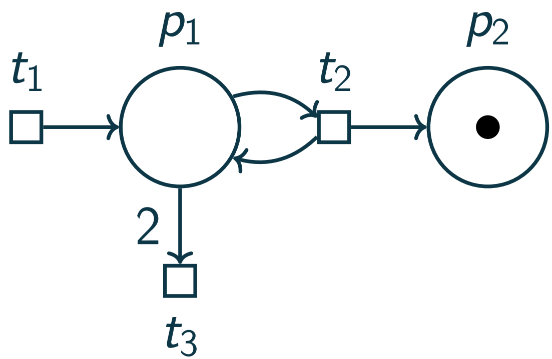
Pre Post

Transitions: $T = \{t_1, t_2, t_3\} \subseteq \mathbb{N}^P \times \mathbb{N}^P$ e.g. $t_2 = ((1,0),(1,1))$

Marking: $P \to \mathbb{N}$

$(0, 1)$

p_1, p_2



Petri nets finitely represent infinite-state systems:
Reachability Graphs

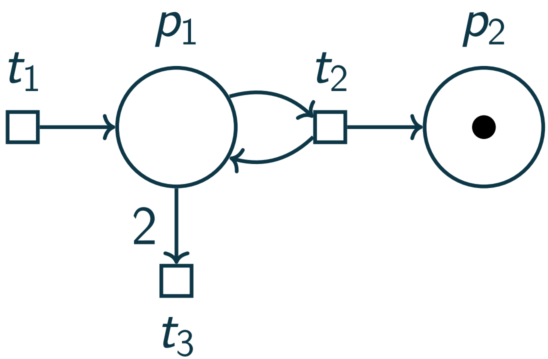
Petri Nets

Places: $P = \{p_1, p_2\}$

Transitions: $T = \{t_1, t_2, t_3\} \subseteq \underbrace{\mathbb{N}^P}^{\text{Pre}} \times \underbrace{\mathbb{N}^P}^{\text{Post}}$ e.g. $t_2 = ((1,0),(1,1))$

Marking: $P \to \mathbb{N}$

$\underset{p_1,\ p_2}{(0,1)}$



Petri nets finitely represent infinite-state systems:
Reachability Graphs

$$
\begin{tikzcd}
(0,0) \arrow[d, "t_1"] & (0,1) \arrow[d, "t_1"'] \\
(1,0) \arrow[r, "t_2"] \arrow[d, "t_1"] & (1,1) \cdots \arrow[d, "t_1"'] \\
(2,0) \arrow[r, "t_2"] \arrow[uu, bend left, "t_3"] & (2,1) \cdots \arrow[uu, bend right, "t_3"'] \\
\vdots & \vdots
\end{tikzcd}
$$

Petri Nets

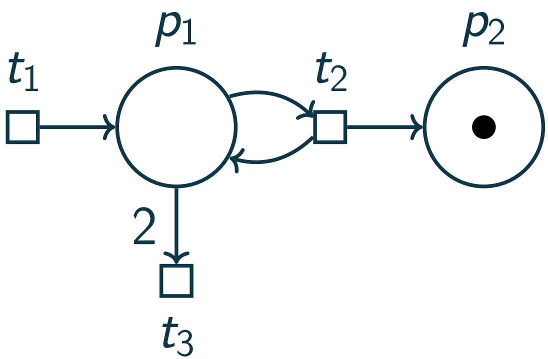
Places: $P = \{p_1, p_2\}$

Transitions: $T = \{t_1, t_2, t_3\} \subseteq \underset{\text{Pre}}{\mathbb{N}^P} \times \underset{\text{Post}}{\mathbb{N}^P}$ e.g. $t_2 = ((1,0),(1,1))$

Marking: $P \to \mathbb{N}$

$(0,1)$

p_1, p_2



Petri nets finitely represent infinite-state systems:
Reachability Graphs

$$\begin{tikzcd}
(0,0) \arrow[d, "t_1"] & (0,1) \arrow[d, "t_1"'] \\
(1,0) \arrow[r, "t_2"] \arrow[d, "t_1"] & (1,1) \cdots \arrow[d, "t_1"'] \\
(2,0) \arrow[r, "t_2"] \arrow[uu, bend left, "t_3"] & (2,1) \cdots \arrow[uu, bend right, "t_3"'] \\
\vdots & \vdots
\end{tikzcd}$$

$(0,1)$ is reachable from $(1,0)$

Petri Nets

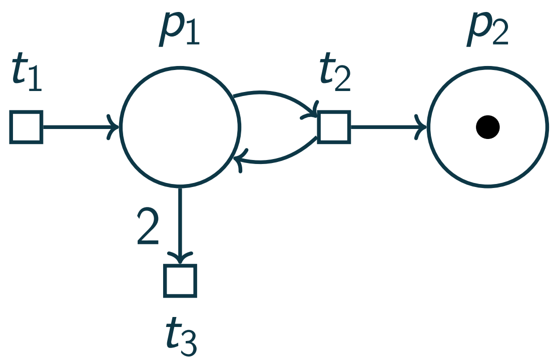
Places: $P = \{p_1, p_2\}$ Pre Post

Transitions: $T = \{t_1, t_2, t_3\} \subseteq \mathbb{N}^P \times \mathbb{N}^P$ e.g. $t_2 = ((1,0),(1,1))$

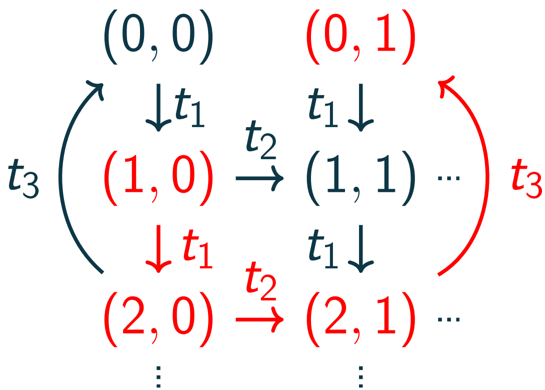
Marking: $P \to \mathbb{N}$

$(0,1)$

p_1, p_2



Petri nets finitely represent infinite-state systems: Reachability Graphs



$(0,1)$ is reachable from $(1,0)$

Witnessing run:

$(1,0) \xrightarrow{t_1} (2,0) \xrightarrow{t_2} (2,1) \xrightarrow{t_3} (0,1)$

Problems for Petri Nets

Reachability: Is there a run that starts with m_{init} and ends with m_{target}? (e.g., $m_{init} = (1, 0), m_{target} = (0, 1)$)

Problems for Petri Nets

Reachability: Is there a run that starts with m_{init}
and ends with m_{target}? (e.g., $m_{init} = (1, 0)$, $m_{target} = (0, 1)$)

Decidable [Mayr, 1980], complexity open for 40+ years

Problems for Petri Nets

Reachability: Is there a run that starts with m_{init}
and ends with m_{target}? (e.g., $m_{init} = (1, 0), m_{target} = (0, 1)$)

Decidable [Mayr, 1980], complexity open for 40+ years

Non-elementary lower bound [Czerwiński et al., 2019]

Ackermannian upper bound [Leroux and Schmitz, 2019]

Problems for Petri Nets

Coverability: Is there a run that starts with m_{init} and ends with a marking where each place has at least as many tokens as in m_{target}?

Problems for Petri Nets

Coverability: Is there a run that starts with m_{init} and ends with a marking where each place has at least as many tokens as in m_{target}?

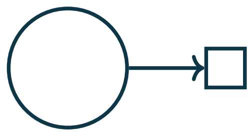
EXPSPACE-complete [Lower bound by Lipton, 1976]
[Upper bound by Rackoff, 1978]

Problems for Petri Nets

Coverability: Is there a run that starts with m_{init} and ends with a marking where each place has at least as many tokens as in m_{target}?

EXPSPACE-complete [Lower bound by Lipton, 1976]
[Upper bound by Rackoff, 1978]

Reduction to reachability: Add transitions that delete tokens



Challenge

What problem do we tackle?

- Coverability: Many competitive solvers

Challenge

What problem do we tackle?

- Coverability: Many competitive solvers
- Reachability: Many interesting applications, but:

Almost no tool support
in the presence of infinite state spaces!

Challenge

What problem do we tackle?

- Coverability: Many competitive solvers
- Reachability: Many interesting applications, but:

Almost no tool support
in the presence of infinite state spaces!

- To show unreachability (safety), approximations can be used, but not clear how they help for reachability

Challenge

What problem do we tackle?

- Coverability: Many competitive solvers
- Reachability: Many interesting applications, but:

Almost no tool support
in the presence of infinite state spaces!

- To show unreachability (safety), approximations can be used, but not clear how they help for reachability

No practically efficient semi-procedures for showing reachability in infinite-state systems

Outline

- Part I: **Applications** - Why is this useful?
- Part II: **Approximations** - Relaxing Reachability
- Part III: **Directed Search** - Searching with Guidance
- Part IV: **Experiments** - Prototype Evaluation

Part I

Applications

Some Applications of Petri Nets

Concurrent Program Analysis

Many threads on same program

Some Applications of Petri Nets

Concurrent Program Analysis

Many threads on same program

`s`: Shared Boolean variable

Initially, `s`$=0$

```
def fun():
  s = 1
  s = 0
  if s == 1:
    raise Err()
```

Some Applications of Petri Nets

Concurrent Program Analysis

Many threads on same program

`s`: Shared Boolean variable

Initially, `s`$= 0$

Problem:

Can any thread reach the error state?

```
def fun():
  s = 1 ← loc₁
  s = 0 ← loc₂
  if s == 1: ← loc₃
    raise Err() ← Err
```

Some Applications of Petri Nets

Concurrent Program Analysis

Many threads on same program

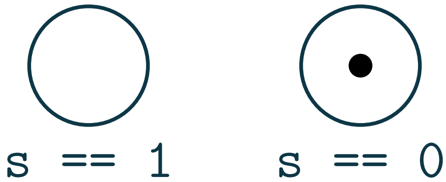
`s`: Shared Boolean variable

Initially, `s`$= 0$

Problem:

Can any thread reach the error state?

```
def fun():
  s = 1 ← loc1
  s = 0 ← loc2
  if s == 1: ← loc3
    raise Err() ← Err
```



Some Applications of Petri Nets

Concurrent Program Analysis

Many threads on same program

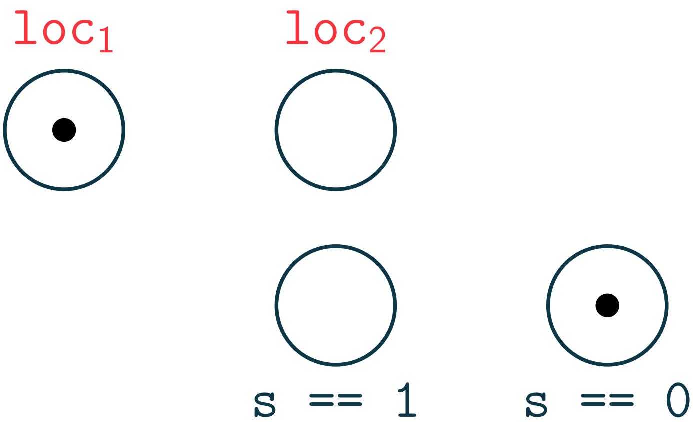
`s`: Shared Boolean variable

Initially, `s`$=0$

Problem:

Can any thread reach the error state?

```
def fun():
  s = 1 ← loc1
  s = 0 ← loc2
  if s == 1: ← loc3
    raise Err() ← Err
```



Some Applications of Petri Nets

Concurrent Program Analysis

Many threads on same program

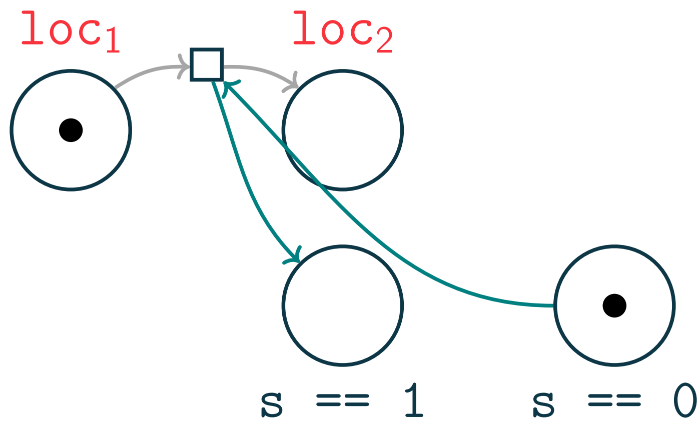
`s`: Shared Boolean variable

Initially, `s`$= 0$

Problem:

Can any thread reach the error state?

```
def fun():
  s = 1 ← loc1
  s = 0 ← loc2
  if s == 1: ← loc3
    raise Err() ← Err
```



Some Applications of Petri Nets

Concurrent Program Analysis

Many threads on same program

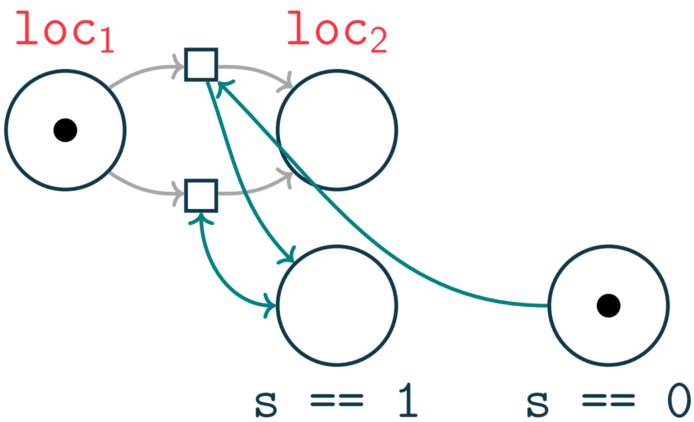
`s`: Shared Boolean variable

Initially, `s`$= 0$

Problem:

Can any thread reach the error state?

```
def fun():
  s = 1 ← loc1
  s = 0 ← loc2
  if s == 1: ← loc3
    raise Err() ← Err
```



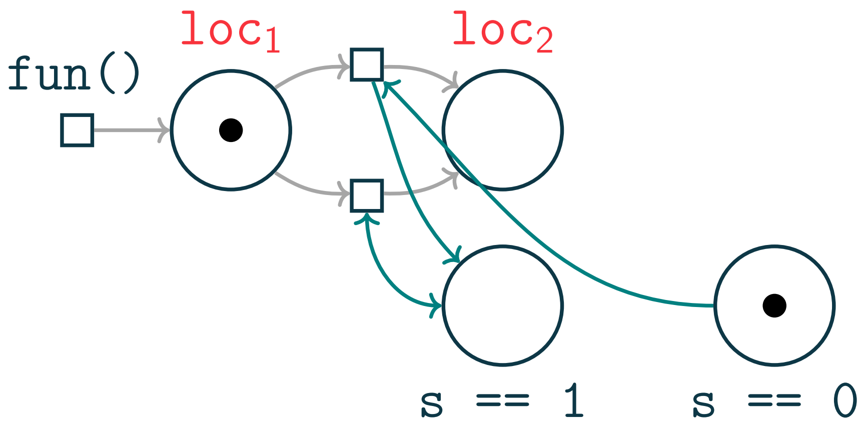
Some Applications of Petri Nets
Concurrent Program Analysis

Many threads on same program

`s`: Shared Boolean variable
Initially, `s`$= 0$

Problem:
Can any thread reach the error state?

```
def fun():
  s = 1 ← loc1
  s = 0 ← loc2
  if s == 1: ← loc3
    raise Err() ← Err
```



Some Applications of Petri Nets

Concurrent Program Analysis

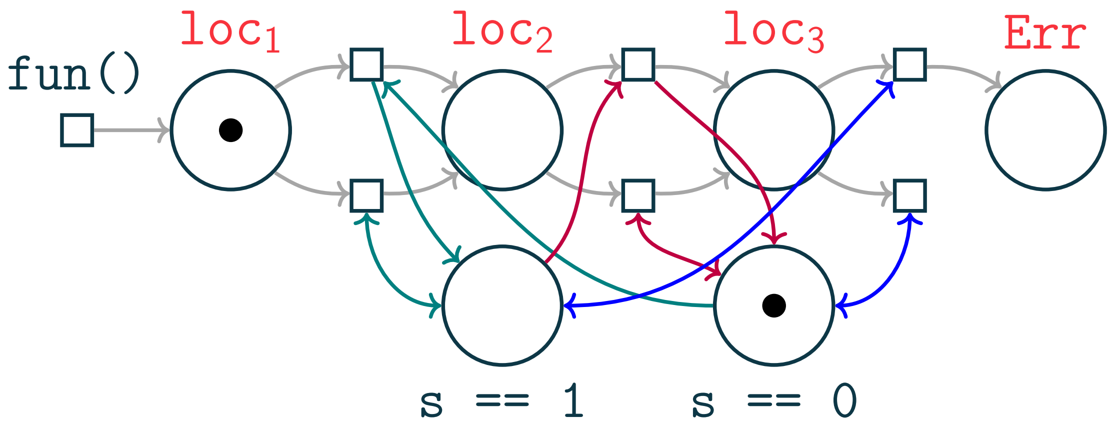
Many threads on same program

s: Shared Boolean variable

Initially, s$=0$

Problem:
Can any thread reach the error state?

```
def fun():
  s = 1 ← loc1
  s = 0 ← loc2
  if s == 1: ← loc3
    raise Err() ← Err
```



Some Applications of Petri Nets

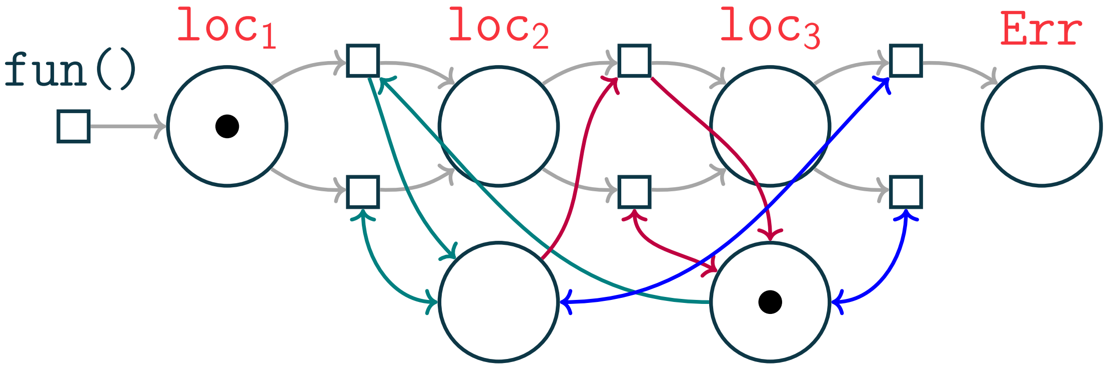
Concurrent Program Analysis

Many threads on same program

s: Shared Boolean variable
Initially, s= 0

Problem:
Can any thread reach the error state?

```
def fun():
  s = 1 ← loc1
  s = 0 ← loc2
  if s == 1: ← loc3
    raise Err() ← Err
```



Can there be at least one token in Err?

Some Applications of Petri Nets

Concurrent Program Analysis

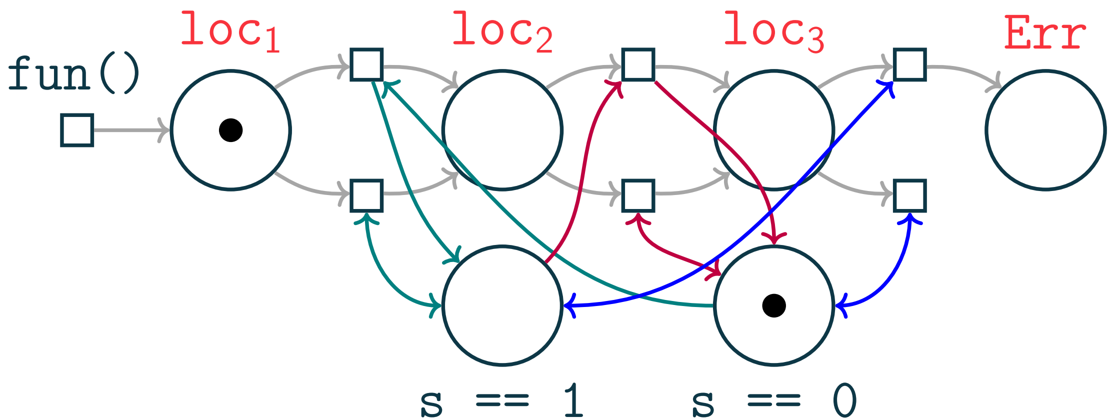
Many threads on same program

`s`: Shared Boolean variable

Initially, `s`$= 0$

```
def fun():
  s = 1  ← loc1
  s = 0  ← loc2
  if s == 1:  ← loc3
    raise Err()  ← Err
```

Problem:
Can any thread reach the error state?



