

# Subexponential lower bounds for randomized pivoting rules for the simplex algorithm

Oliver Friedmann<sup>1</sup>   Thomas Dueholm Hansen<sup>2</sup>   Uri Zwick<sup>3</sup>

<sup>1</sup> Department of Computer Science,  
University of Munich, Germany.

<sup>2</sup> Center for the Theory of Interactive Computation,  
Department of Computer Science,  
Aarhus University, Denmark.

<sup>3</sup> School of Computer Science,  
Tel Aviv University, Israel.

ARC Theory Day, November 11, 2011

- Linear programming and the simplex algorithm.
- Markov decision processes and the connection to linear programming.
- Lower bounds for the simplex algorithm utilizing Markov decision processes.
  - Example: A lower bound for BLAND'S RULE.
  - Gadgets and general ideas for the lower bound for RANDEDGE.
- On lower bounds for the RANDOMFACET pivoting rule and the RANDOMIZED BLAND'S RULE.

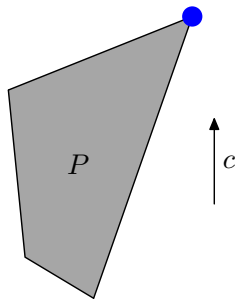
- A **linear program** (LP) is an optimization problem of the form:

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{s.t.} & Ax \leq b \end{array}$$

where  $A$  is an  $n \times d$  matrix and  $b \in \mathbb{R}^n$  and  $c \in \mathbb{R}^d$  are vectors.

- The set of **feasible solutions** is a **convex polytope** (or **polyhedron**):

$$P = \{x \in \mathbb{R}^d \mid Ax \leq b\}$$



- A **linear program** (LP) is an optimization problem of the form:

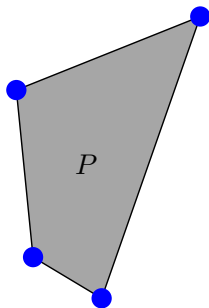
$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{s.t.} & Ax \leq b \end{array}$$

where  $A$  is an  $n \times d$  matrix and  $b \in \mathbb{R}^n$  and  $c \in \mathbb{R}^d$  are vectors.

- The set of **feasible solutions** is a **convex polytope** (or **polyhedron**):

$$P = \{x \in \mathbb{R}^d \mid Ax \leq b\}$$

- **Vertices** (or corners) are **basic feasible solutions**.



- Every linear program has an equivalent **standard form**, defined by an  $n \times m$  matrix  $A$ , with  $m = d + n$ :

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{array}$$

- Every linear program has an equivalent **standard form**, defined by an  $n \times m$  matrix  $A$ , with  $m = d + n$ :

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{array}$$

- A **basis** is a subset  $B \subseteq \{1, \dots, m\}$  of  $n$  linearly independent columns of  $A$ .

- Every linear program has an equivalent **standard form**, defined by an  $n \times m$  matrix  $A$ , with  $m = d + n$ :

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{array}$$

- A **basis** is a subset  $B \subseteq \{1, \dots, m\}$  of  $n$  linearly independent columns of  $A$ .
- Every **basic feasible solution** is defined by a basis by setting **non-basic** variables,  $x_i$  for  $i \notin B$ , to zero.

- Every linear program has an equivalent **standard form**, defined by an  $n \times m$  matrix  $A$ , with  $m = d + n$ :

$$\begin{aligned} & \text{maximize} && c^T x \\ & \text{s.t.} && Ax = b \\ & && x \geq 0 \end{aligned}$$

- A **basis** is a subset  $B \subseteq \{1, \dots, m\}$  of  $n$  linearly independent columns of  $A$ .
- Every **basic feasible solution** is defined by a basis by setting **non-basic** variables,  $x_i$  for  $i \notin B$ , to zero.
- The operation of exchanging a single basic variable in  $B$  with a non-basic variable, producing a new basis  $B'$ , is called **pivoting**.

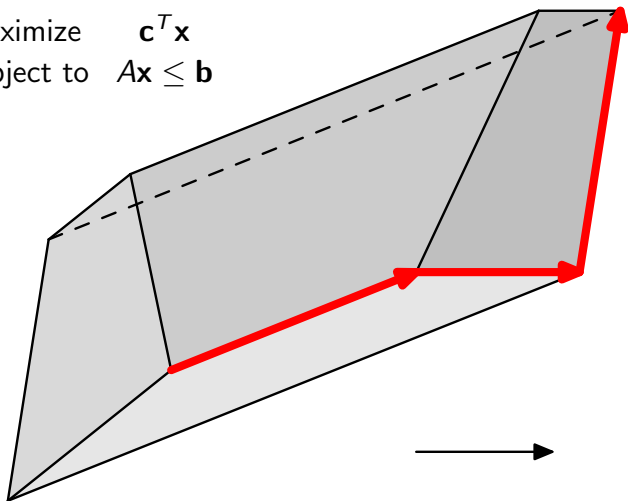
- Every linear program has an equivalent **standard form**, defined by an  $n \times m$  matrix  $A$ , with  $m = d + n$ :

$$\begin{aligned} & \text{maximize} && c^T x \\ & \text{s.t.} && Ax = b \\ & && x \geq 0 \end{aligned}$$

- A **basis** is a subset  $B \subseteq \{1, \dots, m\}$  of  $n$  linearly independent columns of  $A$ .
- Every **basic feasible solution** is defined by a basis by setting **non-basic** variables,  $x_i$  for  $i \notin B$ , to zero.
- The operation of exchanging a single basic variable in  $B$  with a non-basic variable, producing a new basis  $B'$ , is called **pivoting**.
  - Geometrically, pivoting corresponds to moving along edges of the polytope.

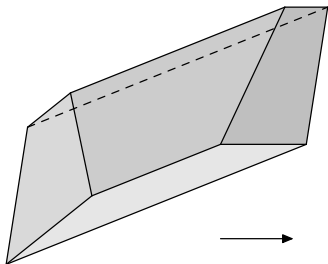
# The simplex algorithm, Dantzig (1947)

$$\begin{aligned} &\text{maximize} && \mathbf{c}^T \mathbf{x} \\ &\text{subject to} && A\mathbf{x} \leq \mathbf{b} \end{aligned}$$



# Example: The tableau method

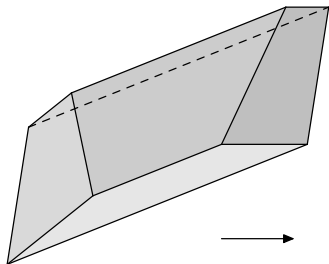
$$\begin{array}{ll} \max & 2x_1 - 2x_2 - x_3 \\ \text{s.t.} & \frac{1}{3}x_1 - \frac{2}{3}x_2 - \frac{2}{3}x_3 \leq 1 \\ & x_2 + x_3 \leq 2 \\ & x_3 \leq 1 \\ & x_1, x_2, x_3 \geq 0 \end{array}$$



- Transform the linear program to standard form by introducing **slack variables**.

# Example: The tableau method

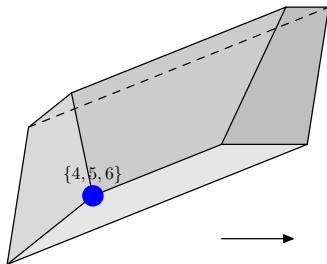
$$\begin{array}{ll} \max & 2x_1 - 2x_2 - x_3 \\ \text{s.t.} & \frac{1}{3}x_1 - \frac{2}{3}x_2 - \frac{2}{3}x_3 + x_4 = 1 \\ & x_2 + x_3 + x_5 = 2 \\ & x_3 + x_6 = 1 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{array}$$



- Transform the linear program to standard form by introducing **slack variables**.

# Example: The tableau method

$$\begin{aligned} \max \quad & 2x_1 - 2x_2 - x_3 \\ \text{s.t.} \quad & x_4 = 1 - \frac{1}{3}x_1 + \frac{2}{3}x_2 + \frac{2}{3}x_3 \\ & x_5 = 2 - x_2 - x_3 \\ & x_6 = 1 - x_3 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{aligned}$$

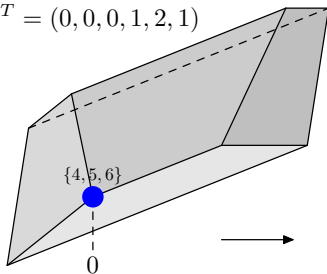


- Transform the linear program to standard form by introducing **slack variables**.
- Pick a basis, in this case  $\{4, 5, 6\}$ , and express the basic variables and the objective function in terms of non-basic variables. This representation is called a **tableau**.

# Example: The tableau method

$$\begin{aligned} \max \quad & 2x_1 - 2x_2 - x_3 \\ \text{s.t.} \quad & x_4 = 1 - \frac{1}{3}x_1 + \frac{2}{3}x_2 + \frac{2}{3}x_3 \\ & x_5 = 2 - x_2 - x_3 \\ & x_6 = 1 - x_3 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{aligned}$$

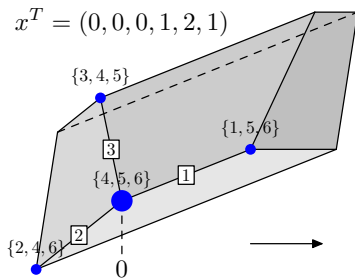
$$x^T = (0, 0, 0, 1, 2, 1)$$



- Transform the linear program to standard form by introducing **slack variables**.
- Pick a basis, in this case  $\{4, 5, 6\}$ , and express the basic variables and the objective function in terms of non-basic variables. This representation is called a **tableau**.
- The corresponding basic solution and its value can be read by setting the non-basic variables to zero.

# Example: The tableau method

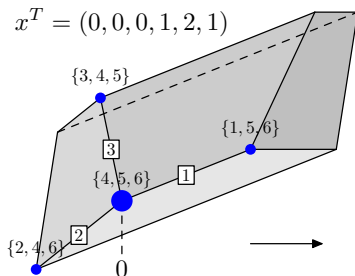
$$\begin{aligned} \max \quad & 2x_1 - 2x_2 - x_3 \\ \text{s.t.} \quad & x_4 = 1 - \frac{1}{3}x_1 + \frac{2}{3}x_2 + \frac{2}{3}x_3 \\ & x_5 = 2 - x_2 - x_3 \\ & x_6 = 1 - x_3 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{aligned}$$



- If the coefficient of a non-basic variable  $x_i$  in the objective function is positive, increasing  $x_i$  will improve the value.

# Example: The tableau method

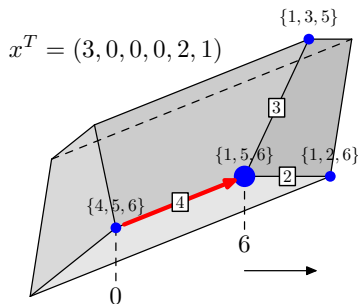
$$\begin{aligned} \max \quad & 2x_1 - 2x_2 - x_3 \\ \text{s.t.} \quad & x_4 = 1 - \frac{1}{3}x_1 + \frac{2}{3}x_2 + \frac{2}{3}x_3 \\ & x_5 = 2 - x_2 - x_3 \\ & x_6 = 1 - x_3 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{aligned}$$



- If the coefficient of a non-basic variable  $x_i$  in the objective function is positive, increasing  $x_i$  will improve the value.
- $x_i$  can be increased until another basic variable  $x_j$  becomes zero, which completes the pivot. The basis is then updated by exchanging  $i$  and  $j$ . If no variable becomes zero the value is **unbounded**.

# Example: The tableau method

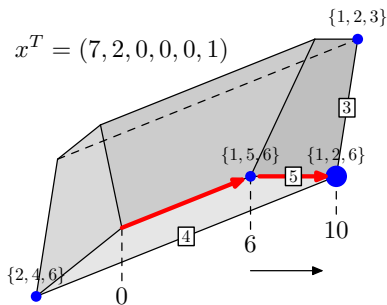
$$\begin{aligned} \max \quad & 6 + 2x_2 + 3x_3 - 6x_4 \\ \text{s.t.} \quad & x_1 = 3 + 2x_2 + 2x_3 - 3x_4 \\ & x_5 = 2 - x_2 - x_3 \\ & x_6 = 1 - x_3 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{aligned}$$



- If the coefficient of a non-basic variable  $x_i$  in the objective function is positive, increasing  $x_i$  will improve the value.
- $x_i$  can be increased until another basic variable  $x_j$  becomes zero, which completes the pivot. The basis is then updated by exchanging  $i$  and  $j$ . If no variable becomes zero the value is **unbounded**.

# Example: The tableau method

$$\begin{aligned} \max \quad & 10 + x_3 - 6x_4 - 2x_5 \\ \text{s.t.} \quad & x_1 = 7 - 3x_4 - 2x_5 \\ & x_2 = 2 - x_3 - x_5 \\ & x_6 = 1 - x_3 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{aligned}$$



- If the coefficient of a non-basic variable  $x_i$  in the objective function is positive, increasing  $x_i$  will improve the value.
- $x_i$  can be increased until another basic variable  $x_j$  becomes zero, which completes the pivot. The basis is then updated by exchanging  $i$  and  $j$ . If no variable becomes zero the value is **unbounded**.



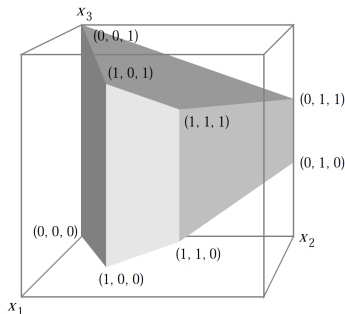
- Two choices must be made when pivoting:
  - ① Which non-basic variable with positive coefficient enters the basis?
  - ② Which basic variable leaves the basis, in case of a tie?

