Optimal Selection via Dynamic Programming

Micha Hofri

Department of Computer Science, WPI

Optimal median selection

Efficient selection is a classic, difficult problem.

A deterministic algorithm for the exact median last was improved in 1999 by Dor & Zwick, requiring (in the worst case) $\approx 2.942n$ comparisons.

Extremely involved . . .

Optimal median selection

Efficient selection is a classic, difficult problem.

A deterministic algorithm for the exact median last was improved in 1999 by Dor & Zwick, requiring (in the worst case) $\approx 2.942n$ comparisons.

Extremely involved . . .

For the expected number of comparisons: Floyd & Rivest showed (1975) it can be done in $(1.5 + o(1))n$.

Cunto & Munro (1989): this bound is tight.

Optimal median selection

Efficient selection is a classic, difficult problem.

A deterministic algorithm for the exact median last was improved in 1999 by Dor & Zwick, requiring (in the worst case) $\approx 2.942n$ comparisons.

Extremely involved . . .

For the expected number of comparisons: Floyd & Rivest showed (1975) it can be done in $(1.5 + o(1))n$.

Cunto & Munro (1989): this bound is tight.

Numerous improvements; some recent. However:

The optimality of these algorithms is asymptotic.

None is acceptable for small sets.

Sicilian median selection

Developed in 1998 by Cantone — and later we discovered that a number of people formulated several analogues earlier — as early as 1978!

Sicilian median selection

Developed in 1998 by Cantone — and later we discovered that a number of people formulated several analogues earlier — as early as 1978!

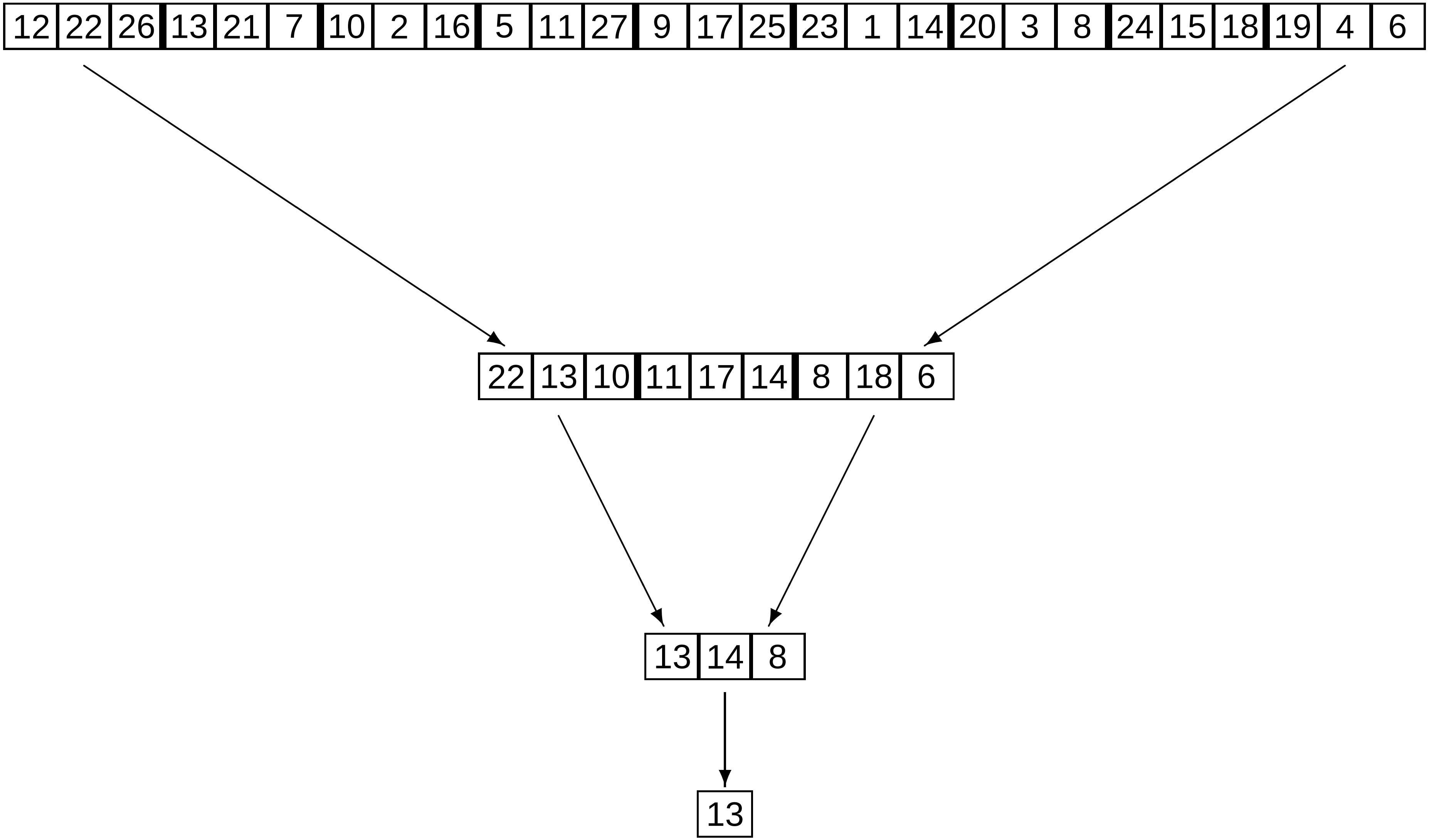
The algorithm is deterministic, and one of its versions uses at most $1.5n$ comparisons,

The expected number is $4/3n$.

