The Complexity of the Simplex Method

John Fearnley  
Rahul Savani

Department of Computer Science  
University of Liverpool
Steve Smale

**Mathematical problems for the next century**


18 unsolved problems in mathematics
18 unsolved problems in mathematics

Problem 9: Is there a strongly polynomial algorithm for LP?
Steve Smale

**Mathematical problems for the next century**


**18 unsolved problems in mathematics**

**Problem 9:** Is there a *strongly* polynomial algorithm for LP?

- running time depends only on the **dimensions** of the LP
- intermediate numbers grow only polynomially
18 unsolved problems in mathematics

Problem 9: Is there a strongly polynomial algorithm for LP?

- running time depends only on the dimensions of the LP
- intermediate numbers grow only polynomially

Yes, if there is a polynomial simplex pivoting rule
Linear programming

\[
\begin{align*}
\text{max} & \quad x_1 \\
\text{s.t.} & \quad \begin{pmatrix} -1 & 2 \\ 2 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 6 \\ 8 \end{pmatrix} \\
& \quad \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \end{pmatrix}
\end{align*}
\]
Linear programming

\[ \text{max } x_1 \]

s.t.  
\[
\begin{pmatrix}
-1 & 2 \\
2 & 1 \\
2 & -1 \\
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
\end{pmatrix}
\leq
\begin{pmatrix}
6 \\
8 \\
4 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
x_1 \\
x_2 \\
\end{pmatrix}
\geq
\begin{pmatrix}
0 \\
0 \\
\end{pmatrix}
\]
The simplex method

\[ \text{max } x_1 \]

\[ \text{s.t. } \begin{pmatrix} -1 & 2 \\ 2 & 1 \\ 2 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 6 \\ 8 \\ 4 \end{pmatrix} \]

\[ \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \end{pmatrix} \]

**Input:** Linear program + starting vertex of the feasible region
The simplex method

max \quad x_1

s.t. \quad \begin{pmatrix} -1 & 2 \\ 2 & 1 \\ 2 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 6 \\ 8 \\ 4 \end{pmatrix}

\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \end{pmatrix}

Vertex (basic feasible solution):

**basis** = dimension-many **linearly independent constraints**

**vertex** given by making basis inequalities **binding**
The simplex method

max \quad x_1

s.t. \quad \begin{pmatrix} -1 & 2 \\ 2 & 1 \\ 2 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 6 \\ 8 \\ 4 \end{pmatrix}

\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \end{pmatrix}

Pivot step:
Traverse edge so as to improve objective function value
One variable enters the basis and another leaves
The simplex method

max \ x_1

s.t. \ 
\begin{pmatrix} -1 & 2 \\ 2 & 1 \\ 2 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 6 \\ 8 \\ 4 \end{pmatrix}

\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \end{pmatrix}

Pivot step:
Choice of entering and leaving variables typically **not unique**
They are chosen according to a **pivot rule**
Largest Coefficient (Dantzig’s pivot rule)
pick entering variable to maximize improvement of objective per unit increase of the entering variable

Largest Increase
pick entering variable that gives largest absolute improvement of the objective

Steepest Edge
choose edge closest in direction to the objective’s direction
Pivot rules

**Largest Coefficient** *(Dantzig’s pivot rule)*

pick entering variable to maximize improvement of objective per unit increase of the entering variable

**Largest Increase**

pick entering variable that gives largest absolute improvement of the objective

**Steepest Edge**

choose edge closest in direction to the objective’s direction

**Next:** introduce the *equational form* in order to understand Dantzig’s pivot rule
Data: $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$

$$\begin{align*}
\max & \quad c^\top x \\
\text{s.t.} & \quad Ax = b \\
& \quad x \geq 0
\end{align*}$$
Basic feasible solution

Data: \( c \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m \)

\[
\begin{align*}
\max & \quad c^\top x \\
\text{s.t.} & \quad Ax = b \\
& \quad x \geq 0
\end{align*}
\]

Let \( B \subset [n] \) with \( |B| = m \), \( N = [n] \setminus B \)
Data: $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$

$$\begin{align*}
\max & \quad c^\top x \\
\text{s.t.} & \quad Ax = b \\
& \quad x \geq 0
\end{align*}$$

