I Can Only Read Equations

Declarative Dynamic Programming with Inverse Coupled Rewrite Systems

Robert Giegerich

Universität Bielefeld
AG Praktische Informatik

Oxford, July 2018
Dynamic Programming is widely used in computer science especially in bioinformatics:

- DNA sequence comparison
- Protein sequence comparison
- RNA sequence comparison
- RNA structure prediction
- RNA structure comparison
- etc

**DP solves exponential search spaces in polynomial time...**

... by use of tabulated recursion
Dynamic Programming is widely used in computer science especially in bioinformatics:

- DNA sequence comparison
- Protein sequence comparison
- RNA sequence comparison
- RNA structure prediction
- RNA structure comparison
- etc

DP solves exponential search spaces in polynomial time...

... by use of tabulated recursion

DP: creative – intricate – tedious – horrifying
String edit distance:

Using Levenshtein's original operations, the edit distance between $a = a_1 \ldots a_n$ and $b = b_1 \ldots b_m$ is given by $d_{mn}$, defined by the recurrence\cite{2}

\[
d_{i0} = \sum_{k=1}^{i} w_{\text{del}}(b_k), \quad \text{for } 1 \leq i \leq m \\
d_{0j} = \sum_{k=1}^{j} w_{\text{ins}}(a_k), \quad \text{for } 1 \leq j \leq n \\
d_{ij} = \begin{cases} d_{i-1,j-1} & \text{for } a_j = b_i \\ \min \left\{ d_{i-1,j} + w_{\text{del}}(b_i), \\
 d_{i,j-1} + w_{\text{ins}}(a_j), \\
 d_{i-1,j-1} + w_{\text{sub}}(a_j, b_i) \right\} & \text{for } a_j \neq b_i \end{cases} \quad \text{for } 1 \leq i \leq m, 1 \leq j \leq n.
\]
Comparative structure prediction: Sankoff’s recurrence Part 1

Let $C(i_1, j_1; i_2, j_2)$ be the minimum cost given that $(i_1, j_1) \in S_1$ and $(i_2, j_2) \in S_2$ without considering the costs of aligning $a^{(1)}_{i_1}$, $a^{(1)}_{j_1}$, $a^{(2)}_{i_2}$ and $a^{(2)}_{j_2}$. If no such pair of structures exists, set $C = \infty$. Then

(13) \hspace{1em} C(i_1, j_1; i_2, j_2) \begin{cases} e(s_1) + e(s_2) + D(i_1 + 1, j_1 - 1; i_2 + 1, j_2 - 1), & s_1, s_2 \text{ hairpins closed by } (i_1, j_1), (i_2, j_2) \text{ respectively}, \\ \min \{e(s_1) + e(s_2) + C(p_1, q_1; p_2, q_2) + D(i_1 + 1, p_1; i_2 + 1, p_2) + D(q_1, j_1 - 1; q_2, j_2 - 1)\}, & s_1, s_2 \text{ are 2-loops closed by } (i_1, j_1), (i_2, j_2) \text{ with } (p_1, q_1), (p_2, q_2) \text{ accessible,} \\ \min_{\begin{array}{c} i_1 < h_1 < j_1 - 1 \\ i_2 < h_2 < j_2 - 1 \end{array}} \{G(i_1 + 1, h_1; i_2 + 1, h_2) + G(h_1 + 1, j_1 - 1; h_2 + 1, j_2 - 1) + 2A\}, & \text{or one of } \begin{cases} s_1 = \phi & \text{and } (p_1, q_1) = (i_1, j_1) \\ s_2 = \phi & \text{and } (p_2, q_2) = (i_2, j_2), \end{cases} \end{cases}
Comparative structure prediction: Sankoff’s recurrence Part 2