Extremely efficient, and easy to implement;

but it only approximates the median.

Example


12 22 26 13 21 7 10 2 16 5 11 27 9 17 25 23 1 14 20 3 8 24 15 18 19 4 6
22 13 10 11 17 14 8 18 6
13 14 8
13

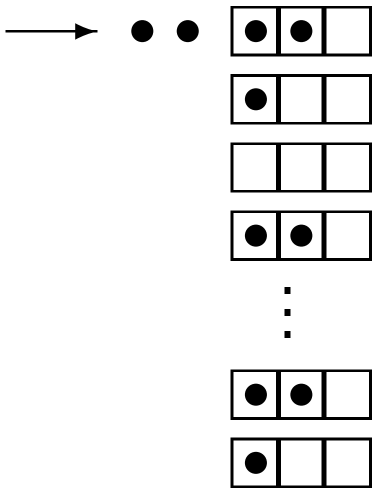
A Sicilian algorithm variant

The procedure above is performed in situ.

A Sicilian algorithm variant

The procedure above is performed in situ.

Essentially the same algorithm can be done "on-line," processing streaming data.

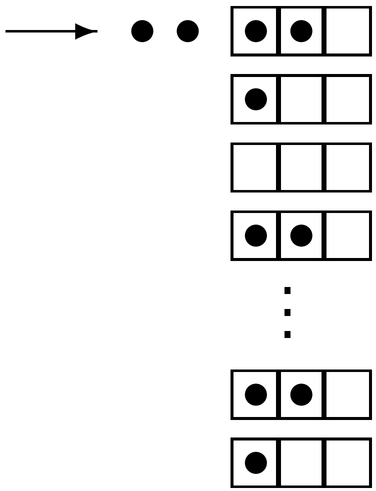


For n entries we need work-area of $4 \log_3 n$ positions;

A Sicilian algorithm variant

The procedure above is performed in situ.

Essentially the same algorithm can be done “on-line,” processing streaming data.



For n entries we need work-area of $4\log_3 n$ positions;

or $(b+1)\log_b n$ when using buffers of size b.

Cost of search – in comparisons

What if we use larger groups then $b = 3$, *e.g.* 5? 7? 9?

Cost of search – in comparisons

What if we use larger groups then $b = 3$, *e.g.* 5? 7? 9?

The expected total number of comparisons when looking

in a list of size n is $\boxed{C_b n,}$ where $C_b = \overline{V}_m(b)/(b-1)$

b	3	5	7	9
$\overline{V}_m(b)$	2.667	5.867	9.305	12.790
$V_m(b)$	3	6	10	14
C_b	1.333	1.4667	1.3293	1.5987

Cost of search – in comparisons

What if we use larger groups then $b = 3$, *e.g.* 5? 7? 9?

The expected total number of comparisons when looking

in a list of size n is $\boxed{C_b n,}$ where $C_b = \overline{V}_m(b)/(b-1)$

b	3	5	7	9
$\overline{V}_m(b)$	2.667	5.867	9.305	12.790
$V_m(b)$	3	6	10	14
C_b	1.333	1.4667	1.3293	1.5987

Here is why: Say $n = b^r$. We need $n/b,\ n/b^2, \ldots n/b^r = 1$

exact medians. The expected cost is

$$n\overline{V}_m(b) \sum_{j=1}^{r} b^{-j} \approx n\overline{V}_m(b)/(b-1) \stackrel{\text{def}}{=} nC_b$$

Searching for a selection algorithm

How to do it? How to locate fast—on the average—the median for $b = 5, 7, 9$? nothing much that I could find, except. . .

Knuth provides a table with these numbers, no algorithm or proof, nothing about derivation; but he has some insights:

Searching for a selection algorithm

How to do it? How to locate fast—on the average—the median for $b = 5, 7, 9$? nothing much that I could find, except...

Knuth provides a table with these numbers, no algorithm or proof, nothing about derivation; but he has some insights:

— Hard problem, mean-optimal much harder;
— different algorithm needed for each size.

Searching for a selection algorithm

How to do it? How to locate fast—on the average—the median for $b = 5, 7, 9$? nothing much that I could find, except. . .

Knuth provides a table with these numbers, no algorithm or proof, nothing about derivation; but he has some insights:

— Hard problem, mean-optimal much harder;
— different algorithm needed for each size.

