

Fixed Parameter Algorithms for
One-Sided Crossing Minimization
Revisited

Vida Dujmović
McGill University

Henning Fernau
Universität Tübingen
The University of Newcastle

Michael Kaufmann
Universität Tübingen

October 15, 2003

Overview

- the problem: OSCM
- the methodology: parameterized algorithmics
- our solutions: kernelization and search tree techniques;
how and why do they work?
- conclusions

One-Sided Crossing Minimization k -OSCM

Given: A bipartite graph $G = (V_1, V_2, E)$ and a linear order $<_1$ on V_1 .

Parameter: k

Question: Is there a linear order $<$ on V_2 such that,

- (1) when the vertices from V_1 are placed on a line (also called layer) L_1 in the order induced by $<_1$ and
 - (2) when the vertices from V_2 are placed on a second layer L_2 (parallel to L_1) in the order induced by $<$,
- then drawing straight lines for each edge in E will introduce no more than k edge crossings.

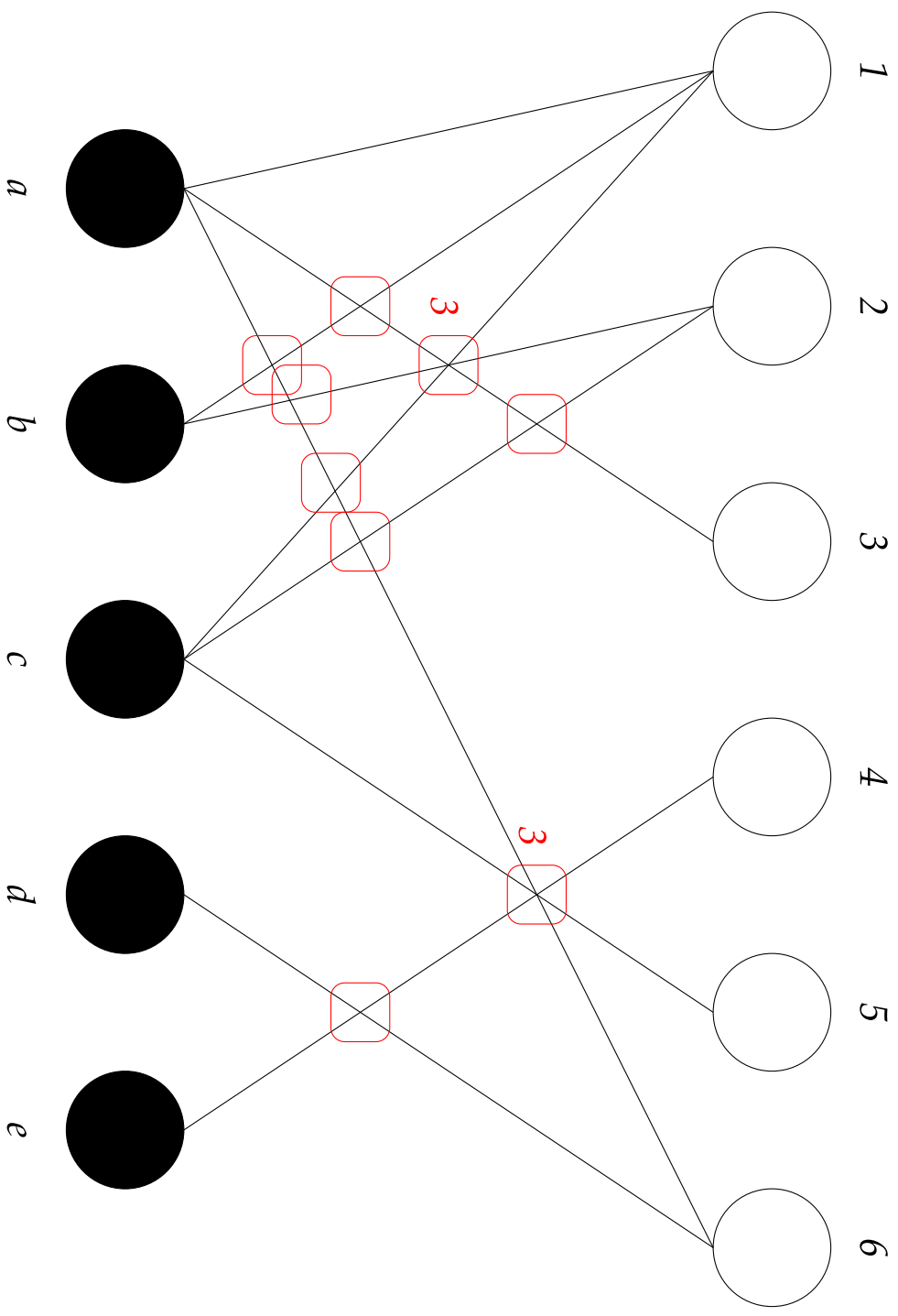
Background: Sugiyama Layering Algorithm

Problem: Construct a “hierarchical” drawing of a given directed graph.

Method: Work in three phases:

- Assign vertices to “layers.”
- **Fix ordering of vertices within layers** s. t. # of crossings is minimized.
- Determine concrete positions of vertices according to ordering.

Doing the 2nd step layerwisely gives OSCM.



