KAIST 수리과학과

Constructive algorithm for path-width of matroids

Sang-il Oum

Joint work with
Jisu Jeong (KAIST)
Eun Jung Kim (CNRS-LAMSADE)

KAIST Department of Mathematical Sciences

GROW2015

2015.10.13 Aussois, France

Arranging vectors:
A problem in coding theory

- **Input**: n vectors $v_1, v_2, \ldots, v_n$ in $\mathbb{F}^m$.

- **Goal**: Find the minimum k such that
  there is a **linear layout** $v_1, v_2, \ldots, v_n$ of the **vectors** such that
  $\dim(\langle v_1\rangle \cap \langle v_2,v_3,\ldots,v_n\rangle) \leq k$,
  $\dim(\langle v_1,v_2\rangle \cap \langle v_3,\ldots,v_n\rangle) \leq k$,
  …
  $\dim(\langle v_1,v_2,v_3,\ldots,v_{n-1}\rangle \cap \langle v_n\rangle) \leq k$.

- Minimum such k = "**Trellis State-Complexity** of a linear code" or
  "**Trellis-Width**" (coding theory)
  or "**Path-width**" (matroid theory) — matroid representable over $\mathbb{F}$

Computing trellis-width (or path-width)

- Deciding trellis-width$\leq$k is NP-complete (Kashyap07, 08)

- What if k is fixed? — Remark in Kashyap's paper (07)

4 Concluding Remarks

The main contribution of this paper was to show that the decision problem Trellis State-Complexity is NP-complete, thus settling a long-standing conjecture. Now, the situation is rather different if we consider a variation of the problem in which the integer w is *not* taken to be a part of the input to the problem. In other words, consider the following problem:

Problem: Weak Trellis State-Complexity
Let $\mathbb{F}_q$ be a fixed finite field, and let w be a fixed positive integer.

Instance: An $m \times n$ generator matrix for a linear code $\mathcal{C}$ over $\mathbb{F}_q$.

Question: Is there a coordinate permutation of $\mathcal{C}$ that yields a code $\mathcal{C}'$ whose minimal trellis has state-complexity at most w?

YES

There is good reason to believe that this problem is solvable in polynomial time.

For fixed k?

- **WQO(Well-quasi-ordering) Conjecture**: $\mathbb{F}$-representable matroids are well-quasi-ordered by the minor relation. (no infinite antichain w.r.t. minors)
- If true, then for every class X of $\mathbb{F}$-representable matroids closed under taking minors, there are **finitely** many $\mathbb{F}$-representable matroids M_1, M_2, …, M_n such that M is in X iff none of M_1,M_2,…,M_n is a minor of M.
- Theorem (Geelen, Gerards, Whittle; 2012+): WQO Conj is true for finite $\mathbb{F}$.
- Corollary: For each k and $\mathbb{F}$, there exists a **finite** list of matroids such that an $\mathbb{F}$-repre. matroid M has path-width$\leq$k iff none in the list is a minor of M.
- Enough to check whether an input matroid has some minors in the finite list (which can be done in poly time for matroids of bounded branch-width, shown by Hlineny.)
- **Trouble**: 1. No algorithm known to construct the list of forbidden minors.
  2. Even if you know the list, this **doesn't provide a linear ordering**!

Known algorithms for path-width/branch-width of matroids

- **STEP 1**: Find an approximate branch-decomposition (Hlineny 2006; $O(n^3)$ algorithm)

- **STEP 2**: Use the dynamic programming to test all forbidden minors for branch-width$\leq k$ or path-width$\leq k$.

- Branch-width: (the size of each forbidden minor)$\leq(6^k-1)/5$ (Geelen, Gerards, Robertson, Whittle 2003)

- Path-width: (#forbidden minors) OPEN!
  No upper bound is known; Finite due to WQO.

- STEP 3: Use step 2 to construct a branch-decomp of width$\leq k$ (Hlineny, Oum)

- For path-width, an efficient algorithm exists but we did not know how to construct.