These choices are specified by a **pivoting rule**.

- LARGESTCOEFFICIENT, Dantzig (1947)
  - The non-basic variable with the largest coefficient enters the basis.
- BLAND'S RULE, Bland (1977)
  - Always pick the available variable with the smallest index, both for entering and leaving the basis.
  - This pivoting rule is guaranteed not to cycle.

# Deterministic pivoting rules

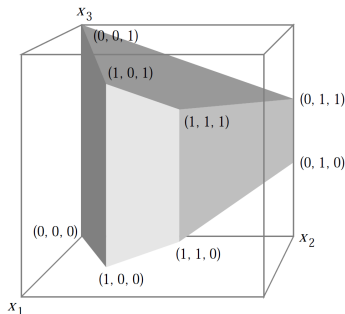
- Klee and Minty (1972): The **LARGESTCOEFFICIENT** pivoting rule may require exponentially many steps; the Klee-Minty cube.<sup>1</sup>



<sup>1</sup>Picture from Gärtner, Henk and Ziegler (1998)

# Deterministic pivoting rules

- Klee and Minty (1972): The `LARGESTCOEFFICIENT` pivoting rule may require exponentially many steps; the Klee-Minty cube.<sup>1</sup>
- Essentially all known natural deterministic pivoting rules are now known to be exponential:
  - `LARGESTINCREASE`: Jeroslow (1973).
  - `STEEPESTEDGE`: Goldfarb and Sit (1979).
  - `BLAND'S RULE`: Avis and Chvátal (1978).
  - `SHADOWVERTEX`: Murty (1980), Goldfarb (1983).
  - See Amenta and Ziegler (1996) for a unified view.



<sup>1</sup>Picture from Gärtner, Henk and Ziegler (1998)

# On the diameter of polytopes

- The **diameter** of a polytope  $P$  is the maximum distance between any two vertices in the edge graph of  $P$ .
- The diameter of  $P$  is a lower bound for the number of steps required for the simplex algorithm to solve a corresponding LP.

# On the diameter of polytopes

- The **diameter** of a polytope  $P$  is the maximum distance between any two vertices in the edge graph of  $P$ .
- The diameter of  $P$  is a lower bound for the number of steps required for the simplex algorithm to solve a corresponding LP.
- Hirsch conjecture (1957): The diameter of any  $n$ -facet polytope in  $d$ -dimensional Euclidean space  $P$  is at most  $n - d$ .
- Counter-example by Santos (2010): Existence of polytopes with diameter  $(1 + \epsilon)(n - d)$ .
  - It remains open whether the diameter is polynomial, or even linear, in  $n$  and  $d$ .

# On the diameter of polytopes

- The **diameter** of a polytope  $P$  is the maximum distance between any two vertices in the edge graph of  $P$ .
- The diameter of  $P$  is a lower bound for the number of steps required for the simplex algorithm to solve a corresponding LP.
- Hirsch conjecture (1957): The diameter of any  $n$ -facet polytope in  $d$ -dimensional Euclidean space  $P$  is at most  $n - d$ .
- Counter-example by Santos (2010): Existence of polytopes with diameter  $(1 + \epsilon)(n - d)$ .
  - It remains open whether the diameter is polynomial, or even linear, in  $n$  and  $d$ .
- Perhaps simple randomized pivoting rules can find a short path with high probability?

# On the diameter of polytopes

- The **diameter** of a polytope  $P$  is the maximum distance between any two vertices in the edge graph of  $P$ .
- The diameter of  $P$  is a lower bound for the number of steps required for the simplex algorithm to solve a corresponding LP.
- Hirsch conjecture (1957): The diameter of any  $n$ -facet polytope in  $d$ -dimensional Euclidean space  $P$  is at most  $n - d$ .
- Counter-example by Santos (2010): Existence of polytopes with diameter  $(1 + \epsilon)(n - d)$ .
  - It remains open whether the diameter is polynomial, or even linear, in  $n$  and  $d$ .
- Perhaps simple randomized pivoting rules can find a short path with high probability?
- Probably not: We present almost exponential lower bounds for the most natural known randomized pivoting rules.
  - The constructed polytopes have low diameter.

- RANDOMEDGE
  - Let a uniformly random non-basic variable with positive coefficient enter the basis.

- RANDOMEDGE
  - Let a uniformly random non-basic variable with positive coefficient enter the basis.
- RANDOMFACET, Kalai (1992) and Matoušek, Sharir and Welzl (1992)
  - Pick a uniformly random facet that contains the current vertex, and recursively find an optimal solution within that facet. If possible, make an improving pivot leaving the facet and repeat.

- RANDOMEDGE
  - Let a uniformly random non-basic variable with positive coefficient enter the basis.
- RANDOMFACET, Kalai (1992) and Matoušek, Sharir and Welzl (1992)
  - Pick a uniformly random facet that contains the current vertex, and recursively find an optimal solution within that facet. If possible, make an improving pivot leaving the facet and repeat.
  - This randomized pivoting rule finds an optimal solution in an expected subexponential,  $2^{O(\sqrt{(n-d)\log n})}$ , number of steps.

- RANDOMEDGE
  - Let a uniformly random non-basic variable with positive coefficient enter the basis.
- RANDOMFACET, Kalai (1992) and Matoušek, Sharir and Welzl (1992)
  - Pick a uniformly random facet that contains the current vertex, and recursively find an optimal solution within that facet. If possible, make an improving pivot leaving the facet and repeat.
  - This randomized pivoting rule finds an optimal solution in an expected subexponential,  $2^{O(\sqrt{(n-d)\log n})}$ , number of steps.
- RANDOMIZED BLAND'S RULE
  - Randomly permute the indices of the variables and use BLAND'S RULE.

## We show that:

- RANDOMEDGE may require  $2^{\Omega(n^{1/4})}$  expected pivoting steps.
- RANDOMFACET may require  $2^{\tilde{\Omega}(n^{1/3})}$  expected pivoting steps.
- RANDOMIZED BLAND'S RULE may require  $2^{\tilde{\Omega}(n^{1/2})}$  expected pivoting steps.

where  $n$  is the number of equality constraints, and the number of variables is  $m = \tilde{O}(n)$ .

## We show that:

- RANDOMEDGE may require  $2^{\Omega(n^{1/4})}$  expected pivoting steps.
- RANDOMFACET may require  $2^{\tilde{\Omega}(n^{1/3})}$  expected pivoting steps.
- RANDOMIZED BLAND'S RULE may require  $2^{\tilde{\Omega}(n^{1/2})}$  expected pivoting steps.

where  $n$  is the number of equality constraints, and the number of variables is  $m = \tilde{O}(n)$ .

## NOTE:

- In our SODA 2011 paper we prove the lower bound for the RANDOMIZED BLAND'S RULE, and **incorrectly** claim that the expected number of pivoting steps is the same for RANDOMFACET.
- We have been able to repair the lower bound, but with a worse bound. The details will appear shortly.

- The proofs are based on strong connections between linear programming and Markov decision processes (MDPs).
- We prove lower bounds for corresponding POLICYITERATION algorithms for MDPs, which immediately translate to lower bounds for the simplex algorithm.

- The proofs are based on strong connections between linear programming and Markov decision processes (MDPs).
- We prove lower bounds for corresponding POLICYITERATION algorithms for MDPs, which immediately translate to lower bounds for the simplex algorithm.
- Friedmann (2009) and Fearnley (2010) gave a similar lower bound construction for Howard's algorithm for solving MDPs (and *parity games*).

- The proofs are based on strong connections between linear programming and Markov decision processes (MDPs).
- We prove lower bounds for corresponding POLICYITERATION algorithms for MDPs, which immediately translate to lower bounds for the simplex algorithm.
- Friedmann (2009) and Fearnley (2010) gave a similar lower bound construction for Howard's algorithm for solving MDPs (and *parity games*).
- Friedmann (2011) used the same technique to prove a lower bound of subexponential form,  $2^{\Omega(\sqrt{n})}$ , for Zadeh's LEASTENTERED pivoting rule (1980).

Superpolynomial lower bounds for `RANDOMEDGE` and `RANDOMFACET` were previously only known in an abstract setting (Acyclic Unique Sink Orientations):

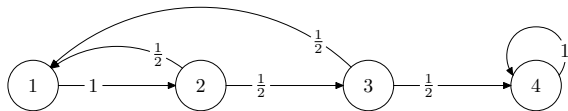
- Matoušek and Szabó (2006):  $2^{\Omega(n^{1/3})}$  lower bound for `RANDOMEDGE`.
- Matoušek (1994):  $2^{\Omega(\sqrt{n})}$  lower bound for `RANDOMFACET`.

Superpolynomial lower bounds for `RANDOMEDGE` and `RANDOMFACET` were previously only known in an abstract setting (Acyclic Unique Sink Orientations):

- Matoušek and Szabó (2006):  $2^{\Omega(n^{1/3})}$  lower bound for `RANDOMEDGE`.
- Matoušek (1994):  $2^{\Omega(\sqrt{n})}$  lower bound for `RANDOMFACET`.

For `RANDOMEDGE` and the `RANDOMIZED BLAND'S RULE` no subexponential upper bounds are known.

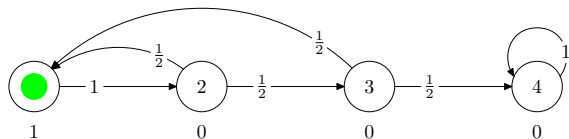
- Linear programming and the simplex algorithm.
- ⇒ ● Markov decision processes and the connection to linear programming.
- Lower bounds for the simplex algorithm utilizing Markov decision processes.
  - Example: A lower bound for BLAND'S RULE.
  - Gadgets and general ideas for the lower bound for RANDOMEDGE.
- On lower bounds for the RANDOMFACET pivoting rule and the RANDOMIZED BLAND'S RULE.



$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- An  $n$ -state **Markov chain** is defined by an  $n \times n$  stochastic matrix  $P$ , with  $P_{i,j}$  being the probability of making a transition from state  $i$  to state  $j$ . I.e.,  $\sum_j P_{i,j} = 1$ .

# Markov chains

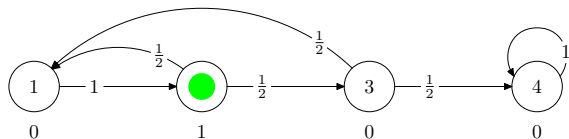


$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- An  $n$ -state **Markov chain** is defined by an  $n \times n$  stochastic matrix  $P$ , with  $P_{i,j}$  being the probability of making a transition from state  $i$  to state  $j$ . I.e.,  $\sum_j P_{i,j} = 1$ .
- Let  $b \in \mathbb{R}^n$  define a probability distribution for picking an initial state.
- $b^T P^k$  is a vector defining the probabilities of being in each state after  $k$  transitions.

$k$	$b^T P^k$
0	1 0 0 0

# Markov chains

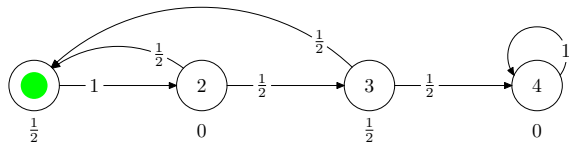


$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- An  $n$ -state **Markov chain** is defined by an  $n \times n$  stochastic matrix  $P$ , with  $P_{i,j}$  being the probability of making a transition from state  $i$  to state  $j$ . I.e.,  $\sum_j P_{i,j} = 1$ .
- Let  $b \in \mathbb{R}^n$  define a probability distribution for picking an initial state.
- $b^T P^k$  is a vector defining the probabilities of being in each state after  $k$  transitions.

$k$	$b^T P^k$			
0	1	0	0	0
1	0	1	0	0

# Markov chains

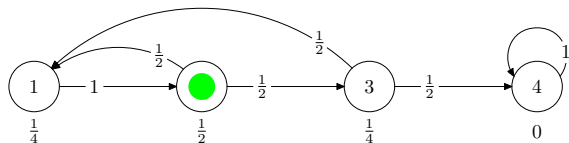


$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- An  $n$ -state **Markov chain** is defined by an  $n \times n$  stochastic matrix  $P$ , with  $P_{i,j}$  being the probability of making a transition from state  $i$  to state  $j$ . I.e.,  $\sum_j P_{i,j} = 1$ .
- Let  $b \in \mathbb{R}^n$  define a probability distribution for picking an initial state.
- $b^T P^k$  is a vector defining the probabilities of being in each state after  $k$  transitions.

$k$	$b^T P^k$			
0	1	0	0	0
1	0	1	0	0
2	$\frac{1}{2}$	0	$\frac{1}{2}$	0

# Markov chains

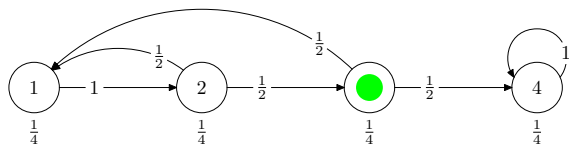


$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- An  $n$ -state **Markov chain** is defined by an  $n \times n$  stochastic matrix  $P$ , with  $P_{i,j}$  being the probability of making a transition from state  $i$  to state  $j$ . I.e.,  $\sum_j P_{i,j} = 1$ .
- Let  $b \in \mathbb{R}^n$  define a probability distribution for picking an initial state.
- $b^T P^k$  is a vector defining the probabilities of being in each state after  $k$  transitions.