Let $B \subset [n]$ with $|B| = m$, $N = [n] \setminus B$

If columns of $A_B$ are **linearly independent** $B$ is a **basis**

$$x_B := A_B^{-1}b \quad \text{and} \quad x_N := 0$$

$x$ is a **basic feasible solution** if $x_B \geq 0$
Consider dual linear program

\[ \begin{align*}
\text{max} & \quad c^T x \\
\text{s.t.} & \quad A x = b \\
\end{align*} \]

\[ \begin{align*}
\text{min} & \quad b^T y \\
\text{s.t.} & \quad A^T y \geq c \\
\end{align*} \]

\[ x \geq 0 \]
Consider dual linear program

\[
\begin{align*}
\max & \quad c^T x \\
\text{s.t.} & \quad Ax = b \\
\min & \quad b^T y \\
\text{s.t.} & \quad A^T y \geq c \\
& \quad x \geq 0
\end{align*}
\]

Each basis $B$ defines a dual solution:

\[y = ((A_B)^{-1})^T c_B\]

If dual solution is feasible, i.e., $A^T y \geq c$, basis is optimal
Consider dual linear program

$$\begin{align*}
\text{max} & \quad c^T x \\
\text{s.t.} & \quad Ax = b & \text{min} & \quad b^T y \\
& \quad x \geq 0 & \text{s.t.} & \quad A^T y \geq c
\end{align*}$$

Each basis $B$ defines a dual solution:

$$y = ((A_B)^{-1})^T c_B$$

If dual solution is feasible, i.e., $A^T y \geq c$, basis is optimal

**Definition (Dantzig’s pivot rule)**

Choose variable with largest coefficient $c - A^T y$ to enter basis
Our main results

Given:
- a linear program $\mathcal{L}$
- an initial basic feasible solution $b$
- a variable $v$

**Definition (problems)**
If Dantzig’s pivot rule is applied to $\mathcal{L}$ starting from $b$
- will $v$ ever enter the basis? ($\text{BASISENTRY}$)
- will $v$ be in the final optimal basis? ($\text{DANTZIGLpSol}$)

**Theorem**
$\text{BASISENTRY}$ and $\text{DANTZIGLpSol}$ are $\text{PSPACE}$-complete
Background results

Exponential-time worst-case examples

Klee and Minty (1972) How good is the simplex algorithm? Inequalities III


Hardness results


Disser and Skutella (2015) The simplex algorithm is NP-mighty SODA
How hard is PSPACE?

PSPACE contains every problem that can be solved by a polynomial space algorithm
Our approach

We construct **Markov decision processes (MDPs)** (discrete-time stochastic control problems)
Our approach

We construct Markov decision processes (MDPs) (discrete-time stochastic control problems)

Policy iteration is a well-known method for solving MDPs
Our approach

We construct **Markov decision processes (MDPs)** (discrete-time stochastic control problems)

**Policy iteration** is a well-known method for solving MDPs

We show that, for our MDPs:

A variant of **single-switch policy iteration** is **exactly simulated by** **Dantzig’s pivot rule** applied to a related LP
Our approach

We construct **Markov decision processes (MDPs)** (discrete-time stochastic control problems)

**Policy iteration** is a well-known method for solving MDPs

We show that, for our MDPs:

A variant of **single-switch policy iteration**

is **exactly simulated by**

**Dantzig’s pivot rule** applied to a related LP

**Main result:** This variant of policy iteration is capable of **iterated circuit evaluation**
Related work: MDPs and LPs

Friedmann, Hansen, and Zwick (2011) Subexponential lower bounds for randomized pivoting rules for solving linear programs *STOC*

Ye (2011) The simplex and policy-iteration methods are strongly polynomial for the Markov decision problem with a fixed discount rate *Mathematics of Operations Research*

Post and Ye (2013) The simplex method is strongly polynomial for deterministic Markov decision processes *SODA*

Hansen, Kaplan, and Zwick (2014) Dantzig’s pivoting rule for shortest paths, deterministic MDPs, and minimum cost to time ratio cycles *SODA*
Roadmap

Markov Decision Processes (MDPs)
- Policy iteration
- Primal and dual linear programs for MDPs
- Equivalence of policy iteration and Dantzig’s pivot rule

The construction
- Iterated circuit evaluation
- The clock: exponential-time MDP examples
- Using the clock to drive iterated circuit evaluation
Markov decision process

- **decision states** (squares) with **actions** (arcs) with corresponding **rewards** (arc weights)

