Edit distance neighbourhoods of input-driven pushdown automata

Alexander Okhotin  

Kai Salomaa

St. Petersburg, Russia  

Kingston, Canada

11 June, A. D. 2017
Edit distance

\[ d(w, w') \]: least number of edit operations.
Edit distance

\(d(w, w')\): least number of edit operations.

- **Insert** one symbol, **remove** one symbol, **[replace one with another]**.
Edit distance

$d(w, w')$: least number of edit operations.

- **Insert** one symbol, **remove** one symbol, [**replace** one with another].
- Folklore $O(mn)$ algorithm for computing the edit distance.
Edit distance

$d(w, w')$: least number of edit operations.

- Insert one symbol, remove one symbol, [replace one with another].
- Folklore $O(mn)$ algorithm for computing the edit distance.
- Sequence alignment.
Edit distance

\(d(w, w')\): least number of edit operations.

- **Insert** one symbol, **remove** one symbol, [**replace** one with another].
- Folklore \(O(mn)\) algorithm for computing the edit distance.
- Sequence alignment.
- Applications: `diff` in Unix, comparing two DNA strands, etc.
Edit distance

$d(w, w')$: least number of edit operations.

- **Insert** one symbol, **remove** one symbol, [**replace** one with another].
- Folklore $O(mn)$ algorithm for computing the edit distance.
- Sequence alignment.
- Applications: `diff` in Unix, comparing two DNA strands, etc.

$d(w, L)$: least number of operations to transform $w$ to some $w' \in L$. 
Edit distance

\[ d(w, w') \]: least number of edit operations.

- **Insert** one symbol, **remove** one symbol, [**replace** one with another].
- Folklore \(O(mn)\) algorithm for computing the edit distance.
- Sequence alignment.
- Applications: `diff` in Unix, comparing two DNA strands, etc.

\[ d(w, L) \]: least number of operations to transform \( w \) to some \( w' \in L \).

- \( w \) misspelled, but assumed to be in \( L \).
Edit distance

\[ d(w, w') \]: least number of edit operations.

- **Insert** one symbol, **remove** one symbol, [**replace** one with another].
- Folklore \( O(mn) \) algorithm for computing the edit distance.
- Sequence alignment.
- Applications: **diff** in Unix, comparing two DNA strands, etc.

\[ d(w, L) \]: least number of operations to transform \( w \) to some \( w' \in L \).

- \( w \) misspelled, but assumed to be in \( L \).
- Number of syntax errors.
Edit distance

$d(w, w')$: least number of edit operations.

- Insert one symbol, remove one symbol, [replace one with another].
- Folklore $O(mn)$ algorithm for computing the edit distance.
- Sequence alignment.
- Applications: diff in Unix, comparing two DNA strands, etc.

$d(w, L)$: least number of operations to transform $w$ to some $w' \in L$.

- $w$ misspelled, but assumed to be in $L$.
- Number of syntax errors.
- Error-correcting parsing algorithms.
Edit distance neighbourhood

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} \]: edit distance \( \ell \)-neighbourhood.
Edit distance neighbourhood

\[ E_\ell(L) = \{ w | d(w, L) \leq \ell \} : \text{edit distance } \ell\text{-neighbourhood.} \]

- Insertion of \( a \in \Sigma \); \( \text{insert}_a(L) = \{ uav | uv \in L \} \).
Edit distance neighbourhood

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} : \text{edit distance } \ell\text{-neighbourhood.} \]

- Insertion of \( a \in \Sigma \); \( \text{insert}_a(L) = \{ uav \mid uv \in L \} \).
- Deletion of \( a \in \Sigma \); \( \text{delete}_a(L) = \{ uv \mid uav \in L \} \).
Edit distance neighbourhood

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} : \text{edit distance } \ell\text{-neighbourhood.} \]

- Insertion of \( a \in \Sigma \); \( \text{insert}_a(L) = \{ uav \mid uv \in L \} \).
- Deletion of \( a \in \Sigma \); \( \text{delete}_a(L) = \{ uv \mid uav \in L \} \).
- 0-neighbourhood: \( E_0(L) = L \). (example: \( L = \{ ab \} \))
Edit distance neighbourhood

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} \]: edit distance \( \ell \)-neighbourhood.

- Insertion of \( a \in \Sigma \); \( \text{insert}_a(L) = \{ uav \mid uv \in L \} \).
- Deletion of \( a \in \Sigma \); \( \text{delete}_a(L) = \{ uv \mid uav \in L \} \).
- 0-neighbourhood: \( E_0(L) = L \). (example: \( L = \{ ab \} \))
- \( E_{\ell+1}(L) = E_\ell(L) \cup \bigcup_{a \in \Sigma} \left( \text{insert}_a(E_\ell(L)) \cup \text{delete}_a(E_\ell(L)) \right) \).
Edit distance neighbourhood

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} \]: edit distance \( \ell \)-neighbourhood.

- Insertion of \( a \in \Sigma \); \( \text{insert}_a(L) = \{ uav \mid uv \in L \} \).
- Deletion of \( a \in \Sigma \); \( \text{delete}_a(L) = \{ uv \mid uav \in L \} \).
- 0-neighbourhood: \( E_0(L) = L \). (example: \( L = \{ab\} \))
- \( E_{\ell+1}(L) = E_\ell(L) \cup \bigcup_{a \in \Sigma} \left( \text{insert}_a(E_\ell(L)) \cup \text{delete}_a(E_\ell(L)) \right) \).
Edit distance neighbourhood

\[ E_\ell(L) = \{w \mid d(w, L) \leq \ell\} : \text{edit distance } \ell\text{-neighbourhood.} \]

- Insertion of \(a \in \Sigma\); \( \text{insert}_a(L) = \{uav \mid uv \in L\} \).
- Deletion of \(a \in \Sigma\); \( \text{delete}_a(L) = \{uv \mid uav \in L\} \).
- 0-neighbourhood: \( E_0(L) = L \). (example: \( L = \{ab\} \))
- \( E_{\ell+1}(L) = E_\ell(L) \cup \bigcup_{a \in \Sigma} \left( \text{insert}_a(E_\ell(L)) \cup \text{delete}_a(E_\ell(L)) \right) \).
Edit distance neighbourhood

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} : \text{edit distance } \ell\text{-neighbourhood}. \]

- Insertion of \( a \in \Sigma \); \( \text{insert}_a(L) = \{ uav \mid uv \in L \} \).
- Deletion of \( a \in \Sigma \); \( \text{delete}_a(L) = \{ uv \mid uav \in L \} \).
- 0-neighbourhood: \( E_0(L) = L \). (example: \( L = \{ ab \} \))
- \( E_{\ell+1}(L) = E_\ell(L) \cup \bigcup_{a \in \Sigma} \left( \text{insert}_a(E_\ell(L)) \cup \text{delete}_a(E_\ell(L)) \right) \).
Edit distance neighbourhood

$E_\ell(L) = \{w \mid d(w, L) \leq \ell\}$: edit distance $\ell$-neighbourhood.