Can there be at least one token in `Err`?

$\Rightarrow$ Coverability problem!

Some Applications of Petri Nets

Program Synthesis [Feng et al., POPL 2017]

Synthesize a function:

```
Area rotate(Area area,
    Point2D point,
    double angle)
```

Some Applications of Petri Nets

Program Synthesis [Feng et al., POPL 2017]

Synthesize a function:

```
Area rotate(Area area,
    Point2D point,
    double angle)
```

Use methods from the `java.awt.geom` library

Some Applications of Petri Nets

Program Synthesis [Feng et al., POPL 2017]

Synthesize a function:

```
Area rotate(Area area,
    Point2D point,
    double angle)
```

Use methods from the `java.awt.geom` library

java.awt.geom
`new AffineTrans()`
`double Point2D.getX()`
`double Point2D.getY()`
`void AffineTrans.` `  setToRotation(double, double, double)`
`Area Area.createTransArea(AffineTrans)`

Some Applications of Petri Nets

Program Synthesis [Feng et al., POPL 2017]

Synthesize a function:

```
Area rotate(Area area,
    Point2D point,
    double angle)
```

Use methods from the `java.awt.geom` library

java.awt.geom
`new AffineTrans()`
`double Point2D.getX()`
`double Point2D.getY()`
`void AffineTrans.` `  setToRotation(double, double, double)`
`Area Area.createTransArea(AffineTrans)`

Goal: Find programs that typecheck, chaining methods from the library

Some Applications of Petri Nets

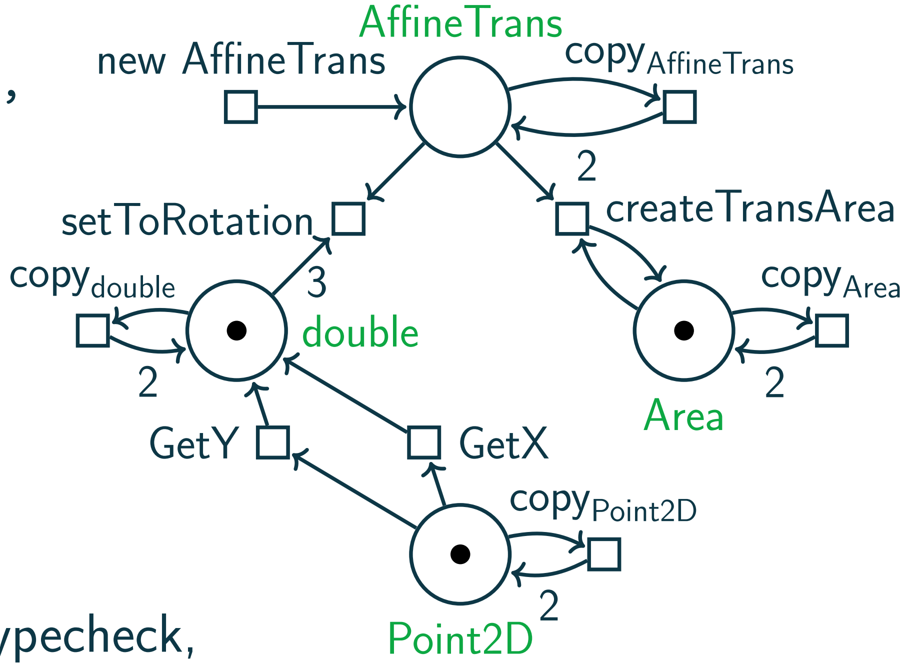
Program Synthesis [Feng et al., POPL 2017]

Synthesize a function:

```
Area rotate(Area area,
    Point2D point,
    double angle)
```

Use methods from the `java.awt.geom` library

```
java.awt.geom
new AffineTrans()
double Point2D.getX()
double Point2D.getY()
void AffineTrans.
  setToRotation(double, double, double)
Area Area.createTransArea(AffineTrans)
```



Goal: Find programs that typecheck, chaining methods from the library

Some Applications of Petri Nets

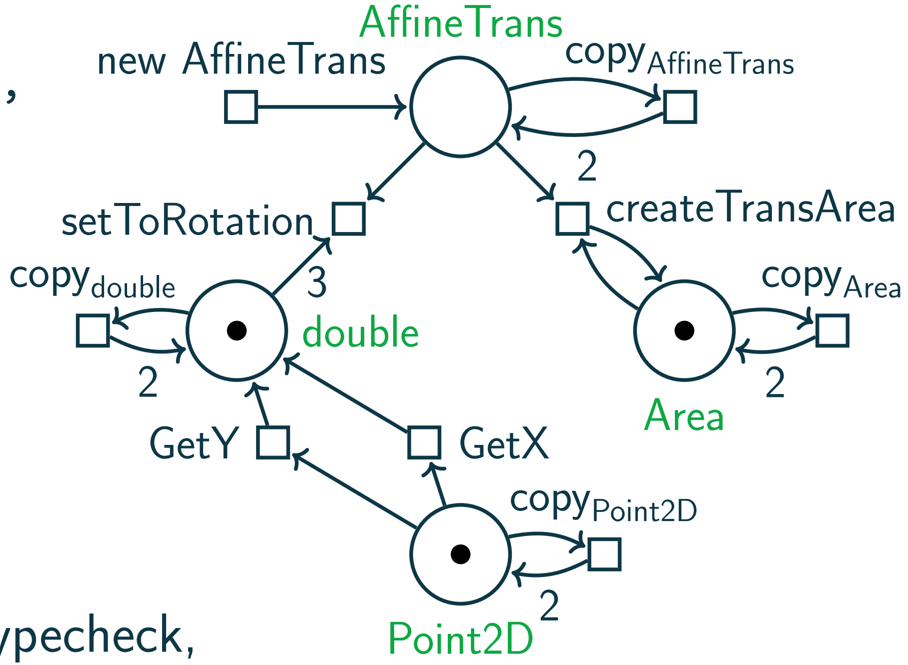
Program Synthesis [Feng et al., POPL 2017]

Synthesize a function:

```
Area rotate(Area area,
    Point2D point,
    double angle)
```

Use methods from the `java.awt.geom` library

```
java.awt.geom
new AffineTrans()
double Point2D.getX()
double Point2D.getY()
void AffineTrans.
  setToRotation(double, double, double)
Area Area.createTransArea(AffineTrans)
```



Goal: Find programs that typecheck, chaining methods from the library

$\Rightarrow$ Typechecking programs correspond to runs starting with a token in `Area`, `Point2D` and `double`, ending with exactly one token in `Area`

Some Applications of Petri Nets

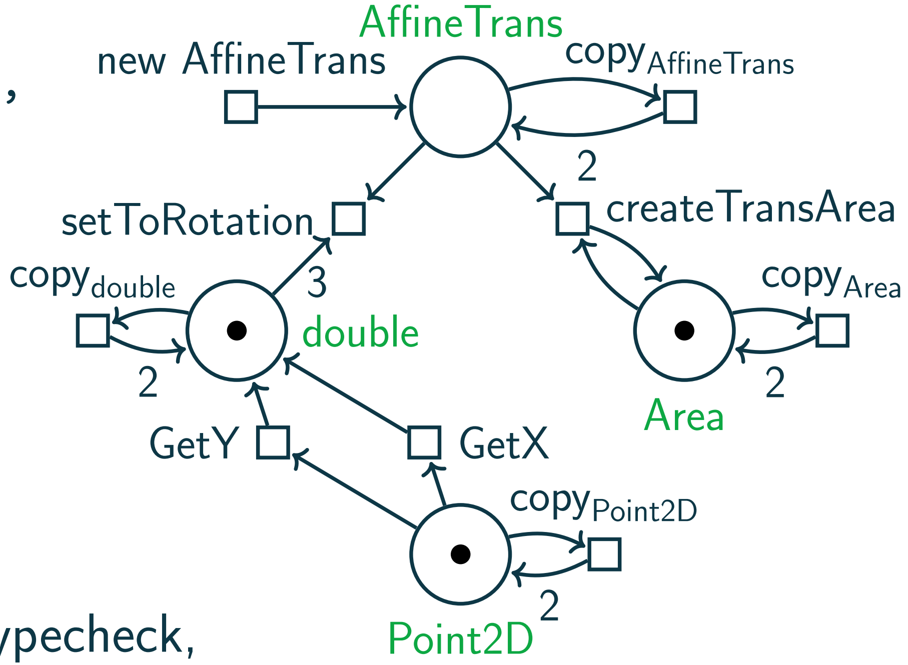
Program Synthesis [Feng et al., POPL 2017]

Synthesize a function:

```
Area rotate(Area area,
    Point2D point,
    double angle)
```

Use methods from the `java.awt.geom` library

java.awt.geom
`new AffineTrans()`
`double Point2D.getX()`
`double Point2D.getY()`
`void AffineTrans.setToRotation(double, double, double)`
`Area Area.createTransArea(AffineTrans)`



Goal: Find programs that typecheck, chaining methods from the library

$\Rightarrow$ Typechecking programs correspond to runs starting with a token in `Area`, `Point2D` and `double`, ending with exactly one token in `Area`

$\Rightarrow$ Reachability Problem!

Some Applications of Petri Nets

More applications

- Scheduling
- Business Processes
- Chemical reaction networks
- . . .

Some Applications of Petri Nets

More applications

- Scheduling
- Business Processes
- Chemical reaction networks
- ...

Common theme: Short witnesses!

Short bug traces = easier to fix

Short synthesized programs = easier to understand

...

State of the Art

Coverability

- [Karp and Miller, 1967]: Karp-Miller trees

State of the Art

Coverability

- [Karp and Miller, 1967]: Karp-Miller trees
- LoLA [Wolf, 2000]: Graph-search techniques (Karp-Miller trees), state space reductions, dedicated data structures, . . .
  still in development (and winning competitions) for 20+ years

State of the Art

Coverability

- [Karp and Miller, 1967]: Karp-Miller trees
- LoLA [Wolf, 2000]: Graph-search techniques (Karp-Miller trees), state space reductions, dedicated data structures, ...
  still in development (and winning competitions) for 20+ years
- [Abdulla et al., 2001]: Backward algorithm for WSTS, implemented in MIST [Ganty et. al, 2007]

State of the Art

Coverability

- [Karp and Miller, 1967]: Karp-Miller trees
- LoLA [Wolf, 2000]: Graph-search techniques (Karp-Miller trees), state space reductions, dedicated data structures, . . .
  still in development (and winning competitions) for 20+ years
- [Abdulla et al., 2001]: Backward algorithm for WSTS, implemented in MIST [Ganty et. al, 2007]
- BFC [Kaiser et al., 2014]: Target Set Widening/Accelerations

State of the Art

Coverability

- [Karp and Miller, 1967]: Karp-Miller trees
- LoLA [Wolf, 2000]: Graph-search techniques (Karp-Miller trees), state space reductions, dedicated data structures, . . .
  still in development (and winning competitions) for 20+ years
- [Abdulla et al., 2001]: Backward algorithm for WSTS, implemented in MIST [Ganty et. al, 2007]
- Bfc [Kaiser et al., 2014]: Target Set Widening/Accelerations
- QCover [Blondin et al., 2016]: Backward algorithm with pruning, based on Continuous Petri Nets (tighter approximation)

State of the Art

Reachability

- [Kosaraju, 1982]: Complete algorithm for reachability, Ackermannian complexity

State of the Art
Reachability

- [Kosaraju, 1982]: Complete algorithm for reachability, Ackermannian complexity
- KReach [Dixon and Lazić, 2020]: Implementation of Kosaraju's 1982 algorithm, works for small examples

State of the Art

Reachability

- [Kosaraju, 1982]: Complete algorithm for reachability, Ackermannian complexity
- KReach [Dixon and Lazić, 2020]: Implementation of Kosaraju's 1982 algorithm, works for small examples
- LoLA: Depth-first search, Random walks

State of the Art

Reachability

- [Kosaraju, 1982]: Complete algorithm for reachability, Ackermannian complexity
- KReach [Dixon and Lazić, 2020]: Implementation of Kosaraju's 1982 algorithm, works for small examples
- LoLA: Depth-first search, Random walks
- Issue: Few benchmarks for reachability with infinite state spaces

State of the Art

Reachability

- [Kosaraju, 1982]: Complete algorithm for reachability, Ackermannian complexity
- KREACH [Dixon and Lazić, 2020]: Implementation of Kosaraju's 1982 algorithm, works for small examples
- LoLA: Depth-first search, Random walks
- Issue: Few benchmarks for reachability with infinite state spaces
- MIST: Standard benchmark suite, but almost no reachability

Part II

Reachability Overapproximations

Approximations

Two sources of hardness in Petri Nets

Approximations

Two sources of hardness in Petri Nets

- Token counts must be integers

Approximations

Two sources of hardness in Petri Nets

- Token counts must be integers
- Token counts must be nonnegative

Approximations

Two sources of hardness in Petri Nets

- Token counts must be integers
- Token counts must be nonnegative

Relaxing either restriction gives us
an **overapproximation** of reachability

Approximations

Two sources of hardness in Petri Nets

- Token counts must be integers
- Token counts must be nonnegative

Relaxing either restriction gives us
an **overapproximation** of reachability

If a target is unreachable in the overapproximation, then it is unreachable in the Petri Net!

[Esparza et al., 2014], [Blondin et al., 2016]

Approximations

Continuous Petri Nets/Continuous token counts

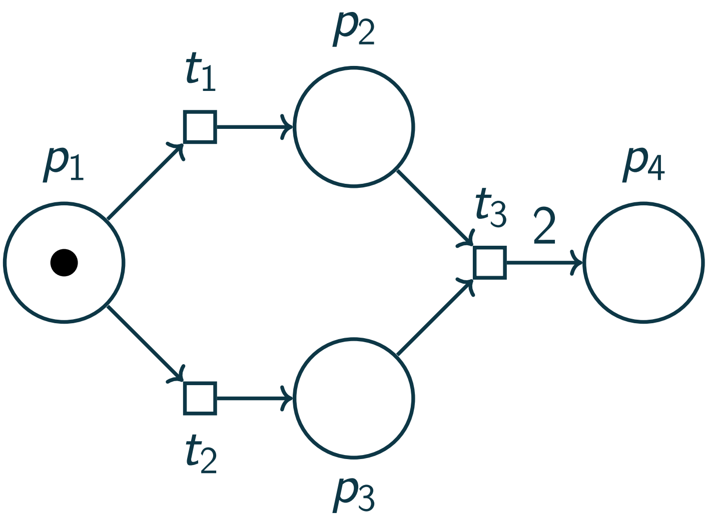
Allow firing transitions by a fraction $\beta \in (0, 1]$

Approximations

Continuous Petri Nets/Continuous token counts

Allow firing transitions by a fraction $\beta \in (0, 1]$

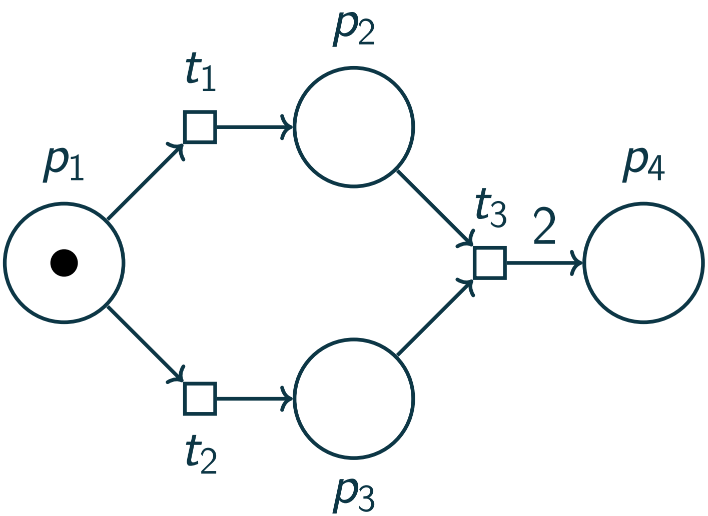
Initial Marking: $(1, 0, 0, 0)$



Approximations

Continuous Petri Nets/Continuous token counts

Allow firing transitions by a fraction $\beta \in (0, 1]$



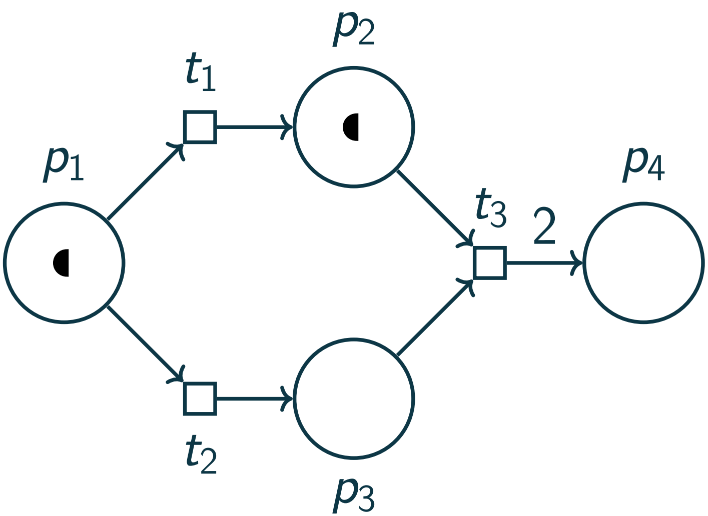
Initial Marking: $(1, 0, 0, 0)$

$(1, 0, 0, 0)$

Approximations

Continuous Petri Nets/Continuous token counts

Allow firing transitions by a fraction $\beta \in (0, 1]$



Initial Marking: $(1, 0, 0, 0)$

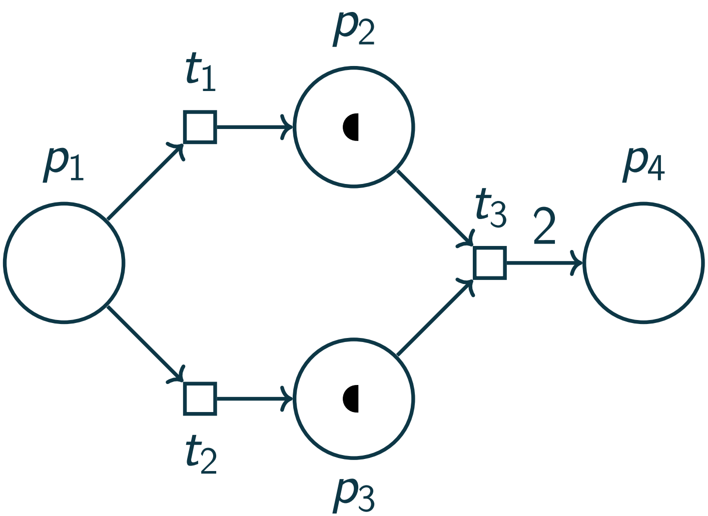
$$(1, 0, 0, 0)$$
$$\xrightarrow{0.5 t_1} (0.5, 0.5, 0, 0)$$

Approximations

Continuous Petri Nets/Continuous token counts

Allow firing transitions by a fraction $\beta \in (0, 1]$

Initial Marking: $(1, 0, 0, 0)$

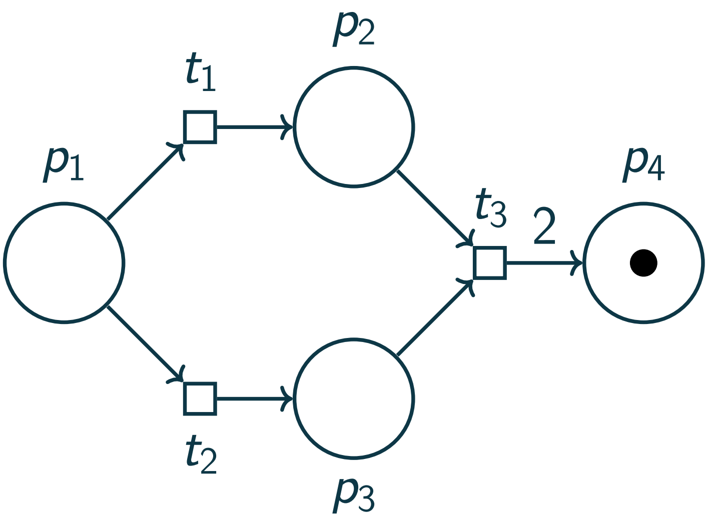


$$(1, 0, 0, 0) \xrightarrow{0.5t_1} (0.5, 0.5, 0, 0) \xrightarrow{0.5t_2} (0, 0.5, 0.5, 0)$$