- **chance states** (circles) with **probabilistic transitions** ((0.5, 0.5) in the example) to other states

![Diagram](image-url)
Deterministic policy

Definition (deterministic policy)

A policy $\sigma$ prescribes an $a \in \text{actions}(s)$ for every $s \in \text{states}$.
Maximize total expected reward

Definition (Total expected reward equations)

\[
\text{Val}^\sigma(s) = \text{reward}(\sigma(s)) + \sum_{s' \in \text{states}} \text{prob}(\sigma(s) \to s') \cdot \text{Val}^\sigma(s')
\]
Maximize total expected reward

Definition (Total expected reward equations)

\[ \text{Val}^{\sigma}(s) = \text{reward}(\sigma(s)) + \sum_{s' \in \text{states}} \text{prob}(\sigma(s) \rightarrow s') \cdot \text{Val}^{\sigma}(s') \]
Maximize total expected reward

**Definition (Total expected reward equations)**

\[
Val^\sigma(s) = \text{reward}(\sigma(s)) + \sum_{s' \in \text{states}} \text{prob}(\sigma(s) \rightarrow s') \cdot Val^\sigma(s')
\]

![Diagram](attachment:image.png)
Maximize total expected reward

**Definition (Total expected reward equations)**

\[ \text{Val}^\sigma(s) = \text{reward}(\sigma(s)) + \sum_{s' \in \text{states}} \text{prob}(\sigma(s) \rightarrow s') \cdot \text{Val}^\sigma(s') \]

\[ \text{Val}(0) = 0 \]
\[ \text{Val}(1) = 1 \quad (\text{Val}(1') = 4) \]
\[ \text{Val}(2) = 0 \quad (\text{Val}(2') = 2) \]
\[ \text{Val}(3) = 1 \quad (\text{Val}(3') = 1) \]

---

**Diagram:**
- States: 0, 1, 2, 3, 3', 2', 1', 0, si, si'
- Edges and their labels:
  - 3 → 0: 0
  - 3 → 2: 0
  - 3 → 1: 0
  - 2 → 0: 0
  - 2 → 1: 0
  - 2 → 1': 0
  - 1 → 0: 0
  - 1 → 1': 0
  - 0 → si: 0
  - 0 → si': 8
- Valuations:
  - Val(0) = 0
  - Val(1) = 1
  - Val(1') = 4
  - Val(2) = 0
  - Val(2') = 2
  - Val(3) = 1
  - Val(3') = 1

---
Optimality equations

Definition (Optimality equations)

\[
\text{Val}(s) = \max_{a \in \text{actions}(s)} \left( \text{reward}(a) + \sum_{s' \in \text{states}} \text{prob}(a \to s') \cdot \text{Val}(s') \right)
\]
Appeal of an action

Definition (Appeal$^\sigma(a)$)

\[
\left(\text{reward}(a) + \sum_{s' \in \text{states}} \text{prob}(a \rightarrow s') \cdot \text{Val}^\sigma(s')\right) - \text{Val}^\sigma(s)
\]

Val(2) = 1  Val(2) = 0  Val(1) = 0  Val(0) = 0

Val(3) = 1  Val(2) = 2  Val(1) = 4

Appeal = 4
Definition (Appeal$^\sigma(a)$)

\[
\left(\text{reward}(a) + \sum_{s' \in \text{states}} \text{prob}(a \rightarrow s') \cdot \text{Val}^\sigma(s')\right) - \text{Val}^\sigma(s)
\]
Appeal of an action

Definition ($\text{Appeal}^\sigma(a)$)

\[
\left(\text{reward}(a) + \sum_{s' \in \text{states}} \text{prob}(a \rightarrow s') \cdot \text{Val}^\sigma(s')\right) - \text{Val}^\sigma(s)
\]

Val(2) = 4  Val(2) = 4  Val(1) = 4  Val(0) = 0

- 3
  - 0
  - 0
  - 0

- 2
  - 0
  - 0
  - 0

- 1
  - 0
  - 0
  - 0

- 0
  - 0
  - 0
  - 0

Val(3') = 3  Val(2') = 2  Val(1') = 4
Policy iteration

**Definition (switchable action)**

action $a$ is **switchable** under $\sigma$ if and only if $\text{Appeal}^\sigma(a) > 0$