What's new? (1/2)

- Theorem [Jeong, Kim, O.]
  O(f(k) n^3)-time algorithm to find
  a linear layout $v_1, v_2, \ldots, v_n$
  of the input **n vectors** in $\mathbb{F}^m$ such that
  $\dim(\langle v_1,v_2,\ldots,v_i\rangle \cap \langle v_{i+1},\ldots,v_n\rangle) \leq \mathbf{k}$ for all i,
  if it exists (when $\mathbb{F}$ is a finite field)

- Outcome: **Fixed Parameter Tractable to decide**
  trellis-width$\leq$k or
  path-width$\leq$k of $\mathbb{F}$-representable matroids

What's new (2/2)
Extension to subspaces

- Theorem [Jeong, Kim, O.]
  O(f(k) n^3)-time algorithm to find
  a linear layout $V_1, V_2, \ldots, V_n$
  of the input **n subspaces** of $\mathbb{F}^m$ such that
  $\dim((V_1+V_2+\ldots+V_i) \cap (V_{i+1}+\ldots+V_n)) \leq \mathbf{k}$ for all i,
  if it exists (when $\mathbb{F}$ is a finite field)

For example, if
$V_i=\langle v_i \rangle$ for all ii,
then
matroid path-width

Corollary to

Linear rank-width

- Cut-rank function $cutrk_G(X)$:=rank of X*(V-X) submatrix of the adjacency matrix of a graph G

- **Linear rank-width** of a graph G:= min k such that ∃linear layout $v_1, v_2, \ldots, v_n$ of the vertices with $cutrk_G(\{v_1,v_2,\ldots,v_i\}) \leq k$ for all i.

- NP-complete to decide linear rank-width≤k.

- THEOREM: For a fixed k, $O(n^3)$-time algorithm to decide linear rank-width≤k. (NEW)

Corollary to

Linear rank-width

If A is the adjacency matrix of G and I is the identity matrix, then

$$\begin{pmatrix} I & A \end{pmatrix}$$

over GF(2)

For a vertex v_i,

let V_i=span of the i-th and (i+n)-th column vectors

EASY FACT: $2 * \text{cutrk}_G(X) = \dim((\Sigma\{V_i : v_i \in X\}) \cap (\Sigma\{V_i : v_i \notin X\}))$

Path-width of $\{V_1, V_2, \ldots, V_n\}$ = 2*(linear rank-width of G)

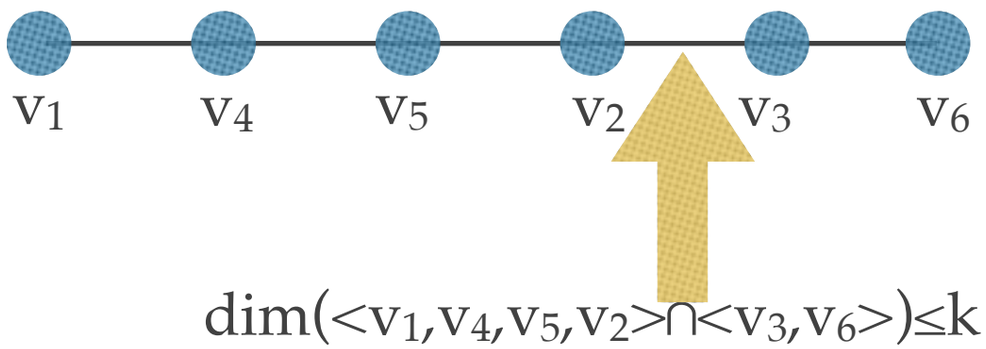
Corollary to Linear clique-width

- Linear clique-width= “linearized version of clique-width”
- EASY FACT: If linear rank-width=k, then linear clique-width$\leq 2^k+1$.
- NP-complete to decide linear clique-width$\leq$k (when k is not fixed) (Fellows, Rosamond, Rotics, Szeider 2009)
- Corollary: The first approximation algorithm for linear clique-width.
  For a fixed k, $O(n^3)$-time algorithm to find a linear clique-width expression of width$\leq 2^k+1$ or confirms that linear clique-width>k.

Bodlaender-Kloks type algorithm for path-width of “subspaces”