- Insertion of $a \in \Sigma$: $\text{insert}_a(L) = \{uav \mid uv \in L\}$.
- Deletion of $a \in \Sigma$: $\text{delete}_a(L) = \{uv \mid uav \in L\}$.
- 0-neighbourhood: $E_0(L) = L$. (example: $L = \{ab\}$)
- $E_{\ell+1}(L) = E_\ell(L) \cup \bigcup_{a \in \Sigma} (\text{insert}_a(E_\ell(L)) \cup \text{delete}_a(E_\ell(L)))$. 

\[
\begin{array}{c}
\varepsilon \\
\downarrow \\
a \\
\downarrow \\
ab \\
\downarrow \\
abb \\
\downarrow \\
abb \\
\downarrow \\
ab \\
\downarrow \\
ab \\
\downarrow \\
abb \\
\end{array}
\]
Edit distance neighbourhood

\[ E_\ell(L) = \{ w | d(w, L) \leq \ell \} \]: edit distance \( \ell \)-neighbourhood.

- Insertion of \( a \in \Sigma \): \( \text{insert}_a(L) = \{ uav | uv \in L \} \).
- Deletion of \( a \in \Sigma \): \( \text{delete}_a(L) = \{ uv | uav \in L \} \).
- 0-neighbourhood: \( E_0(L) = L \).  
  (example: \( L = \{ ab \} \))
- \( E_{\ell+1}(L) = E_\ell(L) \cup \bigcup_{a \in \Sigma} (\text{insert}_a(E_\ell(L)) \cup \text{delete}_a(E_\ell(L))) \).
**Edit distance neighbourhood**

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} \]  
edit distance \( \ell \)-neighbourhood.

- Insertion of \( a \in \Sigma \); \( \text{insert}_a(L) = \{ uav \mid uv \in L \} \).
- Deletion of \( a \in \Sigma \); \( \text{delete}_a(L) = \{ uv \mid uav \in L \} \).
- 0-neighbourhood: \( E_0(L) = L \).  
  (example: \( L = \{ ab \} \))
- \( E_{\ell+1}(L) = E_\ell(L) \cup \bigcup_{a \in \Sigma} (\text{insert}_a(E_\ell(L)) \cup \text{delete}_a(E_\ell(L))) \).
Edit distance neighbourhood

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} : \text{edit distance } \ell\text{-neighbourhood.} \]

- Insertion of \( a \in \Sigma \); \( \text{insert}_a(L) = \{ uav \mid uv \in L \} \).
- Deletion of \( a \in \Sigma \); \( \text{delete}_a(L) = \{ uv \mid uav \in L \} \).
- 0-neighbourhood: \( E_0(L) = L \). (example: \( L = \{ ab \} \))
- \( E_{\ell+1}(L) = E_\ell(L) \cup \bigcup_{a \in \Sigma} \left( \text{insert}_a(E_\ell(L)) \cup \text{delete}_a(E_\ell(L)) \right) \).
**Edit distance neighbourhood**

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} \]: edit distance \( \ell \)-neighbourhood.

- Insertion of \( a \in \Sigma \); \( \text{insert}_a(L) = \{ uav \mid uv \in L \} \).
- Deletion of \( a \in \Sigma \); \( \text{delete}_a(L) = \{ uv \mid uav \in L \} \).
- 0-neighbourhood: \( E_0(L) = L \). (example: \( L = \{ ab \} \))
- \( E_{\ell+1}(L) = E_\ell(L) \cup \bigcup_{a \in \Sigma} \left( \text{insert}_a(E_\ell(L)) \cup \text{delete}_a(E_\ell(L)) \right) \).

---

\[ E_1(L) = \{ ab \} \]

\[ E_2(L) = \{ ab, aba, bab \} \]

\[ E_3(L) = \{ ab, aba, bab, aabb, aaab \} \]
**Edit distance neighbourhood**

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} : \text{edit distance } \ell\text{-neighbourhood.} \]

- Insertion of \( a \in \Sigma \): \( \text{insert}_a(L) = \{ uav \mid uv \in L \} \).
- Deletion of \( a \in \Sigma \): \( \text{delete}_a(L) = \{ uv \mid uav \in L \} \).
- 0-neighbourhood: \( E_0(L) = L \). (example: \( L = \{ ab \} \))
- \( E_{\ell+1}(L) = E_\ell(L) \cup \bigcup_{a \in \Sigma} (\text{insert}_a(E_\ell(L)) \cup \text{delete}_a(E_\ell(L))) \).
**Edit distance neighbourhood**

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} : \text{edit distance } \ell\text{-neighbourhood.} \]

- **Insertion of** \( a \in \Sigma \): \( \text{insert}_a(L) = \{ uav \mid uv \in L \} \).
- **Deletion of** \( a \in \Sigma \): \( \text{delete}_a(L) = \{ uv \mid uav \in L \} \).
- **0-neighbourhood**: \( E_0(L) = L \). \hspace{1cm} \text{(example: } L = \{ ab \} \)
- \( E_{\ell+1}(L) = E_\ell(L) \cup \bigcup_{a \in \Sigma} (\text{insert}_a(E_\ell(L)) \cup \text{delete}_a(E_\ell(L))) \).
Edit distance neighbourhood

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} \]: edit distance \( \ell \)-neighbourhood.