Formal derivation needs notation:

S – the information state about the entries, a digraph.
Edge from large to small.

$U(S)$ – the expected cost to complete the selection, starting at state S.

$\mathscr{S}_r$ – the set of "resolved" states, which hold enough information to determine the selection.

$C(S)$ – set of possible actions at state S.

Searching for a selection algorithm

With this notation we have the natural recurrence (MDP)

$$U(S) = 1 + \min_{\varepsilon \in C(S)} E_\varepsilon U(S \mid \varepsilon), \qquad S \notin \mathscr{S}_r, \tag{1}$$

The expectation is on the outcomes of the action, and 1 is its cost.

Searching for a selection algorithm

With this notation we have the natural recurrence (MDP)

$$U(S) = 1 + \min_{\varepsilon \in C(S)} E_\varepsilon U(S \mid \varepsilon), \qquad S \notin \mathscr{S}_r, \tag{3}$$

The expectation is on the outcomes of the action, and 1 is its cost. The definition of $U(S)$ is completed with

$$U(S) = 0, \qquad S \in \mathscr{S}_r. \tag{4}$$

Searching for a selection algorithm

With this notation we have the natural recurrence (MDP)

$$U(S) = 1 + \min_{\varepsilon \in C(S)} E_\varepsilon U(S \mid \varepsilon), \qquad S \notin \mathscr{S}_r, \tag{5}$$

The expectation is on the outcomes of the action, and 1 is its cost. The definition of $U(S)$ is completed with

$$U(S) = 0, \qquad S \in \mathscr{S}_r. \tag{6}$$

This is a Bellman optimality equation. Satisfies the conditions for the final value $U(S_0)$ to be the optimal cost of resolving S_0.

Any algorithm read off the optimization network is admissible.

Searching for a selection algorithm

With this notation we have the natural recurrence (MDP)

$$U(S) = 1 + \min_{\varepsilon \in C(S)} E_\varepsilon U(S \mid \varepsilon), \qquad S \notin \mathscr{S}_r, \tag{7}$$

The expectation is on the outcomes of the action, and 1 is its cost. The definition of $U(S)$ is completed with

$$U(S) = 0, \qquad S \in \mathscr{S}_r. \tag{8}$$

This is a Bellman optimality equation. Satisfies the conditions for the final value $U(S_0)$ to be the optimal cost of resolving S_0.

Any algorithm read off the optimization network is admissible.

This recurrence is more versatile than may appear.

Optimality in worst-case behavior

If we change the operator E_{ε} for $\max_{\varepsilon}$ we see,

$$U(S) = 1 + \min_{\varepsilon \in C(S)} \max_{\varepsilon} U(S \mid \varepsilon), \qquad S \notin \mathscr{S}_r. \tag{9}$$

This is a recurrence for the optimal worst-case cost, and policy! The same criteria determine termination.

Optimality in worst-case behavior

If we change the operator E_ε for $\max_\varepsilon$ we see,

$$U(S) = 1 + \min_{\varepsilon \in C(S)} \max_{\varepsilon} U(S \mid \varepsilon), \qquad S \notin \mathscr{S}_r. \tag{10}$$

This is a recurrence for the optimal worst-case cost, and policy! The same criteria determine termination.

How can such a recurrence be solved?

Optimality in worst-case behavior

If we change the operator E_ε for $\max_\varepsilon$ we see,

$$U(S) = 1 + \min_{\varepsilon \in C(S)} \max_{\varepsilon} U(S \mid \varepsilon), \qquad S \notin \mathscr{S}_r. \tag{11}$$

This is a recurrence for the optimal worst-case cost, and policy! The same criteria determine termination.

How can such a recurrence be solved?

For optimality — we need to consider (implicitly?) all possible algorithms and choose the best; no guide except the recurrence.

Optimality in worst-case behavior

If we change the operator E_ε for $\max_\varepsilon$ we see,

$$U(S) = 1 + \min_{\varepsilon \in C(S)} \max_\varepsilon U(S \mid \varepsilon), \qquad S \notin \mathscr{S}_r. \tag{12}$$

This is a recurrence for the optimal worst-case cost, and policy! The same criteria determine termination.

How can such a recurrence be solved?

For optimality — we need to consider (implicitly?) all possible algorithms and choose the best; no guide except the recurrence.

There are conjectures; none proved.

Optimality in worst-case behavior

If we change the operator E_ε for $\max_\varepsilon$ we see,

$$U(S) = 1 + \min_{\varepsilon \in C(S)} \max_{\varepsilon} U(S \mid \varepsilon), \qquad S \notin \mathscr{S}_r. \tag{13}$$

This is a recurrence for the optimal worst-case cost, and policy! The same criteria determine termination.

How can such a recurrence be solved?