where

\[
G(i_1, j_1; i_2, j_2) = \min \left\{ \min_{i_1 < h_1 < j_1} \min_{i_2 < h_2 < j_2} \begin{align*}
C(i_1, j_1; i_2, j_2) + 2P + D(i_1, i_1; i_2, i_2) + D(j_1, j_1; j_2, j_2), \\
G(i_1, h_1; i_2, h_2) + (j_1 - h_1 + j_2 - h_2)Q \\
+ D(h_1 + 1, j_1; h_2 + 1, j_2), \\
G(i_1, h_1; i_2, h_2) + G(h_1 + 1, j_1; h_2 + 1, j_2), \\
(h_1 - i_1 + 1 + h_2 - i_2 + 1)Q \\
+ G(h_1 + 1, j_1; h_2 + 1, j_2) + D(i_1, h_1; i_2, h_2) \end{align*} \right\}
\]

and

\[
F(i_1, j_1; i_2, j_2) = \min \left\{ \min_{i_1 \leq h_1 < j_1} \min_{i_2 \leq h_2 < j_2} \begin{align*}
C(i_1, j_1; i_2, j_2) + D(i_1, i_1; i_2, i_2) + D(j_1, j_1; j_2, j_2), \\
F(i_1, h_1; i_2, h_2) + F(h_1 + 1, j_1; h_2 + 1, j_2) \end{align*} \right\}
\]

with initial conditions \( C(i_1, i_1; i_2, i_2) = \infty \) and \( G(i_1, i_1; i_2, j_2) = G(i_1, j_1; i_2, i_2) = \infty \).
ICORE: A declarative approach to Dynamic Programming

I Can Only Read (and write) Equations

ICORE

an

only

ead (and write)
quations

ICORE

Robert Giegerich

Introduction
String edit distance
Framework
ICORE language
RNA Structure
Tree comparison
Implementation
Acknowledgements
Overflow Theory
ICORE: A declarative approach to Dynamic Programming

ICOREs are joint work with Hélène Touzet of Lille University. Paper in *Algorithms* 2014, 7(1), 62-144. mdpi.com
Example 1: String edit distance under unit cost

Edit distance of "DARLING" versus "AIRLINE"

- under edit model MATch, REPlace, DELeTe, INSert
- optimal answer (out of several) is
  D A - R L I N G
  - A I R L I N E
- which has (minimal) score
  1 x DEL, 1 x INS, 1 x REP, 5 x MAT
  1 + 1 + 1 + 5 * 0 = 3

Well, the answer is 3. What exactly was the question?
Example 1: String edit distance under unit cost

Edit distance of "DARLING" versus "AIRLINE"

- under edit model MATch, REPlace, DELete, INSert
- optimal answer (out of several) is
  
  \[ \text{D A R L I N G} \]
  
  \[- \text{A I R L I N E} \]

- which has (minimal) score
  
  \[ 1 \times \text{DEL}, 1 \times \text{INS}, 1 \times \text{REP}, 5 \times \text{MAT} \]
  
  \[ 1 + 1 + 1 + 5 \times 0 = 3 \]

- Well, the answer is 3. What exactly was the question?
Candidate solutions in DP have no data type representing them:
- optimization iterates over *candidate scores*
- candidates in the search space are never constructed explicitly
- some external representation may be constructed on backtrace ...
Invisible data types

Candidate solutions in DP have no data type representing them:

- optimization iterates over candidate scores
- candidates in the search space are never constructed explicitly
- some external representation may be constructed on backtrace ...

CLAIM: There is an invisible candidate data type in every DP algorithm!
Peeking under the hood of dynamic programming
Computer’s view of BPO
Invisible data types forever!

The invisible tree representing the alignment

```
D A - R L I N G
- A I R L I N E
```

- holds 5 character matches, 1 deletion, 1 insertion, and 1 replacement
- evaluates to edit distance 3 under unit cost model
- constitutes an optimal alignment, having minimal cost.