- Insertion of \( a \in \Sigma \); \( \text{insert}_a(L) = \{ uav \mid uv \in L \} \).
- Deletion of \( a \in \Sigma \); \( \text{delete}_a(L) = \{ uv \mid uav \in L \} \).
- 0-neighbourhood: \( E_0(L) = L \). (example: \( L = \{ ab \} \))
- \( E_{\ell+1}(L) = E_\ell(L) \cup \bigcup_{a \in \Sigma} \left( \text{insert}_a(E_\ell(L)) \cup \text{delete}_a(E_\ell(L)) \right) \).

\[
\begin{align*}
\text{aaab} & \rightarrow \text{aabb} & \rightarrow \text{aaba} \\
\text{aa} & \rightarrow \text{bab} & \rightarrow \text{baab} \\
\text{ba} & \rightarrow \text{a} & \rightarrow \text{aab} \\
\text{b} & \rightarrow \text{ab} & \rightarrow \text{aba} \\
\varepsilon & \rightarrow \text{abb} & \rightarrow \text{bab} \\
\text{bb} & \rightarrow \text{bab}
\end{align*}
\]
Edit distance neighbourhood

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} : \text{edit distance } \ell\text{-neighbourhood.} \]

- Insertion of \( a \in \Sigma \); \( \text{insert}_a(L) = \{ uav \mid uv \in L \} \).
- Deletion of \( a \in \Sigma \); \( \text{delete}_a(L) = \{ uv \mid uav \in L \} \).
- 0-neighbourhood: \( E_0(L) = L \). (example: \( L = \{ ab \} \))
- \( E_{\ell+1}(L) = E_\ell(L) \cup \bigcup_{a \in \Sigma} \left( \text{insert}_a(E_\ell(L)) \cup \text{delete}_a(E_\ell(L)) \right) \).
Closure under edit distance neighbourhood

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \}: \text{edit distance } \ell\text{-neighbourhood.} \]
Closure under edit distance neighbourhood

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} : \text{edit distance } \ell\text{-neighbourhood.} \]

- Regular languages: closed (folklore?).

\[ E_1(L) \cap a^*b^* = \{ a^n b^n \mid n \geq 0 \} \cup \{ a^n b^{2n} \mid n \geq 0 \} \]
Closure under edit distance neighbourhood

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} \] : edit distance \( \ell \)-neighbourhood.

- Regular languages: closed (folklore?).
Closure under edit distance neighbourhood

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} : \text{edit distance} \ \ell\text{-neighbourhood.} \]

• Regular languages: closed (folklore?).
  ▶ Salomaa and Schofield (2007): for \( \ell \)-neighbourhood.

\[ L = \{ \text{can}^n \text{bn} \mid n \geq 0 \} \cup \{ \text{da}^n \text{bn}^2 \mid n \geq 0 \} \]

\[ E_1(L) \cap a^*b^* = \{ a^n b^n \mid n \geq 0 \} \cup \{ a^n b^{2n} \mid n \geq 0 \} \]
Closure under edit distance neighbourhood

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} : \text{edit distance } \ell\text{-neighbourhood.} \]

- Regular languages: closed (folklore?).
  - Salomaa and Schofield (2007): for \( \ell \)-neighbourhood.
Closure under edit distance neighbourhood

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} \]: edit distance \( \ell \)-neighbourhood.

- Regular languages: closed (folklore?).
  - Salomaa and Schofield (2007): for \( \ell \)-neighbourhood.


- LR(\(k\)) languages (DPDA)?

The same question for another important family.
Closure under edit distance neighbourhood

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} : \text{edit distance } \ell\text{-neighbourhood.} \]

- Regular languages: closed (folklore?).
  - Salomaa and Schofield (2007): for \( \ell \)-neighbourhood.
- LR(\(k\)) languages (DPDA): not closed.

\[ L = \{ ca^n b^n \mid n \geq 0 \} \cup \{ da^n b^{2n} \mid n \geq 0 \} \]
Closure under edit distance neighbourhood

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} : \text{edit distance } \ell\text{-neighbourhood.} \]

- Regular languages: closed (folklore?).
  - Salomaa and Schofield (2007): for \( \ell \)-neighbourhood.
- LR\((k)\) languages (DPDA): not closed.

\[ L = \{ ca^n b^n \mid n \geq 0 \} \cup \{ da^n b^{2n} \mid n \geq 0 \} \]

\[ E_1(L) \cap a^* b^* = \{ a^n b^n \mid n \geq 0 \} \cup \{ a^n b^{2n} \mid n \geq 0 \} \]
Closure under edit distance neighbourhood

\[ E_\ell(L) = \{ w \mid d(w, L) \leq \ell \} : \text{edit distance } \ell\text{-neighbourhood.} \]

- Regular languages: closed (folklore?).
  - Salomaa and Schofield (2007): for \( \ell\)-neighbourhood.
- LR(\(k\)) languages (DPDA): not closed.

\[ L = \{ ca^n b^n \mid n \geq 0 \} \cup \{ da^n b^{2n} \mid n \geq 0 \} \]
\[ E_1(L) \cap a^* b^* = \{ a^n b^n \mid n \geq 0 \} \cup \{ a^n b^{2n} \mid n \geq 0 \} \]

✓ The same question for another important family.
Input-driven pushdown automata
(a.k.a. visibly pushdown automata, a.k.a. nested word automata)

\[ \Sigma = \Sigma_+ \cup \Sigma_0 \cup \Sigma_- : \text{input alphabet}; \]
Input-driven pushdown automata
(a.k.a. visibly pushdown automata, a.k.a. nested word automata)

- $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$: input alphabet;
- $Q$: finite set of states;

$\delta < : Q \rightarrow Q \times \Gamma$, for $< \in \Sigma_{+1}$;
$\delta c : Q \rightarrow Q$, for $c \in \Sigma_0$.
$\delta > : Q \times (\Gamma \cup \{\bot\}) \rightarrow Q$, for $> \in \Sigma_{-1}$;
Transition by the empty stack ($\bot$).

$F \subseteq Q$: accepting states.
Input-driven pushdown automata
(a.k.a. visibly pushdown automata, a.k.a. nested word automata)

- $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$: input alphabet;
- $Q$: finite set of states;
- $q_0 \in Q$: initial state;
Input-driven pushdown automata
(a.k.a. visibly pushdown automata, a.k.a. nested word automata)

- $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$: input alphabet;
- $Q$: finite set of states;
- $q_0 \in Q$: initial state;
- $\Gamma$: stack alphabet;
- $F \subseteq Q$: accepting states.
Input-driven pushdown automata
(a.k.a. visibly pushdown automata, a.k.a. nested word automata)

- $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$: input alphabet;
- $Q$: finite set of states;
- $q_0 \in Q$: initial state;
- $\Gamma$: stack alphabet;
- $\delta_<: Q \to Q \times \Gamma$, for $< \in \Sigma_{+1}$;
- $\delta_c: Q \to Q$, for $c \in \Sigma_0$.
- $\delta_> : Q \times (\Gamma \cup \{\bot\}) \to Q$, for $> \in \Sigma_{-1}$;
- $F \subseteq Q$: accepting states.
Input-driven pushdown automata
(a.k.a. visibly pushdown automata, a.k.a. nested word automata)

- $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$: input alphabet;
- $Q$: finite set of states;
- $q_0 \in Q$: initial state;
- $\Gamma$: stack alphabet;
- $\delta_{\prec}: Q \to Q \times \Gamma$, for $\prec \in \Sigma_{+1}$;
- $\delta_c: Q \to Q$, for $c \in \Sigma_0$.