$k$	$b^T P^k$			
0	1	0	0	0
1	0	1	0	0
2	$\frac{1}{2}$	0	$\frac{1}{2}$	0
3	$\frac{1}{4}$	$\frac{1}{2}$	0	$\frac{1}{4}$

# Markov chains

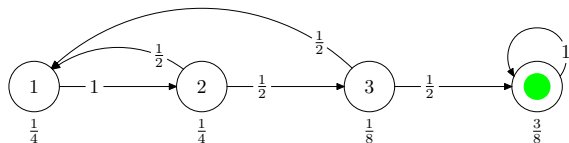


- An  $n$ -state **Markov chain** is defined by an  $n \times n$  stochastic matrix  $P$ , with  $P_{i,j}$  being the probability of making a transition from state  $i$  to state  $j$ . I.e.,  $\sum_j P_{i,j} = 1$ .
- Let  $b \in \mathbb{R}^n$  define a probability distribution for picking an initial state.
- $b^T P^k$  is a vector defining the probabilities of being in each state after  $k$  transitions.

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$k$	$b^T P^k$			
0	1	0	0	0
1	0	1	0	0
2	$\frac{1}{2}$	0	$\frac{1}{2}$	0
3	$\frac{1}{4}$	$\frac{1}{2}$	0	$\frac{1}{4}$
4	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$

# Markov chains



- An  $n$ -state **Markov chain** is defined by an  $n \times n$  stochastic matrix  $P$ , with  $P_{i,j}$  being the probability of making a transition from state  $i$  to state  $j$ . I.e.,  $\sum_j P_{i,j} = 1$ .
- Let  $b \in \mathbb{R}^n$  define a probability distribution for picking an initial state.
- $b^T P^k$  is a vector defining the probabilities of being in each state after  $k$  transitions.

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$k$	$b^T P^k$			
0	1	0	0	0
1	0	1	0	0
2	$\frac{1}{2}$	0	$\frac{1}{2}$	0
3	$\frac{1}{4}$	$\frac{1}{2}$	0	$\frac{1}{4}$
4	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
5	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{3}{8}$
$\vdots$		$\vdots$		

# Markov chains with rewards

- We refer to the act of leaving a state as an **action**.
- A Markov chain with rewards is a Markov chain  $P \in \mathbb{R}^{n \times n}$  where a vector  $c \in \mathbb{R}^n$  associates actions with **rewards** (or **costs**). I.e.,  $c_i$  is the reward for leaving state  $i$ .
- We are interested in the expected **total reward**,  $\sum_{k=0}^{\infty} b^T P^k c$ , accumulated for some initial vector  $b$ . Note that this series generally does not converge.
- **Stopping condition:** A terminal state  $t$  is reached with probability 1 from all states. This ensures convergence.

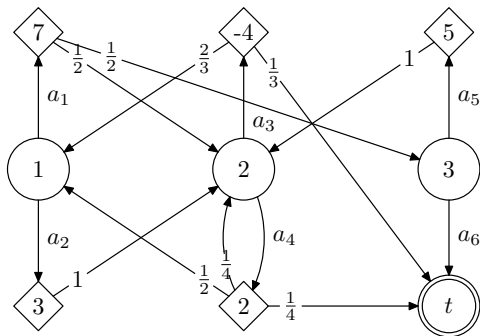
$k$	$b^T P^k$			
0	1	0	0	0
1	0	1	0	0
2	$\frac{1}{2}$	0	$\frac{1}{2}$	0
3	$\frac{1}{4}$	$\frac{1}{2}$	0	$\frac{1}{4}$
4	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
5	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{3}{8}$
$\vdots$		$\vdots$		

# Markov chains with rewards

- We refer to the act of leaving a state as an **action**.
- A Markov chain with rewards is a Markov chain  $P \in \mathbb{R}^{n \times n}$  where a vector  $c \in \mathbb{R}^n$  associates actions with **rewards** (or **costs**). I.e.,  $c_i$  is the reward for leaving state  $i$ .
- We are interested in the expected **total reward**,  $\sum_{k=0}^{\infty} b^T P^k c$ , accumulated for some initial vector  $b$ . Note that this series generally does not converge.

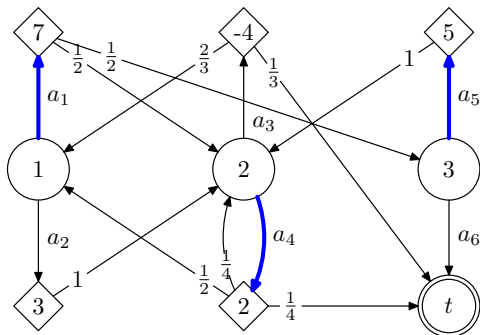
$k$	$b^T P^k$			
0	1	0	0	0
1	0	1	0	0
2	$\frac{1}{2}$	0	$\frac{1}{2}$	0
3	$\frac{1}{4}$	$\frac{1}{2}$	0	$\frac{1}{4}$
4	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
5	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{3}{8}$
$\vdots$		$\vdots$		

- **Stopping condition:** A terminal state  $t$  is reached with probability 1 from all states. This ensures convergence.
- The **value** of state  $i$  is the expected total reward when starting in state  $i$  with probability 1. I.e.,  $b = e_i$ .



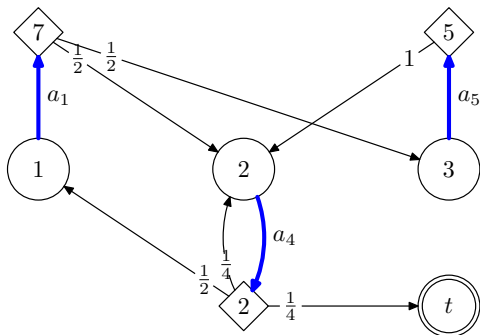
- A Markov decision process (MDP) consists of a set of  $n$  states  $S$ , each state  $i \in S$  being associated with a non-empty set of actions  $A_i$ .
- Each action  $a$  is associated with a reward  $c_a$  and a probability distribution  $P_a \in \mathbb{R}^{1 \times n}$  such that  $P_{a,j}$  is the probability of moving to state  $j$  when using action  $a$ .

# Markov decision processes



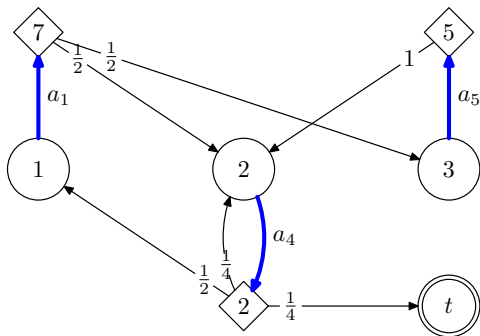
- A **policy**  $\pi$  is a choice of an action from each state.

# Markov decision processes



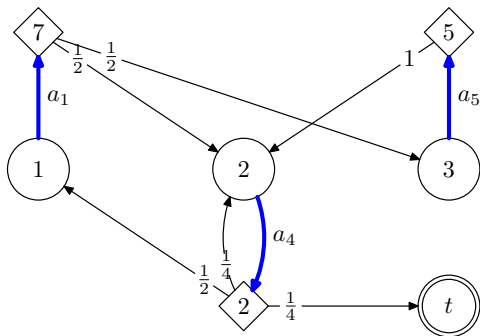
- A **policy**  $\pi$  is a choice of an action from each state.
- A policy  $\pi$  is a Markov chain with rewards.

# Markov decision processes



- A **policy**  $\pi$  is a choice of an action from each state.
- A policy  $\pi$  is a Markov chain with rewards.
- Let  $\text{VAL}_\pi(i) = \sum_{k=0}^{\infty} e_i^T P_\pi^k c_\pi$  be the value of state  $i$  for  $\pi$ .
- A policy  $\pi^*$  is **optimal** if it maximizes the values of all states. I.e.,  $\text{VAL}_{\pi^*}(i) \geq \text{VAL}_\pi(i)$  for all  $\pi$  and  $i$ .

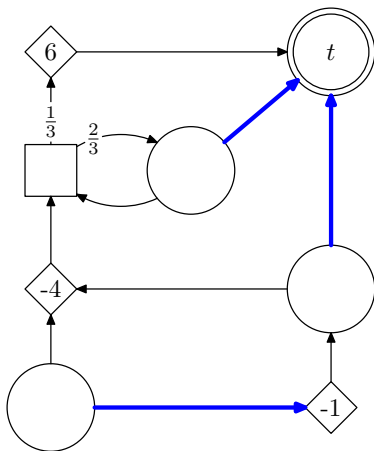
# Markov decision processes



- Shapley (1953), Bellman (1957): There always exists an optimal policy.
- Solving an MDP means finding an optimal policy.

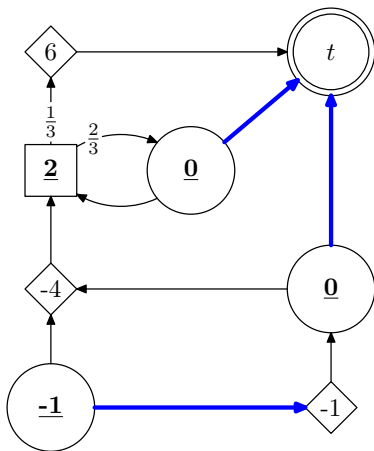
# Example: A simple MDP

- Notation for graphical representation:
  - Circles are states.
  - Diamond-shaped vertices are rewards.
  - Squares are randomization vertices.
- A **policy**  $\pi$  is shown as bold blue arrows.



# Example: A simple MDP

- Notation for graphical representation:
  - Circles are states.
  - Diamond-shaped vertices are rewards.
  - Squares are randomization vertices.
- A **policy**  $\pi$  is shown as bold blue arrows.
- States and randomization vertices are labelled by corresponding values of  $\pi$ .



# Summing values

- Since an optimal policy simultaneously maximizes the values of all states it maximizes the sum of the values:

$$\sum_{i \in S} \text{VAL}_{\pi}(i) = \sum_{k=0}^{\infty} e^T P_{\pi}^k c_{\pi}$$

where  $e^T = (1, 1, \dots, 1)$ .

$k$	$e^T P^k$			
0	1	1	1	1
1	1	1	$\frac{1}{2}$	$\frac{3}{2}$
2	$\frac{3}{4}$	1	$\frac{1}{2}$	$\frac{7}{4}$
3	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{1}{2}$	2
4	$\frac{5}{8}$	$\frac{3}{4}$	$\frac{3}{8}$	$\frac{9}{4}$
$\vdots$				

# Summing values

- Since an optimal policy simultaneously maximizes the values of all states it maximizes the sum of the values:

$$\sum_{i \in S} \text{VAL}_{\pi}(i) = \sum_{k=0}^{\infty} e^T P_{\pi}^k c_{\pi}$$

where  $e^T = (1, 1, \dots, 1)$ .

$k$	$e^T P^k$			
0	1	1	1	1
1	1	1	$\frac{1}{2}$	$\frac{3}{2}$
2	$\frac{3}{4}$	1	$\frac{1}{2}$	$\frac{7}{4}$
3	$\frac{3}{4}$	$\frac{3}{4}$	$\frac{1}{2}$	2
4	$\frac{5}{8}$	$\frac{3}{4}$	$\frac{3}{8}$	$\frac{9}{4}$
$\vdots$				

- Let  $x_a$  be the number of times action  $a$  is used, i.e., the sum of a column in the table. Then:

$$\sum_{i \in S} \text{VAL}_{\pi}(i) = \sum_{i \in S} \sum_{a \in A_i} c_a x_a$$

- The following linear program simultaneously maximizes the values of all states:

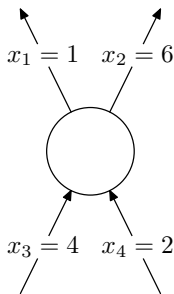
$$\text{maximize } \sum_{i \in S} \sum_{a \in A_i} c_a x_a$$

$$\text{s.t. } \forall i \in S: \sum_{a \in A_i} x_a = 1 + \sum_{j \in S} \sum_{a \in A_j} P_{a,i} x_a$$

$$\forall i \in S, \forall a \in A_i: x_a \geq 0$$

- The constraints ensure “flow conservation”.

Flow conservation:



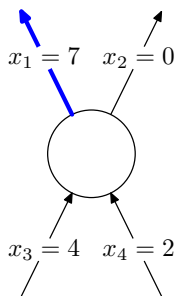
$$x_1 + x_2 = 1 + x_3 + x_4$$

# Basic feasible solutions and policies

$$\begin{aligned} & \text{maximize} && \sum_{i \in S} \sum_{a \in A_i} c_a x_a \\ & \text{s.t.} && \forall i \in S: \sum_{a \in A_i} x_a = 1 + \sum_{j \in S} \sum_{a \in A_j} P_{a,i} x_a \\ & && \forall i \in S, \forall a \in A_i: x_a \geq 0 \end{aligned}$$

- In a basic feasible solution exactly one variable (action) is non-zero for each state.
- Every basic feasible solution corresponds to a policy  $\pi$

Flow conservation:



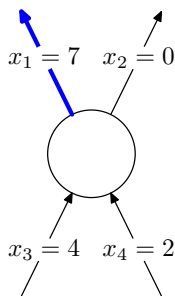
$$x_1 + x_2 = 1 + x_3 + x_4$$

# Basic feasible solutions and policies

$$\begin{aligned} & \text{maximize} && \sum_{i \in S} \sum_{a \in A_i} c_a x_a \\ & \text{s.t.} && \forall i \in S : \sum_{a \in A_i} x_a = 1 + \sum_{j \in S} \sum_{a \in A_j} P_{a,i} x_a \\ & && \forall i \in S, \forall a \in A_i : x_a \geq 0 \end{aligned}$$

- In a basic feasible solution exactly one variable (action) is non-zero for each state.
- Every basic feasible solution corresponds to a policy  $\pi$
- If the stopping condition is satisfied a policy  $\pi$  also corresponds to a basic feasible solution.