- rewrites to inputs "DARLING" and "AIRLINE"
ICORE for String Edit Distance

Rules:

\[ a \cdot X \leftarrow \text{rep}(a, b, X) \rightarrow b \cdot X \]

\[ a \cdot X \leftarrow \text{mat}(a, b, X) \rightarrow b \cdot X \]

\[ a \cdot X \leftarrow \text{del}(a, X) \rightarrow X \]

\[ X \leftarrow \text{ins}(a, X) \rightarrow a \cdot X \]

\[ \varepsilon \leftarrow \text{mty} \rightarrow \varepsilon \]
ICORE for String Edit Distance

Rules:

\[ a \cdot X \leftarrow \text{rep}(a, b, X) \rightarrow b \cdot X \]
\[ a \cdot X \leftarrow \text{mat}(a, b, X) \rightarrow b \cdot X \]
\[ a \cdot X \leftarrow \text{del}(a, X) \rightarrow X \]
\[ X \leftarrow \text{ins}(a, X) \rightarrow a \cdot X \]
\[ \varepsilon \leftarrow \text{mty} \rightarrow \varepsilon \]

The candidate “tree”

\[ c = \text{del}(D, \text{mat}(A, A, \text{ins}(I, \text{mat}(R, R, \text{mat}(L, L, ...))))) \]

rewrites to

\[ \text{DARL} \ldots \leftarrow c \rightarrow \text{AIRL} \ldots \]

and evaluates to

\[ \text{UNITD}(c) = 3 \]
Evaluation algebra $\text{UNITD}$ for unit edit distance

Algebra $\text{UNITD}$

\[
\begin{align*}
\text{rep}(a, b, x) &= x + 1 \\
\text{mat}(a, a, x) &= x \\
\text{del}(a, x) &= x + 1 \\
\text{ins}(a, x) &= x + 1 \\
\text{mty} &= 0 \\
\phi &= \min
\end{align*}
\]

$\text{UNITD}(\text{del}(D, \text{mat}(A, A, \text{ins}(I, \text{mat}(R, R, \text{mat}(L, L, \ldots \text{mty} \ldots))))))) = 3$
The ICORE framework in a nutshell

An Inverse COupled REwrite system has
- input data type(s) (sequences or trees)
- a solution candidate data type
  (= signature of scoring functions)
- evaluation algebra (one or several) satisfying BPO
- rules to rewrite a candidate to its inputs (coupled)
- and one more thing …
ICOREs consist of

- **input signatures** $S_1, S_2, \ldots$, term algebras $T_{S_1}, T_{S_2}, \ldots$
- **core signature** $\zeta$: with term algebra $T_\zeta$ – the “invisible trees”
- **term rewrite system** $T_{S_1} \leftarrow T_\zeta \rightarrow T_{S_2}$
- **evaluation algebra** $A$ including **objective function** $\phi$
- **regular tree grammar** $G$ with $L(G) \subset T_\zeta$ (optional)
The ICORE framework

ICOREs consist of

- input signatures $S_1, S_2, ...$, term algebras $T_{S_1}, T_{S_2}, ...$
- core signature $\zeta$: with term algebra $T_{\zeta}$ – the “invisible trees”
- term rewrite system $T_{S_1} \leftarrow T_{\zeta} \rightarrow T_{S_2}$
- evaluation algebra $A$ including objective function $\phi$
- regular tree grammar $G$ with $L(G) \subset T_{\zeta}$ (optional)

An ICORE problem instance is given by

- inputs $x_1, x_2, ...$

and is solved by computing

$$\phi([A(c) \mid c \in L(G), x_1 \leftarrow_1 c \Rightarrow_2 x_2])$$

efficiently, somehow ...
Three phase view

\[ \phi([A(c) \mid c \in L(G), x_1 \leftarrow_1 c \rightarrow_2 x_2]) \]

1. Construct the search space from the inputs \( x_1, x_2, \ldots \)

\[ C = \{ c \in L(G), x_1 \leftarrow_1 c \rightarrow_2 x_2 \} \]
Three phase view

\[ \phi([A(c) \mid c \in L(G), x_1 \leftrightarrow_1 c \Rightarrow_2 x_2]) \]

1. Construct the search space from the inputs \( x_1, x_2, \ldots \)
   \[ C = \{ c \in L(G), x_1 \leftrightarrow_1 c \Rightarrow_2 x_2 \} \]

2. Evaluate all candidates: \( V = A(C) \)
Three phase view

\[ \phi([A(c) \mid c \in L(G), x_1 \leftarrow_1 c \rightarrow_2 x_2]) \]