Crossing Numbers

For any two distinct vertices $u, v \in V_2$,

$c_{uv} := \#$ crossings in the drawing of $G[\{u, v\} \cup (N(u) \cup N(v))]$ when $u < v$ is assumed.

c_{uv}	a	b	c	d	e
a	–	4	5	0	1
b	1	–	1	0	0
c	3	3	–	0	1
d	3	2	3	–	1
e	2	3	2	0	–

crossings in drawing = $\sum\{c_{uv} \mid u < v\}$.

$$c_{ab} + c_{ac} + c_{ad} + c_{ae} + c_{bc} + c_{bd} + c_{be} + c_{cd} + c_{ce} + c_{de} = 4 + 5 + 0 + 1 + 1 + 0 + 0 + 0 + 1 + 1 = 13.$$

The Curse of Combinatorics

Folklore: Nearly all “interesting” computational problems in graph theory are “(NP-)hard”.

Examples:

- Find a vertex cover of size $\leq k$.
- Find an independent set of size $\geq k$.
- Find a dominating set of size $\leq k$.
- ...

Parameterized Problems and Algorithms

Idea: measure an instance in two ways:

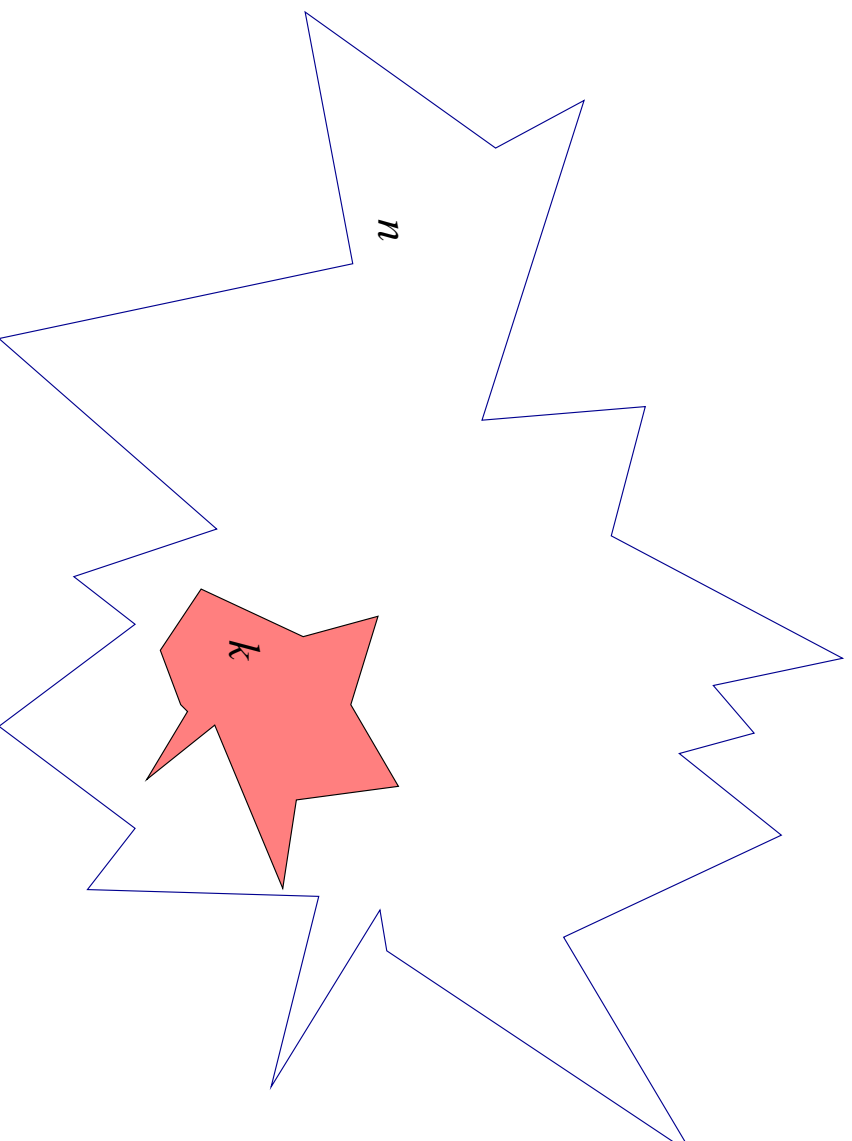
- overall size n and
- **parameter** [size] k .

The parameter can (in principle) be *anything*.

For minimization, take the **natural parameter**: the entity to be minimized.

Controlled Explosion:

The idea of parameterized algorithms: the parameter is **small**.



Parameterized Algorithms and Feasibility

Three equivalent characterizations of **fixed parameter tractable FPT**:

- running time $O(f(k)p(n))$ for arbitrary f and a polynomial p ;
- running time $O(f(k) + p(n))$ for arbitrary f and a polynomial p ;
- being pol.-time reducible to a **problem kernel**, i.e., another instance of the same problem of size $O(f(k))$ and parameter $O(g(k))$ for arbitrary f, g .

Contrast: running time $O(n^k)$.

Parameterized Algorithms Design—Standard Methodology

1. Look for a **suitable parameter**.
2. Find appropriate reduction rules to get a **small kernel**
↪ **data reduction**.
3. Develop **search tree** algorithm.

Remark 1: Other “parameterized methodologies” available, but not “practical.”

Remark 2: Notion of $W[1]$ -hardness ↪ methodology may fail.

Complexity of OSCM

Theorem: (Eades, Wormald 1994) OSCM is NP-complete.

Theorem: (Dujmović and 11 more authors GD'01, ESA '01) OSCM—and similar problems—are in FPT.

Theorem: (Dujmović and Whitesides GD'02) OSCM is solvable in time $\mathcal{O}(1.6182^k n^2)$.