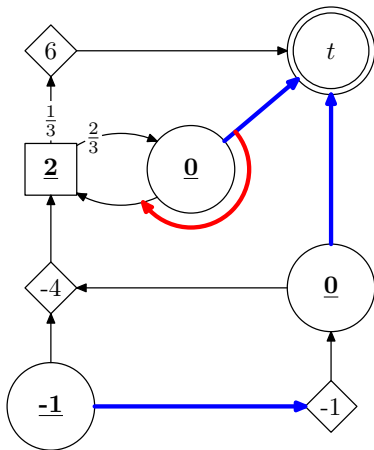
Flow conservation:



$$x_1 + x_2 = 1 + x_3 + x_4$$

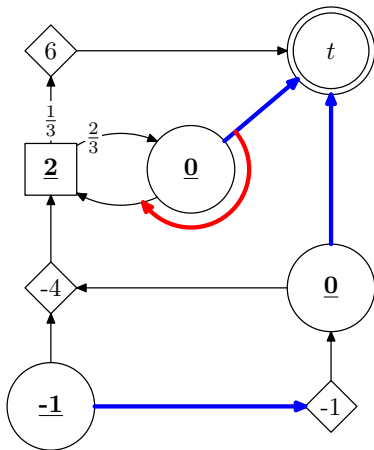
# Improving switches

- An action is an **improving switch** (improving pivot) w.r.t.  $\pi$  if it improves the values.
- It suffices to check whether an action is improving for one step w.r.t. the current values (the coefficient of the objective function is positive).
- A policy  $\pi^*$  is optimal iff there are no improving switches.



# Improving switches

- An action is an **improving switch** (improving pivot) w.r.t.  $\pi$  if it improves the values.
- It suffices to check whether an action is improving for one step w.r.t. the current values (the coefficient of the objective function is positive).
- A policy  $\pi^*$  is optimal iff there are no improving switches.
- For MDPs multiple simultaneous improving switches also lead to better values.



---

**Function** POLICYITERATION( $\pi$ )

---

**while**  $\exists$  *improving switch w.r.t.*  $\pi$  **do**

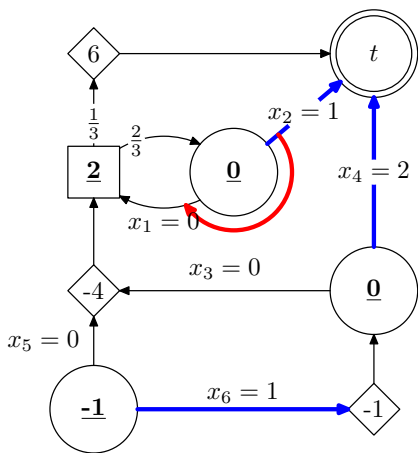
    Update  $\pi$  by performing improving switches

**return**  $\pi$

---

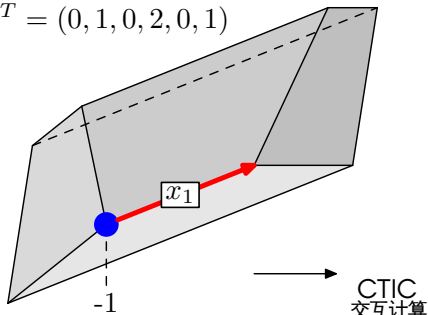
- The simplex algorithm applied to the LP for an MDP is a special case of POLICYITERATION where only one improving switch is performed in each step.

# From MDP to LP

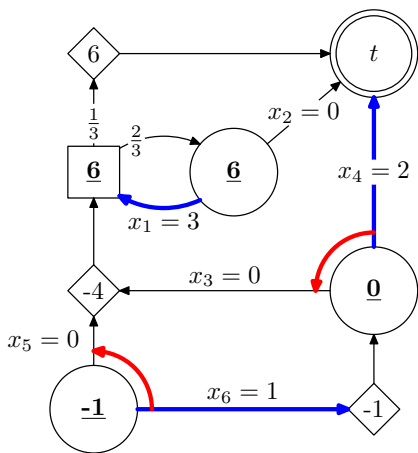


$$\begin{aligned} \max \quad & -1 + 2x_1 - 2x_3 - x_5 \\ \text{s.t.} \quad & x_2 = 1 - \frac{1}{3}x_1 + \frac{2}{3}x_3 + \frac{2}{3}x_5 \\ & x_4 = 2 - x_3 - x_5 \\ & x_6 = 1 - x_5 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{aligned}$$

$$x^T = (0, 1, 0, 2, 0, 1)$$

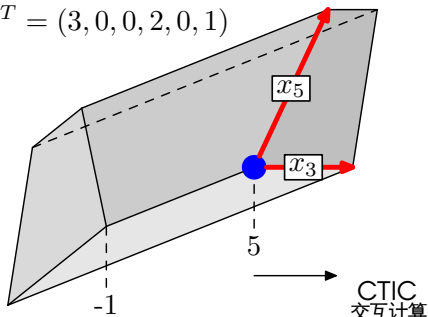


# From MDP to LP

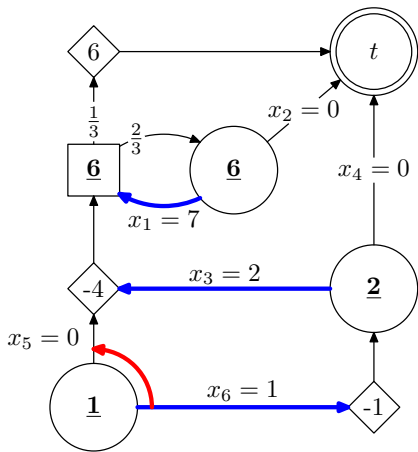


$$\begin{aligned} \max \quad & 5 - 6x_2 + 2x_3 + 3x_5 \\ \text{s.t.} \quad & x_1 = 3 - 3x_2 + 2x_3 + 2x_5 \\ & x_4 = 2 - x_3 - x_5 \\ & x_6 = 1 - x_5 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{aligned}$$

$$x^T = (3, 0, 0, 2, 0, 1)$$

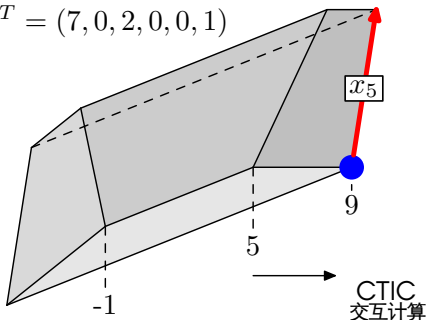


# From MDP to LP

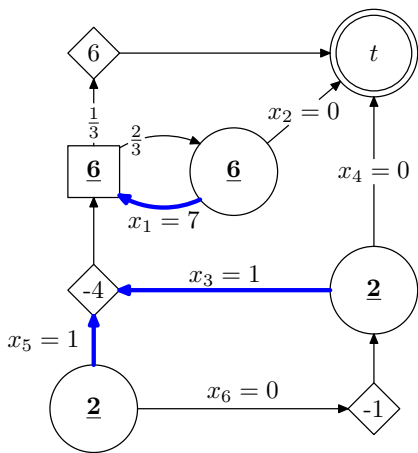


$$\begin{aligned} \max \quad & 9 - 6x_2 - 2x_4 + x_5 \\ \text{s.t.} \quad & x_1 = 7 - 3x_2 - 2x_4 \\ & x_3 = 2 - x_4 - x_5 \\ & x_6 = 1 - x_5 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{aligned}$$

$$x^T = (7, 0, 2, 0, 0, 1)$$

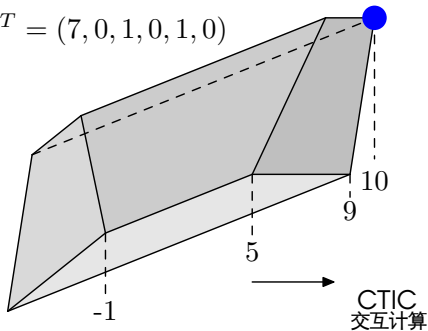


# From MDP to LP



$$\begin{aligned} \max \quad & 10 - 6x_2 - 2x_4 - x_6 \\ \text{s.t.} \quad & x_1 = 7 - 3x_2 - 2x_4 \\ & x_3 = 1 - x_4 + x_6 \\ & x_5 = 1 - x_6 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{aligned}$$

$$x^T = (7, 0, 1, 0, 1, 0)$$



- Linear programming and the simplex algorithm.
- Markov decision processes and the connection to linear programming.
- ⇒ ● Lower bounds for the simplex algorithm utilizing Markov decision processes.
  - Example: A lower bound for BLAND'S RULE.
  - Gadgets and general ideas for the lower bound for RANDOMEDGE.
- On lower bounds for the RANDOMFACET pivoting rule and the RANDOMIZED BLAND'S RULE.

- For a given pivoting rules, we define a family of lower bound MDPs  $G_n$  such that the corresponding simplex algorithm simulates an  $n$ -bit binary counter.
  - To illustrate the main ideas for the lower bound for RANDOMEDGE I first show a simpler construction for BLAND'S RULE.

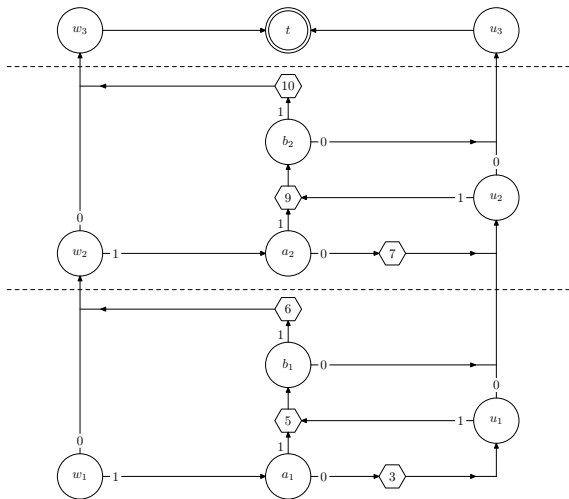
# Lower bound constructions

- For a given pivoting rules, we define a family of lower bound MDPs  $G_n$  such that the corresponding simplex algorithm simulates an  $n$ -bit binary counter.
  - To illustrate the main ideas for the lower bound for RANDOMEDGE I first show a simpler construction for BLAND'S RULE.
- Notation: Integer priority  $p$  corresponds to reward  $(-K)^p$ , where  $K = 3n + 1$ .

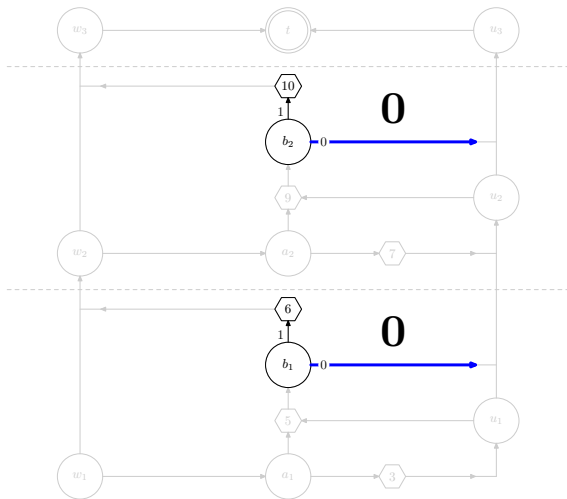
... < 5 < 3 < 1 < 2 < 4 < 6 < ...