Approximations

Continuous Petri Nets/Continuous token counts

Allow firing transitions by a fraction $\beta \in (0, 1]$



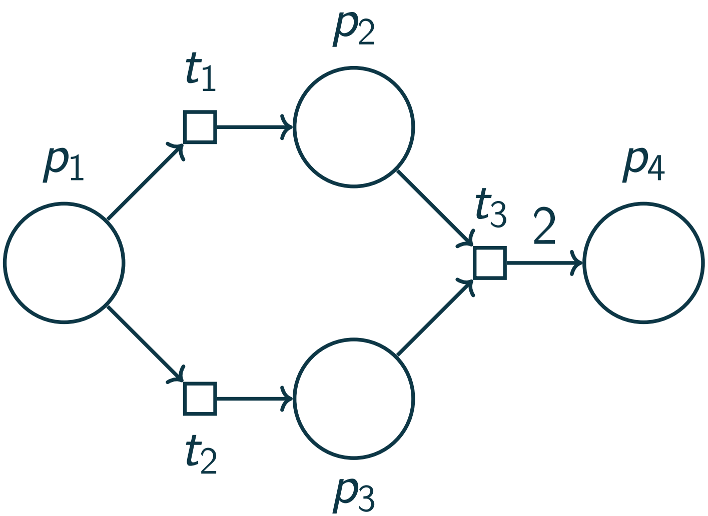
Initial Marking: $(1, 0, 0, 0)$

$$(1, 0, 0, 0) \xrightarrow{0.5t_1} (0.5, 0.5, 0, 0) \xrightarrow{0.5t_2} (0, 0.5, 0.5, 0) \xrightarrow{0.5t_3} (0, 0, 0, 1)$$

Approximations

Continuous Petri Nets/Continuous token counts

Allow firing transitions by a fraction $\beta \in (0, 1]$



Initial Marking: $(1, 0, 0, 0)$

$$(1, 0, 0, 0) \xrightarrow{0.5t_1} (0.5, 0.5, 0, 0) \xrightarrow{0.5t_2} (0, 0.5, 0.5, 0) \xrightarrow{0.5t_3} (0, 0, 0, 1)$$

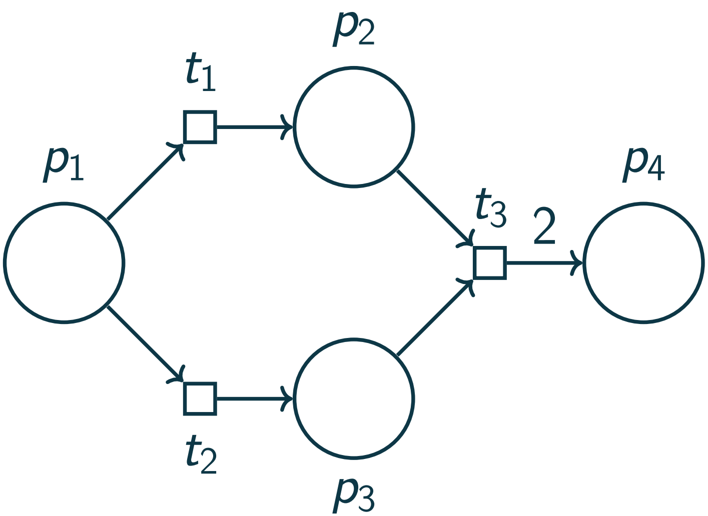
Reachability is **Ptime-complete** [Fraca and Haddad, 2013]

Approximations

Continuous Petri Nets/Continuous token counts

Allow firing transitions by a fraction $\beta \in (0, 1]$

Initial Marking: $(1, 0, 0, 0)$



$$
\begin{aligned}
&(1, 0, 0, 0) \\
&\xrightarrow{0.5 t_1} (0.5, 0.5, 0, 0) \\
&\xrightarrow{0.5 t_2} (0, 0.5, 0.5, 0) \\
&\xrightarrow{0.5 t_3} (0, 0, 0, 1)
\end{aligned}
$$

Reachability is **Ptime-complete** [Fraca and Haddad, 2013]

Alternatively, expressed as a formula

in existential $FO(\mathbb{Q}, +, <)$ — Satisfiability Modulo Theories/SMT

SMT Solving is fast in practice, e.g., via Z3 [Blondin et al., 2016]

Approximations

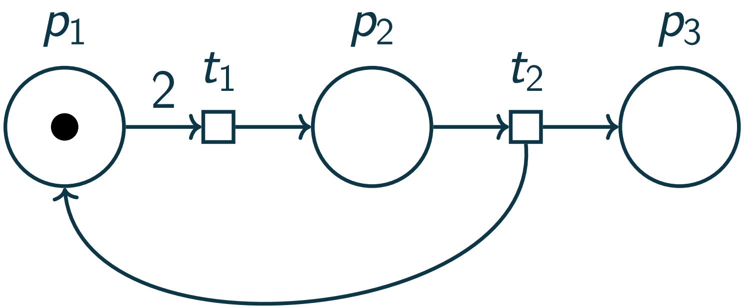
State Equation over $\mathbb{N}$/Negative token counts

Allow firing transitions when it would yield negative tokens

Approximations

State Equation over $\mathbb{N}$/Negative token counts

Allow firing transitions when it would yield negative tokens



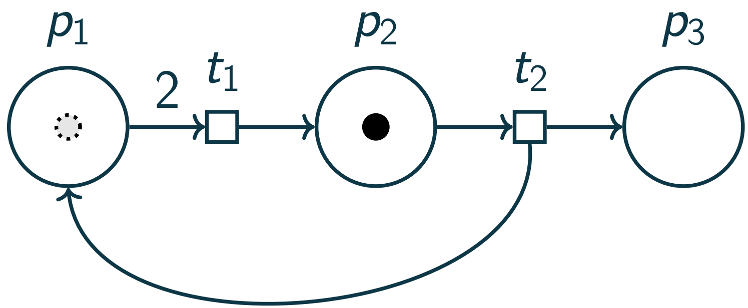
Initial Marking: $(1, 0, 0)$

$(1, 0, 0)$

Approximations

State Equation over $\mathbb{N}$/Negative token counts

Allow firing transitions when it would yield negative tokens



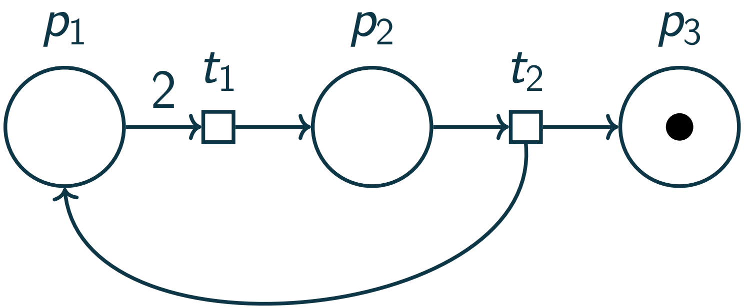
Initial Marking: $(1, 0, 0)$

$(1, 0, 0)$
$\xrightarrow{t_1} (-1, 1, 0)$

Approximations

State Equation over $\mathbb{N}$/Negative token counts

Allow firing transitions when it would yield negative tokens



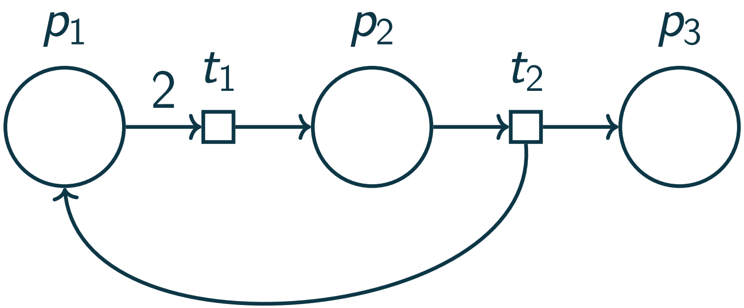
Initial Marking: $(1, 0, 0)$

$$(1, 0, 0) \xrightarrow{t_1} (-1, 1, 0) \xrightarrow{t_2} (0, 0, 1)$$

Approximations

State Equation over $\mathbb{N}$/Negative token counts

Allow firing transitions when it would yield negative tokens



Initial Marking: $(1, 0, 0)$

$(1, 0, 0)$
$\xrightarrow{t_1} (-1, 1, 0)$
$\xrightarrow{t_2} (0, 0, 1)$

Reachability from $(p_{1\,init}, p_{2\,init}, p_{3\,init})$ to $(p_{1\,final}, p_{2\,final}, p_{3\,final})$ if and only if $\exists t_1, t_2 \in \mathbb{N}$ such that:

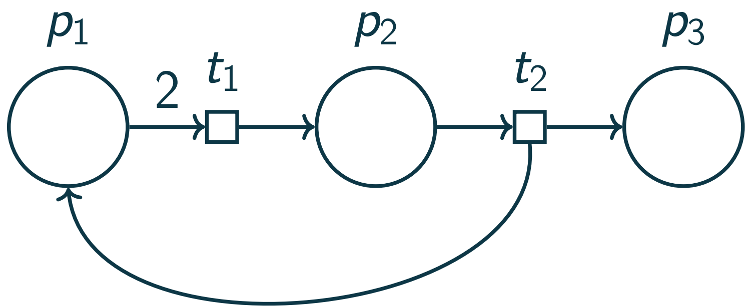
$$p_{1\,final} = p_{1\,init} - 2 \cdot t_1 + t_2$$
$$p_{2\,final} = p_{2\,init} + t_1 - t_2$$
$$p_{3\,final} = p_{3\,init} + t_2$$

$\Rightarrow$ State Equation over $\mathbb{N}$

Approximations

State Equation over $\mathbb{N}$/Negative token counts

Allow firing transitions when it would yield negative tokens



Initial Marking: $(1, 0, 0)$

$(1, 0, 0)$
$\xrightarrow{t_1} (-1, 1, 0)$
$\xrightarrow{t_2} (0, 0, 1)$

Reachability from $(p_{1\,init}, p_{2\,init}, p_{3\,init})$ to $(p_{1\,final}, p_{2\,final}, p_{3\,final})$ if and only if $\exists t_1, t_2 \in \mathbb{N}$ such that:

$$p_{1\,final} = p_{1\,init} - 2 \cdot t_1 + t_2$$
$$p_{2\,final} = p_{2\,init} + t_1 - t_2 \qquad \Rightarrow \text{State Equation over } \mathbb{N}$$
$$p_{3\,final} = p_{3\,init} + t_2$$

Solved via Integer Linear Programming (ILP)

$\Rightarrow$ Computable in NP.

Approximations

State Equation over $\mathbb{Q}$/Continuous, negative token counts

Again amounts to solving the State Equation, but over $\mathbb{Q}$.

Approximations

State Equation over $\mathbb{Q}$/Continuous, negative token counts

Again amounts to solving the State Equation, but over $\mathbb{Q}$.

Solved via Linear Programming (LP)
$\Rightarrow$ Computable in Ptime.

Approximations

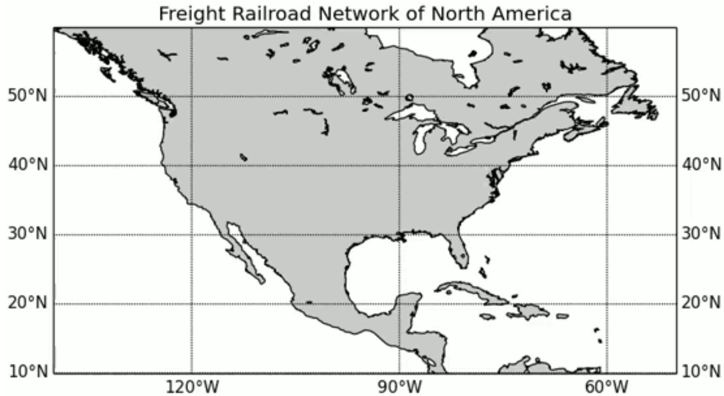
Overview

	Complexity	**Computed Via**
Petri Nets	Non-elementary	Kosaraju's
Continuous Petri Nets	Ptime-complete	SAT/SMT
State Equation over $\mathbb{N}$	NP-complete	Integer Lin. Prog.
State Equation over $\mathbb{Q}$	Ptime-complete	Lin. Prog.

Part III

Directed Search Algorithms

Directed Search Algorithms



Directed Search Algorithms

Directed Search Algorithms

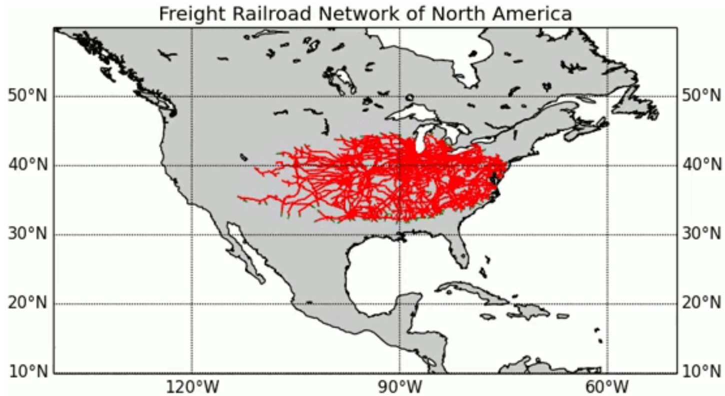
Directed Search Algorithms

Directed Search Algorithms

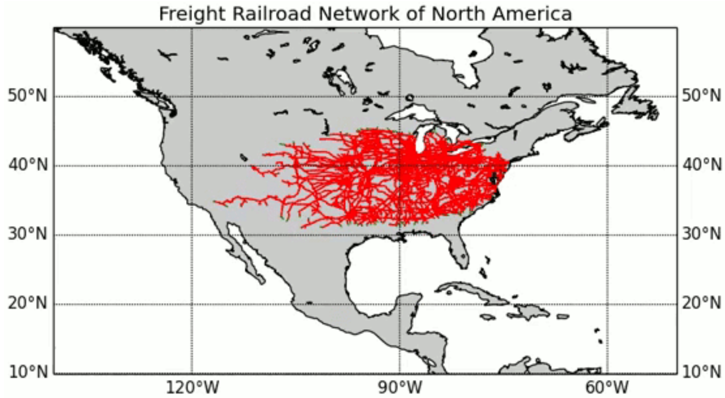
Directed Search Algorithms

Directed Search Algorithms

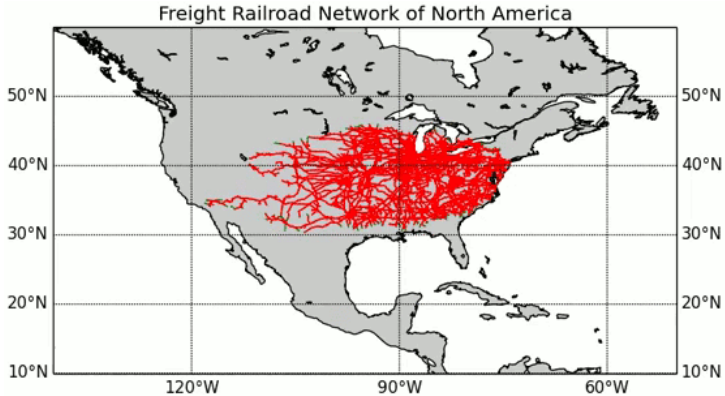
Directed Search Algorithms



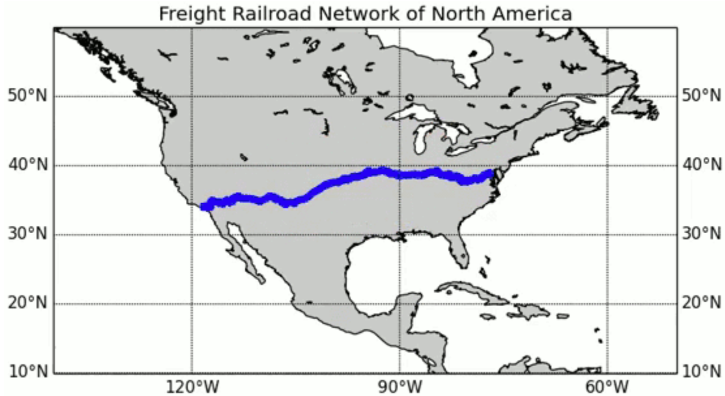
Directed Search Algorithms



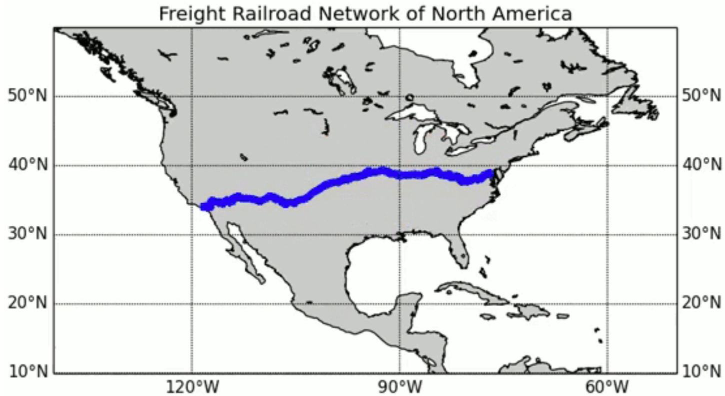
Directed Search Algorithms



Directed Search Algorithms



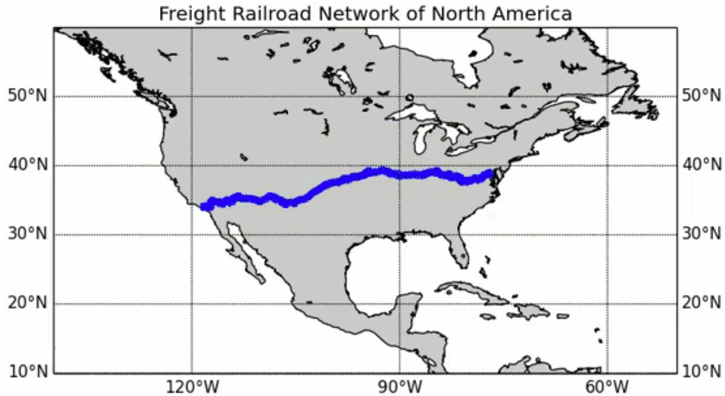
Directed Search Algorithms



Directed Search Algorithms can handle very large graphs

Used successfully in AI, network routing, . . .

Directed Search Algorithms



Directed Search Algorithms can handle very large graphs

Used successfully in AI, network routing, ...

Petri Nets have (infinite) reachability graphs!

Directed Search Algorithms

Directed Search Algorithms can handle very large graphs

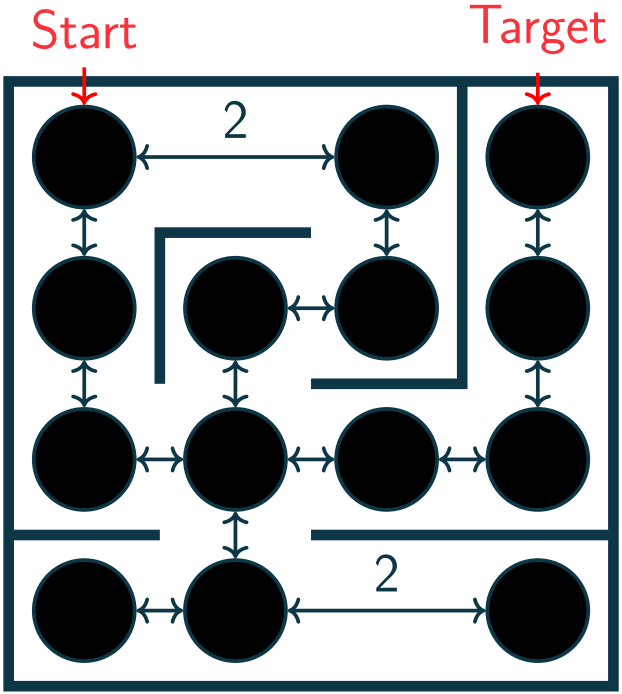
Used successfully in AI, network routing, . . .

Petri Nets have (infinite) reachability graphs!