Theorem: (this paper GD'03) OSCM is solvable in time

$$\mathcal{O}(1.4656^k + kn^2).$$

Remark: $1.4656^{40} \approx 4,000,000$, $1.6182^{40} \approx 200,000,000$

A Stepwise Approach to OSCM

Given an OSCM instance, we *gradually* find a solution by setting the ordering between u and v on V_2 , i.e., we are *committing* $u < v$ or $u < v$.

A **generalized OSCM instance** therefore contains, besides the bipartite graph $G = (V_1, V_2, E)$, a partial ordering on V_2 .

A vertex $v \in V_2$ is **fully committed** if, for all $u \in V_2 \setminus \{u, v\}$, $\{u, v\}$ is committed.

Reduction Rules for OSCM

RR1: For every pair of vertices $\{u, v\}$ from V_2 with $c_{uv} = 0$, commit $u < v$.

Lemma: After exhaustively applying RR1, every pair of vertices $\{u, v\}$ from V_2 which is not committed satisfies $\min\{c_{uv}, c_{vu}\} \geq 1$.

RR2: For every pair of vertices $\{u, v\}$ from V_2 with $N(u) = N(v)$, (arbitrarily) commit $u < v$.

RRLO1: If (G, k, O) is a generalized instance of OSCM s.t. $v \in V_2$ is **fully committed**, reduce to $(G - v, k, O - v)$.

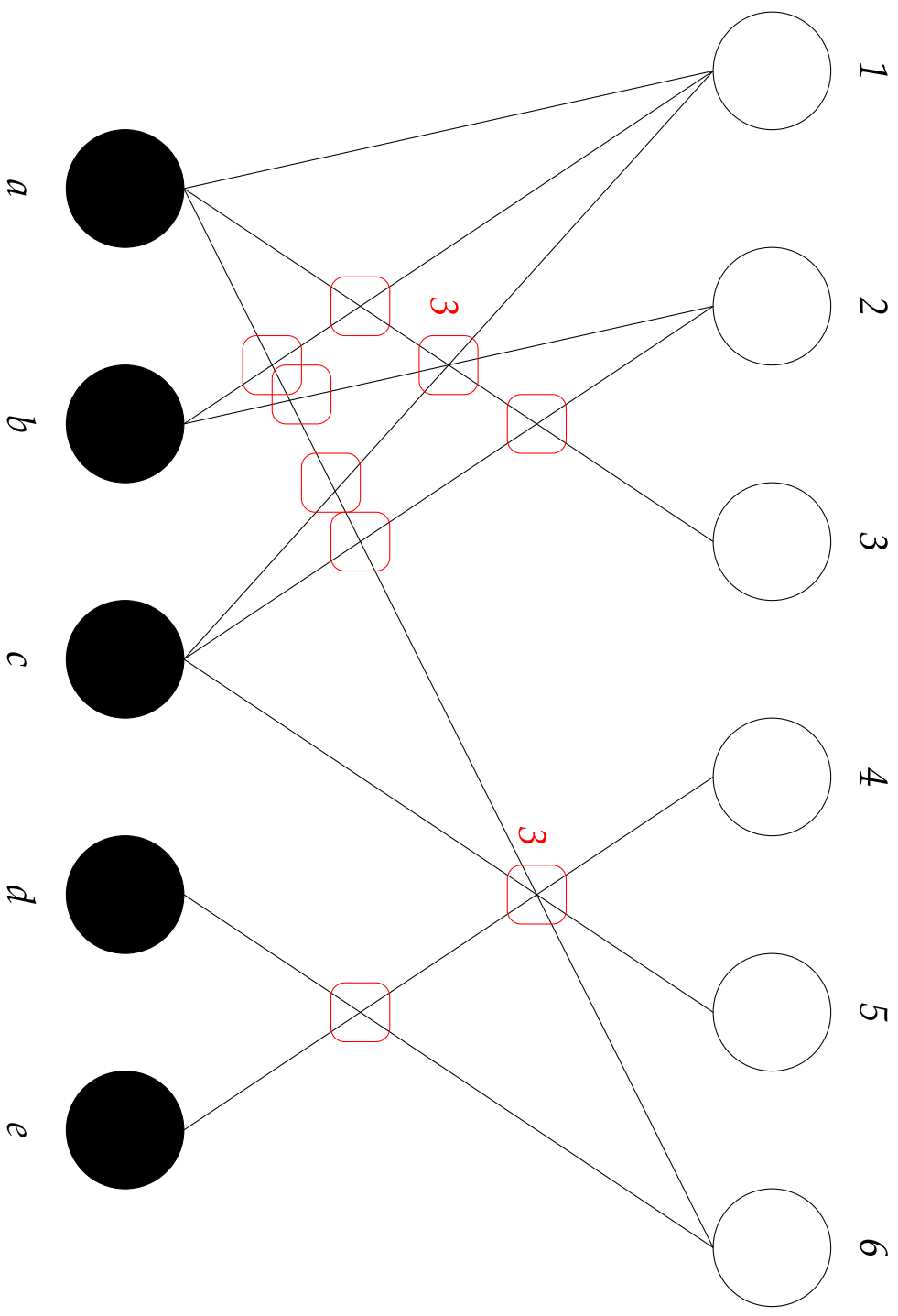
Continuing our Example ...