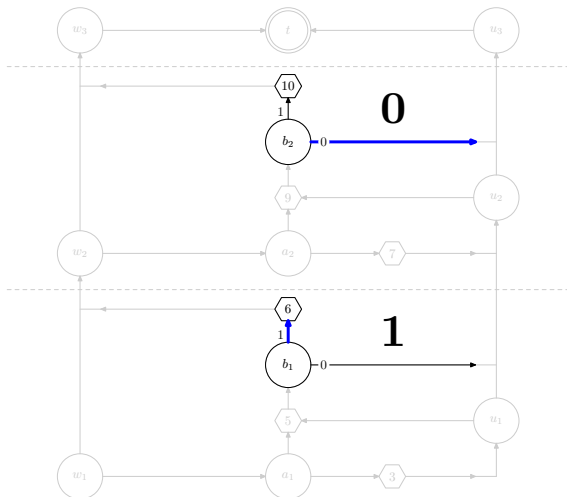
# Lower bound for BLAND'S RULE



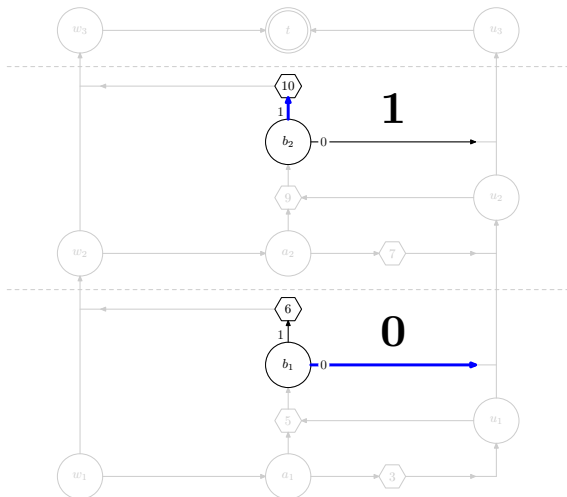
# Lower bound for BLAND'S RULE



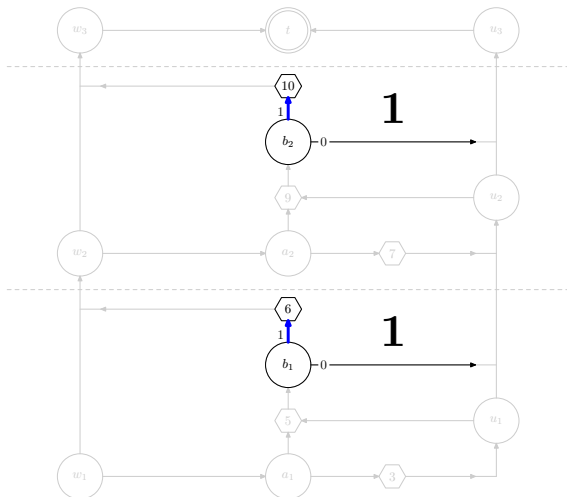
# Lower bound for BLAND'S RULE



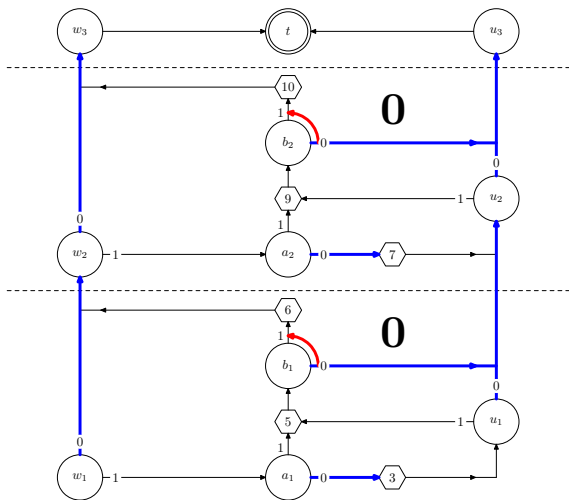
# Lower bound for BLAND'S RULE



# Lower bound for BLAND'S RULE



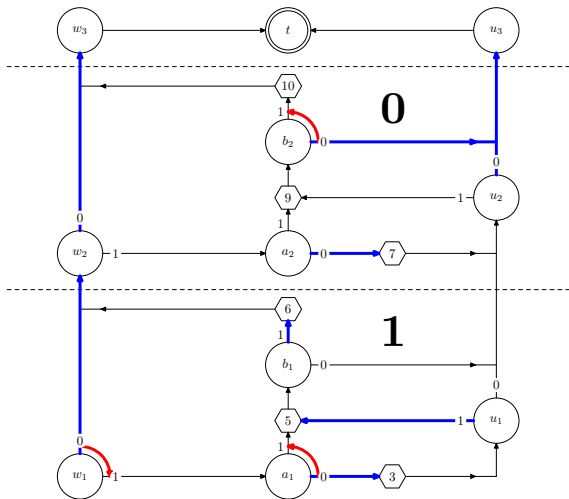
# Lower bound for BLAND'S RULE



Order:  $u_1^1, u_1^2, u_2^1, u_2^2, w_1^1, w_1^2, w_2^1, w_2^2, b_1^0, b_2^0, a_1^0, a_2^0, \underline{a_1^1, a_2^1, b_1^1, b_2^1}$

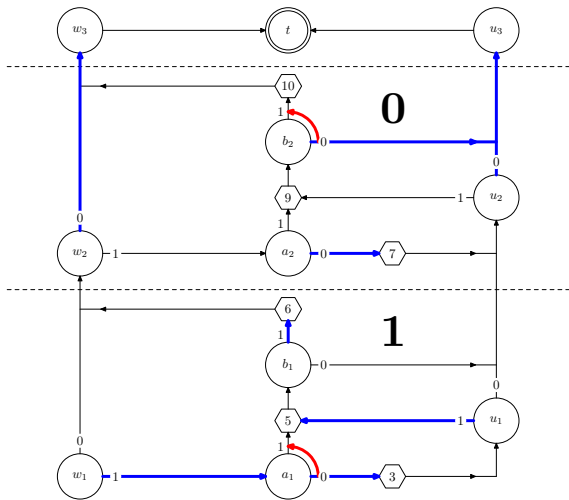


# Lower bound for BLAND'S RULE



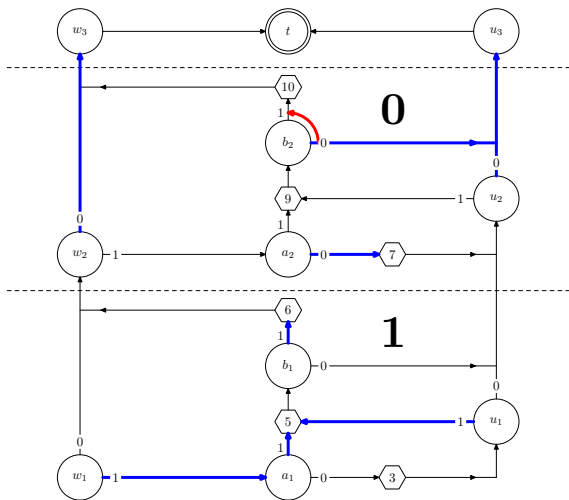
Order:  $u_1^1, u_1^2, u_2^1, u_2^2, w_1^1, w_1^2, w_2^1, w_2^2, b_1^0, b_2^0, a_1^0, a_2^0, \underline{a_1^1, a_2^1, b_1^1, b_2^1}$

# Lower bound for BLAND'S RULE



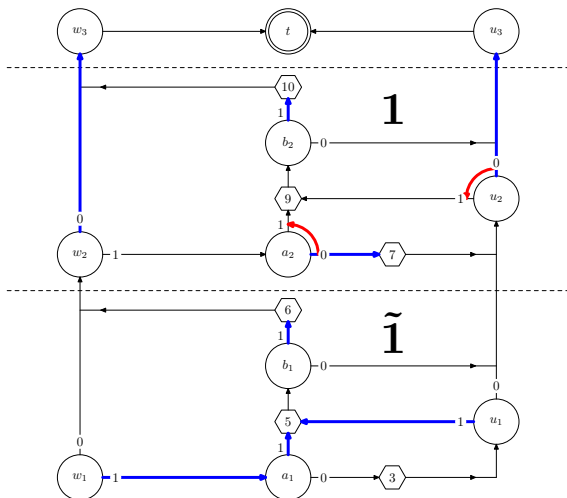
Order:  $u_1^1, u_1^2, u_2^1, u_2^2, w_1^1, w_1^2, w_2^1, w_2^2, b_1^0, b_2^0, a_1^0, a_2^0, \underline{a_1^1, a_2^1, b_1^1, b_2^1}$

# Lower bound for BLAND'S RULE



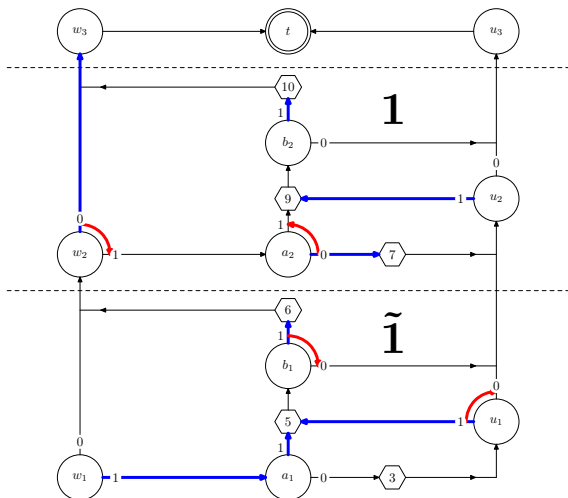
Order:  $u_1^1, u_1^2, u_2^1, u_2^2, w_1^1, w_1^2, w_2^1, w_2^2, b_1^0, b_2^0, a_1^0, a_2^0, \underline{a_1^1, a_2^1, b_1^1, b_2^1}$

# Lower bound for BLAND'S RULE



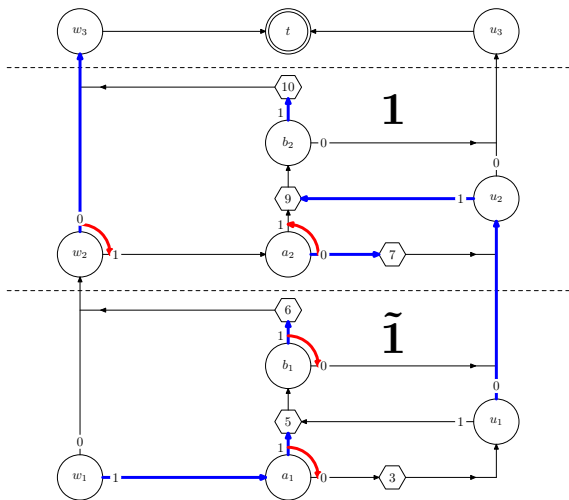
Order:  $u_1^1, u_1^2, u_2^1, u_2^2, w_1^1, w_1^2, w_2^1, w_2^2, b_1^0, b_2^0, a_1^0, a_2^0, \underline{a_1^1, a_2^1, b_1^1, b_2^1}$

# Lower bound for BLAND'S RULE



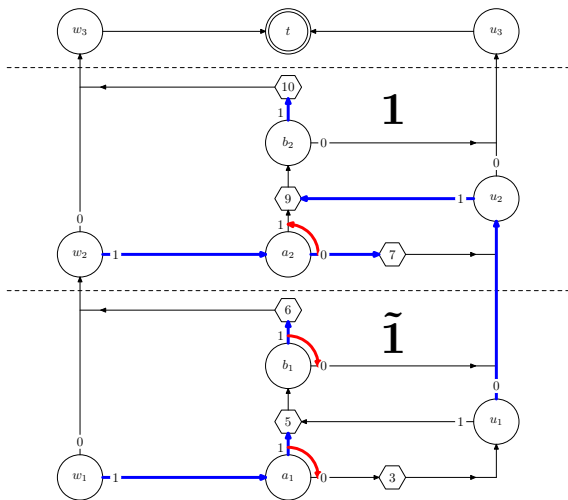
Order:  $u_1^1, u_1^2, u_2^1, u_2^2, w_1^1, w_1^2, w_2^1, w_2^2, b_1^0, b_2^0, a_1^0, a_2^0, \underline{a_1^1, a_2^1, b_1^1, b_2^1}$

# Lower bound for BLAND'S RULE



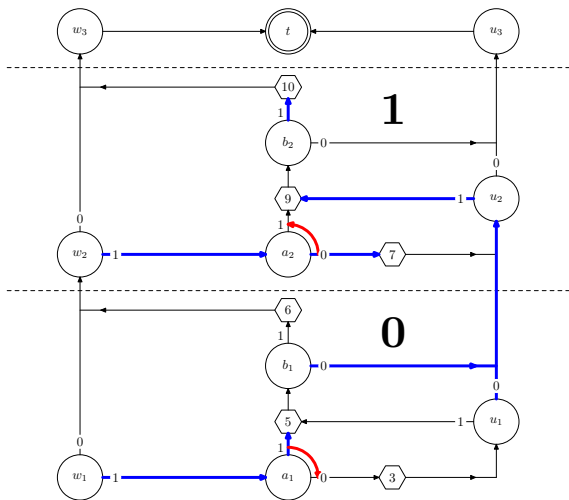
Order:  $u_1^1, u_1^2, u_2^1, u_2^2, w_1^1, w_1^2, w_2^1, w_2^2, b_1^0, b_2^0, a_1^0, a_2^0, \underline{a_1^1, a_2^1, b_1^1, b_2^1}$

# Lower bound for BLAND'S RULE



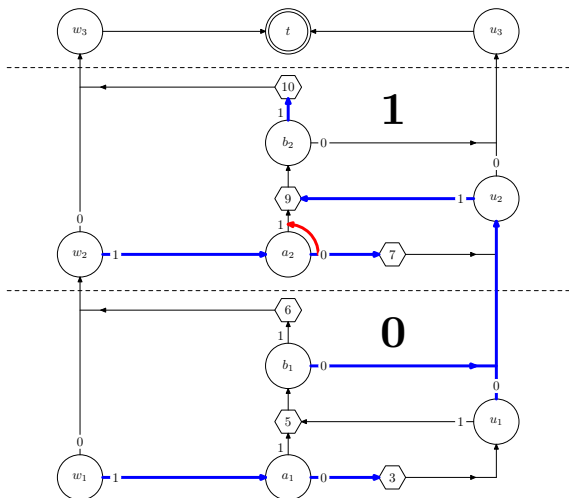
Order:  $u_1^1, u_1^2, u_2^1, u_2^2, w_1^1, w_1^2, w_2^1, w_2^2, b_1^0, b_2^0, a_1^0, a_2^0, \underline{a_1^1, a_2^1, b_1^1, b_2^1}$

# Lower bound for BLAND'S RULE



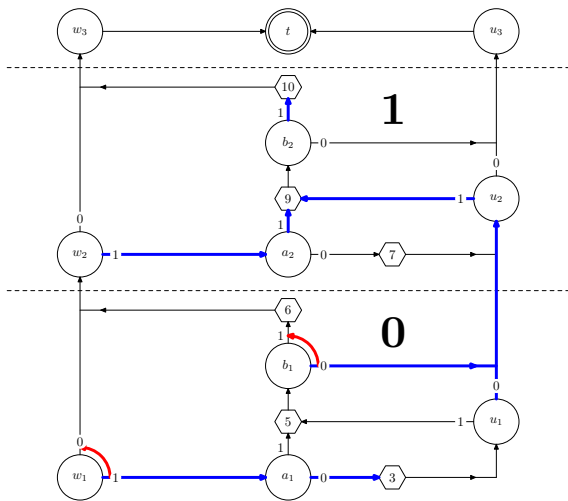
Order:  $u_1^1, u_1^2, u_2^1, u_2^2, w_1^1, w_1^2, w_2^1, w_2^2, b_1^0, b_2^0, a_1^0, a_2^0, \underline{a_1^1, a_2^1, b_1^1, b_2^1}$