1. Construct the search space from the inputs \( x_1, x_2, \ldots \):
   \[ C = \{ c \in L(G), x_1 \leftarrow_1 c \rightarrow_2 x_2 \} \]

2. Evaluate all candidates: \( \mathcal{V} = A(C) \)

3. Choose from all candidate scores:
   \[ \phi(\mathcal{V}) \]
Three phase view

\[ \phi([A(c) \mid c \in L(G), x_1 \xleftarrow{1} c \xrightarrow{2} x_2]) \]

1. Construct the search space from the inputs \( x_1, x_2, \ldots \)

\[ C = \{ c \in L(G), x_1 \xleftarrow{1} c \xrightarrow{2} x_2 \} \]

2. Evaluate all candidates: \( V = A(C) \)

3. Choose from all candidate scores:

\[ \phi(V) \]

\( \phi \) must satisfy **Bellman’s Principle of Optimality** with respect to \( A \). In the implementation, phases must be interleaved.
Find all core terms that rewrite to the given inputs, evaluate them and make your choice!

⇒

ICORE implementation is constructing the inverse of the coupled rewrite relation
Satellite signature $SEQ = \mathcal{A} \cup \{\cdot, \varepsilon\}$ with
- $a : \mathcal{A} \rightarrow \mathcal{A}^*$ -- for $a \in \mathcal{A}$; single letter sequence
- $\cdot : \mathcal{A}^* \times \mathcal{A}^* \rightarrow \mathcal{A}^*$ -- sequence concatenation
- $\varepsilon : \rightarrow \mathcal{A}^*$ -- the empty sequence

Sequences over an alphabet $\mathcal{A}$
ICORE EditDistance: Core

Core signature $ALI$:

- $\text{rep} : A \times A \times ALI \rightarrow ALI$ -- alignment starting with replacement
- $\text{mat} : A \times A \times ALI \rightarrow ALI$ -- alignment starting with match
- $\text{del} : A \times ALI \rightarrow ALI$ -- alignment starting with deletion
- $\text{ins} : A \times ALI \rightarrow ALI$ -- alignment starting with insertion
- $\text{mty} : \rightarrow ALI$ -- empty alignment

Note: $ALI$ is an abstract type name (sort)

$T_{ALI}$: term algebra – the invisible tree domain
ICORE EditDistance: Core

Core signature $ALI$:

\[ \text{rep} : A \times A \times A \rightarrow ALI \quad \text{-- alignment starting with replacement} \]
\[ \text{mat} : A \times A \times ALI \rightarrow ALI \quad \text{-- alignment starting with match} \]
\[ \text{del} : A \times ALI \rightarrow ALI \quad \text{-- alignment starting with deletion} \]
\[ \text{ins} : A \times ALI \rightarrow ALI \quad \text{-- alignment starting with insertion} \]
\[ \text{mty} : \rightarrow ALI \quad \text{-- empty alignment} \]

Note: $ALI$ is an abstract type name (sort)

$T_{ALI}$: term algebra – the invisible tree domain

$T_{ALI}$ represents all possible alignments of any two strings (under the basic edit model)
ICORE EditDistance: Rewrite rules

\[
\begin{align*}
    a \cdot X & \leftarrow \text{rep}(a, b, X) \rightarrow b \cdot X \\
    a \cdot X & \leftarrow \text{mat}(a, a, X) \rightarrow b \cdot X \\
    a \cdot X & \leftarrow \text{del}(a, X) \rightarrow X \\
    X & \leftarrow \text{ins}(a, X) \rightarrow a \cdot X \\
    \varepsilon & \leftarrow \text{mty} \rightarrow \varepsilon
\end{align*}
\]
ICORE EditDistance: Rewrite rules

\[
\begin{align*}
a \cdot X & \leftarrow \text{rep}(a, b, X) \rightarrow b \cdot X \\
a \cdot X & \leftarrow \text{mat}(a, a, X) \rightarrow b \cdot X \\
a \cdot X & \leftarrow \text{del}(a, X) \rightarrow X \\
X & \leftarrow \text{ins}(a, X) \rightarrow a \cdot X \\
\varepsilon & \leftarrow \text{mty} \rightarrow \varepsilon
\end{align*}
\]