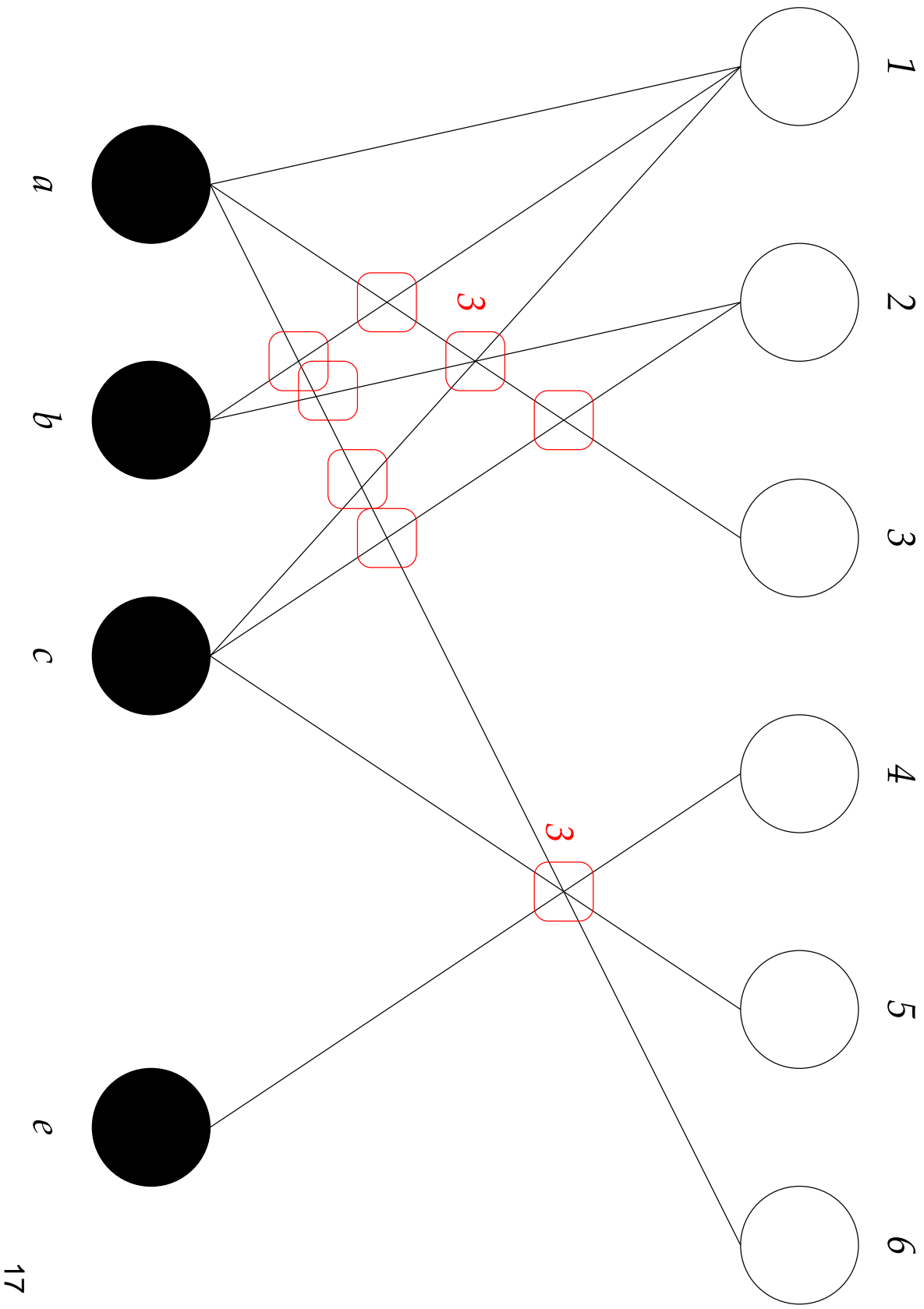
RR1 settles all relations to d

↪

RRLO1 erases d .

Moreover, $b < e$ will be committed.





Towards a Problem Kernel for OSCM

Lemma: Exhaustively applying RR1 and RRL01 $\rightsquigarrow |V_2| \leq 2k$.

With some more “order-related reduction rules”, we were able to prove:

Theorem: Fix some $1 < \alpha \leq 1.5$. \rightsquigarrow

Each instance (G, k) of k -OSCM can be reduced to a generalized instance

$(G' = (V'_1, V'_2, E'), k', O)$ with $|V'_2| \leq \alpha k$.

A Problem Kernel for OSCM

We only need the following (obvious) rule:

RRlarge: If $c_{uv} > k$ then: if $c_{vu} \leq k$ then commit $v < u$ else NO.

Consider $u \in V_2$ has $\deg(u) > 2k + 1$. Then, for $v \in V_2 - u$, $c_{uv} > k$ or $c_{vu} > k$.

Hence, exhaustively performing RRlarge in combination with RRLO1 removes all the vertices of V_2 of degree larger than $2k + 1$.

Theorem: For any $1 < \alpha \leq 1.5$, k -OSCM admits a problem kernel $G = (V_1, V_2, E)$ with $|V_1| \leq (2k + 1)|V_2|$ ($\leq 3k^2 + 1.5k$), $|V_2| \leq \alpha k$, and $|E| \leq (2k + 1)|V_2|$.

Remark: Novel behaviour as kernelization scheme

Some More Helpful Terminology

In a generalized instance, two *unsettled* vertices $u, v \in V_2$ are

- **independent with respect to** $w \in V_2$ if $\{u, w\}$ and $\{v, w\}$ are settled;
- **dependent with respect to** $w \in V_2$ if $\{u, w\}$ or $\{v, w\}$ are unsettled;
- **independent (in \mathcal{O})** if $\{u, v\}$ are independent w.r.t. all $w \in V_2 \setminus \{u, v\}$;
- **dependent (in \mathcal{O})** if $\{u, v\}$ are dependent w.r.t. some $w \in V_2 \setminus \{u, v\}$;
- **transitive with respect to** $w \in V_2$ if either $\{u, w\}$ or $\{v, w\}$ is settled but not both.