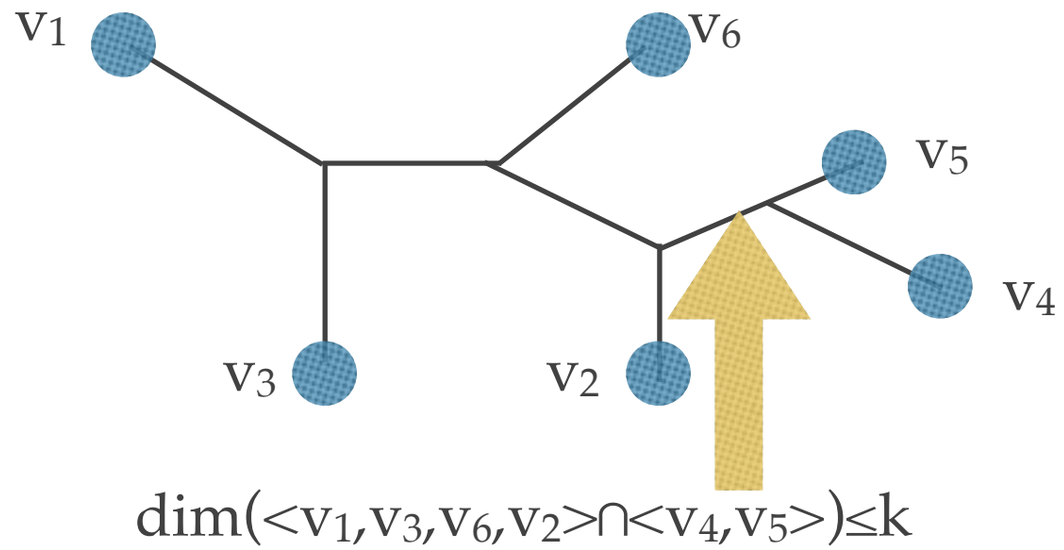
Path-width vs Branch-width of (representable) matroids

Input: $v_1, v_2, \ldots, v_n$: vectors in $\mathbb{F}^m$.
($\mathbb{F}$: finite field)
Output: Yes if $\exists$ permutation π of {1,2, …,n} s.t. …



Path-decomposition of width≤k

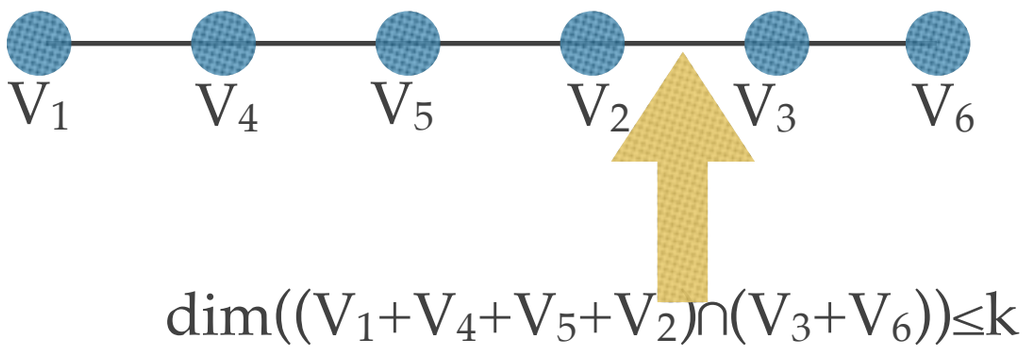
Input: $v_1, v_2, \ldots, v_n$: vectors in $\mathbb{F}^m$.
($\mathbb{F}$: finite field)
Output: Yes if $\exists$ subcubic tree T with a bijection L:{leaves}→{vectors} s.t. …



Branch-decomposition of width≤k

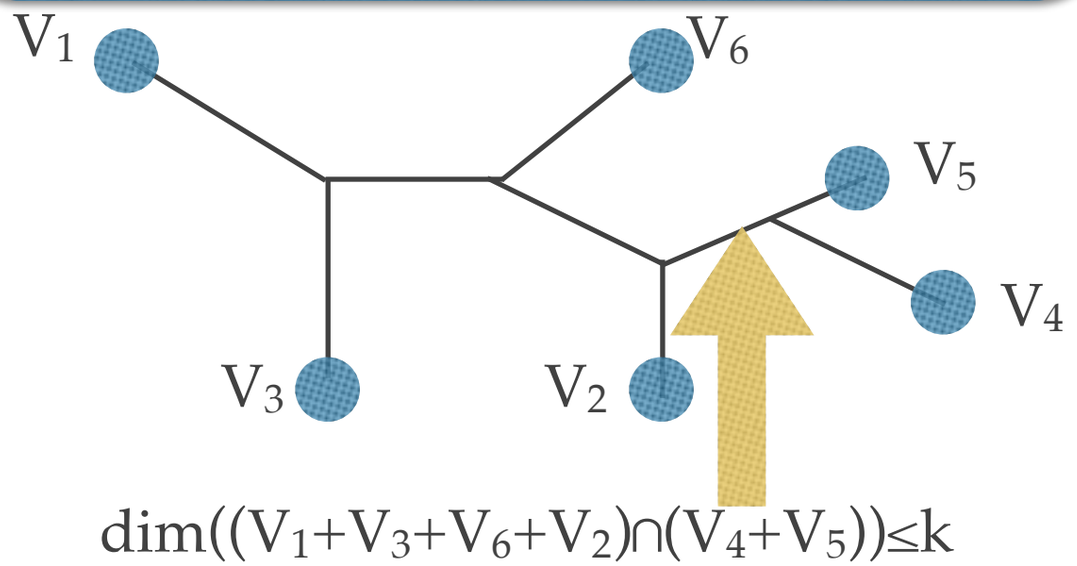
Path-width vs Branch-width of a subspace arrangement