From the ICORE rules you can more or less deduce the rest:

- core signature definition
- input signatures
- evaluation algebra templates
Example 2: RNA secondary structure

An RNA chain molecule folds back onto itself and creates secondary structure via base pairing.
An ambiguous context-free parsing problem:

- Terminal alphabet: $\mathcal{A} = \{a, c, g, u\}$ (four nucleotides)
- RNA language: $\mathcal{A}^*$
- base pairing grammar:

$$
R \rightarrow a \ R \mid c \ R \mid g \ R \mid u \ R \mid P \ R \mid \varepsilon
$$

$$
P \rightarrow a \ P \ u \mid u \ P \ a \mid c \ P \ g \mid g \ P \ c \mid g \ P \ u \mid u \ P \ g \mid R
$$

- Parsing with an Earley or CYK-type parser
RNA structure prediction

Structure prediction computes a structure of minimal free energy

- thermodynamic model with more than 1000 parameters
- search space grows with $O(a^n b n^{-3/2})$ (with $a \approx 1.138$)
- solved via dynamic programming in $O(n^3)$
- convenient dot-bracket representation

```plaintext
accuguagcugaggcucgcaaagcg
.(((.((.)).)).(((.))).)
```

or

```plaintext
.accuguagcugaggcucgcaaagcg
..<<.<<<<<>>.>>..<<<...>>>
```

alternatively

```plaintext
accuguagcugaggcucgcaaagcg
.((.))...(.)...(.)...(.)
```
Input signatures

Satellite signature \textit{RNA}  
\begin{itemize}
  \item alphabet \{A, C, G, U\}
  \item operators \{\cdot, \varepsilon\}
\end{itemize}

\begin{itemize}
\item Satellite signature \textit{DOT}  
  \begin{itemize}
    \item alphabet \{\bullet, <, >\}
    \item operators \{\cdot, \varepsilon\}
  \end{itemize}
\end{itemize}

\begin{itemize}
\item Satellite signature \textit{SAS}  
  \begin{itemize}
    \item alphabet \{\bullet_A, \bullet_C, \bullet_G, \bullet_U, <_A, <_C, <_G, <_U, >_A, >_C, >_G, >_U\}
    \item operators \{\cdot, \varepsilon\}
  \end{itemize}
\end{itemize}
Candidate tree representing a secondary structure

A A C C U G G G A U G U A U

. . ( ( . ) ) . ( ( ( ) ) )
Core signature

A signature representing RNA structures (as terms/trees)

Core signature $STR$

- $mty : \rightarrow S$ -- the empty structure
- $\text{single} : \mathcal{A} \times S \rightarrow S$ -- adding unpaired bases
- $\text{pair} : \mathcal{A} \times S \times \mathcal{A} \rightarrow S$ -- adding base pairs
- $\text{split} : S \times S \rightarrow S$ -- structure branch
Grammar SecStr  (secures a unique representation)

S → mty  |  single  |  split
     |       |     
a    | S    | a    | S
        |     | S    | b

## Grammar

- **S**: start symbol
- **mty**: empty node
- **single**: single node
- **split**: split node
- **pair**: pair node

### Rules

1. Single rule: $a \cdot X$ (1)
2. Split rule: $X \cdot Y$ (2)
3. Pair rule: $a \cdot X \cdot b$ if $basep(a, b)$ (3)
4. Mty rule: $\epsilon$ (4)

### Algebra

- **BPmax single**: $single(a, x) = x$ (5)
- **BPmax split**: $split(x, y) = x + y$ (6)
- **BPmax pair**: $pair(a, x, b) = x + 1$ (7)
- **Mty**: $mty = 0$ (8)
- **φ**: $\phi = \max$ (9)
Grammar \texttt{SecStr} (secures a unique representation)

\begin{align*}
S & \rightarrow \text{mty} \mid \text{single} \mid \text{split} \\
& \quad \quad a \quad S \\
& \quad \quad \text{pair} \quad S \\
& \quad \quad a \quad S \quad b
\end{align*}