# Lower bound for BLAND'S RULE



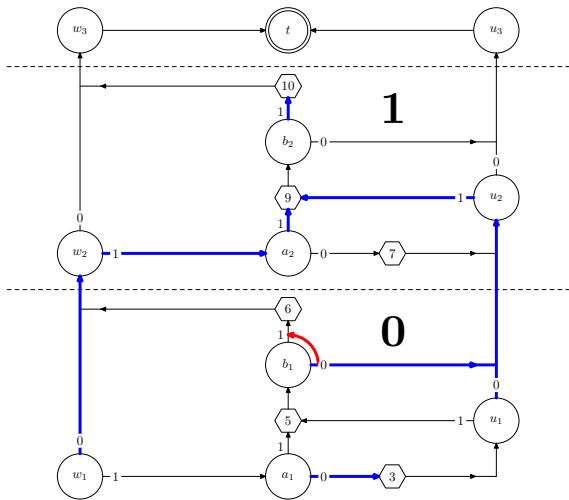
Order:  $u_1^1, u_1^2, u_2^1, u_2^2, w_1^1, w_1^2, w_2^1, w_2^2, b_1^0, b_2^0, a_1^0, a_2^0, \underline{a_1^1, a_2^1, b_1^1, b_2^1}$

# Lower bound for BLAND'S RULE



Order:  $u_1^1, u_1^2, u_2^1, u_2^2, w_1^1, w_1^2, w_2^1, w_2^2, b_1^0, b_2^0, a_1^0, a_2^0, \underline{a_1^1, a_2^1, b_1^1, b_2^1}$

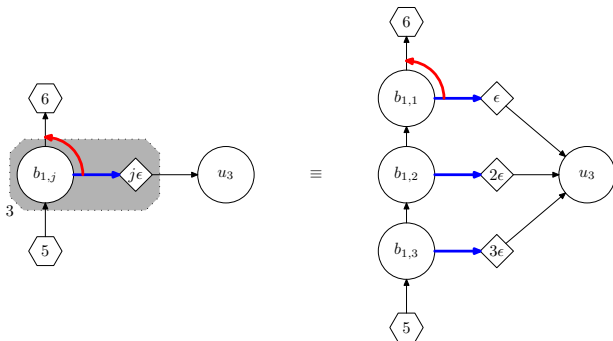
# Lower bound for BLAND'S RULE



Order:  $u_1^1, u_1^2, u_2^1, u_2^2, w_1^1, w_1^2, w_2^1, w_2^2, b_1^0, b_2^0, a_1^0, a_2^0, \underline{a_1^1, a_2^1, b_1^1, b_2^1}$

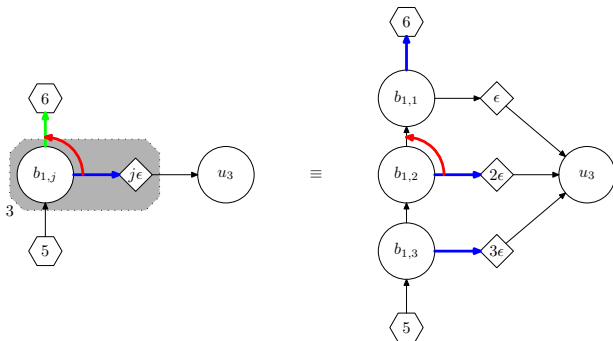
- Let  $k$  be the lowest unset bit. Incrementing the counter happened roughly through five phases:
  - ① Make  $b_k = 1$ .
  - ② Make  $u_k = 1$  and  $u_i = 0$ , for  $i < k$ .
  - ③ Make  $b_i = 0$  and  $a_i = 0$ , for  $i < k$ .
  - ④ Make  $a_k = 1$ .
  - ⑤ Make  $w_k = 1$  and  $w_i = 0$ , for  $i < k$ .
- Only the last part of the ordering, involving  $a_i^1$  and  $b_i^1$  edges, was important.
- To implement a lower bound for RANDOMEDGE we need a gadget to delay improving switches like  $a_i^1$  and  $b_i^1$ .

# Delaying events



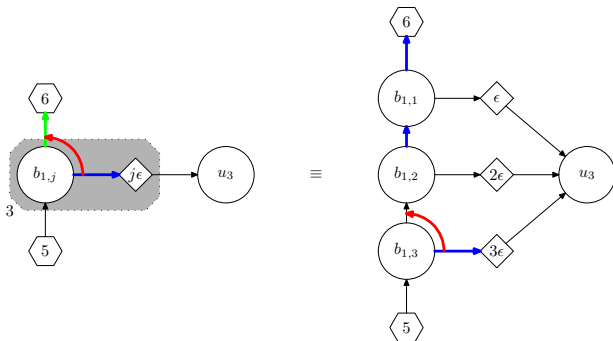
- By replacing a vertex by a chain of vertices, a specific sequence of improving switches has to be performed to get the same effect as performing one improving switch originally.
- At any time there is only one action for which it is improving to move into the chain.

# Delaying events



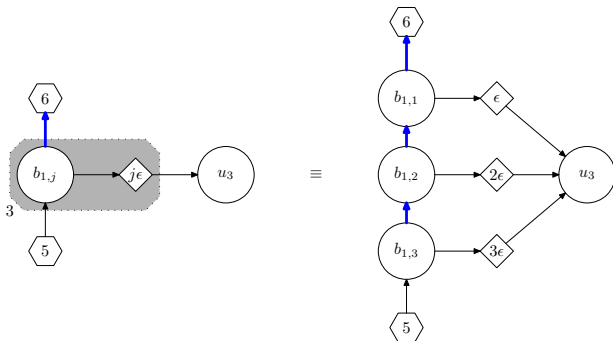
- By replacing a vertex by a chain of vertices, a specific sequence of improving switches has to be performed to get the same effect as performing one improving switch originally.
- At any time there is only one action for which it is improving to move into the chain.

# Delaying events



- By replacing a vertex by a chain of vertices, a specific sequence of improving switches has to be performed to get the same effect as performing one improving switch originally.
- At any time there is only one action for which it is improving to move into the chain.

# Delaying events



- By replacing a vertex by a chain of vertices, a specific sequence of improving switches has to be performed to get the same effect as performing one improving switch originally.
- At any time there is only one action for which it is improving to move into the chain.

# Competing chains

- Suppose a short chain of length  $\ell_i$  is competing with a longer chain of length  $\ell_{i+1}$ .
- There is exactly one improving switch in both chains, and RANDOMEDGE performs either one of them with equal probability.
- Let  $X$  be the number of heads observed in  $\ell_i + \ell_{i+1}$  coin tosses, then by a **Chernoff bound**:

$$\Pr[X \leq \ell_i] \leq e^{\frac{(\ell_{i+1} - \ell_i)^2}{2(\ell_{i+1} + \ell_i)}}$$

- Setting  $\ell_k = \Theta(k^2 n)$  the probability of failure,  $X < \ell_i$ , is at most  $e^{-n}$ .



# Competing chains

- Suppose a short chain of length  $\ell_i$  is competing with a longer chain of length  $\ell_{i+1}$ .
- There is exactly one improving switch in both chains, and RANDOMEDGE performs either one of them with equal probability.
- Let  $X$  be the number of heads observed in  $\ell_i + \ell_{i+1}$  coin tosses, then by a **Chernoff bound**:

$$Pr[X \leq \ell_i] \leq e^{\frac{(\ell_{i+1} - \ell_i)^2}{2(\ell_{i+1} + \ell_i)}}$$

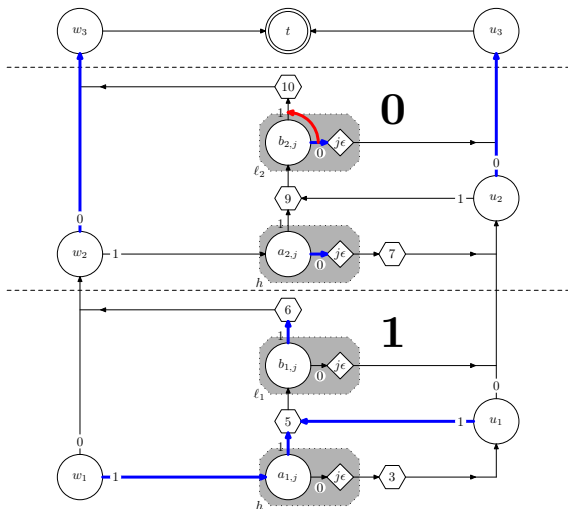
- Setting  $\ell_k = \Theta(k^2 n)$  the probability of failure,  $X < \ell_i$ , is at most  $e^{-n}$ .
- With  $n$  such chains this results in  $N = O(n^4)$  states, giving a lower bound of  $2^{\Omega(N^{1/4})}$  expected pivoting steps for RANDOMEDGE.



# RANDOMEDGE lower bound, first step

- Let  $k$  be the lowest unset bit.  
Incrementing the counter happens through five phases:

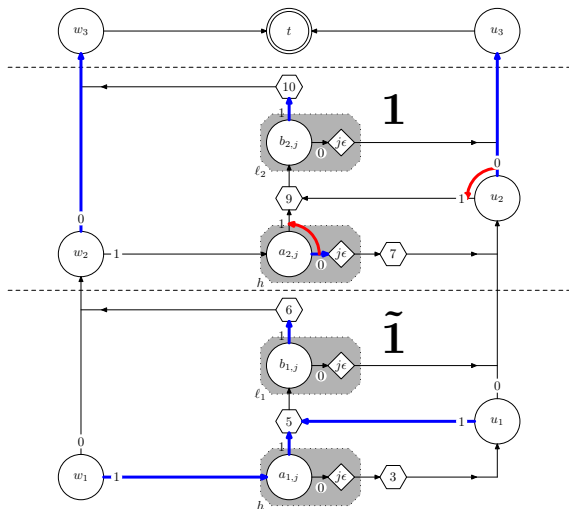
- ⇒
- 1 Make  $b_k = 1$ .
  - 2 Make  $u_k = 1$  and  $u_i = 0$ , for  $i < k$ .
  - 3 Make  $b_i = 0$  and  $a_i = 0$ , for  $i < k$ .
  - 4 Make  $a_k = 1$ .
  - 5 Make  $w_k = 1$  and  $w_i = 0$ , for  $i < k$ .



# RANDOMEDGE lower bound, first step

- Let  $k$  be the lowest unset bit.  
Incrementing the counter happens through five phases:

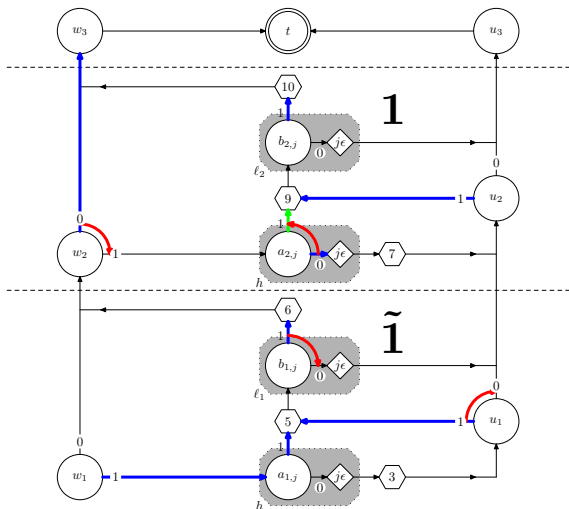
- 1 Make  $b_k = 1$ .
- ⇒ 2 Make  $u_k = 1$  and  $u_i = 0$ , for  $i < k$ .
- 3 Make  $b_i = 0$  and  $a_i = 0$ , for  $i < k$ .
- 4 Make  $a_k = 1$ .
- 5 Make  $w_k = 1$  and  $w_i = 0$ , for  $i < k$ .



# RANDOMEDGE lower bound, first step

- Let  $k$  be the lowest unset bit. Incrementing the counter happens through five phases:

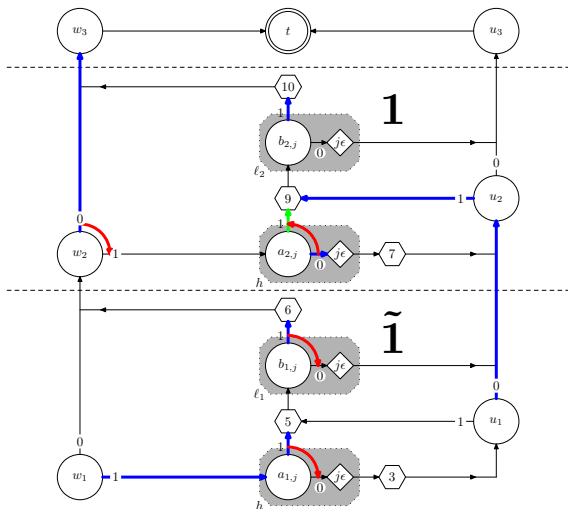
- 1 Make  $b_k = 1$ .
- ⇒ 2 Make  $u_k = 1$  and  $u_i = 0$ , for  $i < k$ .
- 3 Make  $b_i = 0$  and  $a_i = 0$ , for  $i < k$ .
- 4 Make  $a_k = 1$ .
- 5 Make  $w_k = 1$  and  $w_i = 0$ , for  $i < k$ .