Transition by the empty stack ($\bot$).
Stack symbols that are never popped.

$F \subseteq Q$: accepting states.

A. Okhotin, K. Salomaa
Edit distance on input-driven PDAs
CSR 2017 (Kazan) 5 / 15
Input-driven pushdown automata
(a.k.a. visibly pushdown automata, a.k.a. nested word automata)

- $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$: input alphabet;
- $Q$: finite set of states;
- $q_0 \in Q$: initial state;
- $\Gamma$: stack alphabet;
- $\delta_<$: $Q \rightarrow Q \times \Gamma$, for $< \in \Sigma_{+1}$;
- $\delta_c$: $Q \rightarrow Q$, for $c \in \Sigma_0$;
- $\delta_> : Q \times (\Gamma \cup \{\bot\}) \rightarrow Q$, for $> \in \Sigma_{-1}$;
- $F \subseteq Q$: accepting states.

![Graphical representation of an input-driven PDA]
Input-driven pushdown automata
(a.k.a. visibly pushdown automata, a.k.a. nested word automata)

• \( \Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1} \): input alphabet;
• \( Q \): finite set of states;
• \( q_0 \in Q \): initial state;
• \( \Gamma \): stack alphabet;
• \( \delta_< : Q \rightarrow Q \times \Gamma \), for \( < \in \Sigma_{+1} \);
• \( \delta_c : Q \rightarrow Q \), for \( c \in \Sigma_0 \).
• \( \delta_> : Q \times (\Gamma \cup \{\perp\}) \rightarrow Q \), for \( > \in \Sigma_{-1} \);
• Transition by the empty stack (\( \perp \)).
Input-driven pushdown automata
(a.k.a. visibly pushdown automata, a.k.a. nested word automata)

- $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$: input alphabet;
- $Q$: finite set of states;
- $q_0 \in Q$: initial state;
- $\Gamma$: stack alphabet;
- $\delta_\prec: Q \rightarrow Q \times \Gamma$, for $\prec \in \Sigma_{+1}$;
- $\delta_c: Q \rightarrow Q$, for $c \in \Sigma_0$.
- $\delta_\succ: Q \times (\Gamma \cup \{\bot\}) \rightarrow Q$, for $\succ \in \Sigma_{-1}$;
- Transition by the empty stack ($\bot$).
- Stack symbols that are never popped.
Input-driven pushdown automata
(a.k.a. visibly pushdown automata, a.k.a. nested word automata)

- $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$: input alphabet;
- $Q$: finite set of states;
- $q_0 \in Q$: initial state;
- $\Gamma$: stack alphabet;
- $\delta_<$: $Q \rightarrow Q \times \Gamma$, for $< \in \Sigma_{+1}$;
- $\delta_c$: $Q \rightarrow Q$, for $c \in \Sigma_0$.
- $\delta_>$: $Q \times (\Gamma \cup \{\bot\}) \rightarrow Q$, for $> \in \Sigma_{-1}$;
- Transition by the empty stack ($\bot$).
- Stack symbols that are never popped.
- $F \subseteq Q$: accepting states.
Research on IDPDAs

- Languages recognized in space $\frac{\log^2 n}{\log \log n}$ and poly time on TM (Mehlhorn, ICALP 1980).
- NIDPDA equivalent to DIDPDA (von Braunmühl, Verbeek, 1983).
- Thorough study (Alur, Madhusudan, STOC 2004).
- Rediscovered as “visibly pushdown automata”/“nested word automata”.
- Applications to verification.
- $2^\Theta(n^2)$ cost of determinization.
- Closed under Boolean operations, concatenation, Kleene star.

Much ongoing research: algorithms, complexity, closure properties. . .

Closure under edit distance $\ell$-neighbourhood?

For $L$ recognized by an IDPDA, is $E_\ell(L)$ always recognized by one?
Research on IDPDAs

- Languages recognized in space $\frac{\log^2 n}{\log \log n}$ and poly time on TM (Mehlhorn, ICALP 1980).
- ... in space $\log n$ and time $n^2 \log n$ (von Braunmühl, Verbeek, 1983).

Thorough study (Alur, Madhusudan, STOC 2004).

Rediscovered as “visibly pushdown automata”/“nested word automata”

Applications to verification.

$\Theta(n^2)$ cost of determinization.

Closed under Boolean operations, concatenation, Kleene star.

Much ongoing research: algorithms, complexity, closure properties.

Closure under edit distance $\ell$-neighbourhood?

For $L$ recognized by an IDPDA, is $E_\ell(L)$ always recognized by one?
Research on IDPDAs

- Languages recognized in space $\frac{\log^2 n}{\log \log n}$ and poly time on TM (Mehlhorn, ICALP 1980).
- ... in space $\log n$ and time $n^2 \log n$ (von Braunmühl, Verbeek, 1983).
- NIDPDA equivalent to DIDPDA (von Braunmühl, Verbeek, 1983).
Research on IDPDAs

- Languages recognized in space $\frac{\log^2 n}{\log \log n}$ and poly time on TM (Mehlhorn, ICALP 1980).
- ...in space $\log n$ and time $n^2 \log n$ (von Braunmühl, Verbeek, 1983).
- NIDPDA equivalent to DIDPDA (von Braunmühl, Verbeek, 1983).
- Thorough study (Alur, Madhusudan, STOC 2004).
  - Rediscovered as “visibly pushdown automata” / “nested word automata”
Research on IDPDAs

