Space Saving by Dynamic Algebraization

Martin Fürer and Huiwen Yu

Department of Compute Science and Engineering
The Pennsylvania State University
furer@cse.psu.edu
yhw.huiwenyu@gmail.com

June 8, 2014

Space saving by dynamic algebraization

- Outline:

- Exact computations of hard problems on sparse graphs.
- One method: Tree decomposition + Dynamic programming
  – *exponential space*
- Technique: Using algebraic transform to reduce space complexity based on a tree decomposition.
- Applications: Counting perfect matchings, counting set packings, counting set covers.

Saving space using algebraization - an Example of the Fourier transform

Based on work by Lokshtanov and Nederlof, STOC 2010.

- Subset Sum:

Given a set of positive integers $I = \{i_1, i_2, ..., i_n\}$, count the number of subsets of I with element sum equal to t.

- DP in space $O(t)$.
  $s[j, d]$: number of subsets from $\{i_1, ..., i_j\}$ with sum d.
- Define $F(x) = (1 + x^{i_1})(1 + x^{i_2}) \cdots (1 + x^{i_n}) = \sum f_i x^i$.
  1. $f_t = s[n, t]$ is the target number.
  2. DFT: $f_t = \frac{1}{m} \sum_{j=0}^{m-1} \omega^{-jt} F(\omega^j)$, $\omega^m = 1$, $m > nt$.
  3. f_t can be evaluated in polynomial space.

The zeta transform of a function $f \in \mathcal{R}[2^V]$:

$$\zeta f[X] = \sum_{Y \subseteq X} f[Y].$$

The Möbius transform/inversion of f:

$$\mu f[X] = \sum_{Y \subseteq X} (-1)^{|X \setminus Y|} f[Y].$$

It is the inverse transform of the zeta transform:

$$\mu(\zeta f)[X] = f[X].$$

The union product:

$$f *_u g[X] = \sum_{X_1 \bigcup X_2 = X} f(X_1) g(X_2).$$

The union product and the zeta transform:

$$\zeta(f *_u g)[X] = (\zeta f)[X] \cdot (\zeta g)[X]$$

.

Saving space using algebraization - the Framework

Methodology:

- Avoid large computation table for $f[Y]$, $Y \subseteq X$.
- Work with zeta transformed values $\zeta f[Y]$ (with one component at a time).
- At the end do a Möbis transform to get $f[X]$ back.

$$f[X] = \mu(\zeta f)[X]$$

- This is possible, as union products transform into products of the values.

$$\zeta(f *_u g)[X] = (\zeta f)[X] \cdot (\zeta g)[X].$$

We are interested in algorithms not computing union products, but subset convolutions.

union product:

$$f *_u g[X] = \sum_{X_1 \bigcup X_2 = X} f(X_1)g(X_2).$$

Algorithms often use subset convolution:

$$f *_{\mathcal{R}} g[X] = \sum_{Y \subseteq X} f[Y]g[X \setminus Y].$$

But a subset convolution can be simulated by n union products, one for each size of the resulting sets.

Saving space using algebraization - the Framework

- $C \in (\mathbb{Z}[2^V]; \oplus, *)$ outputs $f[V]$.
  Gate $a \to$ a relaxation $\{a_i\}_{i=1}^{|V|}$ of a. ($a_i[X] = a[X]$ if $|X| = i$ or 0 if $|X| < i$.)
  $a = b \oplus c \to a_i = b_i \oplus c_i$.
  $a = b * c \to a_i = \sum_{j=0}^{i} b_j *_u c_{i-j}$, for $0 \le i \le |V|$.
- $C_1 \in (\mathbb{Z}[2^V]; \oplus, *_u)$ outputs $f_{|V|}[V]$.
  $*_u$ gate $\to \odot$.
  constant gate $a \to \zeta a$.
- $C_2 \in (\mathbb{Z}[2^V]; \oplus, \odot)$, for every gate $a \in C_1$, the corresponding gate in C_2 outputs ζa.
- $C_2 \to 2^{|V|}$ disjoint circuits C^Y over $(\mathbb{Z}[2^V]; +, \cdot)$, $\forall Y \subseteq V$.
  C^Y outputs $(\zeta f)[Y]$.
- $$f[V] = \sum_{X \subseteq V} (-1)^{|V \setminus X|} (\zeta f)[X].$$

Tree decomposition

$G = (V, E)$, a tree decomposition of G is a tree $\mathcal{T} = (V_\mathcal{T}, E_\mathcal{T})$, $\forall x \in V_\mathcal{T}$ associate with a set B_x (a bag of x),

1. $\forall x, y$, node $z \in$ path connecting x and y in $\mathcal{T}$, $B_x \cap B_y \subseteq B_z$.
2. $\forall e = \{u, v\} \in E$, $\exists x$ such that $u, v \in B_x$. (e is associated with x.)
3. $\bigcup_{x \in V_\mathcal{T}} B_x = V$.

Treewidth: $\max_{x \in V_\mathcal{T}} |B_x| - 1$. NP

A path decomposition - $\mathcal{T}$ is a path.