Input: $V_1, V_2, \ldots, V_n$: subspaces in $\mathbb{F}^m$.
($\mathbb{F}$: finite field)
Output: Yes if $\exists$ permutation π of {1,2, …,n} s.t. …



Path-decomposition of width≤k

Input: $V_1, V_2, \ldots, V_n$: subspaces in $\mathbb{F}^m$.
($\mathbb{F}$: finite field)
Output: Yes if $\exists$ subcubic tree T with a bijection L:{leaves}→{subspaces} s.t. …



Branch-decomposition of width≤k

Our algorithm

- Input: n subspaces
- Assume that we are given a branch-decomposition of width w.
- Task: Do the dynamic programming to enumerate all “partial solutions” of width at most k.

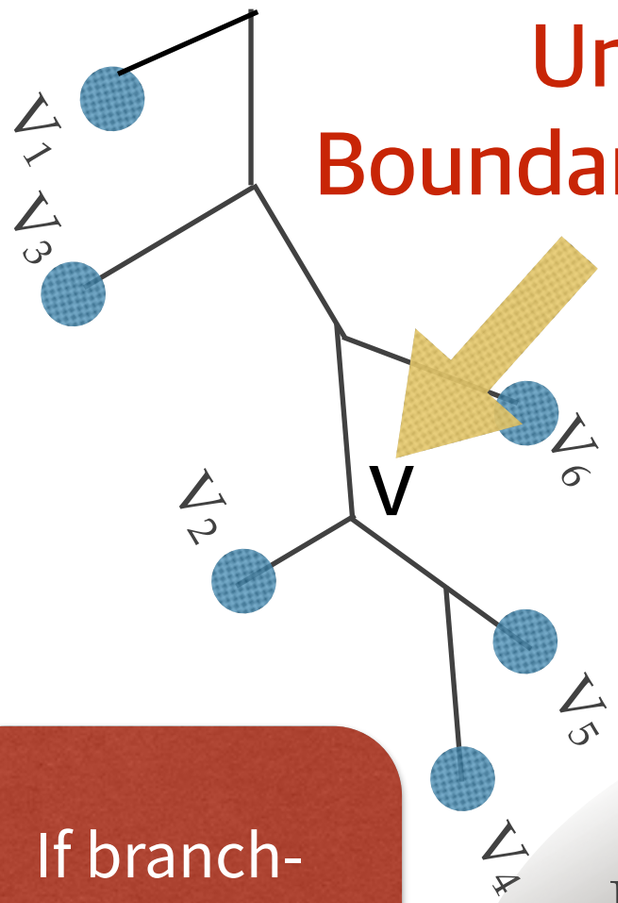
We will discuss how to provide such a decomposition later

How to run dynamic-programming on a “branch-decomposition of subspaces”