---

**Algorithm 1** generic policy iteration

```plaintext
while the set $A$ of switchable actions is non-empty do
    switch a non-empty subset of $A$
end while
```

**single-switch** policy iteration $\leftrightarrow$ simplex method
Policy iteration

Definition (switchable action)

action $a$ is switchable under $\sigma$ if and only if $\text{Appeal}^\sigma(a) > 0$

Algorithm 4 Dantzig’s switching rule

while the set $A$ of switchable actions is non-empty do

switch the action with highest appeal

end while

single-switch policy iteration $\leftrightarrow$ simplex method
Definition (Appeal$^\sigma(a)$)

$$\left(\text{reward}(a) + \sum_{s' \in \text{states}} \text{prob}(a \rightarrow s') \cdot \text{Val}^\sigma(s')\right) - \text{Val}^\sigma(s)$$
Definition (Appeal$^\sigma(a)$)

\[
\left( \text{reward}(a) + \sum_{s' \in \text{states}} \text{prob}(a \rightarrow s') \cdot \text{Val}^\sigma(s') \right) - \text{Val}^\sigma(s)
\]
Given:
- a Markov decision process $\mathcal{M}$
- a starting policy $\sigma$
- an action $a$

**Definition (problems)**
If Dantzig’s switching rule is applied to $\mathcal{M}$ starting from $\sigma$
- will $a$ ever be switched? ($\text{ACTIONSwitch}$)
- will $a$ be used by the final optimal policy? ($\text{DANTZIGMdpSol}$)

**Theorem**
$\text{ACTIONSwitch}$ and $\text{DANTZIGMdpSol}$ are $\text{PSPACE}$-complete
Value LP

minimize $\frac{1}{|\text{states}|} \sum_{s \in \text{states}} val(s)$

s.t. $val(s) \geq reward(a) + \sum_{s' \in \text{states}} \text{prob}(a \rightarrow s') \cdot val(s')$

$\forall s \in \text{states},$
$\forall a \in \text{actions}(s)$
Apply simplex to dual of Value LP

\[
\begin{align*}
\max & \quad \sum_{a \in \text{actions}} \text{reward}(a) \cdot x_a \\
\text{s.t.} & \quad \sum_{a \in \text{actions}(j)} x_a - \sum_{s \in \text{states}} \sum_{a \in \text{actions}(s)} \text{prob}(a \to s) \cdot x_a = \frac{1}{|\text{states}|} \\
& \quad \forall j \in \text{states} \\
& \quad x_a \geq 0 \\
& \quad \forall a \in \text{actions}
\end{align*}
\]
Apply simplex to dual of Value LP

\[
\begin{align*}
\text{max} \quad & \sum_{a \in \text{actions}} \text{reward}(a) \cdot x_a \\
\text{s.t.} \quad & \sum_{a \in \text{actions}(j)} x_a - \sum_{s \in \text{states}} \sum_{a \in \text{actions}(s)} \text{prob}(a \to s) \cdot x_a = \frac{1}{|\text{states}|} \\
& \forall j \in \text{states} \\
& x_a \geq 0 \\
& \forall a \in \text{actions} \\
\end{align*}
\]

\[x_a = E[\# \text{ of times } a \text{ will be used}]\]  
(\text{starting from a state chosen uniformly at random})
Recall Dantzig’s pivot rule

\[
\begin{align*}
\text{max} \quad & c^\top x \\
\text{s.t.} \quad & Ax = b \\
\text{min} \quad & b^\top y \\
\text{s.t.} \quad & A^\top y \geq c \\
\end{align*}
\]

\[x \geq 0\]

Each basis \(B\) defines a dual solution:

\[y = ((A_B)^{-1})^\top c_B\]

If dual solution is feasible, i.e., \(A^\top y \geq c\), basis is optimal

**Definition (Dantzig’s pivot rule)**

Choose variable with largest coefficient \(c - A^\top y\) to enter basis
Value LP

\[
\text{minimize } \frac{1}{|\text{states}|} \sum_{s \in \text{states}} \text{val}(s)
\]

\[
\text{s.t. } \text{val}(s) \geq \text{reward}(a) + \sum_{s' \in \text{states}} \text{prob}(a \to s') \cdot \text{val}(s')
\]

\[
\forall s \in \text{states},
\forall a \in \text{actions}(s)
\]
Equivalence

Reduced costs of Dual LP = appeals of actions, so:

**Policy iteration** with greedy single switch

is exactly simulated by

**Dantzig’s pivot rule** applied to the Dual LP.
Roadmap

Markov Decision Processes (MDPs)
- Policy iteration
- Primal and dual linear programs for MDPs
- Equivalence of policy iteration and Dantzig’s pivot rule

The construction
- Iterated circuit evaluation
- The clock: exponential-time MDP examples
- Using the clock to drive iterated circuit evaluation
Given

- $F : \{0, 1\}^n \mapsto \{0, 1\}^n$ encoded by boolean circuit
- input bitstring $B \in \{0, 1\}^n$; bit $z \in \{1, \ldots, n\}$

Definition (two starting problems for reductions)

- Is the $z$th bit of $F^{2^n}(B)$ equal to 1?
- Is the $z$th bit of $F^i(B)$ equal to 1 for some even $i \leq 2^n$?
Circuit iteration

Given

- $F : \{0, 1\}^n \mapsto \{0, 1\}^n$ encoded by boolean circuit
- input bitstring $B \in \{0, 1\}^n$; bit $z \in \{1, \ldots, n\}$

Definition (two starting problems for reductions)

- Is the $z$th bit of $F^{2^n}(B)$ equal to 1?
- Is the $z$th bit of $F^i(B)$ equal to 1 for some even $i \leq 2^n$?

These problems are **PSPACE-complete** since $F$ can simulate one iteration of a **space-bounded Turing machine**
Construction overview

Circuit 1

Circuit 0

CLOCK

\[
\begin{cases}
\text{Low} & t \text{ even} \\
\text{High} & t \text{ odd}
\end{cases}
\]

\[
\begin{cases}
\text{High} & t \text{ even} \\
\text{Low} & t \text{ odd}
\end{cases}
\]

Time: \( t \in \{0, 1, \ldots, 2^n - 1\} \)
The need for appeal reduction

We need to be able to control when actions are switched. E.g., for the clock to work we need to ensure that we always switch at the leftmost switchable state.
Appeal reduction gadget for action \((s, t)\)

\[
1 - p
\]

\[
p
\]
Appeal reduction gadget for action \((s, t)\)

\((s, t)\) used: gadget has no effect on value of \(s\)
Appeal reduction gadget for action \((s, t)\)

\[(s, t) \text{ used: gadget has no effect on value of } s\]

\[(s, t') \text{ used: use } p \text{ to set appeal of } (s, t) \text{ as low as needed}\]
Reflected binary code (Gray code)

It underlies both:

- exponential-time examples for policy iteration of Condon and Melekopoglou (1994), on which are clock is based; and

- exponential-time examples for simplex of Klee and Minty (1972)

\[
\begin{array}{ccc}
0 & 0 & 1 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
0 & 1 & 1 \\
0 & 1 & 0 \\
1 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 0 \\
\end{array}
\]
Reflective binary code (Gray code)

It underlies both:

- exponential-time examples for policy iteration of Condon and Melekopoglou (1994), on which are clock is based; and

- exponential-time examples for simplex of Klee and Minty (1972)
Reflected binary code (Gray code)

It underlies both:

- exponential-time examples for policy iteration of Condon and Melekopoglou (1994), on which are clock is based; and
- exponential-time examples for simplex of Klee and Minty (1972)

```
 0  0  1
 1  0  1
 1  1  1
 0  1  1
```

```plaintext
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Reflected binary code (Gray code)

It underlies both:

- exponential-time examples for policy iteration of Condon and Melekopoglou (1994), on which are clock is based; and
- exponential-time examples for simplex of Klee and Minty (1972)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
### The clock

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$\sigma(3)$</th>
<th>$\sigma(2)$</th>
<th>$\sigma(1)$</th>
<th>Val($c_0$)</th>
<th>Val($c_1$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The diagram shows the transitions between states $c_0$, $c_1$, $3'$, $2'$, $1'$, $si$, and $si'$. The transitions are labeled with $\alpha_1$, $\alpha_2$, and $\alpha_3$. The equation $T \cdot 2^{3+1}$ is also shown.
# The clock

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$\sigma(3)$</th>
<th>$\sigma(2)$</th>
<th>$\sigma(1)$</th>
<th>Val($c_0$)</th>
<th>Val($c_1$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

![Diagram of the clock](attachment:image.png)
The clock

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$\sigma(3)$</th>
<th>$\sigma(2)$</th>
<th>$\sigma(1)$</th>
<th>Val($c_0$)</th>
<th>Val($c_1$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The diagram shows the clock state transitions and iteration steps.
# The clock

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$\sigma(3)$</th>
<th>$\sigma(2)$</th>
<th>$\sigma(1)$</th>
<th>Val($c_0$)</th>
<th>Val($c_1$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

![Diagram of the clock system](image-url)
## The clock

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$\sigma(3)$</th>
<th>$\sigma(2)$</th>
<th>$\sigma(1)$</th>
<th>$\text{Val}(c_0)$</th>
<th>$\text{Val}(c_1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The diagram illustrates the states and transitions of the clock mechanism. The states are labeled $c_0$, $c_1$, $3'$, $2'$, $1'$, and $si'$. The transitions are labeled with $\alpha_1$, $\alpha_2$, $\alpha_3$, and $T \cdot 2^{3+1}$. The values $c_0$ and $c_1$ are the inputs, and $si'$ is the output.
The clock

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$\sigma(3)$</th>
<th>$\sigma(2)$</th>
<th>$\sigma(1)$</th>
<th>Val($c_0$)</th>
<th>Val($c_1$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Diagram:

- $c_0$ to $3$ with $\alpha_3$
- $3'$ to $3$ with $\alpha_3$
- $3'$ to $2'$ with $\alpha_2$
- $2'$ to $2$ with $\alpha_2$
- $2$ to $1$ with $\alpha_1$
- $1$ to $1'$ with $\alpha_1$
- $1'$ to $0$ with $\alpha_1$
- $0$ to $c_1$ with $\alpha_1$
- $c_1$ to $3'$ with $\alpha_3$
- $3'$ to $3$ with $\alpha_3$
- $3$ to $2$ with $\alpha_2$
- $2$ to $2'$ with $\alpha_2$
- $2'$ to $1'$ with $\alpha_1$
- $1'$ to $1$ with $\alpha_1$
- $1$ to $0$ with $\alpha_1$
- $0$ to $si$ with $\alpha_1$
- $si$ to $si'$ with $0$
- $T \cdot 2^{3+1}$ to $si'$
The clock

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$\sigma(3)$</th>
<th>$\sigma(2)$</th>
<th>$\sigma(1)$</th>
<th>Val($c_0$)</th>
<th>Val($c_1$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Diagram:

- $c_0$ to $3$ with $\alpha_3$
- $3'$ to $2'$ with $\alpha_2$
- $2'$ to $1'$ with $\alpha_1$
- $1'$ to $0$ with $\alpha_1$
- $0$ to $s_i$ with $T \cdot 2^{3+1}$
- $s_i$ to $0$
- $c_1$ to $3'$ with $\alpha_3$
- $3'$ to $2'$ with $\alpha_2$
- $2'$ to $1'$ with $\alpha_1$
- $1'$ to $0$ with $\alpha_1$
- $0$ to $s_i'$ with $T \cdot 2^{3+1}$
- $s_i'$ to $0$
### The clock

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$\sigma(3)$</th>
<th>$\sigma(2)$</th>
<th>$\sigma(1)$</th>
<th>Val($c_0$)</th>
<th>Val($c_1$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

---

### Diagram

- $c_0$: 0
- $c_1$: 3' → 2' → 1' → 0 → $si'$. 
- $si$: 0 → $T \cdot 2^{3+1}$.
The clock

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$\sigma(3)$</th>
<th>$\sigma(2)$</th>
<th>$\sigma(1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Val($c_0$)</th>