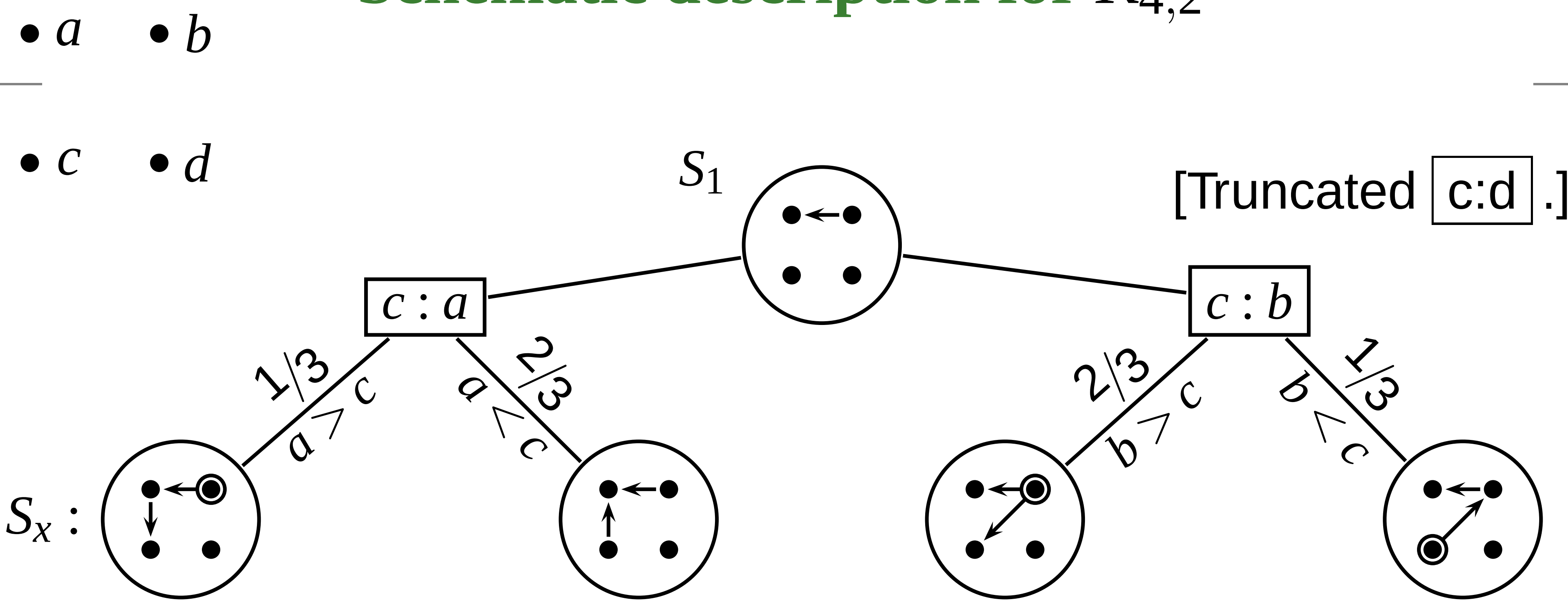
For optimality — we need to consider (implicitly?) all possible algorithms and choose the best; no guide except the recurrence.

There are conjectures; none proved.