- Languages recognized in space \( \frac{\log^2 n}{\log \log n} \) and poly time on TM (Mehlhorn, ICALP 1980).
- ... in space \( \log n \) and time \( n^2 \log n \) (von Braunmühl, Verbeek, 1983).
- NIDPDA equivalent to DIDPDA (von Braunmühl, Verbeek, 1983).
- Thorough study (Alur, Madhusudan, STOC 2004).
  - Rediscovered as “visibly pushdown automata” / “nested word automata”
  - Applications to verification.
Research on IDPDAs

- Languages recognized in space $\frac{\log^2 n}{\log \log n}$ and poly time on TM (Mehlhorn, ICALP 1980).
- ... in space $\log n$ and time $n^2 \log n$ (von Braunmühl, Verbeek, 1983).
- NIDPDA equivalent to DIDPDA (von Braunmühl, Verbeek, 1983).
- Thorough study (Alur, Madhusudan, STOC 2004).
  - Rediscovered as “visibly pushdown automata” / “nested word automata”
  - Applications to verification.
  - $2^{\Theta(n^2)}$ cost of determinization.
Research on IDPDAs

- Languages recognized in space $\frac{\log^2 n}{\log \log n}$ and poly time on TM (Mehlhorn, ICALP 1980).
- ... in space $\log n$ and time $n^2 \log n$ (von Braunmühl, Verbeek, 1983).
- NIDPDA equivalent to DIDPDA (von Braunmühl, Verbeek, 1983).
- Thorough study (Alur, Madhusudan, STOC 2004).
  - Rediscovered as “visibly pushdown automata” / “nested word automata”
  - Applications to verification.
  - $2^{\Theta(n^2)}$ cost of determinization.
  - Closed under Boolean operations, concatenation, Kleene star.

Much ongoing research: algorithms, complexity, closure properties...
Research on IDPDAs

- Languages recognized in space $\frac{\log^2 n}{\log \log n}$ and poly time on TM (Mehlhorn, ICALP 1980).
- ... in space $\log n$ and time $n^2 \log n$ (von Braunmühl, Verbeek, 1983).
- NIDPDA equivalent to DIDPDA (von Braunmühl, Verbeek, 1983).
- Thorough study (Alur, Madhusudan, STOC 2004).
  - Rediscovered as “visibly pushdown automata” / “nested word automata”
  - Applications to verification.
  - $2^{\Theta(n^2)}$ cost of determinization.
  - Closed under Boolean operations, concatenation, Kleene star.
- Much ongoing research: algorithms, complexity, closure properties...
Research on IDPDAs

- Languages recognized in space $\frac{\log^2 n}{\log \log n}$ and poly time on TM (Mehlhorn, ICALP 1980).
- ... in space $\log n$ and time $n^2 \log n$ (von Braunmühl, Verbeek, 1983).
- NIDPDA equivalent to DIDPDA (von Braunmühl, Verbeek, 1983).
- Thorough study (Alur, Madhusudan, STOC 2004).
  - Rediscovered as “visibly pushdown automata” / “nested word automata”
  - Applications to verification.
  - $2^{\Theta(n^2)}$ cost of determinization.
  - Closed under Boolean operations, concatenation, Kleene star.
- Much ongoing research: algorithms, complexity, closure properties...

Closure under edit distance $\ell$-neighbourhood?

For $L$ recognized by an IDPDA, is $E_\ell(L)$ always recognized by one?
Centered around ordinary grammars (Chomsky’s “context-free”).
The big picture: IDPDAs among formal grammars

- Centered around ordinary grammars (Chomsky’s “context-free”).
- Pushdown automata: IDPDA, TDPDA (Kutrib et al. 2015), DPDA.
The big picture: IDPDAs among formal grammars

- Centered around ordinary grammars (Chomsky’s “context-free”).
- Pushdown automata: IDPDA, TDPDA (Kutrib et al. 2015), DPDA.
- Other well-known subfamilies.
Centered around ordinary grammars (Chomsky’s “context-free”).

Pushdown automata: IDPDA, TDPDA (Kutrib et al. 2015), DPDA.

Other well-known subfamilies.

Extensions with Boolean operations (Okhotin, 2001, 2004).
Centered around ordinary grammars (Chomsky’s “context-free”).

Pushdown automata: IDPDA, TDPDA (Kutrib et al. 2015), DPDA.

Other well-known subfamilies.

Extensions with Boolean operations (Okhotin, 2001, 2004).

Complexity.
Edit distance neighbourhood for IDPDA

- Three symbol types: $c \in \Sigma_0$, $< \in \Sigma_{+1}$, $> \in \Sigma_{-1}$. 
Three symbol types: $c \in \Sigma_0$, $\prec \in \Sigma_{+1}$, $\succ \in \Sigma_{-1}$.

insert$_c$, delete$_c$, insert$_\prec$, delete$_\prec$, insert$_\succ$, delete$_\succ$. 
Edit distance neighbourhood for IDPDA

- Three symbol types: $c \in \Sigma_0$, $< \in \Sigma_{+1}$, $> \in \Sigma_{-1}$.
- $\text{insert}_c$, $\text{delete}_c$, $\text{insert}<$, $\text{delete}<$, $\text{insert}>$, $\text{delete}>$.
- $E_{\ell+1}(L)$: union over operations on individual symbols.
Edit distance neighbourhood for IDPDA

- Three symbol types: $c \in \Sigma_0 \quad < \in \Sigma_+ \quad > \in \Sigma_-.$
- $\text{insert}_c,$ $\text{delete}_c,$ $\text{insert}_<,$ $\text{delete}_<,$ $\text{insert}_>,$ $\text{delete}_>.$
- $E_{\ell+1}(L):$ union over operations on individual symbols.
- ✓ Six individual constructions for IDPDA.
Edit distance neighbourhood for IDPDA

- Three symbol types: $c \in \Sigma_0$, $< \in \Sigma_{+1}$, $> \in \Sigma_{-1}$.
- $\text{insert}_c$, $\text{delete}_c$, $\text{insert}_<$, $\text{delete}_<$, $\text{insert}_>$, $\text{delete}_>$.
- $E_{\ell+1}(L)$: union over operations on individual symbols.
- ✓ Six individual constructions for IDPDA.
- ✓ Cases to be presented: $\text{insert}_c$, $\text{insert}_<$, $\text{delete}_<$. 

A. Okhotin, K. Salomaa

Edit distance on input-driven PDAs

CSR 2017 (Kazan)
Inserting a neutral symbol

- Old NIDPDA reads $uv$, new NIDPDA reads $ucv$. 

The same matching of brackets. New NIDPDA guesses $c$ and ignores it.