First: Refresher on Directed Search Algorithms

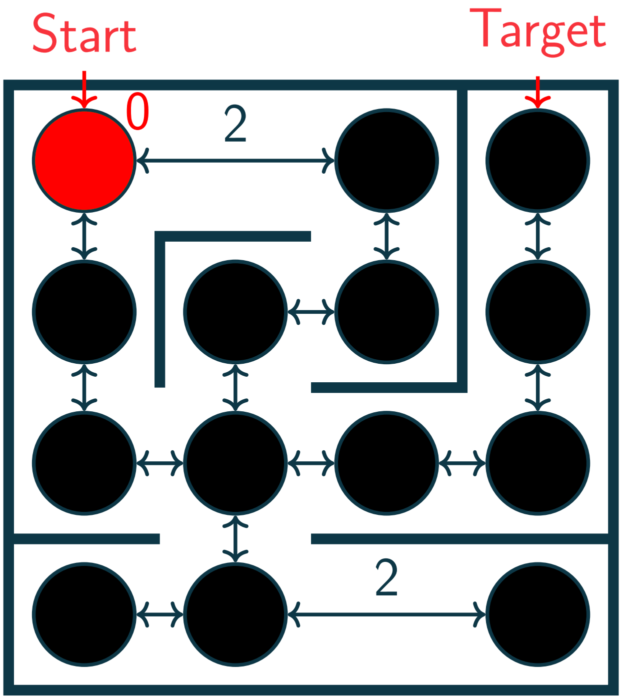
Afterwards: How to apply them to Petri nets

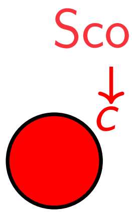
Directed Search Algorithms
Dijkstra's/Breadth First Search



Directed Search Algorithms

Dijkstra's/Breadth First Search





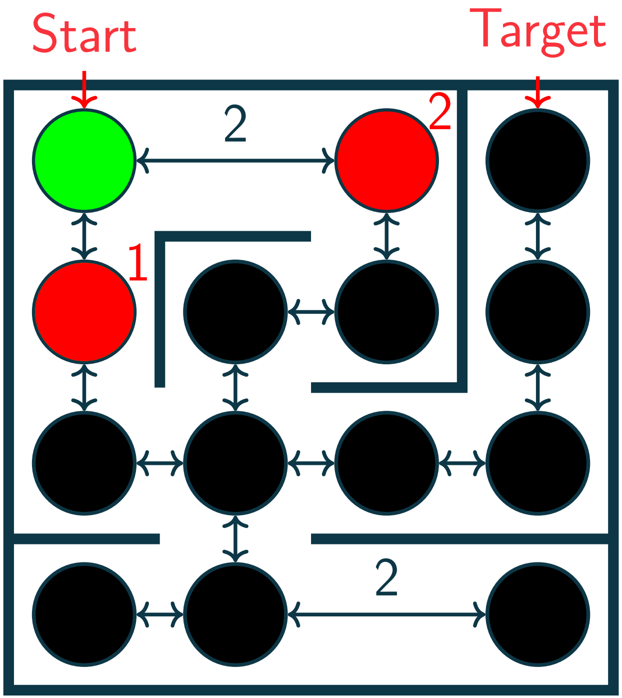
Frontier nodes:
Next nodes to explore

Explore node with
lowest score in frontier

Dijkstra's:
Score = Distance from Start

Directed Search Algorithms

Dijkstra's/Breadth First Search



Score

c

Frontier nodes:
Next nodes to explore

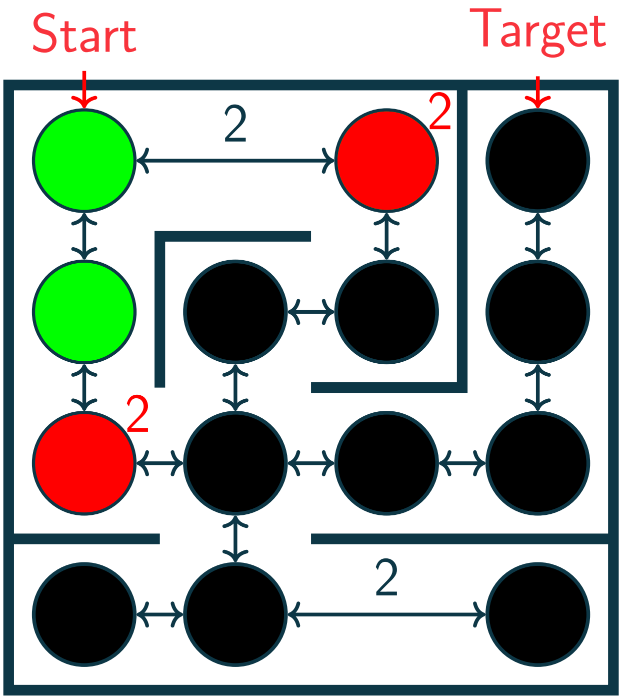
Closed nodes:
Nodes that were explored

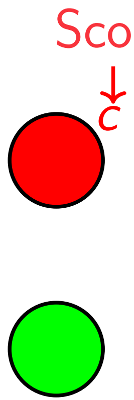
Explore node with
lowest score in frontier

Dijkstra's:
Score = Distance from Start

Directed Search Algorithms

Dijkstra's/Breadth First Search





Frontier nodes:
Next nodes to explore

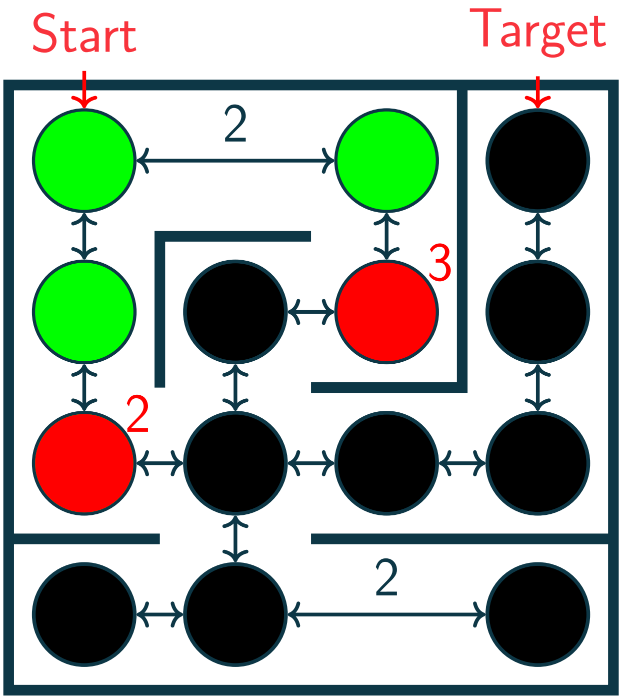
Closed nodes:
Nodes that were explored

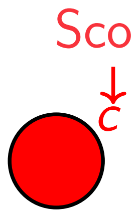
Explore node with
lowest score in frontier

Dijkstra's:
Score = Distance from Start

Directed Search Algorithms

Dijkstra's/Breadth First Search





Frontier nodes:
Next nodes to explore

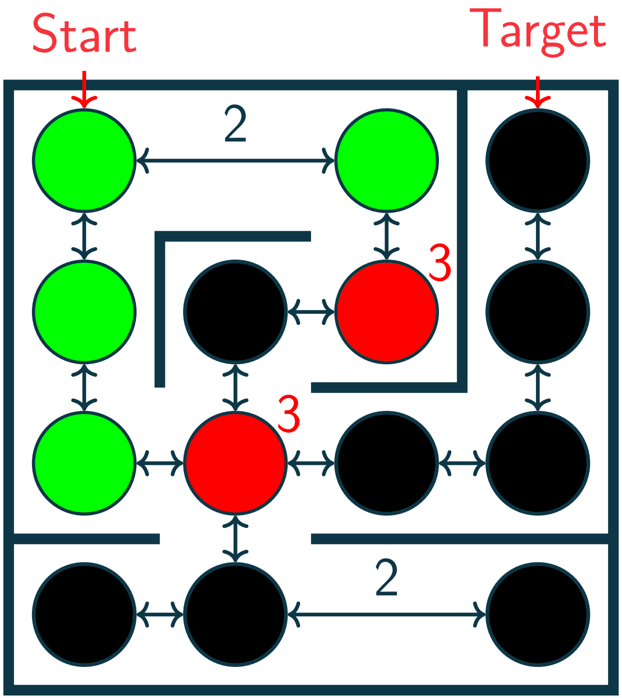
Closed nodes:
Nodes that were explored

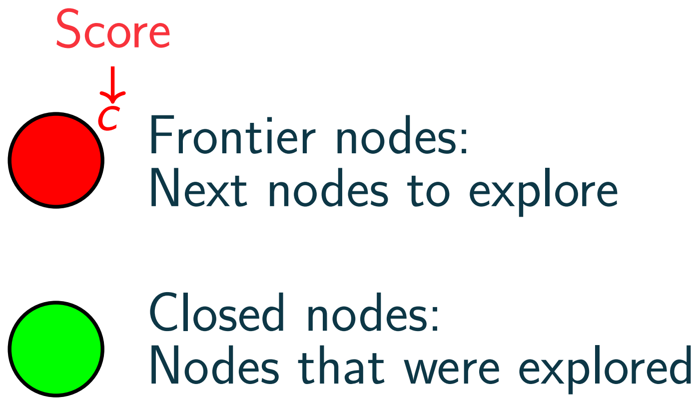
Explore node with lowest score in frontier

Dijkstra's:
Score = Distance from Start

Directed Search Algorithms

Dijkstra's/Breadth First Search



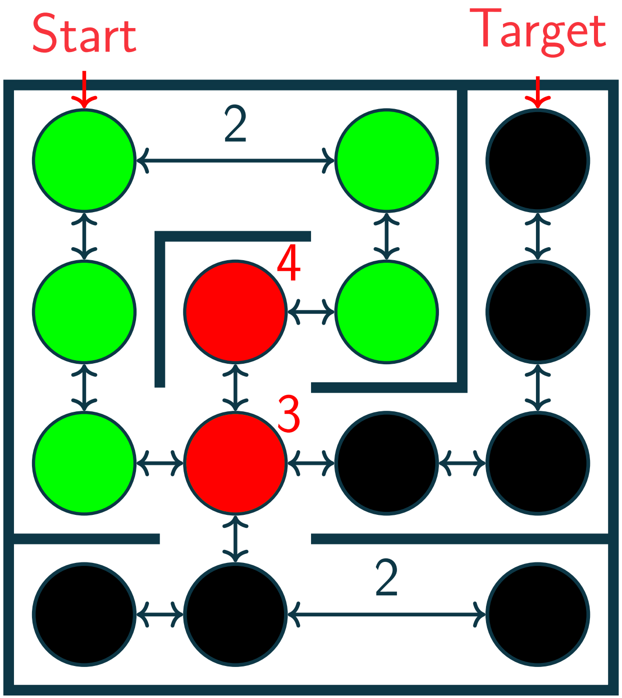


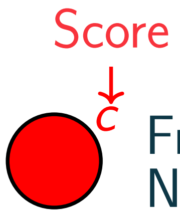
Explore node with
lowest score in frontier

Dijkstra's:
Score = Distance from Start

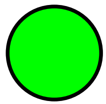
Directed Search Algorithms

Dijkstra's/Breadth First Search





Frontier nodes:
Next nodes to explore



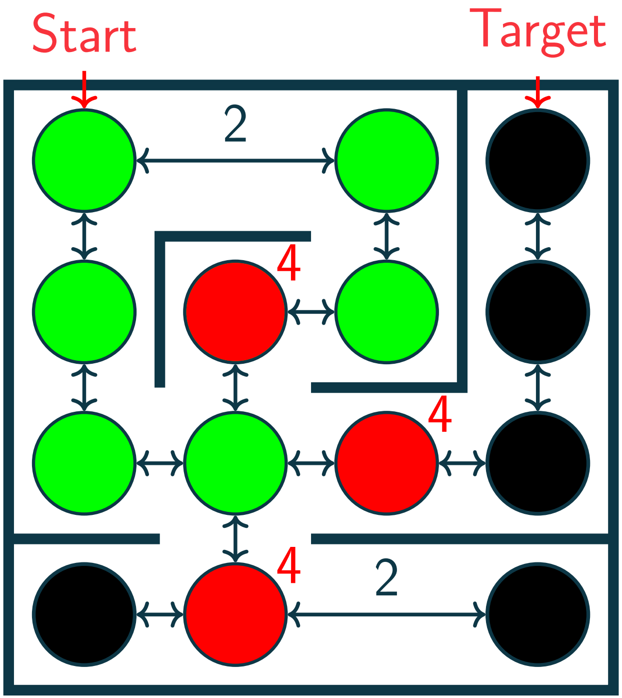
Closed nodes:
Nodes that were explored

Explore node with
lowest score in frontier

Dijkstra's:
Score = Distance from Start

Directed Search Algorithms

Dijkstra's/Breadth First Search



Frontier nodes:
Next nodes to explore

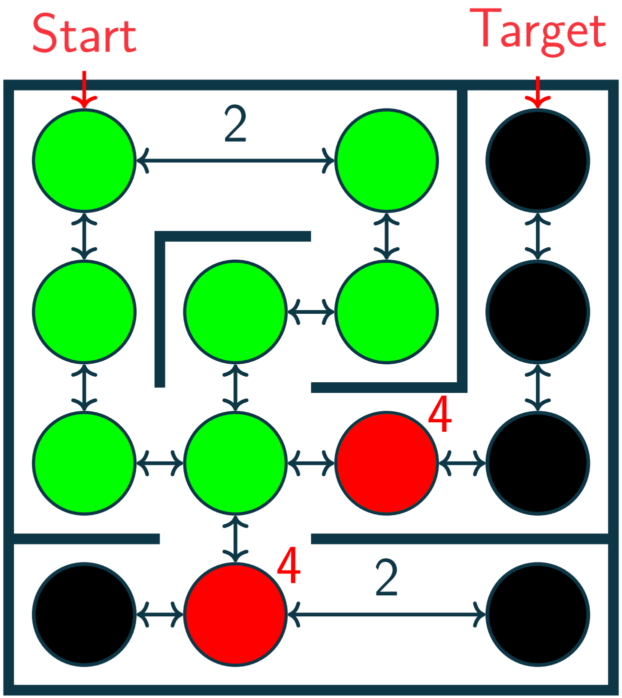
Closed nodes:
Nodes that were explored

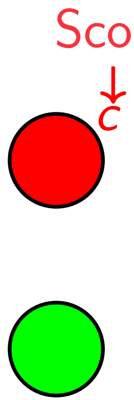
Explore node with
lowest score in frontier

Dijkstra's:
Score = Distance from Start

Directed Search Algorithms

Dijkstra's/Breadth First Search





Frontier nodes:
Next nodes to explore

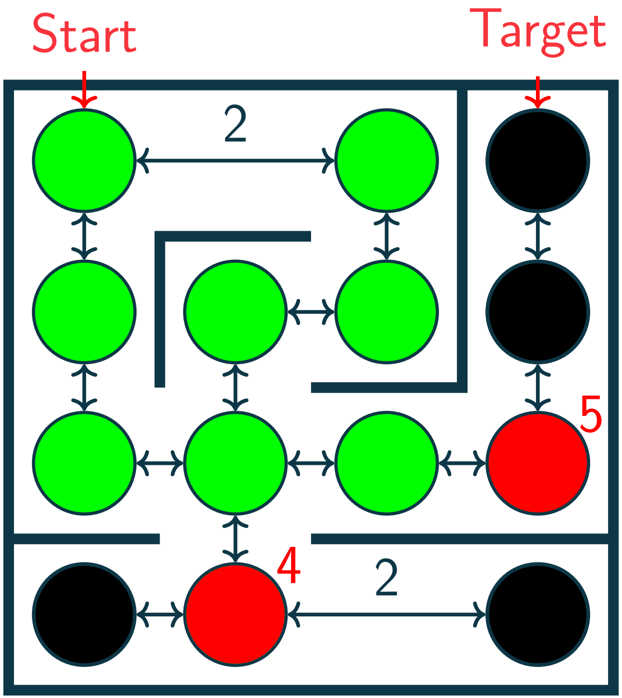
Closed nodes:
Nodes that were explored

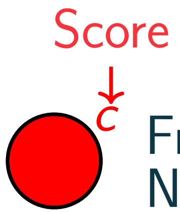
Explore node with
lowest score in frontier

Dijkstra's:
Score = Distance from Start

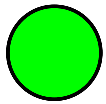
Directed Search Algorithms

Dijkstra's/Breadth First Search





Frontier nodes:
Next nodes to explore



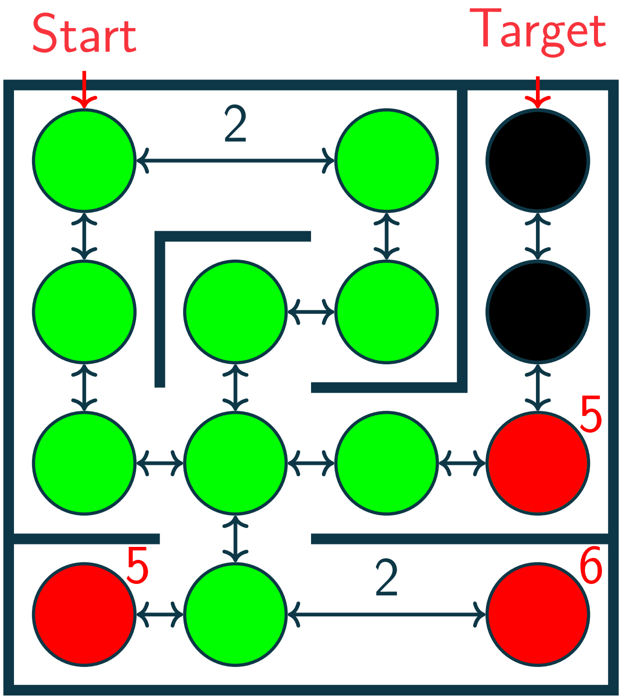
Closed nodes:
Nodes that were explored

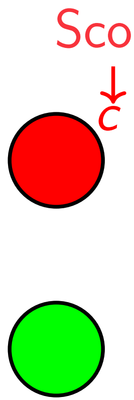
Explore node with
lowest score in frontier

Dijkstra's:
Score = Distance from Start

Directed Search Algorithms

Dijkstra's/Breadth First Search





Frontier nodes:
Next nodes to explore

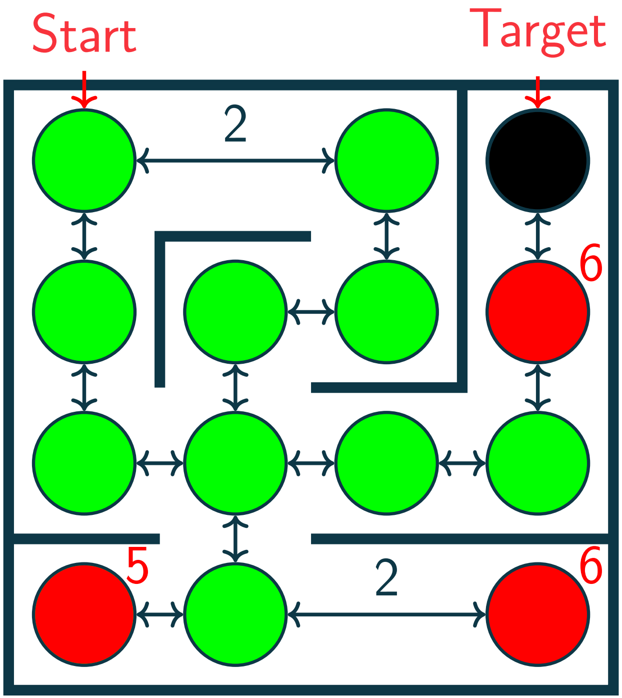
Closed nodes:
Nodes that were explored

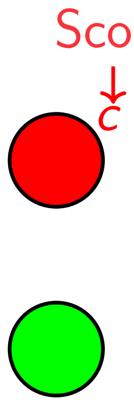
Explore node with
lowest score in frontier

Dijkstra's:
Score = Distance from Start

Directed Search Algorithms

Dijkstra's/Breadth First Search





Frontier nodes:
Next nodes to explore

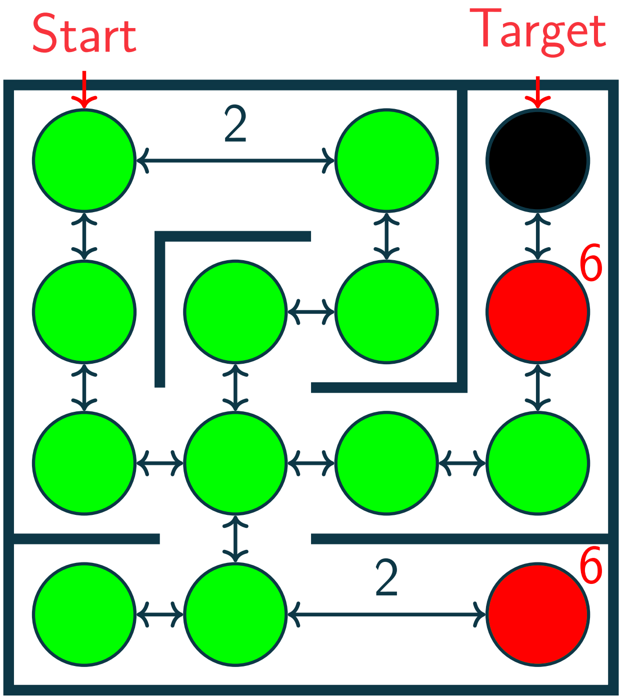
Closed nodes:
Nodes that were explored

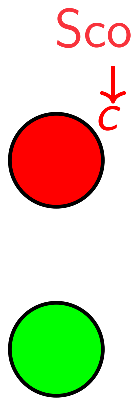
Explore node with
lowest score in frontier