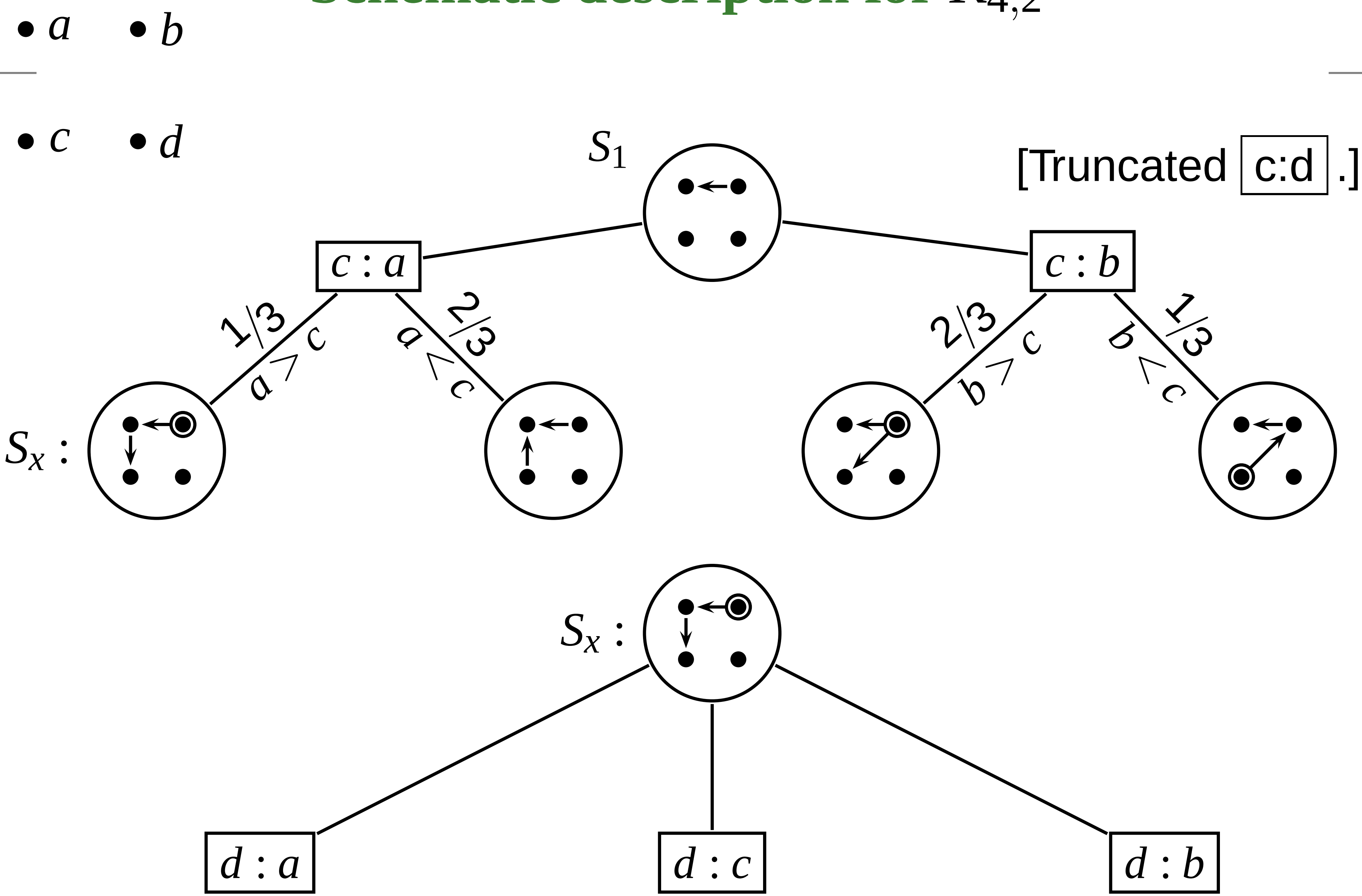
Difficulty: Billions-and-billions to choose from!