A pair $\{u, v\} \subset V_2$ with $c_{uv} = i$ and $c_{vu} = j$ is also called an *i/j pattern*.

Another “Ordering” Rule

RRL02: If u and v are independent in a generalized OSCM instance $I = (G, k, O)$ and if $c_{uv} < c_{vu}$, then reduce to $I' = (G, k - c_{uv}, O \cup \{(u, v)\})$.

Kernelization: the Complete Preprocessing

Compute the crossing numbers c_{ab} for all pairs (a, b) .

If $k < \sum_{a,b \in V_2} \min(c_{ab}, c_{ba})$ then answer NO.

If $k \geq \sum_{a,b \in V_2} \max(c_{ab}, c_{ba})$ then YES; output an arbitrary linear order.

Apply reduction rules RR1, RR2 (and RR3).

Apply reduction rules RRL01, RRL02, and RRLarge exhaustively.

The running time is dominated by the computation of the crossing numbers, taking time $\mathcal{O}(kn^2)$ even when combined with RRLarge.

A Search Tree Algorithm for OSCM

Each **node of a search tree** is associated with a generalized OSCM instance $I = (G = (V_1, V_2, E), k, O)$.

Initially (root node), I is determined by the kernelization.

Then, a recursive algorithm is applied (next slide).

Afterwards, we argue about **running time** and **correctness**.

A Search Tree Algorithm for OSCM

```
In a node of a search tree with instance  $I = (G, k', 0)$  do
0: if  $k' < 0$  return NO.
1: Apply RRL01, RRL02 and RRLarge exhaustively.
2: if in  $I$  there is an unsettled  $i/j$  pattern  $\{a, b\}$  such that  $i + j \geq 4$ 
    branch on  $a < b$  and  $b < a$ ; update  $0$  and  $k'$  accordingly
3: else if in  $I$  there is a dependent 2/1 pattern  $\{a, b\}$ 
    branch on  $a < b$  and  $b < a$ ; update  $0$  and  $k'$  accordingly
4: else
    //the remaining 2/1 patterns are independent; thus, RRL02 applies
    resolve all remaining 1/1 patterns arbitrarily
    update  $I$  and  $k'$  accordingly; if  $k' < 0$  return NO else YES.
```

Running Time Analysis

Each internal node of our search tree has two branches.

(b_1, b_2) **branching vector**: one branch lowers the parameter k' by b_1 and the other by b_2 .

Main Lemma: Let $\{u, v\}$ be a pair of **dependent** vertices forming 2/1 pattern in step 3 of the algorithm. Then $\{u, v\}$ is **transitive**.

Run Time Theorem: The search tree takes $\mathcal{O}(1.4656^k k^2)$ time.

Each node that branches on a transitive 2/1 pattern commits by transitivity an extra i/j pattern where $i, j > 0$. \rightsquigarrow In each internal node: $b_1 + b_2 \geq 4$.

\rightsquigarrow Each node of the search tree has branching vector (2,2) or (3,1).

Deduce the size $s(k)$ of the search tree:

- The recurrence corresponding to the (2,2) branching vector is

$$s(k) = 2s(k-2) + \mathcal{O}(1).$$

Solving this recurrence gives $s(k) < 1.4143^k$.

- The recurrence corresponding to (1,3) is

$$s(k) = s(k-1) + s(k-3) + \mathcal{O}(1).$$

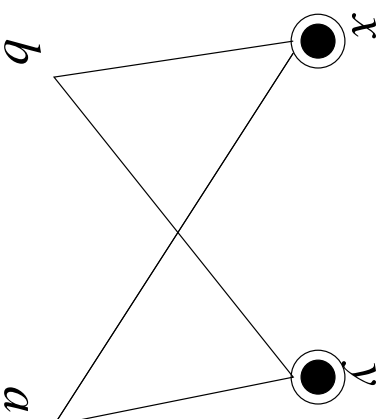
Solving this recurrence gives $s(k) < 1.4656^k$.

The correctness of the Main Lemma relies on a detailed analysis of 1/1 and of 2/1 patterns.

The main ingredients of this analysis are:

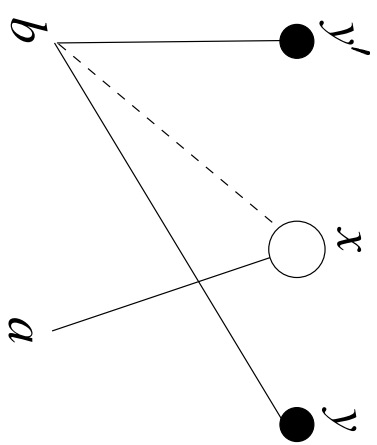
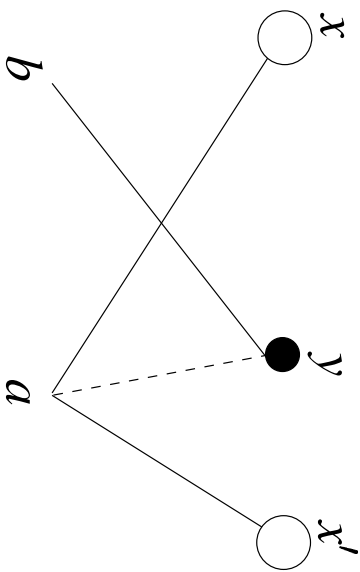
- A complete characterization of 1/1 patterns.
- A complete characterization of 2/1 patterns.
- Reduction rules for “half of the cases.”
- Showing the Main Lemma for the remaining cases.