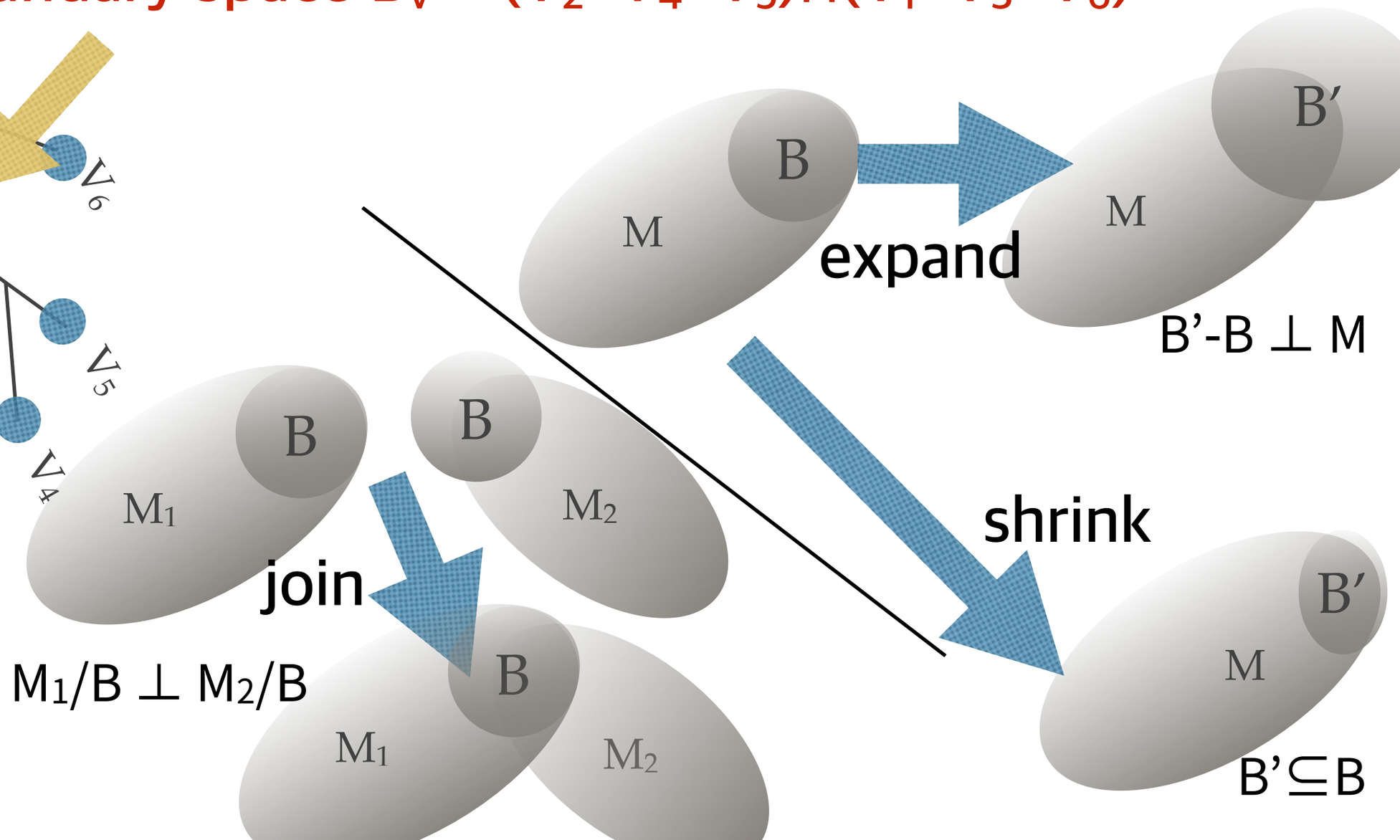
Dynamic programming on a branch-decomposition

Underlying space: $M_v := V_2+V_4+V_5$

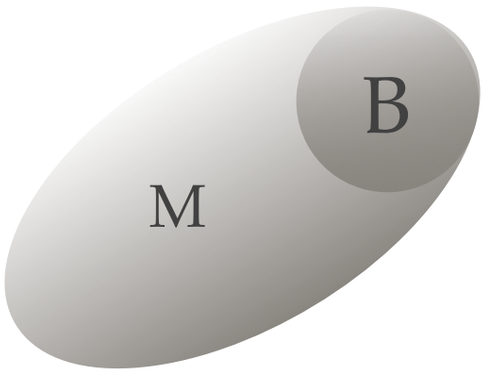
Boundary space $B_v := (V_2+V_4+V_5) \cap (V_1+V_3+V_6)$



If branch-width$\leq$w, then we can keep dim (B)$\leq$2w.



What do we keep?