$$u_n = 3^{n(n-1)/2}; \qquad u_8 = 2.28768 \times 10^{13}$$

Schematic description for $K_{4,2}$

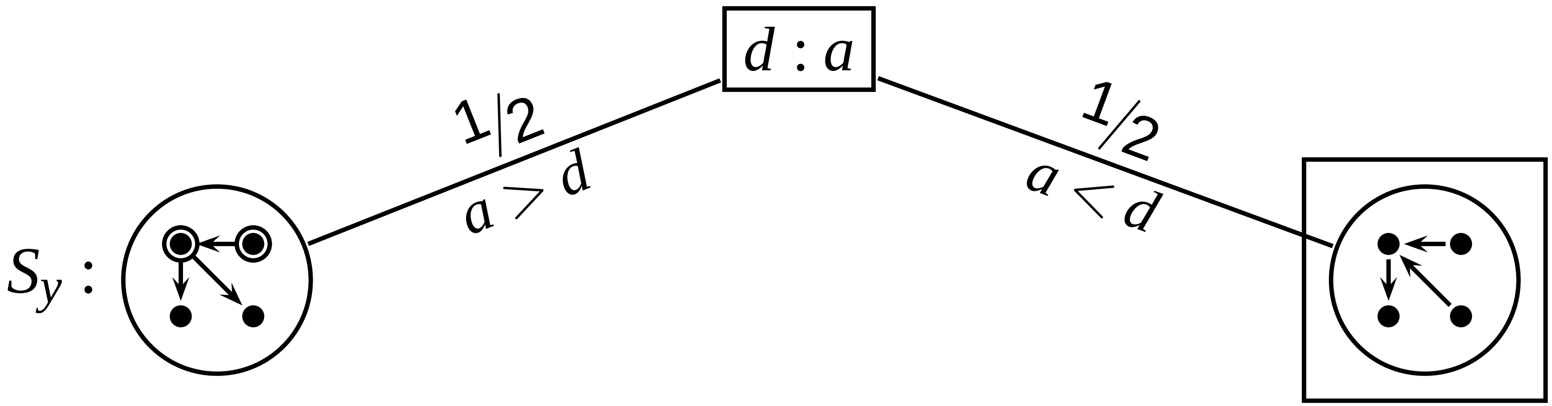


Schematic description for $K_{4,2}$



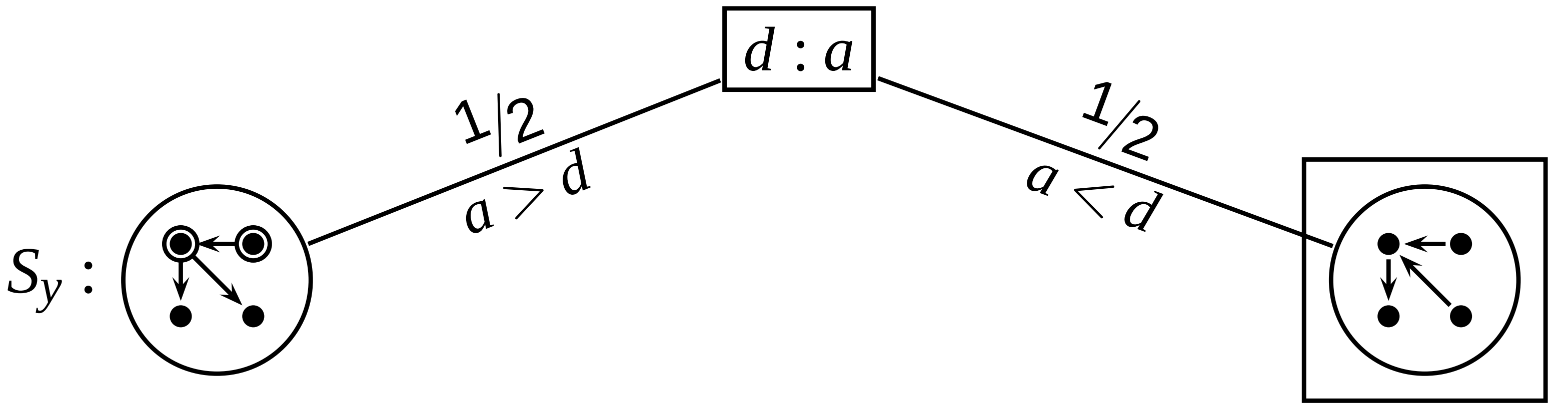
Schematic description for $K_{4,2}$, Cont.

We first examine the action $\{d : a\}$ in state S_x:



Schematic description for $K_{4,2}$, Cont.

We first examine the action $\{d : a\}$ in state S_x:

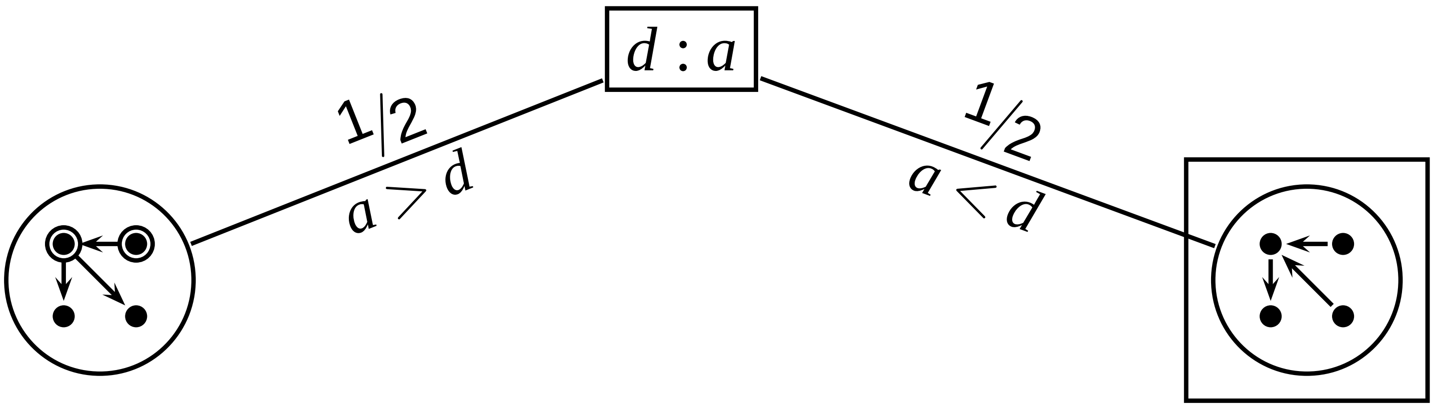


The state on the right is done, it is in $\mathscr{S}_r$; the sought value is a. On the left, the larger of c and d would be. We found $EU(S_x \mid d : a) = 1/2$.

Schematic description for $K_{4,2}$, Cont.

We first examine the action $\{d : a\}$ in state S_x:

S_y :