Remember whether $c$ was encountered before: $\tilde{q}$ vs. $q$.

Theorem

Let $L$ be recognized by NIDPDA with states $Q$, stack symbols $\Gamma$.

$\text{insert } c \quad (L)$ recognized by NIDPDA with states $Q \cup \tilde{Q}$, stack $\Gamma$. 

A. Okhotin, K. Salomaa

Edit distance on input-driven PDAs

CSR 2017 (Kazan)
Inserting a neutral symbol

- Old NIDPDA reads $uv$, new NIDPDA reads $ucv$.

- The same matching of brackets.
Inserting a neutral symbol

- Old NIDPDA reads $uv$, new NIDPDA reads $ucv$.
- The same matching of brackets.
Inserting a neutral symbol

- Old NIDPDA reads $uv$, new NIDPDA reads $ucv$.

- The same matching of brackets.
- New NIDPDA guesses $c$ and ignores it.
Inserting a neutral symbol

- Old NIDPDA reads $uv$, new NIDPDA reads $ucv$.

- The same matching of brackets.
- New NIDPDA guesses $c$ and ignores it.
- Remember whether $c$ was encountered before: $\tilde{q}$ vs. $q$. 
Inserting a neutral symbol

- Old NIDPDA reads $uv$, new NIDPDA reads $ucv$.

- The same matching of brackets.
- New NIDPDA guesses $c$ and ignores it.
- Remember whether $c$ was encountered before: $\tilde{q}$ vs. $q$. 

\[ A. \text{ Okhotin}, \ K. \text{ Salomaa} \]
Inserting a neutral symbol

- Old NIDPDA reads $uv$, new NIDPDA reads $ucv$.

- The same matching of brackets.
- New NIDPDA guesses $c$ and ignores it.
- Remember whether $c$ was encountered before: $\tilde{q}$ vs. $q$. 
Inserting a neutral symbol

- Old NIDPDA reads $uv$, new NIDPDA reads $ucv$.

- The same matching of brackets.
- New NIDPDA guesses $c$ and ignores it.
- Remember whether $c$ was encountered before: $\tilde{q}$ vs. $q$. 
Inserting a neutral symbol

- Old NIDPDA reads $uv$, new NIDPDA reads $ucv$.

- The same matching of brackets.
- New NIDPDA guesses $c$ and ignores it.
- Remember whether $c$ was encountered before: $\tilde{q}$ vs. $q$. 

\[ < \quad q \quad q_1 \quad r \]
\[ p \]
\[ < \quad > \quad s \]

\[ \tilde{p} \quad \tilde{q} \quad q \quad q_1 \quad r \]
\[ c \quad < \quad > \quad s \]
Inserting a neutral symbol

- Old NIDPDA reads $uv$, new NIDPDA reads $ucv$.

The same matching of brackets.

- New NIDPDA guesses $c$ and ignores it.
- Remember whether $c$ was encountered before: $\tilde{q}$ vs. $q$.

Theorem

- Let $L$ be recognized by NIDPDA with states $Q$, stack symbols $\Gamma$. 

Inserting a neutral symbol

- Old NIDPDA reads $uv$, new NIDPDA reads $ucv$.

- The same matching of brackets.
- New NIDPDA guesses $c$ and ignores it.
- Remember whether $c$ was encountered before: $\tilde{q}$ vs. $q$.

**Theorem**

- Let $L$ be recognized by NIDPDA with states $Q$, stack symbols $\Gamma$.
- $\text{insert}_c(L)$ recognized by NIDPDA with states $Q \cup \tilde{Q}$, stack $\Gamma$. 
Inserting a left bracket

- Old NIDPDA reads $uv$, new NIDPDA reads $u \ll v$.
Inserting a left bracket

- Old NIDPDA reads $uv$, new NIDPDA reads $u \ll v$.

- Matching between brackets is shifted!
Inserting a left bracket

- Old NIDPDA reads $uv$, new NIDPDA reads $u \ll v$.

- Matching between brackets is shifted!
- Begin in tilde-states;

$$\begin{align*}
q & \rightarrow p \\
\tilde{q} & \rightarrow r
\end{align*}$$
Inserting a left bracket

- Old NIDPDA reads $uv$, new NIDPDA reads $u \ll v$.

- Matching between brackets is shifted!
- Begin in tilde-states; guess $\ll$, push $\square$, move to no-tilde.

\[
\begin{align*}
q &\rightarrow p \\
\sim q &\rightarrow q
\end{align*}
\]
Inserting a left bracket

- Old NIDPDA reads $uv$, new NIDPDA reads $u\ll v$.

- Matching between brackets is shifted!
- Begin in tilde-states; guess $\ll$, push $\square$, move to no-tilde.
- Guess $s$ upon popping $\square$. 
Inserting a left bracket

- Old NIDPDA reads $uv$, new NIDPDA reads $u \ll v$.

- Matching between brackets is shifted!
- Begin in tilde-states; guess $\ll$, push $\square$, move to no-tilde.
- Guess $s$ upon popping $\square$.
- Verify the guess when popping $s$. 

\[ q \rightarrow p \rightarrow r \]

\[ \tilde{q} \rightarrow (r,s) \]
Inserting a left bracket

- Old NIDPDA reads $uv$, new NIDPDA reads $u \ll v$.

Matching between brackets is shifted!
- Begin in tilde-states; guess $\ll$, push $\Box$, move to no-tilde.
- Guess $s$ upon popping $\Box$.
- Verify the guess when popping $s$.

Theorem

- Let $L$ be recognized by NIDPDA with states $Q$, stack symbols $\Gamma$. 
Inserting a left bracket

- Old NIDPDA reads $uv$, new NIDPDA reads $u\ll v$.

![Diagram showing the transition from old to new NIDPDA reading]

- Matching between brackets is shifted!
- Begin in tilde-states; guess $\ll$, push $\Box$, move to no-tilde.
- Guess $s$ upon popping $\Box$.
- Verify the guess when popping $s$.

**Theorem**

- Let $L$ be recognized by NIDPDA with states $Q$, stack symbols $\Gamma$.
- $\text{insert}_{\ll}(L)$ recognized by NIDPDA with states $Q \cup \tilde{Q} \cup (Q \times \Gamma)$, stack symbols $\Gamma \cup \{\Box\} \cup (\Gamma \times \Gamma)$. 
Deleting a left bracket

- Old NIDPDA reads $u \ll v$, new NIDPDA reads $uv$. 
Deleting a left bracket

- Old NIDPDA reads $u\ll v$, new NIDPDA reads $uv$.