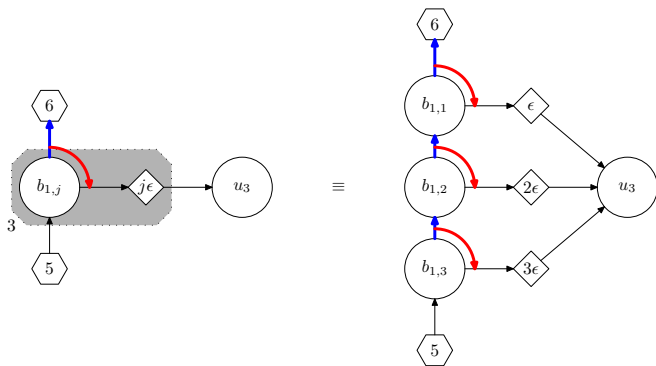
# RANDOMEDGE lower bound, first step

- Let  $k$  be the lowest unset bit. Incrementing the counter happens through five phases:

- ① Make  $b_k = 1$ .
- ② Make  $u_k = 1$  and  $u_i = 0$ , for  $i < k$ .
- ⇒ ③ Make  $b_i = 0$  and  $a_i = 0$ , for  $i < k$ .
- ④ Make  $a_k = 1$ .
- ⑤ Make  $w_k = 1$  and  $w_i = 0$ , for  $i < k$ .



# Fast resetting

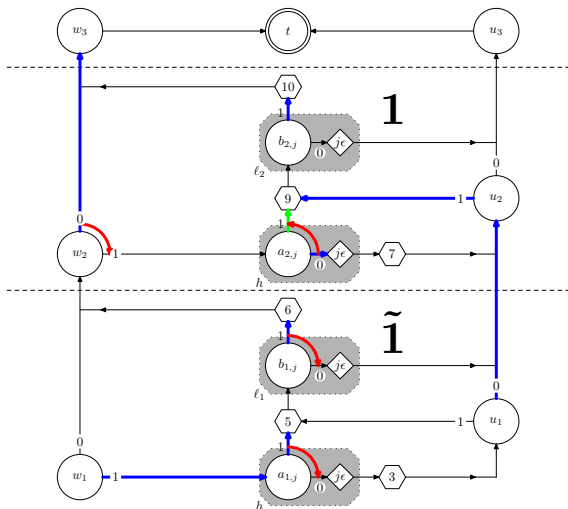


- Moving in the other directions happens much faster since all actions are improving switches simultaneously.

# RANDOMEDGE lower bound, first step

- Let  $k$  be the lowest unset bit. Incrementing the counter happens through five phases:

- ① Make  $b_k = 1$ .
- ② Make  $u_k = 1$  and  $u_i = 0$ , for  $i < k$ .
- ⇒ ③ Make  $b_i = 0$  and  $a_i = 0$ , for  $i < k$ .
- ④ Make  $a_k = 1$ .
- ⑤ Make  $w_k = 1$  and  $w_i = 0$ , for  $i < k$ .



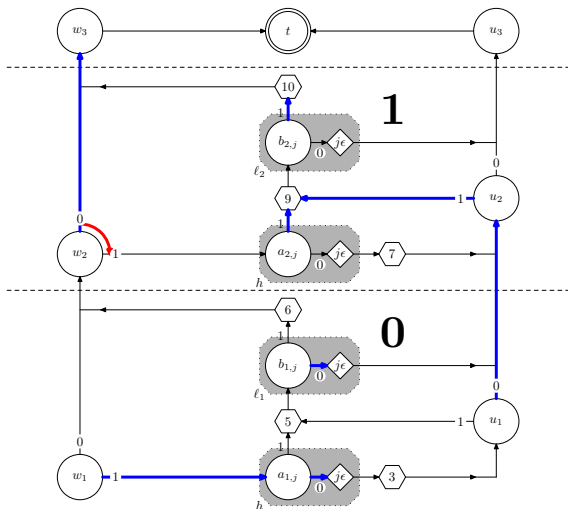




# RANDOMEDGE lower bound, first step

- Let  $k$  be the lowest unset bit. Incrementing the counter happens through five phases:

- 1 Make  $b_k = 1$ .
  - 2 Make  $u_k = 1$  and  $u_i = 0$ , for  $i < k$ .
  - 3 Make  $b_i = 0$  and  $a_i = 0$ , for  $i < k$ .
  - 4 Make  $a_k = 1$ .
  - 5 Make  $w_k = 1$  and  $w_i = 0$ , for  $i < k$ .
- ⇒

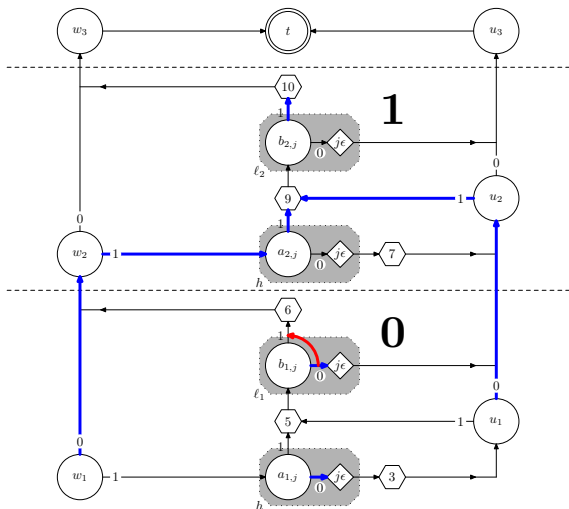




# RANDOMEDGE lower bound, first step

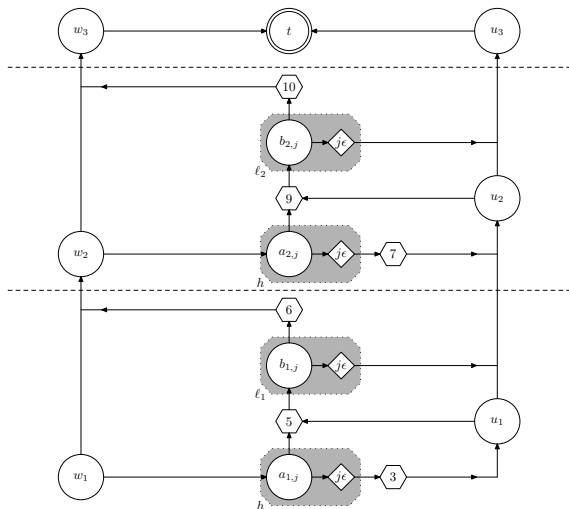
- Let  $k$  be the lowest unset bit.  
Incrementing the counter happens through five phases:

- ⇒
- 1 Make  $b_k = 1$ .
  - 2 Make  $u_k = 1$  and  $u_i = 0$ , for  $i < k$ .
  - 3 Make  $b_i = 0$  and  $a_i = 0$ , for  $i < k$ .
  - 4 Make  $a_k = 1$ .
  - 5 Make  $w_k = 1$  and  $w_i = 0$ , for  $i < k$ .



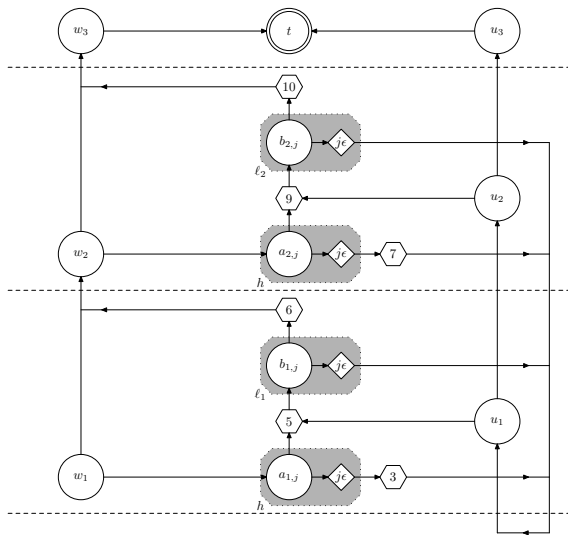
# RANDOMEDGE lower bound, full construction

- It remains to reset partially set higher bits to get a “fresh start”.



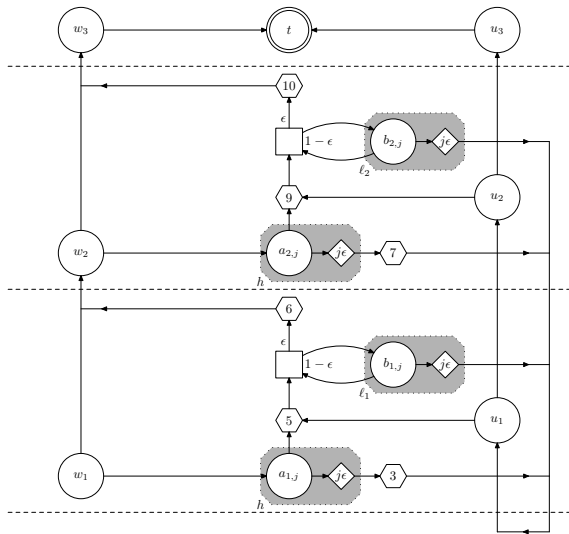
# RANDOMEDGE lower bound, full construction

- It remains to reset partially set higher bits to get a “fresh start”.



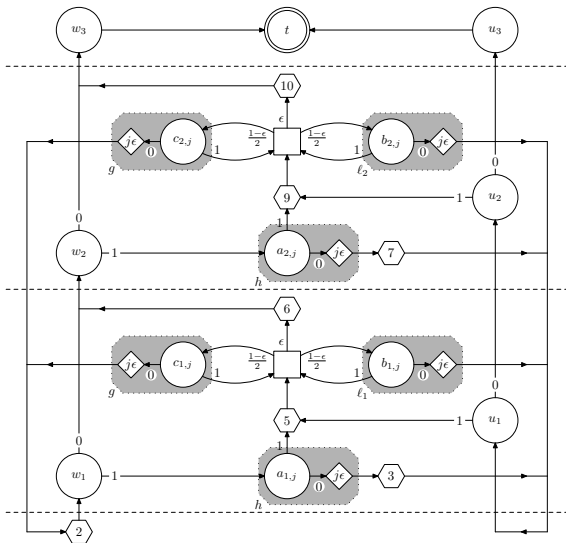
# RANDOMEDGE lower bound, full construction

- It remains to reset partially set higher bits to get a “fresh start”.



# RANDOMEDGE lower bound, full construction

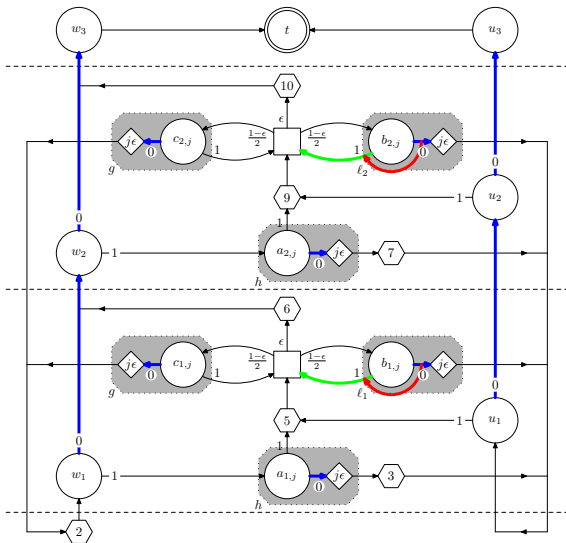
- It remains to reset partially set higher bits to get a “fresh start”.



# RANDOMEDGE lower bound, full construction

- It remains to reset partially set higher bits to get a “fresh start”.

- ⇒
- 1 Make  $b_k = 1$ .
  - 2 Make  $c_k = 1$ .
  - 3 Make  $u_k = 1$  and  $u_i = 0$ , for  $i < k$ .
  - 4 Make  $b_i = 0$  and  $a_i = 0$ , for  $i < k$ . Reset  $b_i$  for unset bits  $i > k$ .
  - 5 Make  $a_k = 1$ .
  - 6 Make  $w_k = 1$  and  $w_i = 0$ , for  $i < k$ .
  - 7 Reset  $c_i$  for all unset bits.