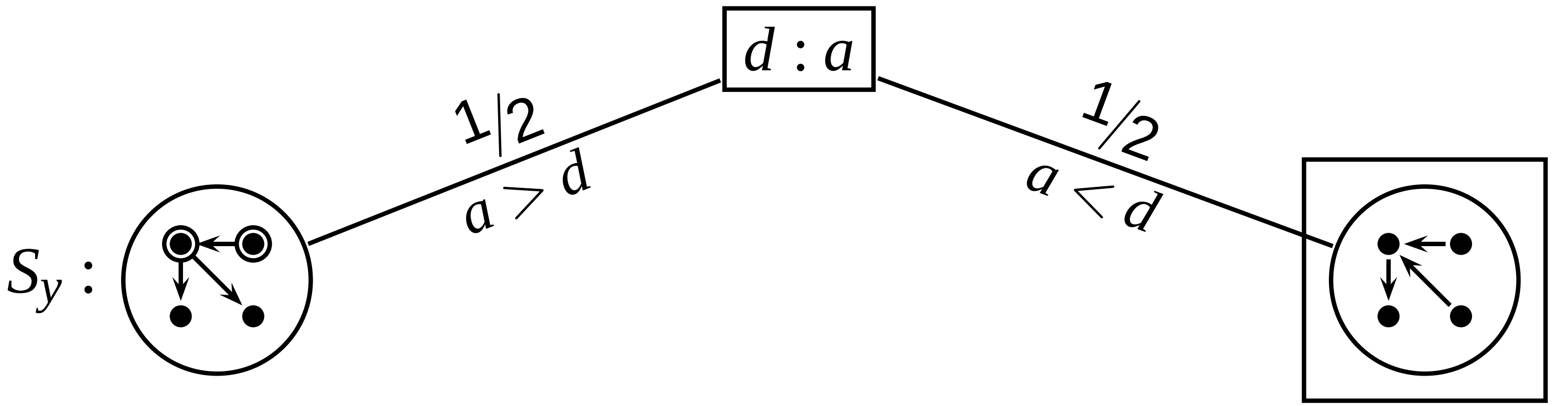
The state on the right is done, it is in $\mathscr{S}_r$; the sought value is a.
On the left, the larger of c and d would be. We found
$EU(S_x \mid d : a) = 1/2$.
Similarly developing the other actions possible at S_x we find

$EU(S_x \mid d : c) = 3/4$. $EU(S_x \mid d : b) = 5/3$:

The comparison $\{d : b\}$ is nearly useless!

Schematic description for $K_{4,2}$, Cont.

We first examine the action $\{d : a\}$ in state S_x:



The state on the right is done, it is in $\mathscr{S}_r$; the sought value is a.
On the left, the larger of c and d would be. We found
$EU(S_x \mid d : a) = 1/2$.
Similarly developing the other actions possible at S_x we find

$EU(S_x \mid d : c) = 3/4$. $EU(S_x \mid d : b) = 5/3$:

The comparison $\{d : b\}$ is nearly useless!

A general rule: exclude entries known to be too extreme.

Computational process

Teasing the optimal policy tree out of the state-activity network:

Computational process

Teasing the optimal policy tree out of the state-activity network:

(1). Create the network via a BFS-like process.

(2). Resolution: answering the question $S \stackrel{?}{\in} \mathscr{S}_r$.
Probabilities can be computed then as well.

Computational process

Teasing the optimal policy tree out of the state-activity network:

(1). Create the network via a BFS-like process.

(2). Resolution: answering the question $S \overset{?}{\in} \mathscr{S}_r$.
Probabilities can be computed then as well.

(3). Use DFS-like process to compute $U(S)$ for all states in the network.

This is when the optimization criterion applies.

When a state ($\notin \mathscr{S}_r$) is done, one of its actions is marked.

Computational process

Teasing the optimal policy tree out of the state-activity network:

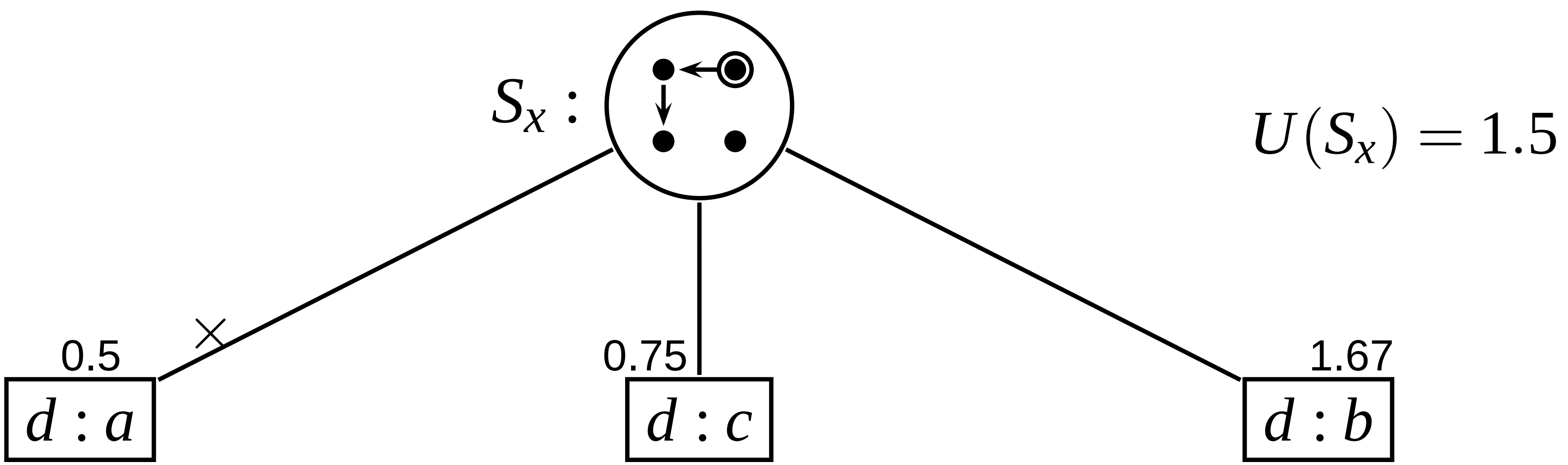
(1). Create the network via a BFS-like process.