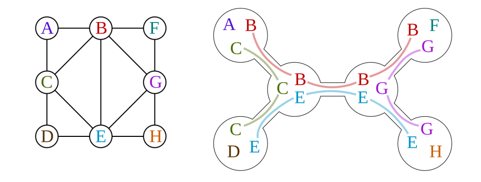


Figure : Illustrative figure for tree decomposition. (Pic. from wikipedia)

Nice tree decomposition

The degree of any node ≤ 2.
$c \rightarrow$ the only child of x.
$c_1, c_2 \rightarrow$ two children of x.

Any node x in a nice tree decomposition:

1. An introduce vertex node (introduce vertex v), $B_x = B_c \bigcup \{v\}$.
2. An introduce edge node (introduce edge $e = \{u, v\}$), $u, v \in B_x$, e is associated with x, $B_x = B_c$.
3. A forget vertex node (forget vertex v), $B_x = B_c \setminus \{v\}$.
4. A join node, x has two children, $B_x = B_{c_1} = B_{c_2}$.

Further transform the leaf nodes and the root into empty node.

Use counting perfect matchings as an example.

- Subtree $\mathcal{T}_x$ rooted at x;

T_x: nodes in $\mathcal{T}_x$ but B_x.
$X \subseteq B_x$: $Y_X \leftarrow X \cup T_x$.
$f_x[X]$: $\sharp$ perfect matchings in Y_X.

1. An introduce vertex node. $B_x = B_c \cup \{v\}$.

$$f_x[X] = \begin{cases} f_c[X] & v \notin X \\ 0 & v \in X \end{cases}$$

$$(\zeta f_x)[X] = \begin{cases} (\zeta f_c)[X] & v \notin X \\ (\zeta f_c)[X \setminus \{v\}] & v \in X \end{cases}$$

- $f_x[X]$: $\sharp$ perfect matchings in Y_X.

3. A forget vertex node (forget vertex v), $B_x = B_c \setminus \{v\}$.
$f_x[X] = f_c[X \cup \{v\}]$.

$$
\begin{aligned}
(\zeta f_x)[X] &= \sum_{X' \subseteq X} f_x[X'] = \sum_{X' \subseteq X} f_c[X' \cup \{v\}] \\
&= (\zeta f_c)[X \cup \{v\}] - (\zeta f_c)[X].
\end{aligned}
$$

4. A join node, x has two children and $B_x = B_{c_1} = B_{c_2}$.

$$
f_x[X] = \sum_{X' \subseteq X} f_{c_1}[X'] f_{c_2}[X \setminus X'] = f_{c_1} * f_{c_2}[X].
$$

5. A leaf node, a leaf of $\mathcal{T}$. $f_x[\emptyset] = 1$.

2. An introduce edge node (introduce edge $e = \{u, v\}$), $B_x = B_c$.

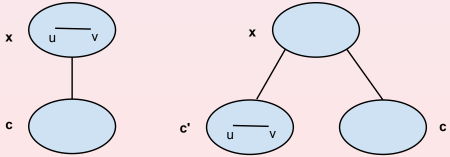
$$f_x[X] = \begin{cases} f_c[X] & e \not\subseteq X \\ f_c[X] + f_c[X \setminus \{u, v\}] & e \subseteq X \end{cases}$$

$$(\zeta f_x)[X] = \begin{cases} (\zeta f_c)[X] & e \not\subseteq X \\ (\zeta f_c)[X] + (\zeta f_c)[X \setminus \{u, v\}] & e \subseteq X \end{cases}$$

- Modified the construction:

Add a new child node c', $B_x = B_{c'}$. Introduce e in $B_{c'}$.

$x \to$ join node.



- Algorithm:

- Follow depth-first search in-order of the tree.
- Branch on the forget vertex node.

$$(\zeta f_x)[X] = (\zeta f_c)[X \cup \{v\}] - (\zeta f_c)[X]$$

- Otherwise, “point-wise” addition or multiplication.

Introduce vertex node:

$$(\zeta f_x)[X] = \begin{cases} (\zeta f_c)[X] & v \notin X \\ (\zeta f_c)[X \setminus \{v\}] & v \in X \end{cases}$$

Join node:

$$(\zeta f_x)[X] = (\zeta f_{c_1})[X] \cdot (\zeta f_{c_2})[X]$$

- Complexity:
  - time: $O((|E|+|V|)2^h)$, $h =$ max number of forget nodes along any root-to-leaf path.

$$T[j] \leq 2 \cdot 2^{\max\{h_1,h_2\}} \max\{T[j_1], T[j_2]\}$$

  - space: poly, handle one subset at a time.
- Parameter h:
  - max number of vertices along any root-to-leaf path.
  - Equivalent to “tree-depth”.
  - $k+1 \leq h \leq O(\log|V|)(k+1)$.

Saving space on tree decomposition - Main result

Theorem

For any graph $G = (V, E)$ and a modified nice tree decomposition $\mathcal{T}$ on G. Assume the number of forget nodes along any path from the root to a leaf in $\mathcal{T}$ is at most h. Let f be a function evaluated by a circuit C over $(\mathbb{Z}[2^V]; \oplus, *)$ with constants being singletons. Assume $f[V] \leq m$ for integer m. We can compute $f[V]$ in time $O((|V| + |E|)2^h)$ and in space $O(|V||C| \log m)$.