Characterizing 1/1 Patterns, 1st Part

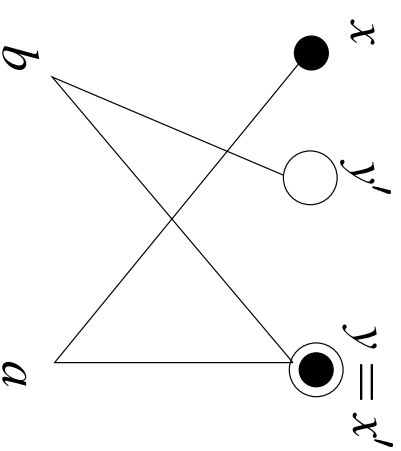
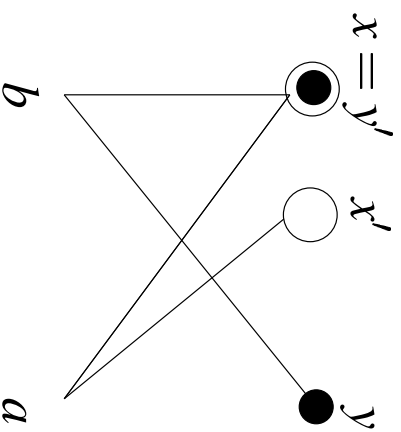


Lemma: This situation is resolved by RR2 ($N(a) = N(b)$).

Characterizing 1/1 Patterns, 2nd Part



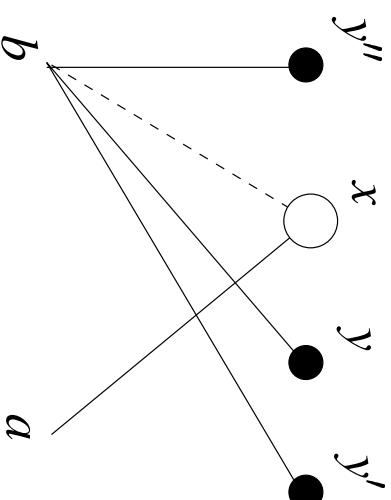
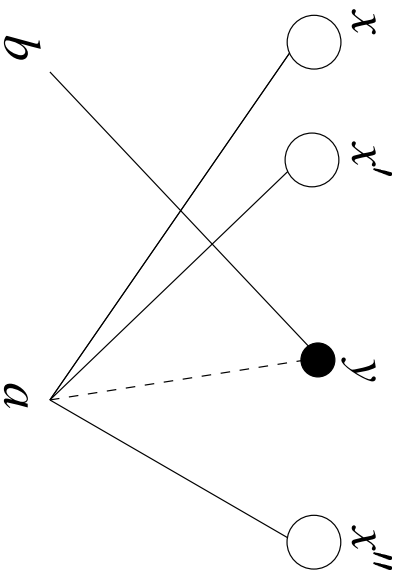
Characterizing 2/1 Patterns, 1st Part



RR3: In these situations, always let $a < b$.

Lemma: These situations are resolved by RR3.

Characterizing 2/1 Patterns, 2nd Part



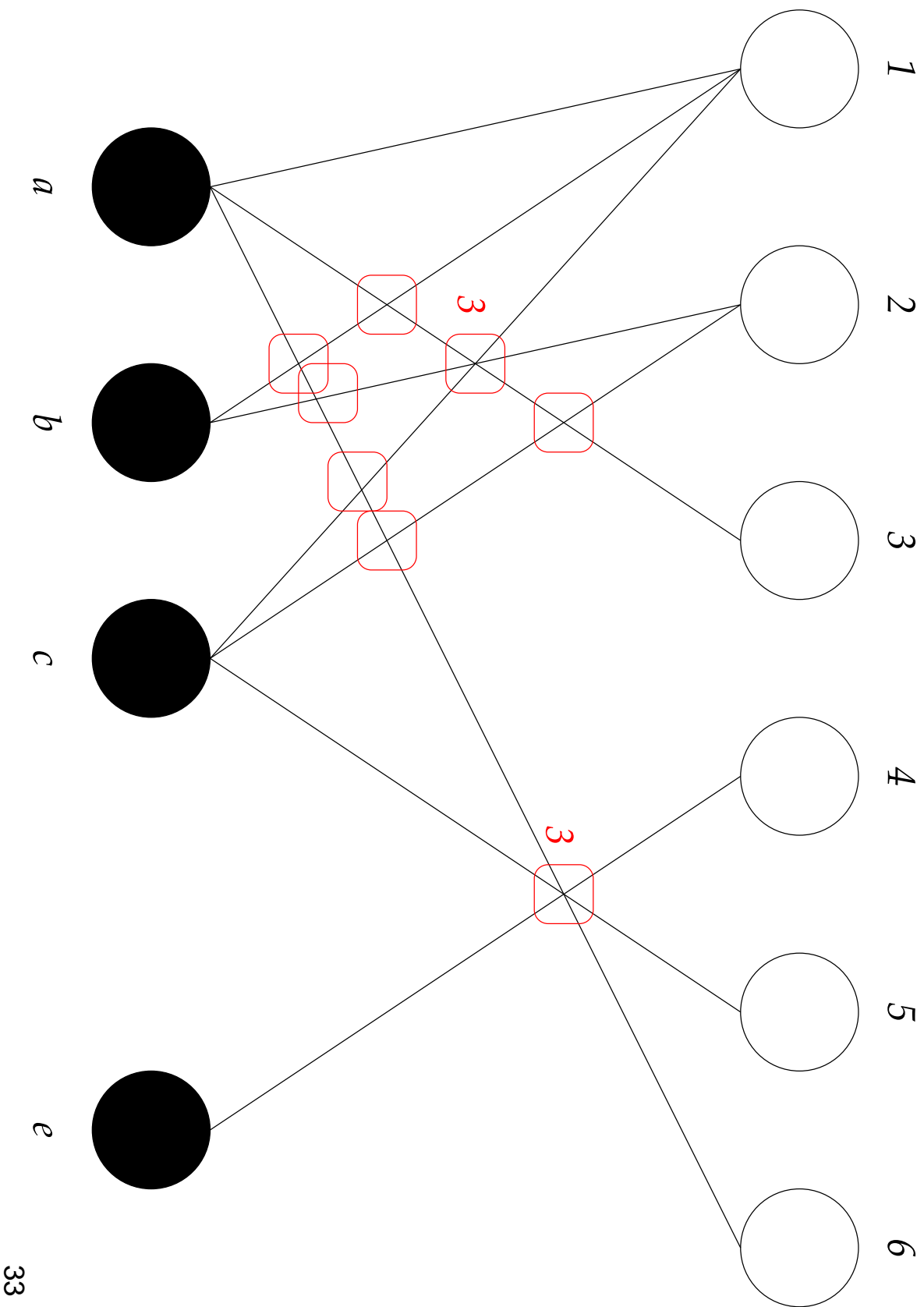
Continuing Our Example . . .