(2). Resolution: answering the question $S \overset{?}{\in} \mathscr{S}_r$.
Probabilities can be computed then as well.

(3). Use DFS-like process to compute $U(S)$ for all states in the network.

This is when the optimization criterion applies.

When a state ($\notin \mathscr{S}_r$) is done, one of its actions is marked.



Computational process, Cont.

(4). A resolved state is marked with the item selected.

(5). Now an entire optimal policy can be captured.

Here is a function to locate $K_{4,2}$, as generated by the search:

```
int K42(int *Q) {
if (Q[0] > Q[1]) swap(&Q[0], &Q[1]);  // symmetrization;
L0: if(Q[2] > Q[3]) goto L1;  else goto L2;
L1: if(Q[0] > Q[3]) goto L3;  else goto L4;
L2: if(Q[0] > Q[2]) goto L5;  else goto L6;
L3: if(Q[0] > Q[2])  return Q[2];  else return Q[0];
L4: if(Q[3] > Q[1])  return Q[1];  else return Q[3];
L5: if(Q[0] > Q[3])  return Q[3];  else return Q[0];
L6: if(Q[2] > Q[1])  return Q[1];  else return Q[2];
 }
```

This looks like a full, perfect binary tree. Same cost (4) on all paths. Highly unusual.

Note: the states corresponding to L1 and L2 are isomorphic, and so are the four last states, L3 to L6.

Computational process, Cont.

The notion of state equivalence is key to the calculation.

Computational process, Cont.

The notion of state equivalence is key to the calculation.

d_n — number of labeled transitive DAGs, on n nodes.

d_n^* — number of non-isomorphic labeled transitive DAGs. . . .

x_n — number of states needed to find median.

n	4	5	6	7	8	9
d_n	219	4231	130,023	6,129,859	431,723,379	44,511,042,511
d_n^*	16	63	318	2045	16,999	183,231
x_n		54	291	1971	16,618	181,773

Computational process, Cont.

The notion of state equivalence is key to the calculation.

d_n — number of labeled transitive DAGs, on n nodes.

d_n^* — number of non-isomorphic labeled transitive DAGs. . . .

x_n — number of states needed to find median.

n	4	5	6	7	8	9
d_n	219	4231	130,023	6,129,859	431,723,379	44,511,042,511
d_n^*	16	63	318	2045	16,999	183,231
x_n		54	291	1971	16,618	181,773

All grow superexponentially — main limitation of the process.

Computational process, Cont.

The notion of state equivalence is key to the calculation.

d_n — number of labeled transitive DAGs, on n nodes.

d_n^* — number of non-isomorphic labeled transitive DAGs. . . .

x_n — number of states needed to find median.

n	4	5	6	7	8	9
d_n	219	4231	130,023	6,129,859	431,723,379	44,511,042,511
d_n^*	16	63	318	2045	16,999	183,231
x_n		54	291	1971	16,618	181,773

All grow superexponentially — main limitation of the process.

The largest we did was $n = 11$:

$$d_{11} = 1,396,281,677,105,899; \qquad d_{11}^* = 46,749,427.$$

Observations

So far, no predictable structure, except in full sorting.

Observations

So far, no predictable structure, except in full sorting.

Single-selection has a large cost-range.

Example: $K_{9,5}$ needs 8 to 16 comparisons, (average 12.7896.)

$\{K_{9,3}, K_{9,6}\}$ pairs are produced after 11 to 14 comparisons.

Observations

So far, no predictable structure, except in full sorting.

Single-selection has a large cost-range.
Example: $K_{9,5}$ needs 8 to 16 comparisons, (average 12.7896.)
$\{K_{9,3}, K_{9,6}\}$ pairs are produced after 11 to 14 comparisons.

With $b = 5$ or 7 the Sicilian algorithm shows some improvement of the accuracy. Here is mean error in approximate median:

Observations

So far, no predictable structure, except in full sorting.

Single-selection has a large cost-range.
Example: $K_{9,5}$ needs 8 to 16 comparisons, (average 12.7896.)
$\{K_{9,3}, K_{9,6}\}$ pairs are produced after 11 to 14 comparisons.

With $b = 5$ or 7 the Sicilian algorithm shows some improvement of the accuracy. Here is mean error in approximate median:

n	$b = 3$	$b = 5$	$b = 7$
10	0.5215	0.6913	0
100	5.2242	4.2694	5.5255
1000	26.101	17.999	31.838
10,000	144.19	83.239	92.766
100,000	648.86	412.33	310.71
1,000,000	3471.2	2801.1	1297.9
10,000,000	11901	5787.7	4782.8