Counting perfect matchings on grids - Balanced tree decomposition on grids

Monomer-Dimer problem - an important problem in statistical physics.

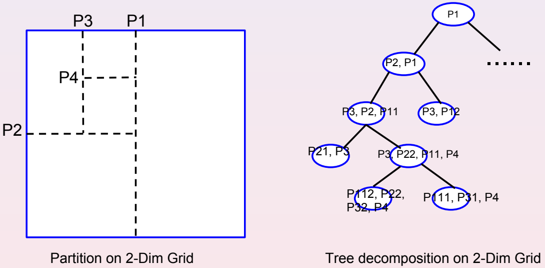


Figure : Illustrative figure for balanced tree decomposition on 2-dimensional grids. Always bipartition the longer side of the grid/subgrid. P_i represent a balanced vertex separator. Denote the left/top half of P_i by P_{i1}, and the right/bottom part by P_{i2}.

Lemma

The treewidth of the tree decomposition on $G_d(n)$ is $\frac{3}{2}n^{d-1}$. The maximum number of forget nodes along any path from the root to a leaf is at most $\frac{2^d-1}{2^{d-1}-1}n^{d-1}$.

Theorem

The problem of counting perfect matchings on grids of dimension d and uniform length n can be solved in time $O^*(2^{\frac{2^d-1}{2^{d-1}-1}n^{d-1}})$ and in polynomial space.

Comparison to other algorithms

- DP (Dynamic Programming) based on path decomposition. Construct n nodes $\{p_1, p_2, ..., p_n\}$ associated with a bag of vertices with x_1 coordinate equal to j, for $j = 0, 1, ..., n-1$. For any p_j, p_{j+1}, start from p_j, add a sequence of nodes by alternating between adding a vertex of $x_1 = j+1$ and deleting its neighbor with $x_1 = j$. Pathwidth of n^{d-1}. Time $O^*(2^{n^{d-1}})$, space $O^*(2^{n^{d-1}})$.

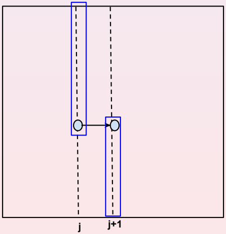


Figure : Path decomposition on grids

Comparison to other algorithms

- DP based on path decomposition on a subgrid. Extract from $G_d(n)$ a subgrid of pathwidth $O(\log n)$, delete a portion of vertices from $G_d(n)$ to turn a "cube"-shaped grid into a long "stripe" with $O(\log n)$ cross-section area. Remove $O(\frac{n^d}{(\log n)^{1/(d-1)}})$ vertices.
  Time $2^{O\left(\frac{n^d}{(\log n)^{1/(d-1)}}\right)}$, poly-space.

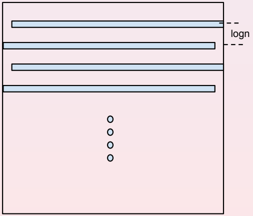


Figure : Path decomposition on subgrids.

Comparison to other algorithms

- Branching algorithm. First find a balanced separator S and partitioning the graph into $A \cup S \cup B$. Enumerate every subset $X \subseteq S$. A vertex in X either matches to vertices in A or to vertices in B. Vertices in $S \setminus X$ are matched within S. Recurse on A and B. $T_d(n) \leq 2T_d(\frac{n-|S|}{2}) \sum_{X \subseteq S} 2^{|X|} T_{d-1}(|S \setminus X|)$. Separator size $O(n^{d-1})$, $T_{d-1}(|S \setminus X|) = 2^{O(n^{d-2})}$.
Time $O^*(3^{\frac{2^d-1}{2^{d-1}-1} n^{d-1}})$, poly-space.

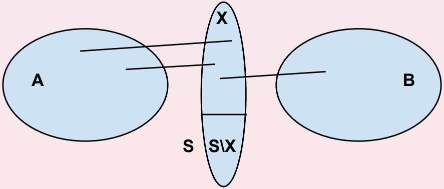


Figure : Balanced graph decomposition

Summary

- Comparison:

- *DP on path decomposition, pathwidth* $O(n^{d-1})$*: time* $O^*(2^{n^{d-1}})$*, space* $O^*(2^{n^{d-1}})$*.*
- DP on path decomposition, pathwidth $O(\log n)$: time $2^{O\left(\frac{n^d}{(\log n)^{1/(d-1)}}\right)}$, poly-space.
- Branching: time $O(3^h)$, $h = \frac{2^d-1}{2^{d-1}-1} n^{d-1}$, poly-space.
- DP by algebrazation: time $O(2^h)$, poly-space.

- Results generalized to more general grids, different length in each dimension.

Extensions

- Extensions to other problems:
  - YES: matching polynomial, counting set packings, counting set covers.
  - NO: independent sets (not a convolution), Hamiltonian paths, Steiner tree (poly-space?).
- Open problems:
  - Find other graph decompositions of large subgraphs?
  - Do other problems fit in this dynamic algebraization framework?