For a path-decomposition $(S_1,S_2,\ldots,S_m)$

For each “gap”, there is a triple (L,R,λ)

S_1 | S_2 | S_3 | S_4 | S_5 | S_6 | S_7

“Left subspace” shown on B

$L_3=(S_1+S_2+S_3)\cap B$

“Right subspace” shown on B

$R_3=(S_4+S_5+S_6+S_7)\cap B$

Extra connectivity not shown in B

$\lambda_3=\dim(S_1+S_2+S_3)\cap(S_4+S_5+S_6+S_7)-\dim((S_1+S_2+S_3)\cap(S_4+S_5+S_6+S_7)\cap B)$

For (M,B), we only need to keep a sequence of (L,R,λ) in order to determine whether we have a path-decomposition

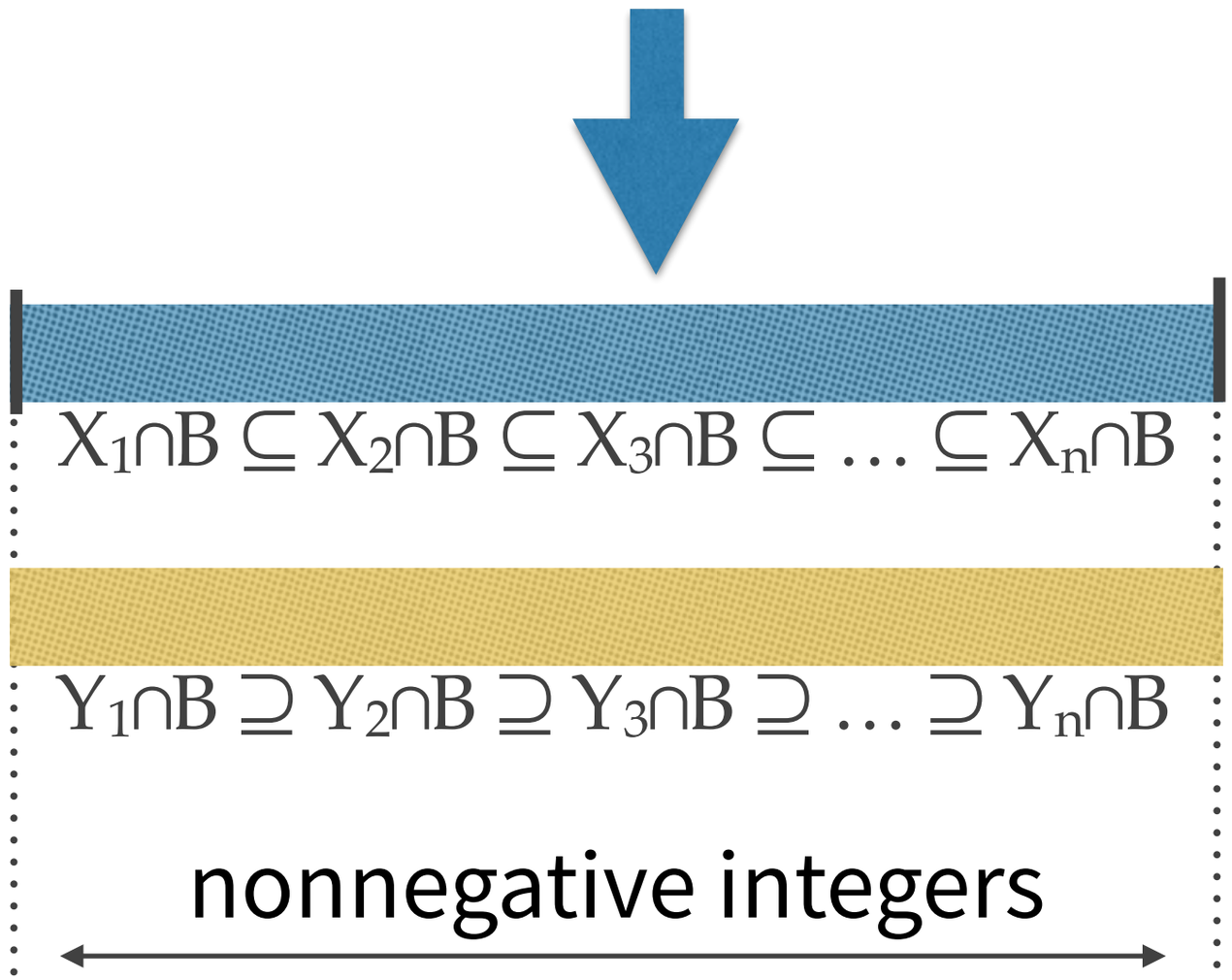
B-trajectory for a subspace B

- statistic:=triple (L,R,λ) of subspaces L, R of B and $\lambda \geq 0$.
- **B-trajectory**:=a finite sequence of statistics $\Gamma=a_1,a_2,\ldots,a_n$ such that
  $L(a_1)\subseteq L(a_2)\subseteq\ldots\subseteq L(a_n)$,
  $R(a_1)\supseteq R(a_2)\supseteq\ldots\supseteq R(a_n)$.
- Width of a B-trajectory:= max λ.
- The **canonical B-trajectory** of a path-decomposition $(S_1,S_2,\ldots,S_n)$