Dijkstra's:
Score = Distance from Start

Directed Search Algorithms

Dijkstra's/Breadth First Search





Frontier nodes:
Next nodes to explore

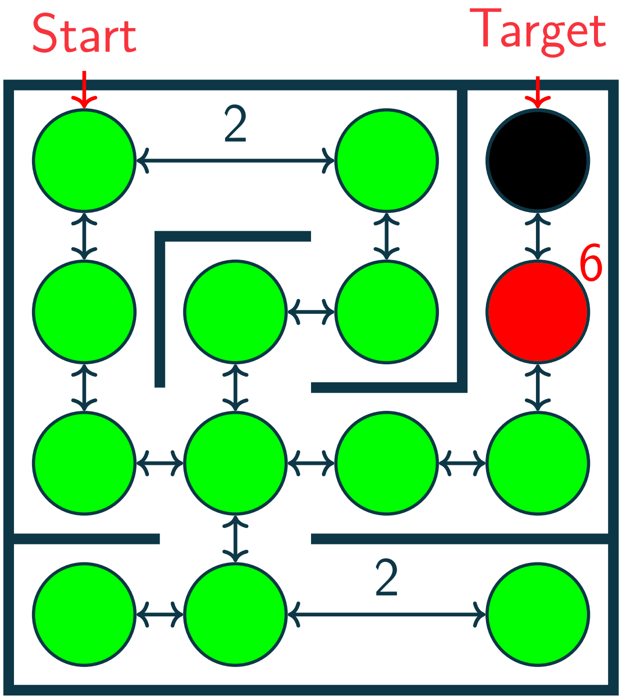
Closed nodes:
Nodes that were explored

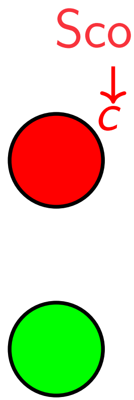
Explore node with
lowest score in frontier

Dijkstra's:
Score = Distance from Start

Directed Search Algorithms

Dijkstra's/Breadth First Search





Frontier nodes:
Next nodes to explore

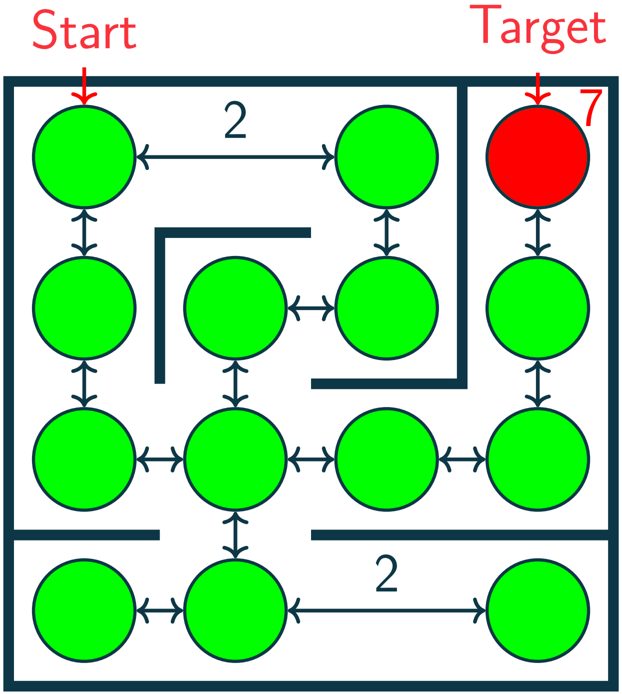
Closed nodes:
Nodes that were explored

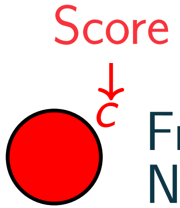
Explore node with
lowest score in frontier

Dijkstra's:
Score = Distance from Start

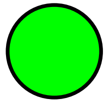
Directed Search Algorithms

Dijkstra's/Breadth First Search





Frontier nodes:
Next nodes to explore



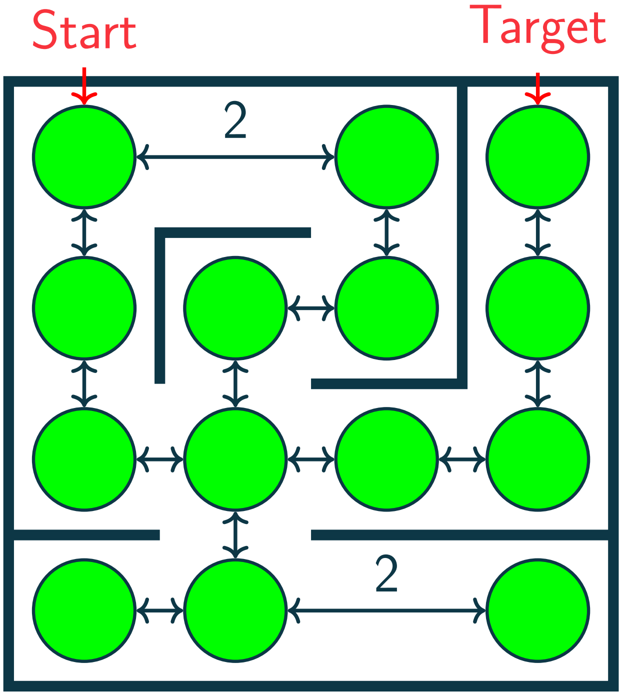
Closed nodes:
Nodes that were explored

Explore node with
lowest score in frontier

Dijkstra's:
Score = Distance from Start

Directed Search Algorithms

Dijkstra's/Breadth First Search



Score

c Frontier nodes:
Next nodes to explore

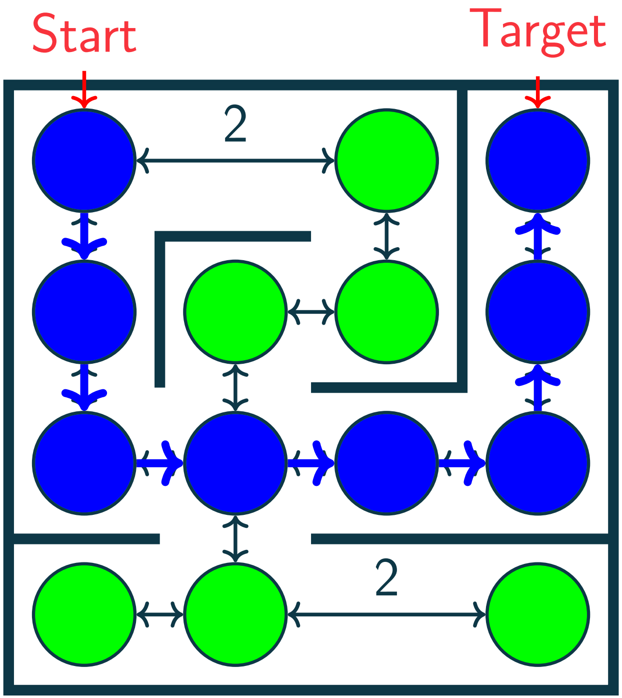
Closed nodes:
Nodes that were explored

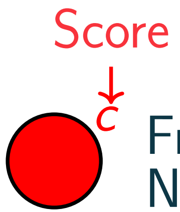
Explore node with
lowest score in frontier

Dijkstra's:
Score = Distance from Start

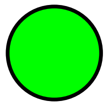
Directed Search Algorithms

Dijkstra's/Breadth First Search





Frontier nodes:
Next nodes to explore



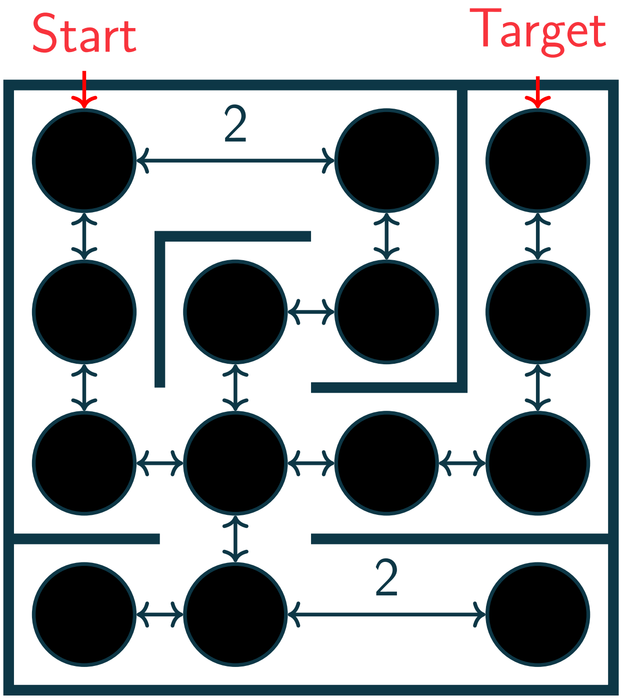
Closed nodes:
Nodes that were explored

Explore node with
lowest score in frontier

Dijkstra's:
Score = Distance from Start

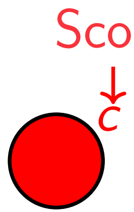
Directed Search Algorithms

Greedy Best-First Search (GBFS)

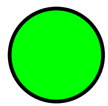


Heuristic estimation needed!

Heuristic: Grid-distance



Frontier nodes:
Next nodes to explore



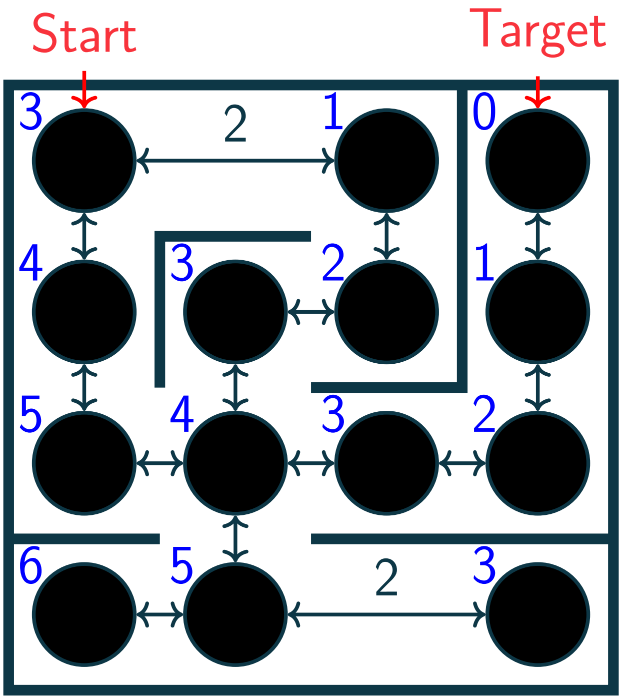
Closed nodes:
Nodes that were explored

Explore node with
lowest score in frontier

Greedy Best-First Search:
Score = Distance to Target

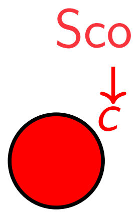
Directed Search Algorithms

Greedy Best-First Search (GBFS)

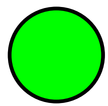


Heuristic estimation needed!

Heuristic: Grid-distance



Frontier nodes:
Next nodes to explore



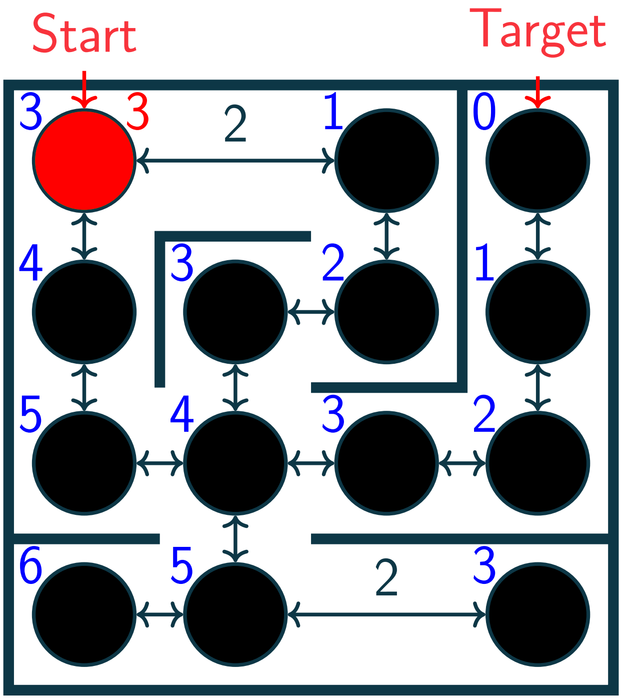
Closed nodes:
Nodes that were explored

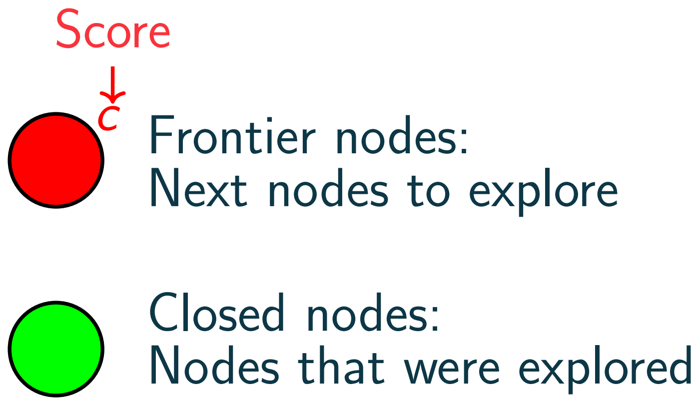
Explore node with
lowest score in frontier

Greedy Best-First Search:
Score = Distance to Target

Directed Search Algorithms

Greedy Best-First Search (GBFS)





Heuristic estimation needed!

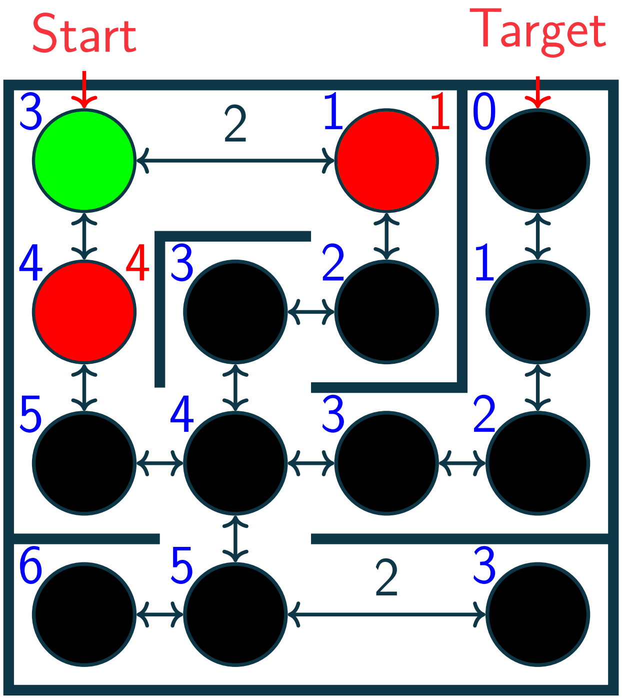
Heuristic: Grid-distance

Explore node with lowest score in frontier

Greedy Best-First Search:
Score = Distance to Target

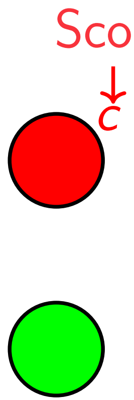
Directed Search Algorithms

Greedy Best-First Search (GBFS)



Heuristic estimation needed!

Heuristic: Grid-distance



Frontier nodes:
Next nodes to explore

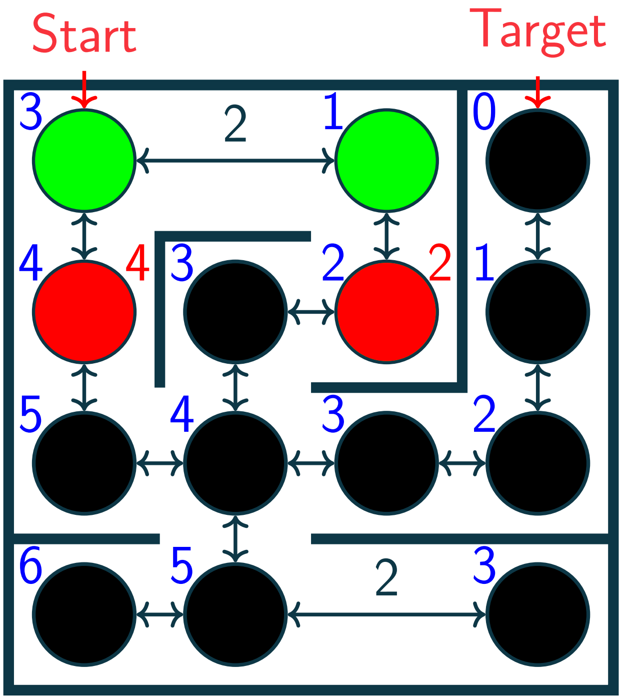
Closed nodes:
Nodes that were explored

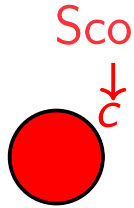
Explore node with
lowest score in frontier

Greedy Best-First Search:
Score = Distance to Target

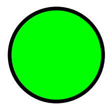
Directed Search Algorithms

Greedy Best-First Search (GBFS)





Frontier nodes:
Next nodes to explore



Closed nodes:
Nodes that were explored

Explore node with
lowest score in frontier

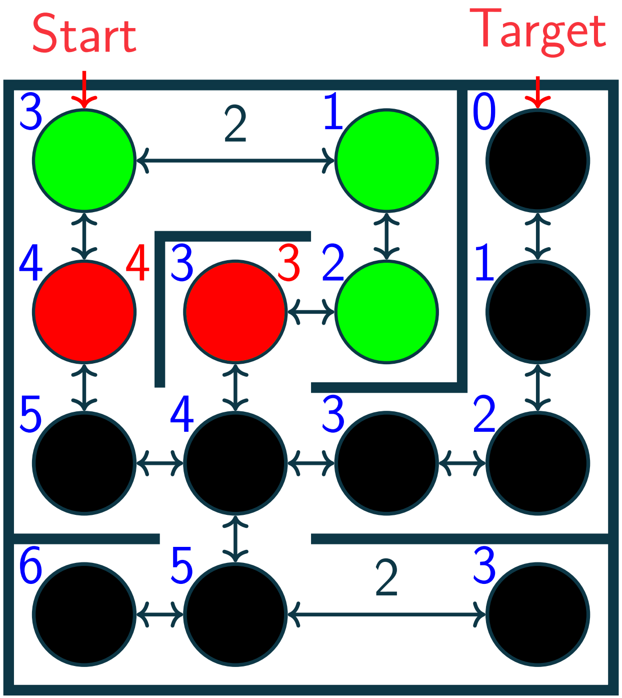
Greedy Best-First Search:
Score = Distance to Target

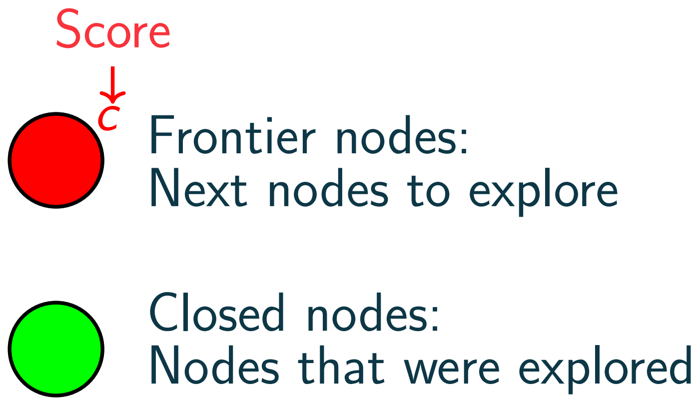
Heuristic estimation needed!