- Matching between brackets is shifted again.
Deleting a left bracket

- Old NIDPDA reads $u \ll v$, new NIDPDA reads $uv$.

- Matching between brackets is shifted again.
Deleting a left bracket

- Old NIDPDA reads $u \ll v$, new NIDPDA reads $uv$.

- Matching between brackets is shifted again.
- Guess where $\ll$ was, enter state $(q, s)$.
Deleting a left bracket

- Old NIDPDA reads \( u \ll v \), new NIDPDA reads \( uv \).

- Matching between brackets is shifted again.
- Guess where \( \ll \) was, enter state \( (q, s) \).

Theorem
Let \( L \) be recognized by NIDPDA with states \( Q \), stack symbols \( \Gamma \).

\[ \text{delete } \ll (L) \text{ recognized by NIDPDA with states } Q \cup \tilde{Q} \cup (Q \times \Gamma), \text{ stack symbols } \Gamma \cup (\Gamma \times \Gamma). \]
Deleting a left bracket

- Old NIDPDA reads $u \ll v$, new NIDPDA reads $uv$.

- Matching between brackets is shifted again.
- Guess where $\ll$ was, enter state $(q, s)$.

A. Okhotin, K. Salomaa
Edit distance on input-driven PDAs
CSR 2017 (Kazan) 11 / 15
Deleting a left bracket

- Old NIDPDA reads $u \ll v$, new NIDPDA reads $uv$.

- Matching between brackets is shifted again.
- Guess where $\ll$ was, enter state $(q, s)$.
- When popping $t$, act as if $s$ is popped, store $t$. 

\[ (q, s) \rightarrow (r_1, s) \]
\[ \tilde{p} \rightarrow (r_1, s) \]
Deleting a left bracket

- Old NIDPDA reads $u \ll v$, new NIDPDA reads $uv$.

Matching between brackets is shifted again.
- Guess where $\ll$ was, enter state $(q, s)$.
- When popping $t$, act as if $s$ is popped, store $t$. 
Deleting a left bracket

- Old NIDPDA reads $u \ll v$, new NIDPDA reads $uv$.

![Diagram showing the process of deleting a left bracket]

- Matching between brackets is shifted again.
- Guess where $\ll$ was, enter state $(q, s)$.
- When popping $t$, act as if $s$ is popped, store $t$.

Theorem

- Let $L$ be recognized by NIDPDA with states $Q$, stack symbols $\Gamma$. 
Deleting a left bracket

- Old NIDPDA reads $u \ll v$, new NIDPDA reads $uv$.

- Matching between brackets is shifted again.
- Guess where $\ll$ was, enter state $(q, s)$.
- When popping $t$, act as if $s$ is popped, store $t$.

### Theorem

- Let $L$ be recognized by NIDPDA with states $Q$, stack symbols $\Gamma$.
- delete$_{\ll}(L)$ recognized by NIDPDA with states $Q \cup \tilde{Q} \cup (Q \times \Gamma)$, stack symbols $\Gamma \cup (\Gamma \times \Gamma)$. 

A. Okhotin, K. Salomaa
Edit distance on input-driven PDAs
CSR 2017 (Kazan) 11 / 15
Lower bound for edit distance neighbourhood

Goal: $n$-state NIDPDA for $L_n$, large NIDPDA required for $E_1(L_n)$. 

Lemma $L_n$ is recognized by a DIDPDA with $O(n)$ states and $n$ stack symbols.

Guess $i$, push $i$, remember $i$ in the state.

Verify $a_i$, read and remember $b_j$.

Verify $b_i$. On $>$, pop $i$ and verify $a_i$.

Lemma Any NIDPDA for delete $<$ ($L_n$) needs at least $n^2$ states.

Has to deal with $c_i + k a_i b_j > b_j a_i$, cannot use the stack.
Lower bound for edit distance neighbourhood

Goal: $n$-state NIDPDA for $L_n$, large NIDPDA required for $E_1(L_n)$.

- $\Sigma_+ = \{<\}$, $\Sigma_- = \{>\}$ $\Sigma_0 = \{a, b, c, $\}$. 

Lemma $L_n$ is recognized by a DIDPDA with $O(n)$ states and $n$ stack symbols.

Guess $i$, push $i$, remember $i$ in the state.
Verify $a_i$, read and remember $b_j$.
Verify $b_i$. On $>$, pop $i$ and verify $a_i$.

Lemma Any NIDPDA for $\text{delete} < (L_n)$ needs at least $n^2$ states.

Has to deal with $c_i + k a_i b_j > b_j > a_i$, cannot use the stack.
Lower bound for edit distance neighbourhood

Goal: $n$-state NIDPDA for $L_n$, large NIDPDA required for $E_1(L_n)$.

- $\Sigma_{+1} = \{<\}$, $\Sigma_{-1} = \{>\}$ $\Sigma_0 = \{a, b, c, \}$.
- $L_n = \{c^i < c^k a^i b^j > b^i > a^i \mid 1 \leq i, j \leq n, \ k \geq 0\}$
Lower bound for edit distance neighbourhood

Goal: $n$-state NIDPDA for $L_n$, large NIDPDA required for $E_1(L_n)$.

- $\Sigma_{+1} = \{<\}$, $\Sigma_{-1} = \{>\}$, $\Sigma_0 = \{a, b, c, $\}$.
- $L_n = \{c^i < c^k a^i b^j $b^j > a^i | 1 \leq i, j \leq n, k \geq 0\}$

Lemma

$L_n$ is recognized by a DIDPDA with $O(n)$ states and $n$ stack symbols.
Lower bound for edit distance neighbourhood

Goal: \( n \)-state NIDPDA for \( L_n \), large NIDPDA required for \( E_1(L_n) \).

- \( \Sigma_{+1} = \{<\} \), \( \Sigma_{-1} = \{>\} \), \( \Sigma_0 = \{a, b, c, $\} \).
- \( L_n = \{c^i < c^k a^i b^j $ b^j > a^i | 1 \leq i, j \leq n, k \geq 0\} \)

Lemma

\( L_n \) is recognized by a DIDPDA with \( O(n) \) states and \( n \) stack symbols.