How to read the following search tree graphics:

- Nodes contain the parameter value.
- Edges are labelled with the committed orders; blue labels indicate the orders committed by transitivity.

We start with displaying the reduced generalized graph instance (and the corresponding table), assuming we started with the instance $(G, 7)$.

A natural heuristic for working on the search tree is to first treat “dearest branches.”



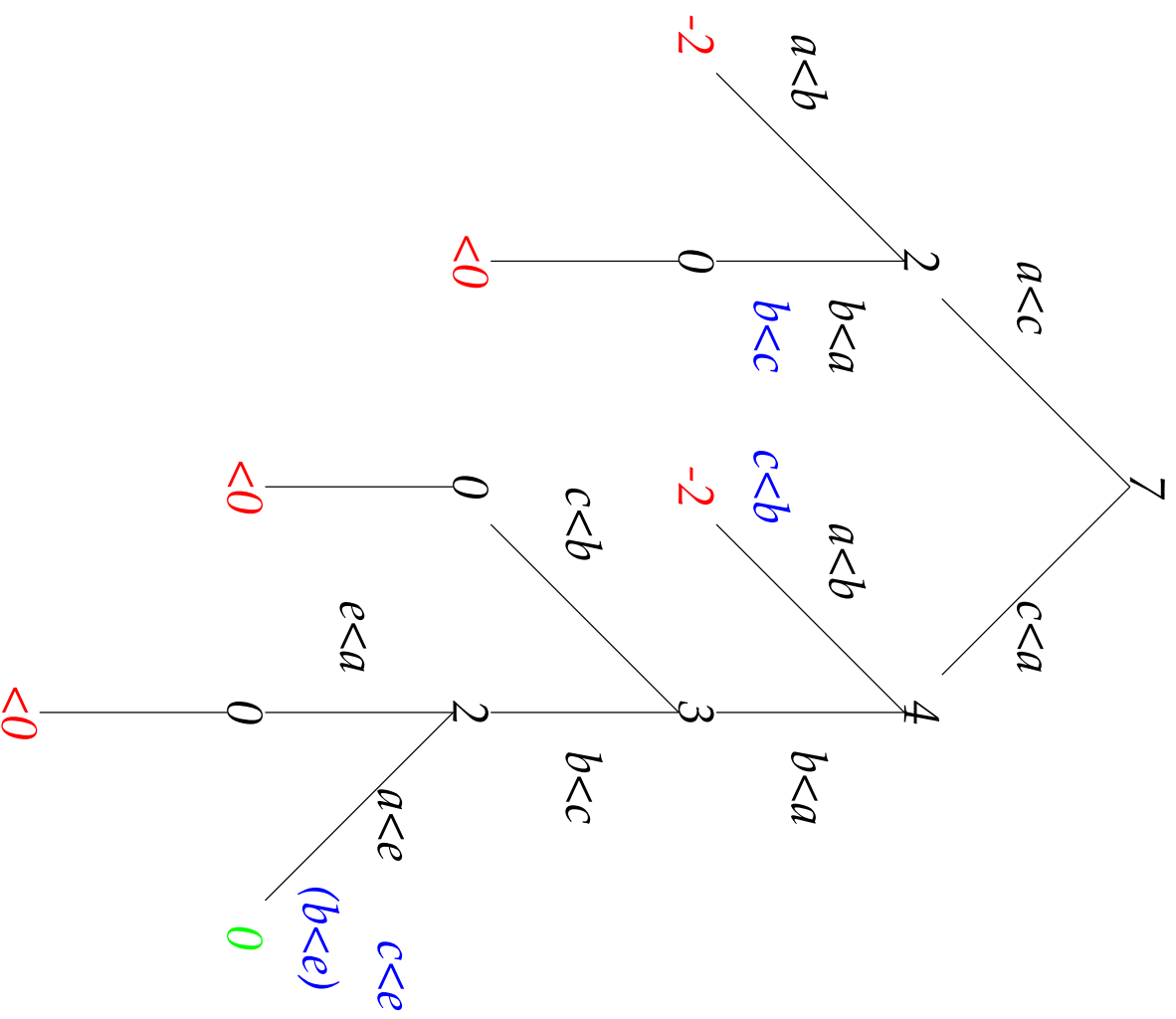
Crossing Numbers

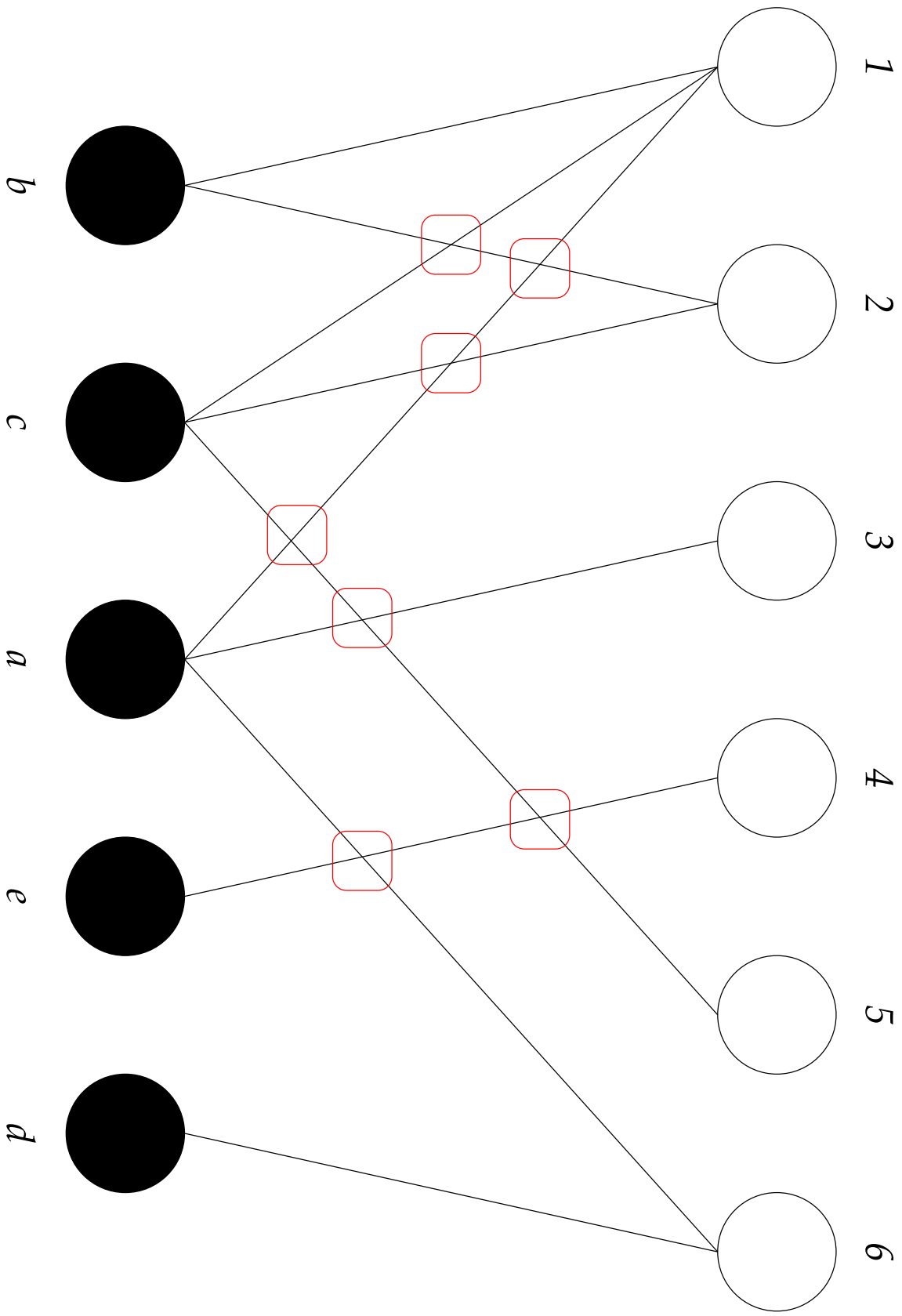
c_{uv}	a	b	c	e
a	–	4	5	1
b	1	–	1	0
c	3	3	–	1
e	2	3	2	–

crossings in drawing = $\sum \{c_{uv} \mid u < v\}$.

Lowerbound: $\sum_{x,y} \min\{c_{xy}, c_{yx}\}$.

Here lowerbound = 7.





Conclusions: Summary

- Parameterized algorithmics can **exactly solve NP-hard problems**:
 - providing a well-founded notion of **data reduction**, and
 - putting the **combinatorial explosion** in some “harmless spot.”

- In the case of OSCM, we **showed** how to develop **kernelization rules** and a **search tree** algorithm, yielding an

$$\mathcal{O}(1.4656^k + kn^2)$$

algorithm, the **parameter** being the # of permissible crossings.

- Our **small constants** may justify the layerwise sweep technique traditionally used in the Sugiyama approach.

Conclusions: Outlook

- Further improvements of the exponential base of our algorithm for OSCM might be possible at the expense of intensive tedious case analysis or further structural insights.
- The parameterized approach should be explored for other areas in graph drawing. Typical small entities are:
 - # crossings
 - # auxiliary points
 - avg./max. distance between neighbouring vertices, etc.