Heuristic: Grid-distance

Directed Search Algorithms

Greedy Best-First Search (GBFS)





Explore node with
lowest score in frontier

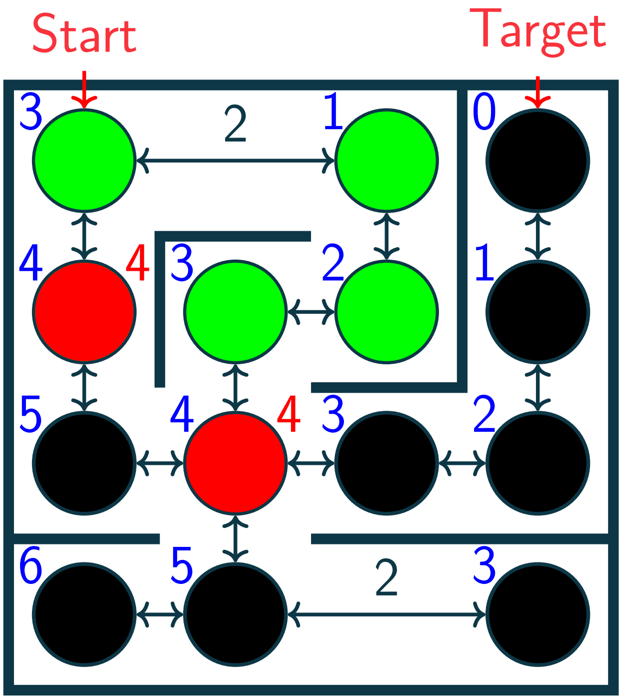
Greedy Best-First Search:
Score = Distance to Target

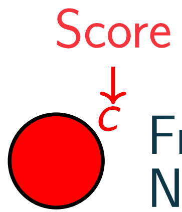
Heuristic estimation needed!

Heuristic: Grid-distance

Directed Search Algorithms

Greedy Best-First Search (GBFS)





Frontier nodes:
Next nodes to explore

Closed nodes:
Nodes that were explored

Heuristic estimation needed!

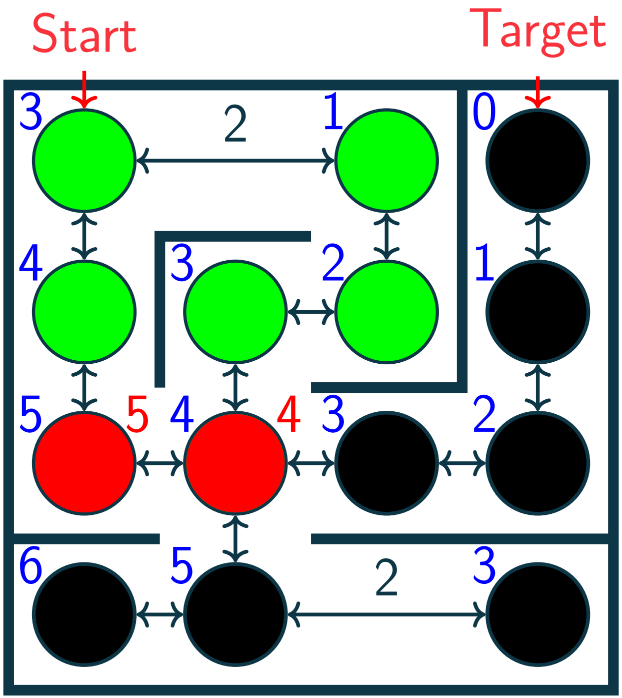
Heuristic: Grid-distance

Explore node with lowest score in frontier

Greedy Best-First Search:
Score = Distance to Target

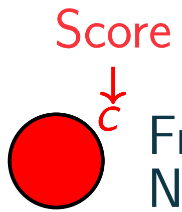
Directed Search Algorithms

Greedy Best-First Search (GBFS)

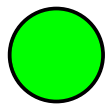


Heuristic estimation needed!

Heuristic: Grid-distance



Frontier nodes:
Next nodes to explore



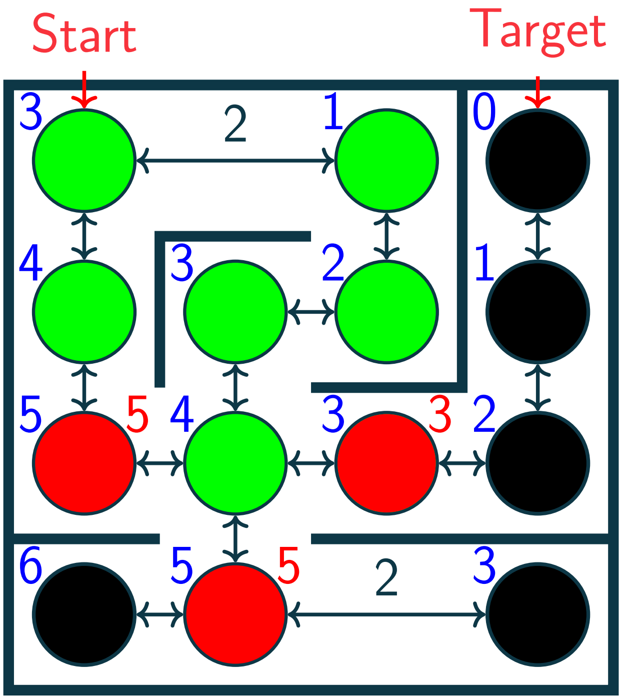
Closed nodes:
Nodes that were explored

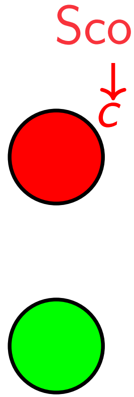
Explore node with
lowest score in frontier

Greedy Best-First Search:
Score = Distance to Target

Directed Search Algorithms

Greedy Best-First Search (GBFS)





Frontier nodes:
Next nodes to explore

Closed nodes:
Nodes that were explored

Explore node with
lowest score in frontier

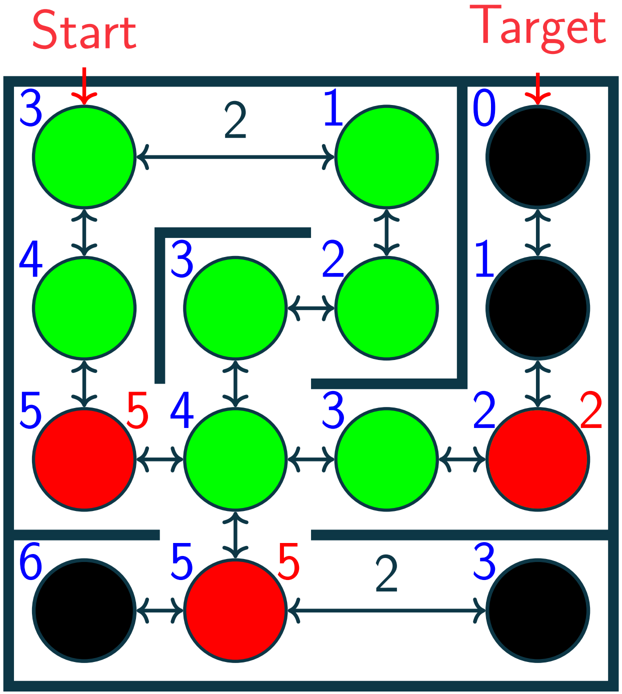
Greedy Best-First Search:
Score = Distance to Target

Heuristic estimation needed!

Heuristic: Grid-distance

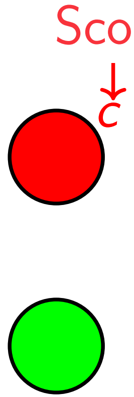
Directed Search Algorithms

Greedy Best-First Search (GBFS)



Heuristic estimation needed!

Heuristic: Grid-distance



Frontier nodes:
Next nodes to explore

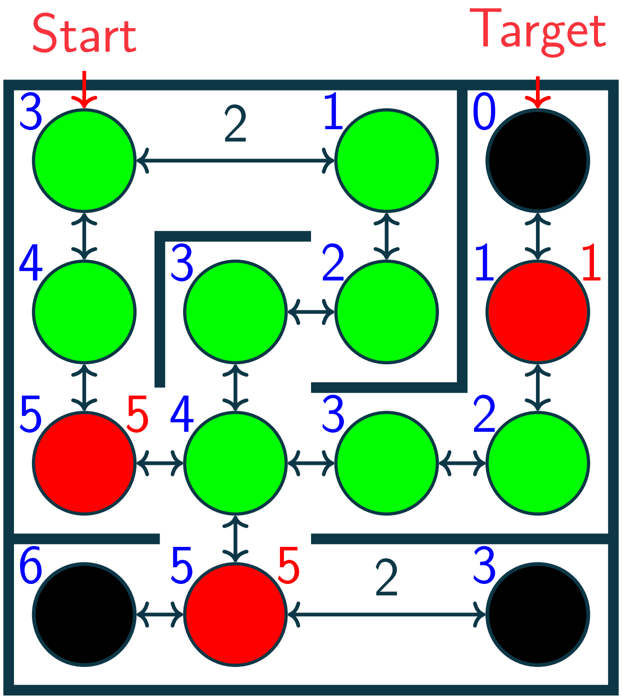
Closed nodes:
Nodes that were explored

Explore node with
lowest score in frontier

Greedy Best-First Search:
Score = Distance to Target

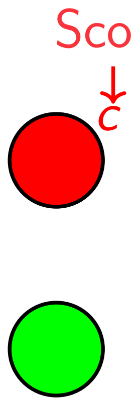
Directed Search Algorithms

Greedy Best-First Search (GBFS)



Heuristic estimation needed!

Heuristic: Grid-distance



Frontier nodes:
Next nodes to explore

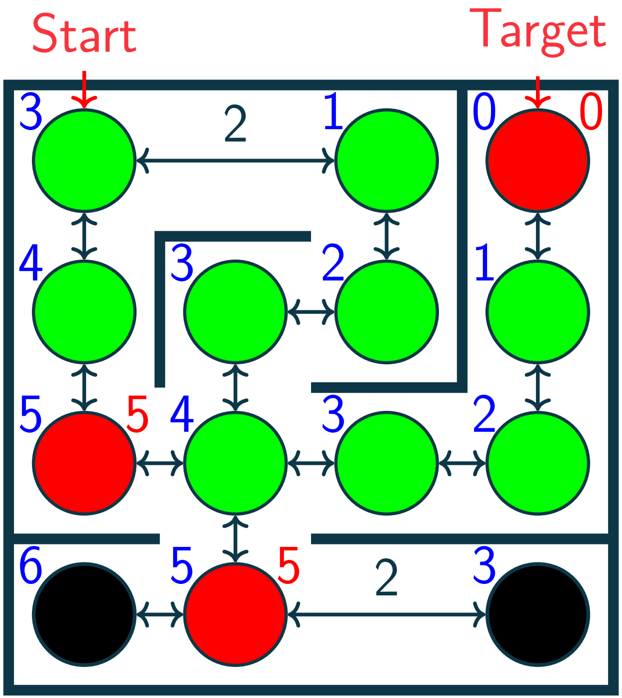
Closed nodes:
Nodes that were explored

Explore node with
lowest score in frontier

Greedy Best-First Search:
Score = Distance to Target

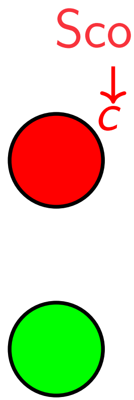
Directed Search Algorithms

Greedy Best-First Search (GBFS)



Heuristic estimation needed!

Heuristic: Grid-distance



Frontier nodes:
Next nodes to explore

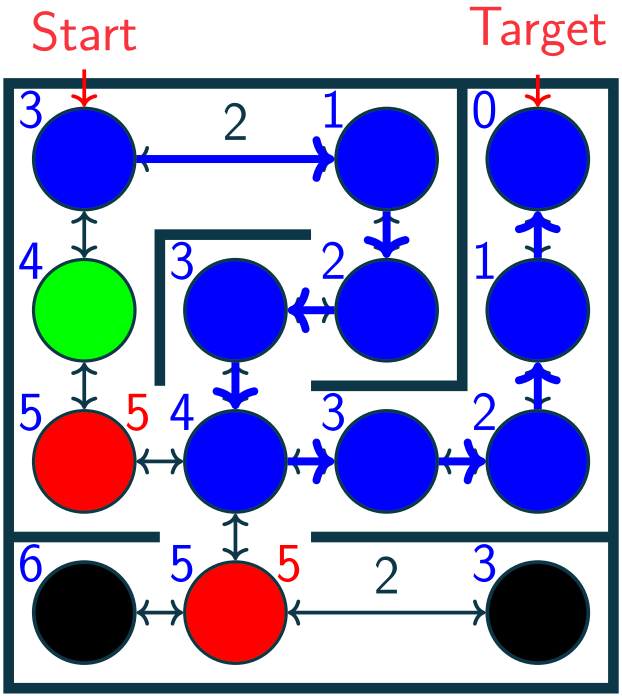
Closed nodes:
Nodes that were explored

Explore node with
lowest score in frontier

Greedy Best-First Search:
Score $=$ Distance to Target

Directed Search Algorithms

Greedy Best-First Search (GBFS)



Heuristic estimation needed!

Heuristic: Grid-distance

Score

c

Frontier nodes:
Next nodes to explore

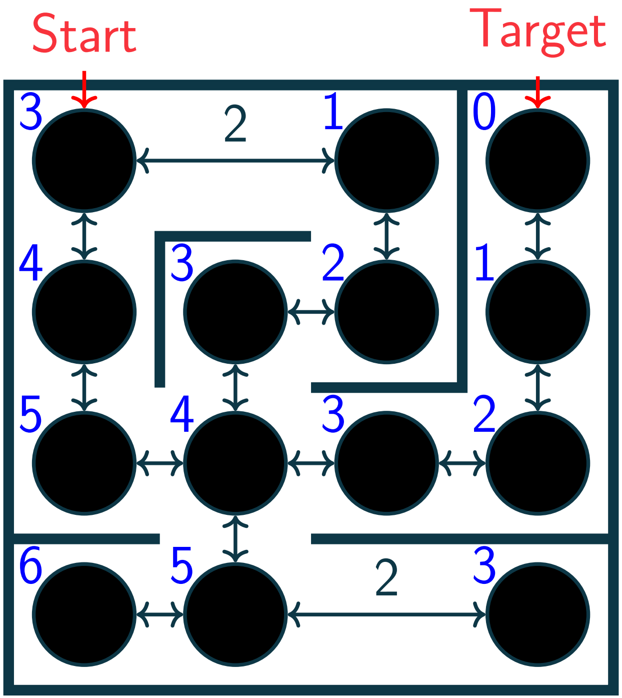
Closed nodes:
Nodes that were explored

Explore node with
lowest score in frontier

Greedy Best-First Search:
Score = Distance to Target

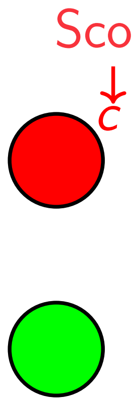
Directed Search Algorithms

A*



Heuristic estimation needed!

Heuristic here: Grid-distance



Frontier nodes:
Next nodes to explore

Closed nodes:
Nodes that were explored

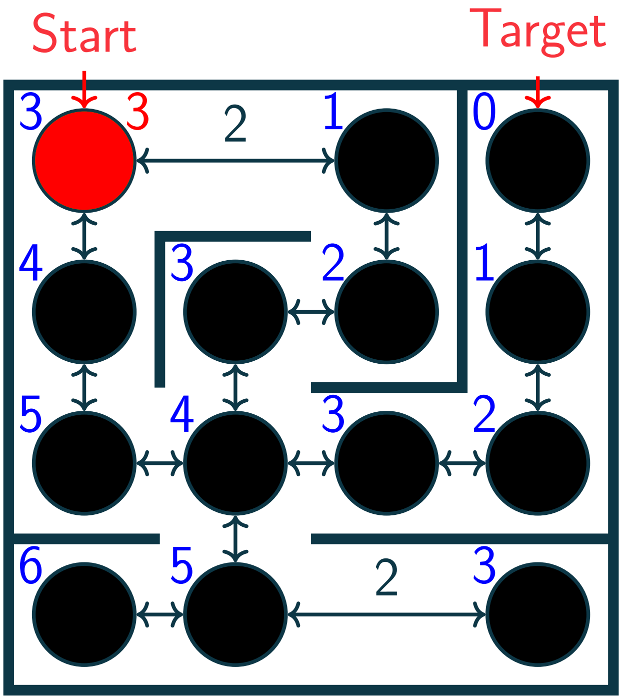
Explore node with
lowest score in frontier

A*:

Score = Distance from Start + Distance to Target

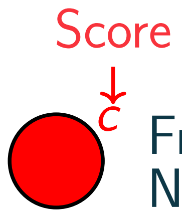
Directed Search Algorithms

A^*

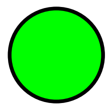


Heuristic estimation needed!

Heuristic here: Grid-distance



Frontier nodes:
Next nodes to explore



Closed nodes:
Nodes that were explored

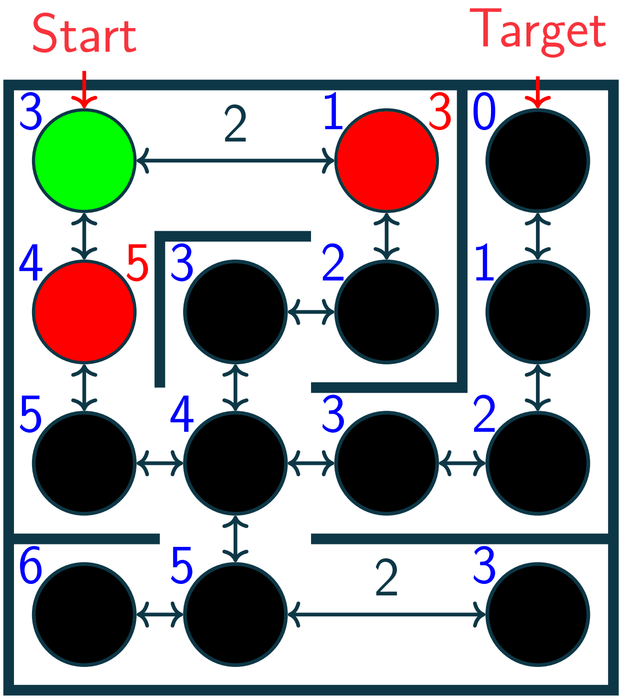
Explore node with lowest score in frontier

A^*:

Score = Distance from Start + Distance to Target

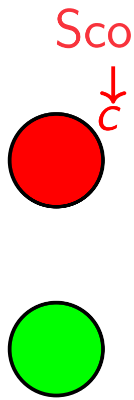
Directed Search Algorithms

A*



Heuristic estimation needed!

Heuristic here: Grid-distance



Frontier nodes:
Next nodes to explore

Closed nodes:
Nodes that were explored

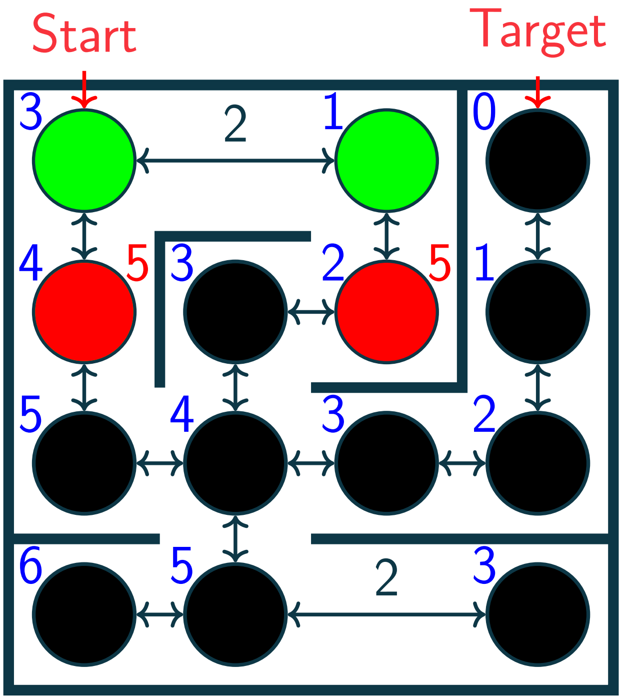
Explore node with
lowest score in frontier

A*:

Score = Distance from Start + Distance to Target

Directed Search Algorithms

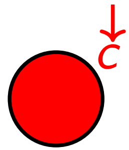
A*



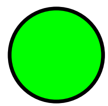
Heuristic estimation needed!