- Guess \( i \), push \( i \), remember \( i \) in the state.
Lower bound for edit distance neighbourhood

Goal: \( n \)-state NIDPDA for \( L_n \), large NIDPDA required for \( E_1(L_n) \).

- \( \Sigma_{+1} = \{<\} \), \( \Sigma_{-1} = \{>\} \) \( \Sigma_0 = \{a, b, c, $\} \).
- \( L_n = \{c^i< c^k a^i b^j$ b^j > a^i | 1 \leq i, j \leq n, k \geq 0\} \)

Lemma

\( L_n \) is recognized by a DIDPDA with \( O(n) \) states and \( n \) stack symbols.

- Guess \( i \), push \( i \), remember \( i \) in the state.
- Verify \( a^i \), read and remember \( b^i \).
Lower bound for edit distance neighbourhood

**Goal:** $n$-state NIDPDA for $L_n$, large NIDPDA required for $E_1(L_n)$.

- $\Sigma_{+1} = \{<\}$, $\Sigma_{-1} = \{>\}$ $\Sigma_0 = \{a, b, c, $\}$.
- $L_n = \{c^i < c^k a^i b^j > b^j a^i \mid 1 \leq i, j \leq n, k \geq 0\}$

**Lemma**

$L_n$ is recognized by a DIDPDA with $O(n)$ states and $n$ stack symbols.

- Guess $i$, push $i$, remember $i$ in the state.
- Verify $a^i$, read and remember $b^i$.
- Verify $b^i$. On $>$, pop $i$ and verify $a^i$. 
Lower bound for edit distance neighbourhood

Goal: \( n \)-state NIDPDA for \( L_n \), large NIDPDA required for \( E_1(L_n) \).

- \( \Sigma_{+1} = \{<\}, \Sigma_{-1} = \{>\} \) \( \Sigma_0 = \{a, b, c, \$\} \).
- \( L_n = \{c^i < c^k a^i b^j \$ b^j > a^i | 1 \leq i, j \leq n, k \geq 0\} \)

Lemma

\( L_n \) is recognized by a DIDPDA with \( O(n) \) states and \( n \) stack symbols.

- Guess \( i \), push \( i \), remember \( i \) in the state.
- Verify \( a^i \), read and remember \( b^i \).
- Verify \( b^i \). On \( > \), pop \( i \) and verify \( a^i \).

Lemma

Any NIDPDA for \( \text{delete}_<(L_n) \) needs at least \( n^2 \) states.
Lower bound for edit distance neighbourhood

Goal: $n$-state NIDPDA for $L_n$, large NIDPDA required for $E_1(L_n)$.

- $\Sigma_{+1} = \{<\}$, $\Sigma_{-1} = \{>\}$ $\Sigma_0 = \{a, b, c, \}$.
- $L_n = \{c^i < c^k a^i b^i \$ b^j > a^i \mid 1 \leq i, j \leq n, \ k \geq 0\}$

Lemma

$L_n$ is recognized by a DIDPDA with $O(n)$ states and $n$ stack symbols.

- Guess $i$, push $i$, remember $i$ in the state.
- Verify $a^i$, read and remember $b^i$.
- Verify $b^i$. On $>$, pop $i$ and verify $a^i$.

Lemma

Any NIDPDA for delete$<$($L_n$) needs at least $n^2$ states.

- Has to deal with $c^{i+k} a^i b^j \$ b^j > a^i$, cannot use the stack.
Edit distance neighbourhood for DIDPDA

- The construction essentially uses nondeterminism.
Edit distance neighbourhood for DIDPDA

- The construction essentially uses nondeterminism.
- If DIDPDA is required, can determinize an NIDPDA, cost $2\Theta(n^2)$.
The construction essentially uses nondeterminism.
If DIDPDA is required, can determinize an NIDPDA, cost $2^\Theta(n^2)$.
Total cost: $2^O(n^4)$.
Edit distance neighbourhood for DIDPDA

- The construction essentially uses nondeterminism.
- If DIDPDA is required, can determinize an NIDPDA, cost $2^{\Theta(n^2)}$.
- Total cost: $2^{O(n^4)}$.
- Direct construction of DIDPDA?
Edit distance neighbourhood for DIDPDA

- The construction essentially uses nondeterminism.
- If DIDPDA is required, can determinize an NIDPDA, cost $2^{\Theta(n^2)}$.
- Total cost: $2^{O(n^4)}$.
- Direct construction of DIDPDA?

**Theorem**

- $L$ recognized by DIDPDA with states $Q$, stack symbols $\Gamma$. 

The construction essentially uses nondeterminism.
If DIDPDA is required, can determinize an NIDPDA, cost $2^{\Theta(n^2)}$.
Total cost: $2^{O(n^4)}$.
Direct construction of DIDPDA?

**Theorem**
- $L$ recognized by DIDPDA with states $Q$, stack symbols $\Gamma$.
- $\text{delete} \ll (L)$ recognized by DIDPDA with states $Q \times 2^Q \times (\Gamma \cup \{\bot\}) \times Q^Q$, stack symbols $\Sigma_{+1} \times \Gamma \times 2^Q \times (\Gamma \cup \{\bot\}) \times Q^Q$. 

A. Okhotin, K. Salomaa

Edit distance on input-driven PDAs

CSR 2017 (Kazan)
The construction essentially uses nondeterminism.

If DIDPDA is required, can determinize an NIDPDA, cost $2^{\Theta(n^2)}$.

Total cost: $2^{O(n^4)}$.

Direct construction of DIDPDA?

**Theorem**

$L$ recognized by DIDPDA with states $Q$, stack symbols $\Gamma$.

$\text{delete}(L)$ recognized by DIDPDA with states $Q \times 2^{Q \times (\Gamma \cup \{\bot\})} \times \Gamma \times 2^{Q \times (\Gamma \cup \{\bot\})} \times Q^Q$, stack symbols $\Sigma_{+1} \times \Gamma \times 2^{Q \times (\Gamma \cup \{\bot\})} \times Q^Q$.

Matching $2^{\Omega(n^2)}$ lower bound.
Conclusion

Which families are closed under edit distance neighbourhood?
Conclusion

Which families are closed under edit distance neighbourhood?

- Regular languages.
Conclusion

Which families are closed under edit distance neighbourhood?

- Regular languages.
Conclusion

Which families are closed under edit distance neighbourhood?

- Regular languages.
- IDPDA.
Conclusion

Which families are closed under edit distance neighbourhood?

- Regular languages.
- IDPDA.