Theorem (Friedmann, Hansen and Zwick (2011))

*The worst-case expected number of pivoting steps performed by RANDOMEDGE on linear programs with  $n$  equalities and  $2n$  non-negative variables is  $2^{\Omega(n^{1/4})}$ .*

# The linear program

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^n \sum_{j=1}^h ((-K)^{4i-1} + j\epsilon) a_{i,j}^0 + \\ & \sum_{i=1}^n ((-K)^{4i+1} + \epsilon(-K)^{4i+2})(a_{i,1}^1 + u_i^1) + \\ & \sum_{i=1}^n (\epsilon(-K)^{4i+2})(b_{i,1}^1 + c_{i,1}^1) + \\ & \sum_{i=1}^n \sum_{j=1}^{\ell_i} j\epsilon b_{i,j}^0 + \sum_{i=1}^n \sum_{j=1}^g j\epsilon c_{i,j}^0 \end{aligned}$$

subject to

$$\begin{aligned} \forall 1 \leq i \leq n : \quad & a_{i,h}^0 + a_{i,h}^1 = 1 + w_i^1 \\ \forall 1 \leq i \leq n, \forall 1 \leq j < h : \quad & a_{i,j}^0 + a_{i,j}^1 = 1 + a_{i,j+1}^1 \\ \forall 1 \leq i \leq n : \quad & b_{i,\ell_i}^0 + b_{i,\ell_i}^1 = 1 + \frac{1-\epsilon}{2} (a_{i,1}^1 + b_{i,1}^1 + c_{i,1}^1 + u_i^1) \\ \forall 1 \leq i \leq n, \forall 1 \leq j < \ell_i : \quad & b_{i,j}^0 + b_{i,j}^1 = 1 + b_{i,j+1}^1 \\ \forall 1 \leq i \leq n : \quad & c_{i,g}^0 + c_{i,g}^1 = 1 + \frac{1-\epsilon}{2} (a_{i,1}^1 + b_{i,1}^1 + c_{i,1}^1 + u_i^1) \\ \forall 1 \leq i \leq n, \forall 1 \leq j < g : \quad & c_{i,j}^0 + c_{i,j}^1 = 1 + c_{i,j+1}^1 \\ & u_1^0 + u_1^1 = 1 + \sum_{i=1}^n \left( \sum_{j=1}^h a_{i,j}^0 + \sum_{j=1}^{\ell_i} b_{i,j}^0 \right) \\ \forall 2 \leq i \leq n : \quad & u_i^0 + u_i^1 = 1 + u_{i-1}^0 \\ & w_1^0 + w_1^1 = 1 + \sum_{i=1}^n \sum_{j=1}^g c_{i,j}^0 \\ \forall 2 \leq i \leq n : \quad & w_i^0 + w_i^1 = 1 + w_{i-1}^0 + \epsilon (a_{i,1}^1 + b_{i,1}^1 + c_{i,1}^1 + u_i^1) \end{aligned}$$

$$a_{i,j}^0, a_{i,j}^1, b_{i,j}^0, b_{i,j}^1, c_{i,j}^0, c_{i,j}^1, u_i^0, u_i^1, w_i^0, w_i^1 \geq 0$$

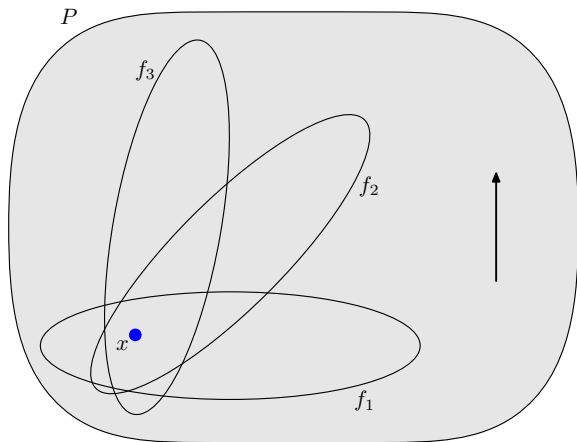
- Linear programming and the simplex algorithm.
  - Markov decision processes and the connection to linear programming.
  - Lower bounds for the simplex algorithm utilizing Markov decision processes.
    - Example: A lower bound for BLAND'S RULE.
    - Gadgets and general ideas for the lower bound for RANDOMEDGE.
- ⇒ ● On lower bounds for the RANDOMFACET pivoting rule and the RANDOMIZED BLAND'S RULE.

# The RANDOMFACET pivoting rule

- RANDOMFACET, Kalai (1992):
  - ① Pick a uniformly random facet  $f$  that contains the current basic feasible solution  $x$ .
  - ② Recursively find the optimal solution  $x'$  within the picked facet  $f$ .
  - ③ If possible, make an improving pivot from  $x'$ , leaving the facet  $f$ , and repeat from (1). Otherwise return  $x'$ .

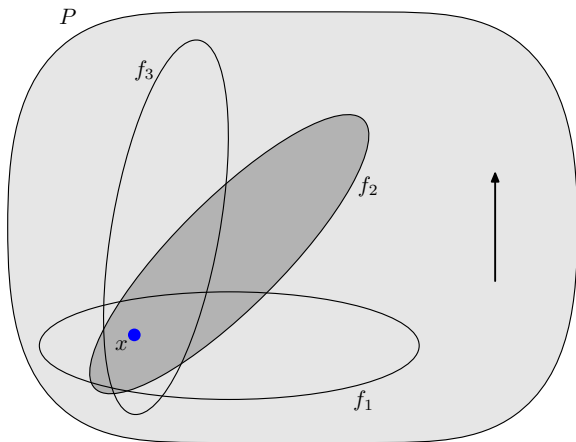
- RANDOMFACET, Kalai (1992):
  - ① Pick a uniformly random facet  $f$  that contains the current basic feasible solution  $x$ .
  - ② Recursively find the optimal solution  $x'$  within the picked facet  $f$ .
  - ③ If possible, make an improving pivot from  $x'$ , leaving the facet  $f$ , and repeat from (1). Otherwise return  $x'$ .
- A **dual** variant of the RANDOMFACET pivoting rule was discovered independently by Matoušek, Sharir and Welzl (1992).

# The RANDOMFACET pivoting rule



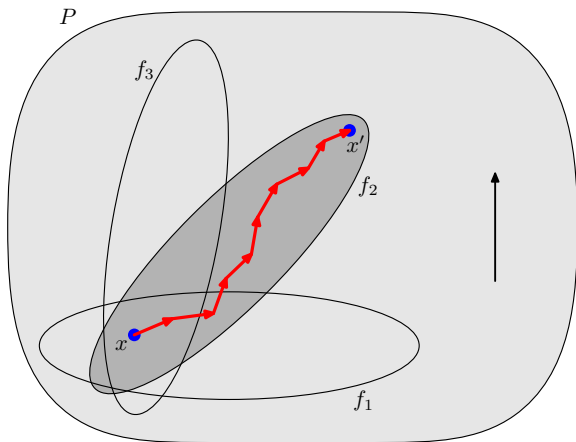
- Pick a uniformly random facet  $f_i$  that contains the current basic feasible solution  $x$ .

# The RANDOMFACET pivoting rule



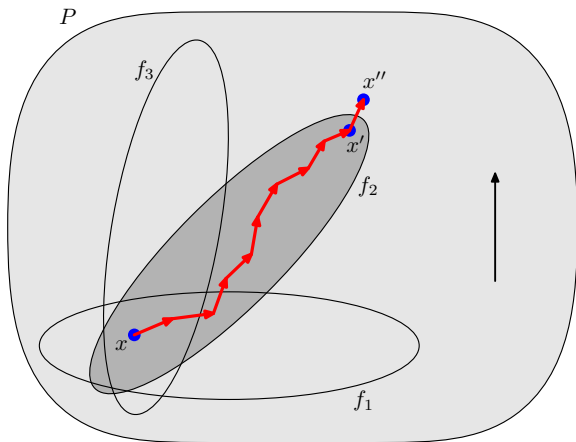
- Pick a uniformly random facet  $f_i$  that contains the current basic feasible solution  $x$ .

# The RANDOMFACET pivoting rule



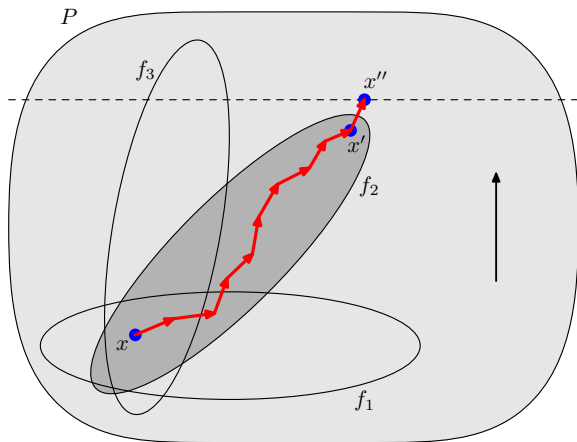
- Recursively find the optimal solution  $x'$  within the picked facet  $f_i$ .

# The RANDOMFACET pivoting rule



- If possible, make an improving pivot from  $x'$ , leaving the facet  $f_i$ , and repeat from the beginning. Otherwise return  $x'$ .

# The RANDOMFACET pivoting rule



- Note that if the facets  $f_1, \dots, f_d$  containing  $x$  are ordered according to their optimal value, then from  $x''$  we never visit  $f_1, \dots, f_i$  again.

# The RANDOMFACET pivoting rule

- The number of pivoting steps for a linear program with  $d$  variables and  $n$  constraints, including non-negativity constraints, is at most:

$$f(d, n) \leq f(d - 1, n - 1) + 1 + \frac{1}{d} \sum_{i=1}^d f(d, n - i)$$

with  $f(d, n) = 0$  for  $n \leq d$ .

# The RANDOMFACET pivoting rule

- The number of pivoting steps for a linear program with  $d$  variables and  $n$  constraints, including non-negativity constraints, is at most:

$$f(d, n) \leq f(d-1, n-1) + 1 + \frac{1}{d} \sum_{i=1}^d f(d, n-i)$$

with  $f(d, n) = 0$  for  $n \leq d$ .

- Solving the corresponding recurrence gives:

$$f(d, n) \leq 2^{O(\sqrt{(n-d) \log n})}$$

$$\begin{aligned} & \text{maximize} && \sum_{i \in S} \sum_{a \in A_i} c_a x_a \\ & \text{s.t.} && \forall i \in S : \sum_{a \in A_i} x_a = 1 + \sum_{j \in S} \sum_{a \in A_j} P_{a,i} x_a \\ & && \forall i \in S, \forall a \in A_i : x_a \geq 0 \end{aligned}$$

- Staying within a facet means that the corresponding inequality is tight, meaning that a variable is fixed to zero. This corresponds to removing the action.
- The `RANDOMFACET` pivoting rule removes random unused actions and solves the corresponding MDP recursively.

# The RANDOMIZED BLAND'S RULE

- The non-basic variable with the largest index is not included in the basis until there are no other improving pivots.
- This corresponds to staying within the facet until the optimal vertex is reached, which for MDPs corresponds to removing the action.

- When constructing lower bounds for RANDOMFACET and RANDOMIZED BLAND'S RULE, instead of delaying improving switches, the challenge is to make sure that certain actions are not removed before certain other actions.

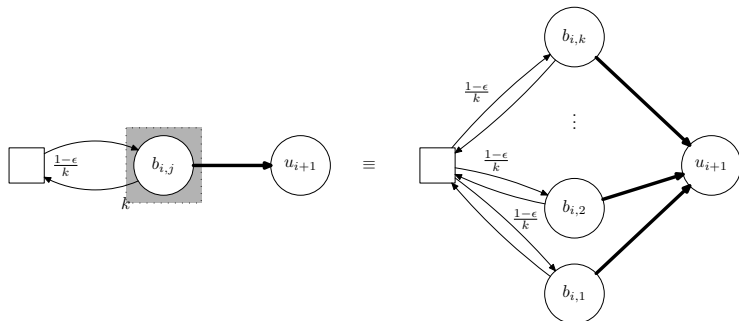
- When constructing lower bounds for RANDOMFACET and RANDOMIZED BLAND'S RULE, instead of delaying improving switches, the challenge is to make sure that certain actions are not removed before certain other actions.
- Suppose an action  $a$  must not be removed before another action  $b$ .

- When constructing lower bounds for RANDOMFACET and RANDOMIZED BLAND'S RULE, instead of delaying improving switches, the challenge is to make sure that certain actions are not removed before certain other actions.
- Suppose an action  $a$  must not be removed before another action  $b$ .
- To achieve this with high probability we make use of redundancy: Let  $a$  and  $b$  be copied  $k$  times, in such a way that we only require that at least one copy of  $b$  is removed before all copies of  $a$  are removed.

- When constructing lower bounds for RANDOMFACET and RANDOMIZED BLAND'S RULE, instead of delaying improving switches, the challenge is to make sure that certain actions are not removed before certain other actions.
- Suppose an action  $a$  must not be removed before another action  $b$ .
- To achieve this with high probability we make use of redundancy: Let  $a$  and  $b$  be copied  $k$  times, in such a way that we only require that at least one copy of  $b$  is removed before all copies of  $a$  are removed.
- The only *bad* permutation for RANDOMIZED BLAND'S RULE is then:  $aa \dots abb \dots b$

- When constructing lower bounds for RANDOMFACET and RANDOMIZED BLAND'S RULE, instead of delaying improving switches, the challenge is to make sure that certain actions are not removed before certain other actions.
- Suppose an action  $a$  must not be removed before another action  $b$ .
- To achieve this with high probability we make use of redundancy: Let  $a$  and  $b$  be copied  $k$  times, in such a way that we only require that at least one copy of  $b$  is removed before all copies of  $a$  are removed.
- The only *bad* permutation for RANDOMIZED BLAND'S RULE is then:  $aa \dots abb \dots b$
- The probability of choosing a bad permutation is  $\frac{(k!)^2}{(2k)!} \leq \frac{1}{2^k}$ .

# Different gadgets



- We use different gadgets to ensure that we get the correct behaviour with high probability.
- Bold edges indicate multiple redundant actions.



- We prove the same subexponential lower bounds for RANDOMEDGE, RANDOMFACET and the RANDOMIZED BLAND'S RULE when used to solve *parity games*.

- We prove the same subexponential lower bounds for RANDOMEDGE, RANDOMFACET and the RANDOMIZED BLAND'S RULE when used to solve *parity games*.
- **Open problem:** Strongly polynomial time algorithm for solving MDPs?

- We prove the same subexponential lower bounds for RANDOMEDGE, RANDOMFACET and the RANDOMIZED BLAND'S RULE when used to solve *parity games*.
- **Open problem:** Strongly polynomial time algorithm for solving MDPs?
- **Open problem:** Subexponential upper bound for the RANDOMIZED BLAND'S RULE?

- We prove the same subexponential lower bounds for RANDOMEDGE, RANDOMFACET and the RANDOMIZED BLAND'S RULE when used to solve *parity games*.
- **Open problem:** Strongly polynomial time algorithm for solving MDPs?
- **Open problem:** Subexponential upper bound for the RANDOMIZED BLAND'S RULE?
- **Open problem:** Prove or disprove the polynomial Hirsch conjecture to settle whether there is hope for a polynomial pivoting rule for the simplex algorithm.
  - Check out the Polymath3 project on Gil Kalai's blog.

Thank you for listening!