Heuristic here: Grid-distance

Score



Frontier nodes:
Next nodes to explore



Closed nodes:
Nodes that were explored

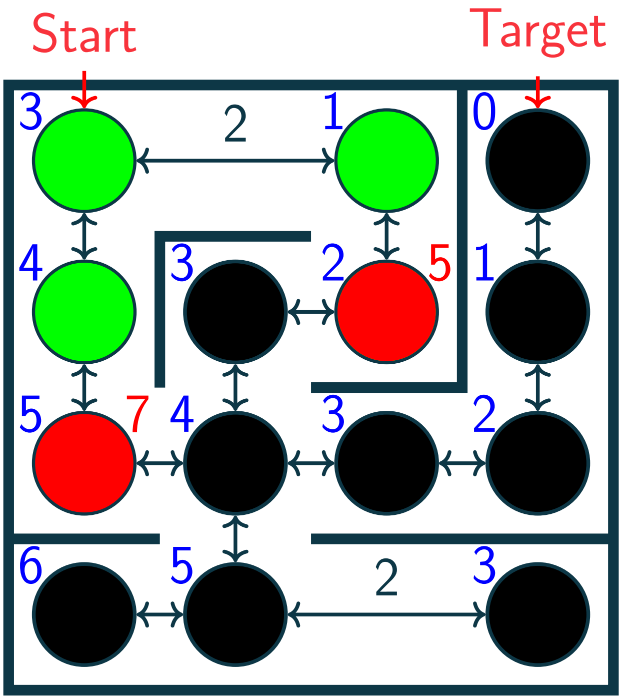
Explore node with
lowest score in frontier

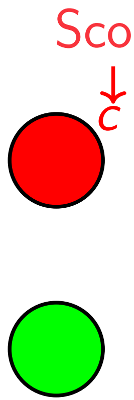
A*:

Score = Distance from Start + Distance to Target

Directed Search Algorithms

A*





Frontier nodes:
Next nodes to explore

Closed nodes:
Nodes that were explored

Heuristic estimation needed!

Heuristic here: Grid-distance

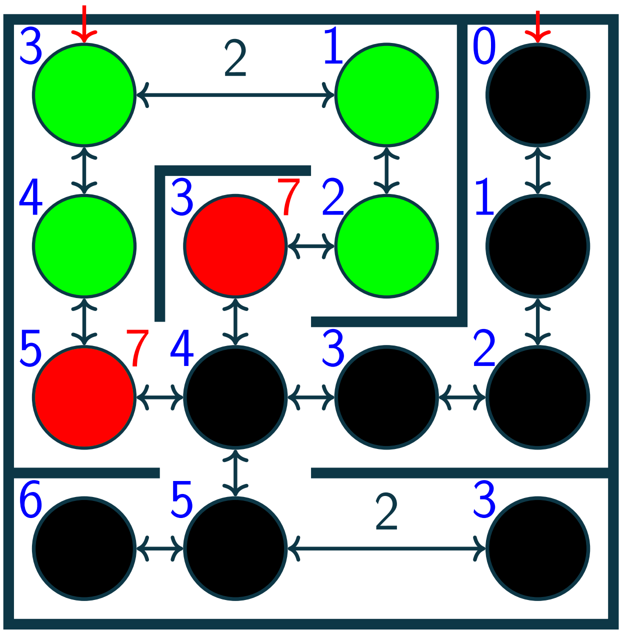
Explore node with lowest score in frontier

A*:

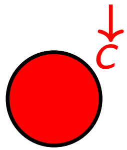
Score = Distance from Start + Distance to Target

Directed Search Algorithms

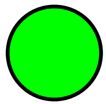
A*



Score



Frontier nodes:
Next nodes to explore



Closed nodes:
Nodes that were explored

Explore node with
lowest score in frontier

A*:

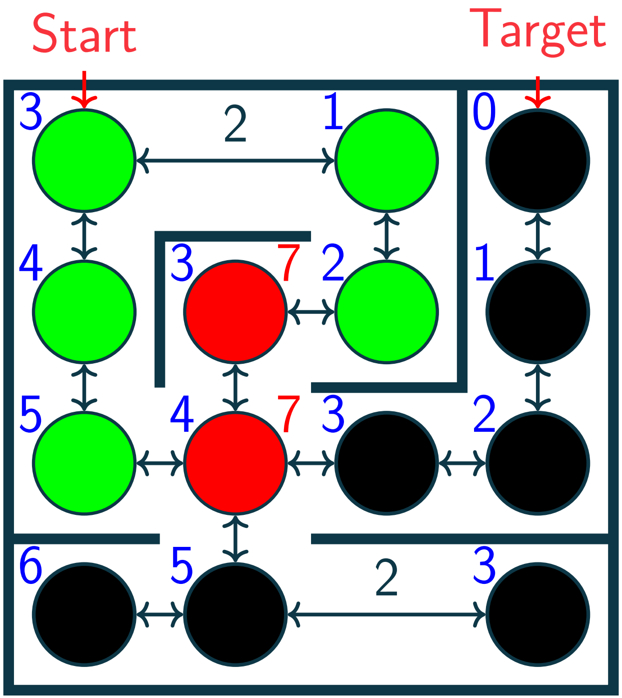
Score = Distance from Start +
Distance to Target

Heuristic estimation needed!

Heuristic here: Grid-distance

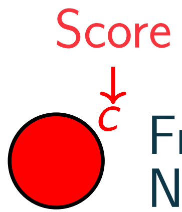
Directed Search Algorithms

A*

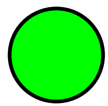


Heuristic estimation needed!

Heuristic here: Grid-distance



Frontier nodes:
Next nodes to explore



Closed nodes:
Nodes that were explored

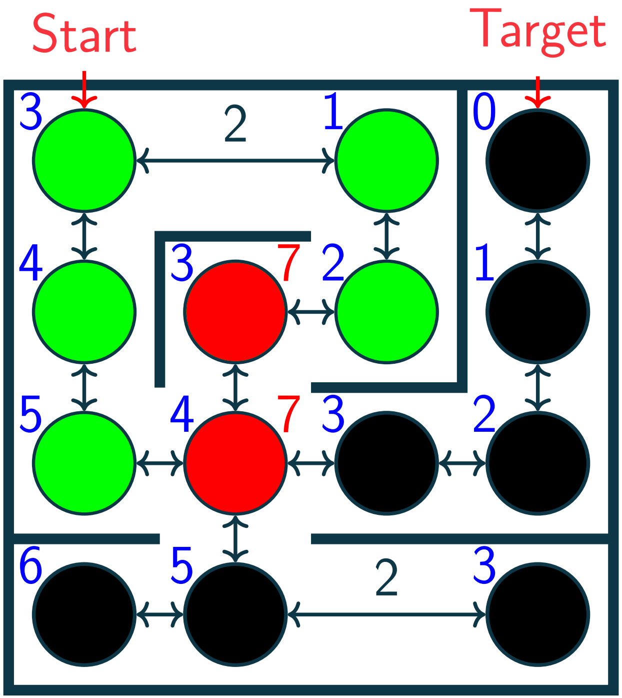
Explore node with
lowest score in frontier

A*:

Score = Distance from Start + Distance to Target

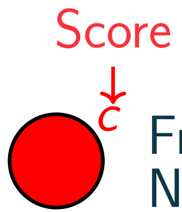
Directed Search Algorithms

A*

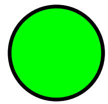


Heuristic estimation needed!

Heuristic here: Grid-distance



Frontier nodes:
Next nodes to explore



Closed nodes:
Nodes that were explored

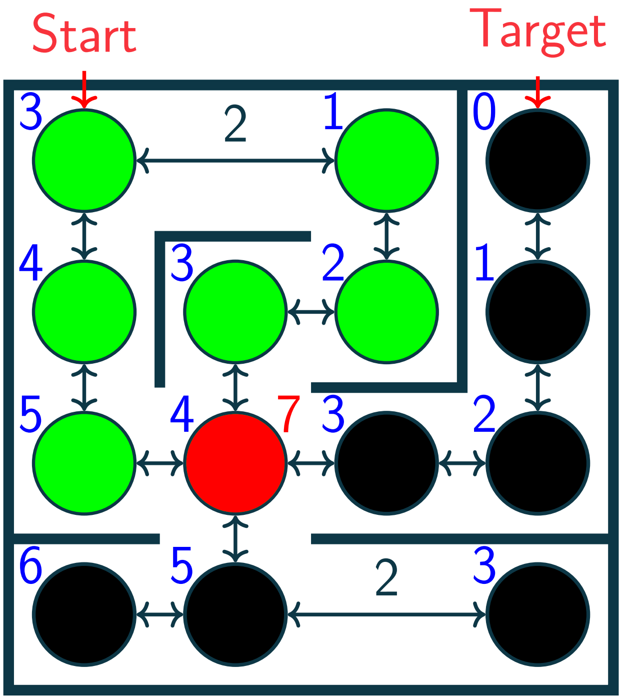
Explore node with lowest score in frontier

A*:

Score $=$ Distance from Start $+$ Distance to Target

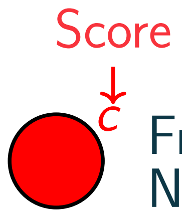
Directed Search Algorithms

A*

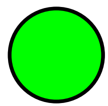


Heuristic estimation needed!

Heuristic here: Grid-distance



Frontier nodes:
Next nodes to explore



Closed nodes:
Nodes that were explored

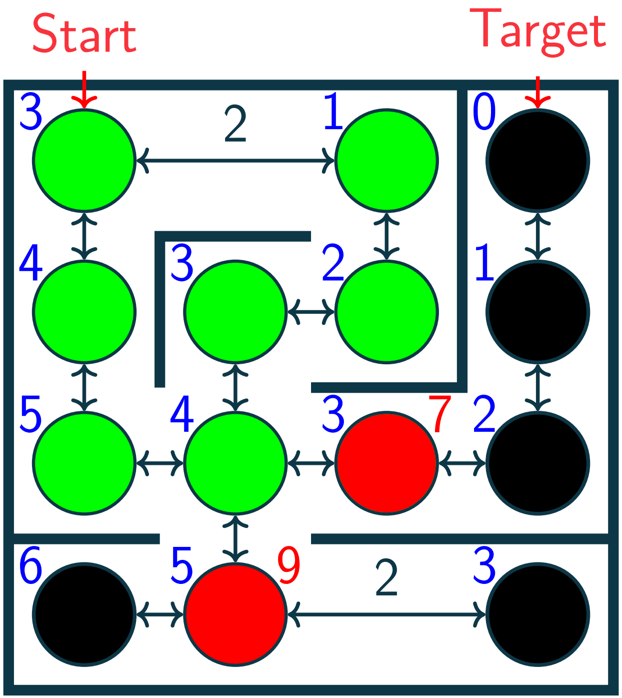
Explore node with
lowest score in frontier

A*:

Score = Distance from Start + Distance to Target

Directed Search Algorithms

A*



Frontier nodes:
Next nodes to explore

Closed nodes:
Nodes that were explored

Heuristic estimation needed!

Heuristic here: Grid-distance

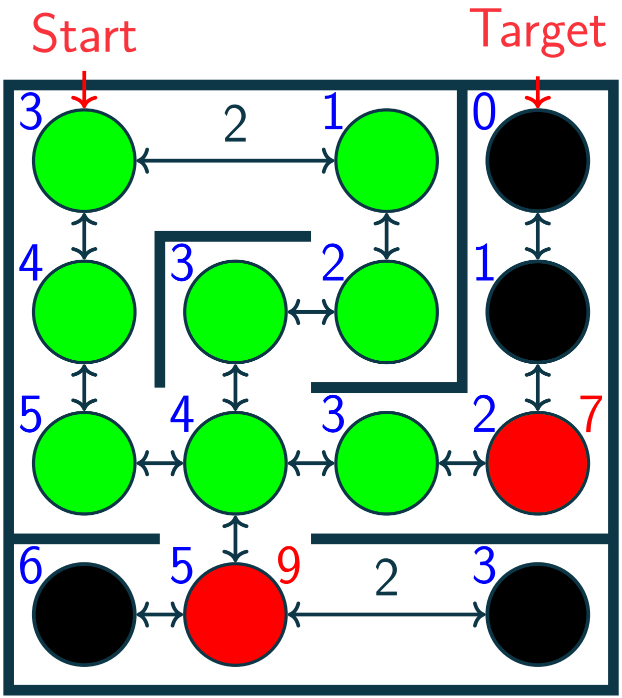
Explore node with
lowest score in frontier

A*:

Score = Distance from Start + Distance to Target

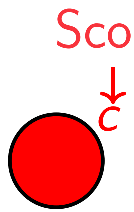
Directed Search Algorithms

A*

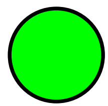


Heuristic estimation needed!

Heuristic here: Grid-distance



Frontier nodes:
Next nodes to explore



Closed nodes:
Nodes that were explored

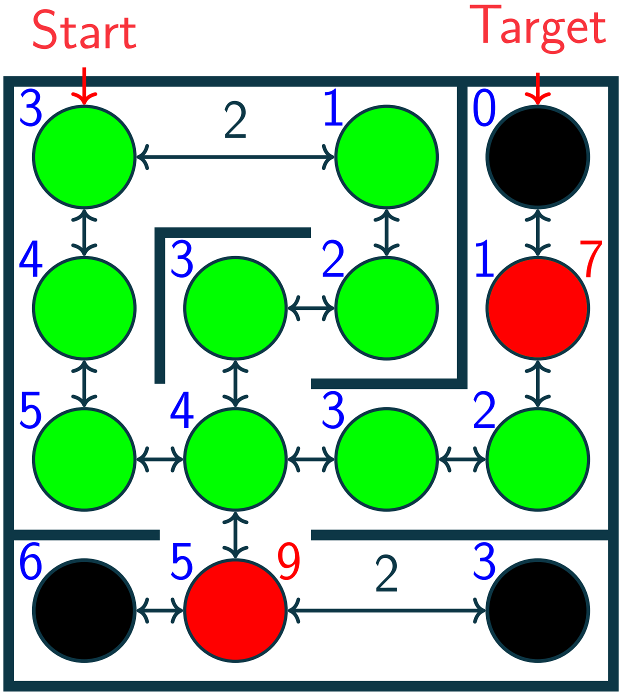
Explore node with
lowest score in frontier

A*:

Score = Distance from Start + Distance to Target

Directed Search Algorithms

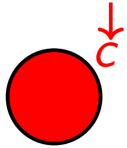
A*



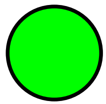
Heuristic estimation needed!

Heuristic here: Grid-distance

Score



Frontier nodes:
Next nodes to explore



Closed nodes:
Nodes that were explored

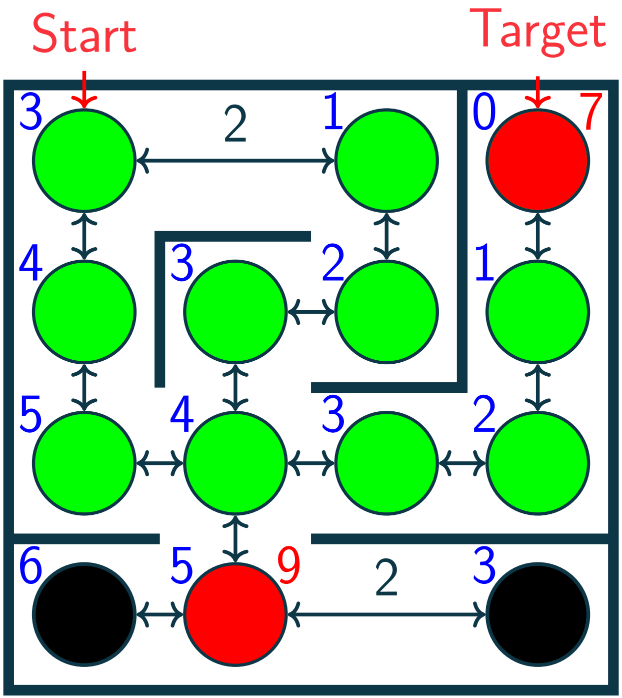
Explore node with
lowest score in frontier

A*:

Score = Distance from Start + Distance to Target

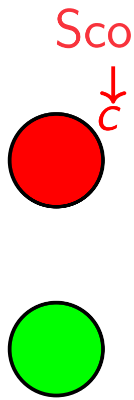
Directed Search Algorithms

A*



Heuristic estimation needed!

Heuristic here: Grid-distance



Frontier nodes:
Next nodes to explore

Closed nodes:
Nodes that were explored

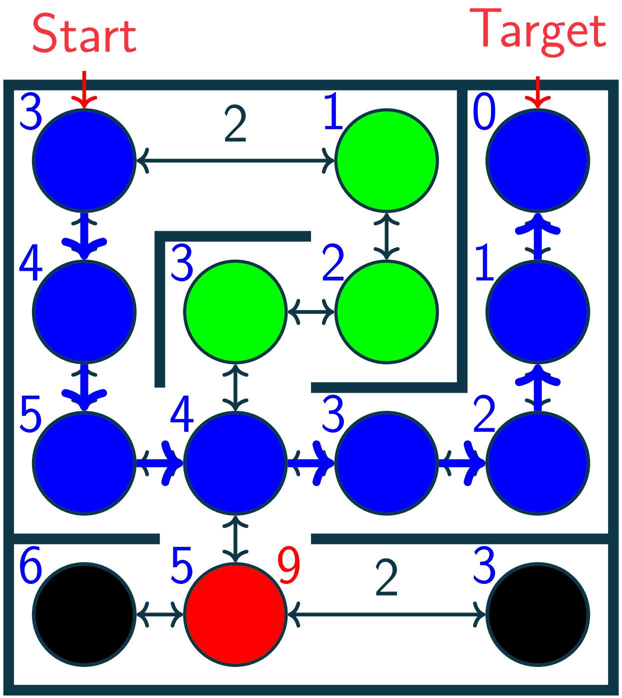
Explore node with
lowest score in frontier

A*:

Score = Distance from Start + Distance to Target

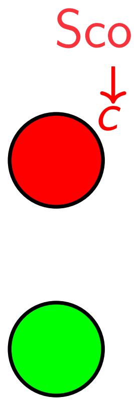
Directed Search Algorithms

A*



Heuristic estimation needed!

Heuristic here: Grid-distance



Frontier nodes:
Next nodes to explore

Closed nodes:
Nodes that were explored

Explore node with
lowest score in frontier

A*:

Score = Distance from Start +
Distance to Target

Directed Search Algorithms

Guarantees on infinite graphs (with finite branching)

If the target is reachable what guarantees do we have, depending on the heuristic h?

Directed Search Algorithms

Guarantees on infinite graphs (with finite branching)

If the target is reachable what guarantees do we have, depending on the heuristic h?

	Termination	Shortest Path
Dijkstra	✔	✔
GBFS	✖	✖
A*	✔	Admissible, consistent h

Admissible: h never overestimates distance to target

Consistent: If b is a successor of a, then
$$h(b) \geq h(a) - \underbrace{c(a, b)}_{\text{cost to reach b from a}}$$

Directed Search Algorithms

Guarantees on infinite graphs (with finite branching)

If the target is reachable what guarantees do we have, depending on the heuristic h?

	Termination	Shortest Path
Dijkstra	✔	✔
GBFS	✖	✖
A*	✔	Admissible, consistent h

Admissible: h never overestimates distance to target

Consistent: If b is a successor of a, then
$$h(b) \geq h(a) - c(a, b)$$

$\uparrow$ cost to reach b from a

$$\begin{tikzcd}
 & b \arrow[dr, "h(b)"] & \\
a \arrow[ur, "c(a,b)"] \arrow[rr, "h(a)"'] & & \text{target}
\end{tikzcd}$$

Directed Search Algorithms

Guarantees on infinite graphs (with finite branching)

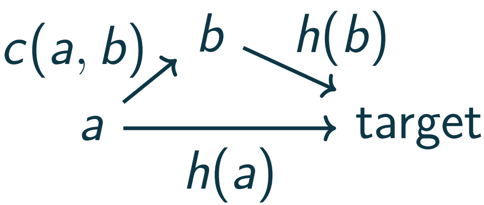
If the target is reachable what guarantees do we have, depending on the heuristic h?

	Termination	Shortest Path
Dijkstra	✔	✔
GBFS	✖	✖
A*	✔	Admissible, consistent h

Admissible: h never overestimates distance to target

Consistent: If b is a successor of a, then
$$h(b) \geq h(a) - c(a, b)$$

↑
cost to reach b from a



$\boxed{\text{Can we get better guarantees for GBFS?}}$

Termination for GBFS

Observation: Result from this work

GBFS terminates on infinite graphs with a reachable target
if the employed heuristic is unbounded

Termination for GBFS

Observation: Result from this work

GBFS terminates on infinite graphs with a reachable target
if the employed heuristic is unbounded

Unbounded heuristic:
On any infinite simple path, the heuristic reaches arbitrarily large heuristic scores

Termination for GBFS

Observation: Result from this work

GBFS terminates on infinite graphs with a reachable target
if the employed heuristic is unbounded

Unbounded heuristic:
On any infinite simple path, the heuristic reaches arbitrarily large heuristic scores

Idea: GBFS cannot follow any infinite path forever without making progress towards the target

From Overapproximations to Heuristics

How do we obtain heuristics from overapproximations?

Shortest path in approx. $\leq$ Shortest path in Petri net

Overapproximations only allow more behaviours!

From Overapproximations to Heuristics

How do we obtain heuristics from overapproximations?

Shortest path in approx. $\leq$ Shortest path in Petri net

Overapproximations only allow more behaviours!