A path-decomposition $(S_1,S_2,\ldots,S_{n-1})$

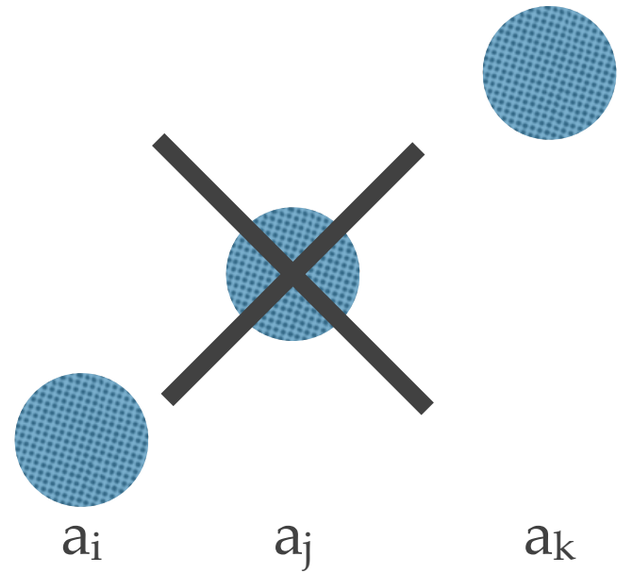
$$X_i := S_1+S_2+\ldots+S_i$$

$$Y_i := S_{i+1}+\ldots+S_{n-1}$$



How to compress B-trajectories?
Typical sequences of Bodlaender and Kloks

- Reducing operation for a sequence of integers:
  In a sequence $a_1,a_2,\ldots,a_n$ of integers,
  remove a_j if a_j is between a_i and a_k and i<j<k.



  - τ(1 2 5 3 9 2) = (1 9 2)
  - τ(1 3 8 4 5 6 3 7) = (1 8 3 7)

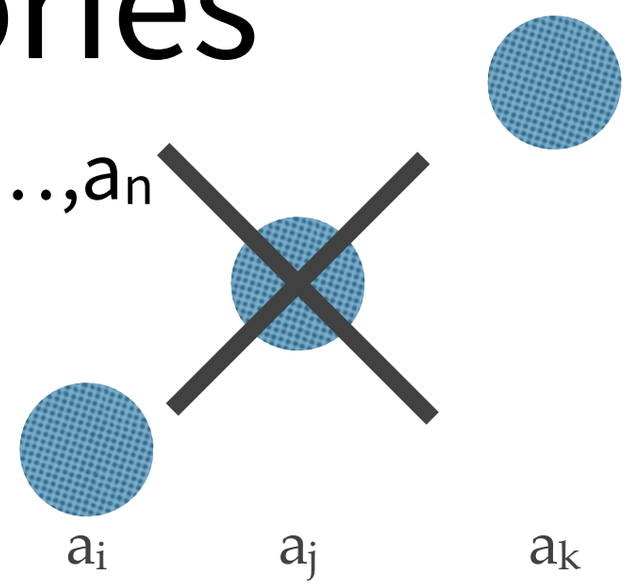
- A sequence is typical if no further reducing is possible.

- Lemma (Bodlaender and Kloks 1996):
  (#typical sequences consisting of {0,1,2,…,k}) $\leq (8/3)2^{2k}$.

How to compress B-trajectories?

"Compact" B-trajectories

- Reducing operation for a B-trajectory $\Gamma=a_1,a_2,\ldots,a_n$
  remove a_j if
  $L(a_i)=L(a_k)$, $R(a_i)=R(a_k)$, and
  $\lambda(a_j)$ is between $\lambda(a_i)$ and $\lambda(a_k)$ and $i<j<k$.



- A B-trajectory is <u>compact</u> if no further reducing is possible.
  The compactification $\tau(\Gamma)$ is a compact B-trajectory obtained by reducing from Γ.