<th>Val($c_1$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ \sigma(3) \cdot \sigma(2) \cdot \sigma(1) \]

\[ \alpha_3 \cdot \alpha_2 \cdot \alpha_1 \]

\[ T \cdot 2^{3+1} \]
Construction overview

Circuit 1

Circuit 0

STORE

COMPUTE

CLOCK

\( c_0 \)

\( c_1 \)

In

Out

High

Low

Time: \( t \) even
Construction overview

Circuit 1

STORE

Circuit 0

COMPUTE

CLOCK

\( c_0 \) High

\( c_1 \) Low

In

Out

Time: \( t \) odd
Inside one of the circuits

We only use Not and Or gates

![Circuit Diagram]
Circuit padding

We only use Not and Or gates

We add padding gates to ensure that all outputs have the same depth
Gadgets for gates

We only use Not and Or gates

We add padding gates to ensure that all outputs have the same depth

Each gate is represented by a gadget
Gadgets

Appeal Reduction Gadget

\[ t \leftarrow +r_d, p, +r_f \]

\[ s \]

\[ t \leftarrow +r_f \]

\[ s' \]

\[ 1 - p \]

\[ +r_f \]

\[ p \]

\[ +r_d \]

\[ s \]
### Appeal Reduction Gadget

- $t \leftarrow +r_d, p, +r_f$
- $s \leftarrow +r_f$
- $1 - p \leftarrow p, +r_d$

### Input bit

- $\text{one}_i$
- $\text{zero}_i$

### Or gate

- $\text{c}_i$
- $x_i$
- $\text{out}_i$
- $v_i$

### Not gate

- $\text{c}_i$
- $a_i$
- $\text{out}_i$
- $\text{out}_{\ln l}$
Proving PSPACE-completeness

We want to reduce from these problems

- Is the $z$th bit of $F^i(B)$ equal to 1 for some even $i \leq 2^n$?
- Is the $z$th bit of $F^{2^n}(B)$ equal to 1?

To these problems

- Will Dantzig’s switching rule ever switch $a$?
- Will $a$ be used by the final optimal policy found by Dantzig’s switching rule?
Will action \( a \) ever be switched?

**Definition (PSPACE-complete problem)**

Is the \( z \)-th bit of \( F^i(B) \) equal to 1 for some even \( i \in \{1, \ldots, 2^n\} \)?

**Input bit \( z \):**

- Action used at \( \text{out}^0_z \) encodes the \( z \)-th bit \( F^i(B) \)
- This bit starts as a 0 in \( B \)
- This action is switched at some point

\[ \quad \leftrightarrow \quad \]

the \( z \)-th bit flips in some iteration of \( F \) on \( B \)
Will action $a$ ever be switched?

**Definition (PSPACE-complete problem)**

Is the $z$-th bit of $F^i(B)$ equal to 1 for some even $i \in \{1, \ldots, 2^n\}$?

Input bit $z$:

- action used at $out_z^0$ encodes the $z$-th bit $F^i(B)$
- this bit starts as a 0 in $B$
- this action is switched at some point

$\iff$

the $z$-th bit flips in some iteration of $F$ on $B$
Is action $a$ used in the solution?

**Definition (PSPACE-complete problem)**

Is the $z$-th bit of $F^{2^n}(B)$ equal to 1?

---

**Diagram:**

- $\text{one}_z^0$
- $\text{zero}_z^0$
- $\text{out}_{z}^0$
- $\text{c}_0$
- $\text{c}_1$
- $\text{out}_{\text{in}(z)}^1$

**Lock:**

- $\text{one}_z^0$
- $\text{b}_2$
- $\text{b}_1$
- $\text{si}$
- $\text{zero}_z^0$

---

After $2^n$ ticks of the clock, the **lock gadget** makes $\text{out}_{z}^0$ indifferent between its actions (for ever more)
Recap results

Given:
- an MDP $M$
- a starting policy $\sigma$
- an action $a$
- a LP $L$
- an initial bfs $b$
- a variable $v$

Theorem

The following four problems are all PSPACE-complete:

If Dantzig’s switching rule is applied to $M$ starting from $\sigma$
- will $a$ ever be switched?
- will $a$ be used by the final optimal policy?

If Dantzig’s pivot rule is applied to $L$ starting from $b$
- will $v$ ever enter the basis?
- will $v$ be in the final optimal basis?
Conclusion

- Dantzig’s pivot rule for LPs can solve any problem in \text{PSPACE}.
- Thus, it solves problems (presumably) much harder than LP.
- On the other hand, it performs well in practice on interesting LPs.
Conclusion

- Dantzig’s pivot rule for LPs can solve any problem in \textit{PSPACE}.
- Thus, it solves problems (presumably) \textbf{much harder} than LP.
- On the other hand, it performs well in practice on interesting LPs.

Open problems:

- Analyse other pivot rules (a recipe for \textit{PSPACE}-hardness?)
- Construct a polynomial-time pivot rule!
- The computational power of other algorithms.
Thank you

John Fearnley and Rahul Savani

The Complexity of the Simplex Method

To appear at STOC 2015