Define h_{approx}: For frontier node m, $h_{approx}(m)$ is the length of the shortest path from m to *target* in the overapproximation *approx*

From Overapproximations to Heuristics

How do we obtain heuristics from overapproximations?

Shortest path in approx. $\leq$ Shortest path in Petri net

Overapproximations only allow more behaviours!

Define h_{approx}: For frontier node m, $h_{approx}(m)$ is the length of the shortest path from m to *target* in the overapproximation *approx*

Observation: Result from this work

For any Petri net reachability overapproximation *approx*, h_{approx} is unbounded, admissible and consistent!

Applying Directed Search to Petri Nets

- Key insight: Modern ILP/SMT solvers allow computing shortest paths for reachability overapproximations fast
  $\Rightarrow$ ILP/SMT allow optimization of solutions

- Directed search based on reachability overapproximations gives formal guarantees:
  Shortest path (A*), Termination (GBFS)

- Highly efficient in practice $\Rightarrow$ rest of the talk

Part IV

Experimental Results

Experimental Results

Prototype implemented in C#,
Gurobi for (integer) linear programming,
Z3 for SAT/SMT

Experimental Results

Prototype implemented in $\mathrm{C}\#$,
Gurobi for (integer) linear programming,
Z3 for SAT/SMT

Reachability benchmarks: program synthesis, random walks on nets from program analysis

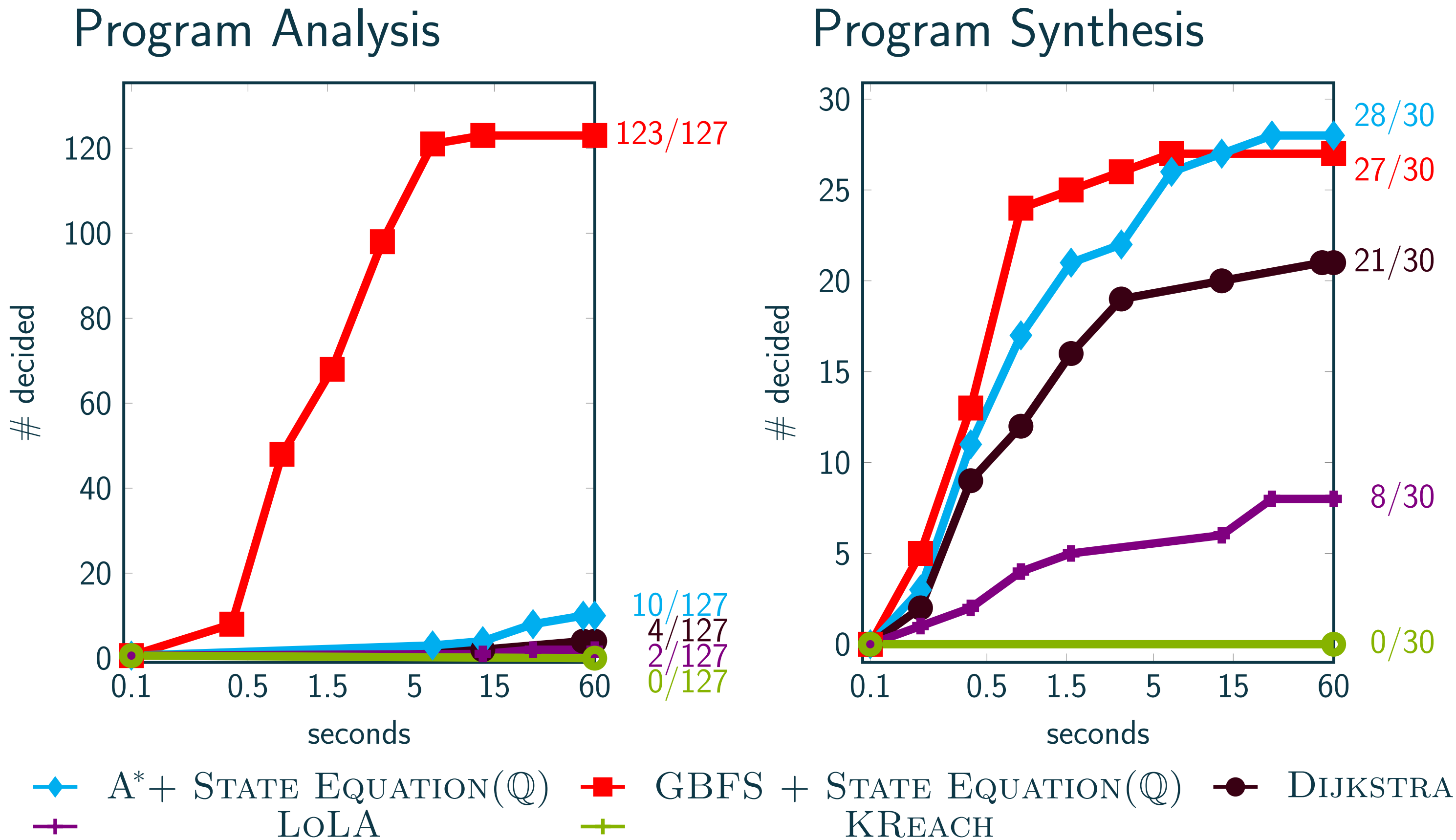
Experimental Results

Prototype implemented in C#,
Gurobi for (integer) linear programming,
Z3 for SAT/SMT

Reachability benchmarks: program synthesis,
random walks on nets from program analysis

Focus is on reachable instances
(but exploratory results confirm known effectiveness of approximations for unreachable instances)

Experimental Results: Reachability

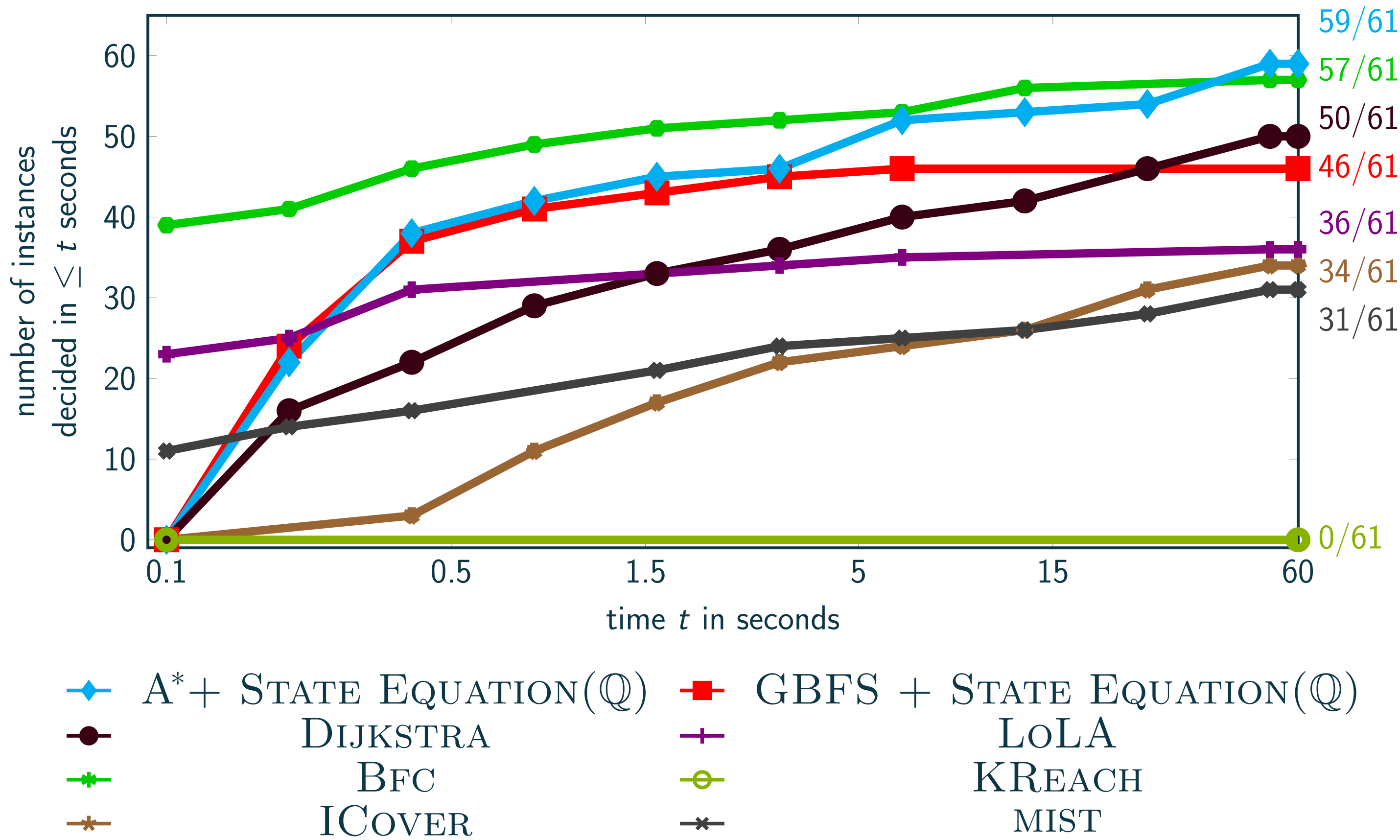


Guided Search outperforms existing approaches (by orders of magnitude)

Experimental Results: Coverability

Even competitive against dedicated coverability solvers

Program Analysis



Experimental Results

What about the other approximations?

- State Equation over $\mathbb{N}$ slightly worse performance than over $\mathbb{Q}$

Experimental Results

What about the other approximations?

- State Equation over $\mathbb{N}$ slightly worse performance than over $\mathbb{Q}$
- Continuous Petri Nets are much slower (Gurobi vs Z3), extra accuracy does not outweigh slowdown

Experimental Results

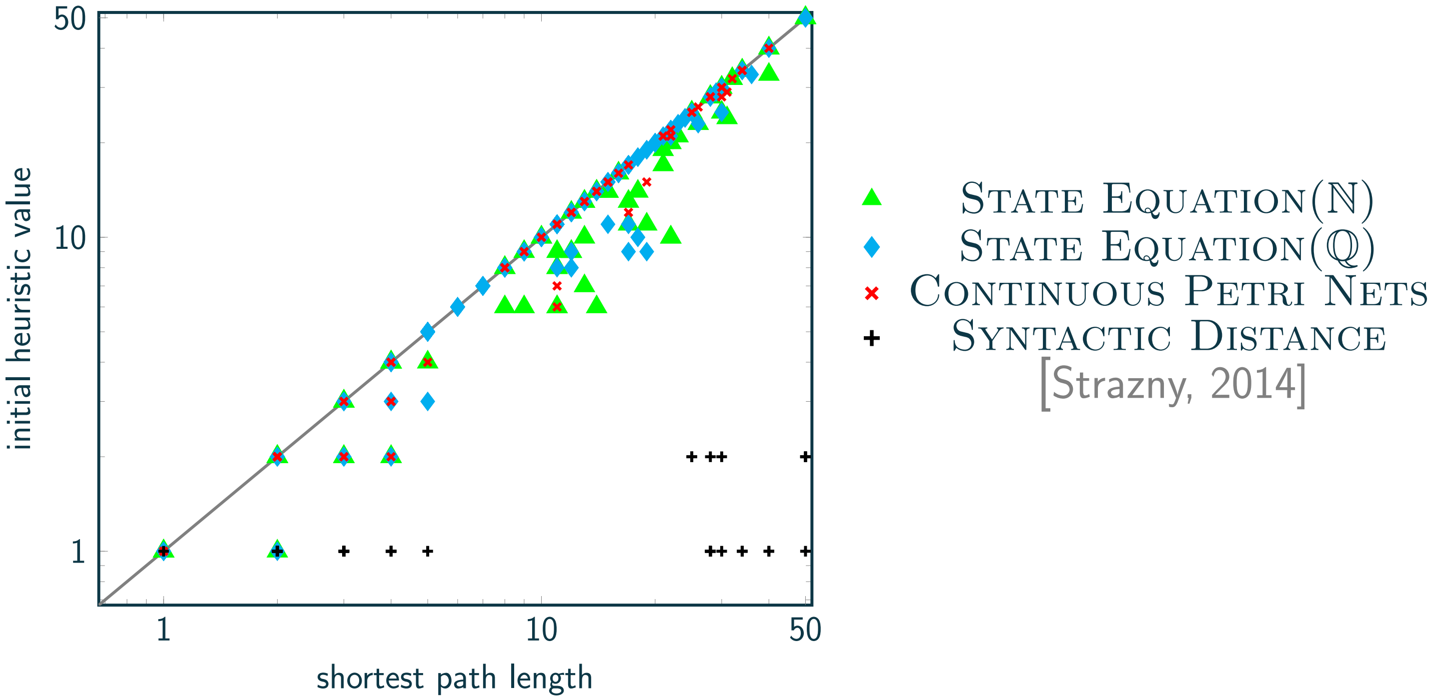
What about the other approximations?

- State Equation over $\mathbb{N}$ slightly worse performance than over $\mathbb{Q}$
- Continuous Petri Nets are much slower (Gurobi vs Z3), extra accuracy does not outweigh slowdown
- Much more accurate than previously considered ad-hoc heuristics

Experimental Results

What about the other approximations?

- State Equation over $\mathbb{N}$ slightly worse performance than over $\mathbb{Q}$
- Continuous Petri Nets are much slower (Gurobi vs Z3), extra accuracy does not outweigh slowdown
- Much more accurate than previously considered ad-hoc heuristics

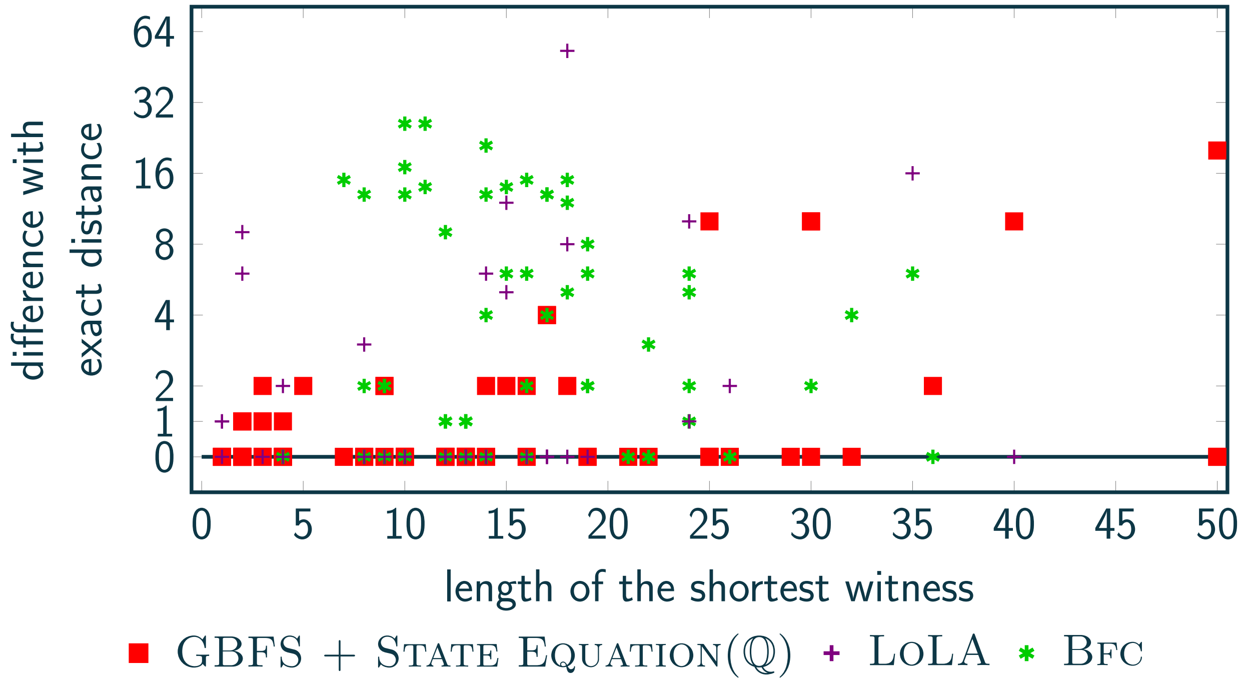


Experimental Results: Witness Length

- Caveat: Some approaches do not guarantee shortest paths

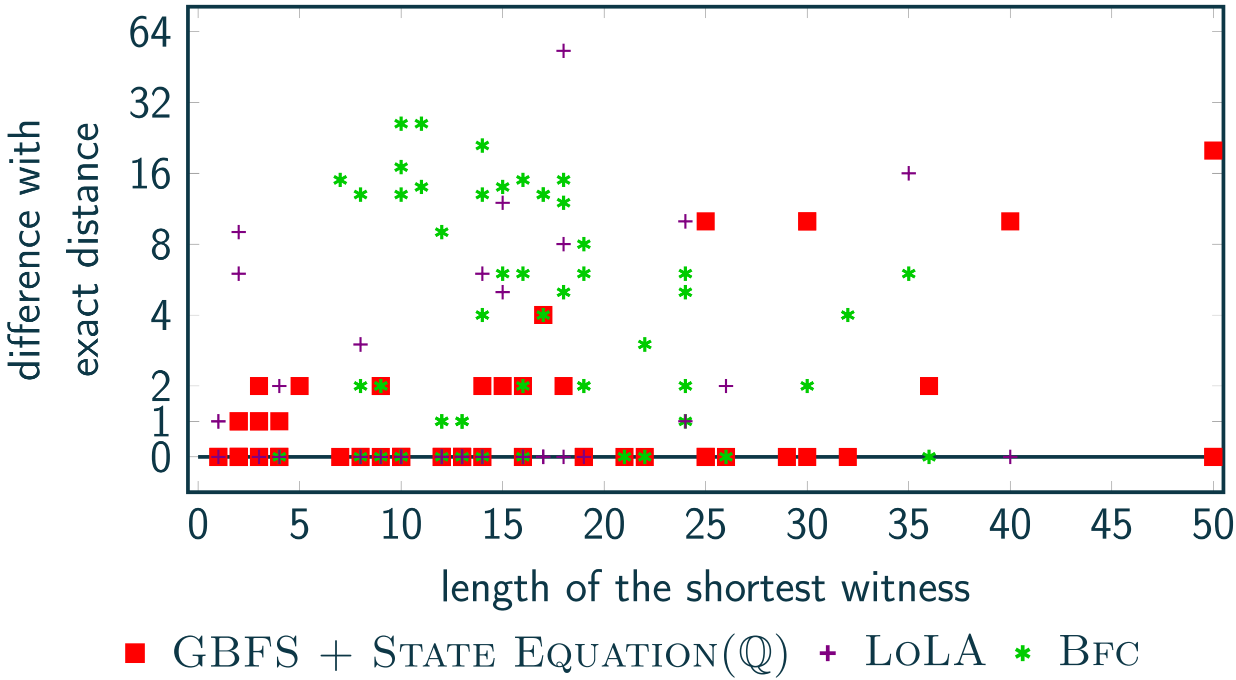
Experimental Results: Witness Length

- Caveat: Some approaches do not guarantee shortest paths



Experimental Results: Witness Length

- Caveat: Some approaches do not guarantee shortest paths



- A* is competitive against Bfc and much faster than LoLA while guaranteeing a shortest path

Conclusion

Challenge:

No practically efficient semi-procedures for showing reachability in infinite-state systems

Conclusion

Challenge:

No practically efficient semi-procedures for showing reachability in infinite-state systems

- Provide a tool handling large examples in practice, which outperforms the state-of-the-art by orders of magnitude

Conclusion

Challenge:

No practically efficient semi-procedures for showing reachability in infinite-state systems

- Provide a tool handling large examples in practice, which outperforms the state-of-the-art by orders of magnitude
- Key insight: Petri Nets have easy-to-compute approximations ... that are surprisingly accurate

Conclusion

Challenge:

No practically efficient semi-procedures for showing reachability in infinite-state systems

- Provide a tool handling large examples in practice,
  which outperforms the state-of-the-art by orders of magnitude
- Key insight: Petri Nets have easy-to-compute approximations
  . . . that are surprisingly accurate
- Typically used for showing unreachability. . .
  . . . but we show they can be adapted for directed search

Conclusion

Challenge:

No practically efficient semi-procedures for showing reachability in infinite-state systems

- Provide a tool handling large examples in practice, which outperforms the state-of-the-art by orders of magnitude
- Key insight: Petri Nets have easy-to-compute approximations
  ... that are surprisingly accurate
- Typically used for showing unreachability...
  ... but we show they can be adapted for directed search
- Directed search using these approximations is efficient
  ... even against domain specific solvers for coverability

Outlook

- Extension to directed model checking:
  For example, finding cycles with conditions (for LTL, ...)
- Representing overapproximations concisely
- Classes of Petri Nets with guarantees on heuristic accuracy
- Applying directed search to (undecidable) extensions
  (Transfer nets, reset nets, colored nets, ...)