- Lemma: if dim(B)=w and $|\mathbb{F}|$=q, then
  (#compact B-trajectories of width$\leq$k) $\leq 2^{9w+2}q^{w(w-1)}2^{2(2w+1)k}$

What to store during dynamic programming?
"Full Sets"

- For a subspace arrangement $\boldsymbol{\mathcal{V}}=\{V_1,V_2,..,V_n\}$,

  the **full set** $FS_k(\boldsymbol{\mathcal{V}},B)$:=set of all B-trajectories of width$\leq k$ that are better than some B-trajectories, realizable in $\boldsymbol{\mathcal{V}}$.

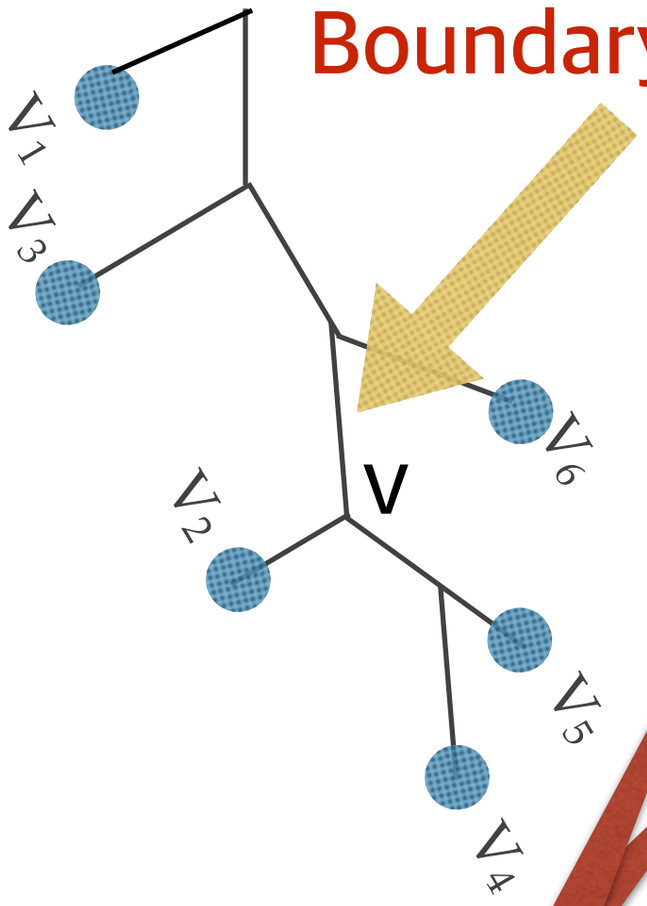
- $FS_k(\boldsymbol{\mathcal{V}},\{0\})\neq\emptyset$ if and only if path-width$\leq k$.

We aim to compute $FS_k(\boldsymbol{\mathcal{V}},\{0\})$ by the dynamic programming

Computing the Full Set

Underlying space: $M_v := V_2+V_4+V_5$

Boundary space $B_v := (V_2+V_4+V_5) \cap (V_1+V_3+V_6)$



Expand

Join

Shrink

At each leaf v, compute FS(M_v, B_v) "easy"

At each internal node v

with two children x and y,

- compute FS(M_x, B_x+B_y) from FS(M_x, B_x)
- compute FS(M_y, B_x+B_y) from FS(M_y, B_y)
- compute FS(M_v, B_x+B_y) from FS(M_x, B_x+B_y) and FS(M_y, B_x+B_y)
- compute FS(M_v,B_v) from FS(M_v, B_x+B_y)

O(n)-time to compute $FS_k(\mathcal{V},\{0\})$.

How to provide an initial "approximate" branch-decomposition

- Method 1: Iterative compression; O(n) overhead
  Modify the output for $V_1,\ldots,V_{i-1}$ of width$\leq$k to be the branch-decomposition of width$\leq$k+1.
  TOTAL: $O(n^4)$ time (simpler but slower)

- Method 2: Use the algorithm by Hlineny and Oum (2008) that provides a branch-decomposition of width$\leq$k in $O(n^3)$-time (faster)
  TOTAL: $O(n^3)$ time.

Concluding remarks

- By backtracking, we can construct a linear layout of width$\leq$k.
- Similar idea works for rank-width of graphs and branch-width of matroids representable over a fixed finite field.
- Two bottlenecks for a faster algorithm:
  - Finding an approximate branch-decomposition.
  - Preprocessing the approximate branch-decomposition to make it more useful for dynamic programming.
    (e.g. Precompute a basis for each boundary space B)

THANK YOU FOR YOUR ATTENTION!