Rules

\begin{align*}
\text{single}(a, X) & \rightarrow a \cdot X \\
\text{split}(X, Y) & \rightarrow X \cdot Y \\
\text{pair}(a, X, b) & \rightarrow a \cdot X \cdot b \text{ if } \text{basep}(a, b) \\
\text{mty} & \rightarrow \varepsilon
\end{align*}
Grammar **SecStr** (secures a unique representation)

$$S ightarrow mty \mid single \mid split$$

- **single**($a, X$) $\rightarrow$ $a \cdot X$ (1)
- **split**($X, Y$) $\rightarrow$ $X \cdot Y$ (2)
- **pair**($a, X, b$) $\rightarrow$ $a \cdot X \cdot b$ if basep($a, b$) (3)
- $mty \rightarrow \varepsilon$ (4)

**Algebra BPmax**

- **single**($a, x$) = $x$ (5)
- **split**($x, y$) = $x + y$ (6)
- **pair**($a, x, b$) = $x + 1$ (7)
- $mty = 0$ (8)
- $\phi = max$ (9)
More algebras

Algebra \texttt{DotPar}

\begin{align*}
\text{single}(a,x) &= \texttt{'}\cdot\texttt{'} \cdot x \\
\text{split}(x,y) &= x \cdot y \\
\text{pair}(a,x,b) &= \texttt{'}(\texttt{'} \cdot x \cdot \texttt{'})' \\
\text{mty} &= \varepsilon \\
\phi &= \texttt{id}
\end{align*}
More algebras

Algebra DotPar

\[
\begin{align*}
single(a, x) &= ' .' \cdot x \\
split(x, y) &= x \cdot y \\
pair(a, x, b) &= '(\ ' \cdot x \cdot ')'
\end{align*}
\]

\[
\begin{align*}
mty &= \varepsilon \\
\phi &= \text{id}
\end{align*}
\]

Algebra SCFG

\[
\begin{align*}
single(a, x) &= \pi_a \ast x \\
split(x, y) &= \pi_{\text{split}} \ast x \ast y \\
pair(a, x, b) &= \pi_{ab} \ast x
\end{align*}
\]

\[
\begin{align*}
mty &= 1 \\
\phi &= \text{max}
\end{align*}
\]
Consider the candidate
\[ t = \text{single}(A, \text{split}((C, \text{single}(U, \text{mty}), G), \text{single}(G, \text{mty}))) \].

We find that
\[ \text{DotPar}(t) = ".(.)." \]
\[ \text{BPmax}(t) = 1, \]
and
\[ t \Rightarrow \text{ACUGG} \]
Again: Find all such \( t \), evaluate and choose ...
Scale-up to related problems

Fold 1 sequence

\[
\text{single}(a, X) \rightarrow a \cdot X \\
\text{split}(X, Y) \rightarrow X \cdot Y \\
\text{pair}(a, X, b) \rightarrow a \cdot X \cdot b \text{ if basep}(a, b) \\
\text{mty} \rightarrow \varepsilon
\]
Scale-up to related problems

Fold 1 sequence

\[ \text{single}(a, X) \rightarrow a \cdot X \]
\[ \text{split}(X, Y) \rightarrow X \cdot Y \]
\[ \text{pair}(a, X, b) \rightarrow a \cdot X \cdot b \text{ if basep}(a, b) \]
\[ \text{mty} \rightarrow \varepsilon \]

Joint folding:

\[ c \cdot X \leftarrow \text{single}(c, a, X) \rightarrow a \cdot X \]
\[ X \cdot Y \leftarrow \text{split}(X, Y) \rightarrow X \cdot Y \]
\[ c \cdot X \cdot d \leftarrow \text{pair}(c, a, X, d, b) \rightarrow a \cdot X \cdot b \text{ if basep}(a, b) \]
\[ & \text{basep}(c, d) \]
\[ \varepsilon \leftarrow \text{mty} \rightarrow \varepsilon \]
Scale-up to related problems

Fold 1 sequence

\[
\text{single}(a, X) \rightarrow a \cdot X \\
\text{split}(X, Y) \rightarrow X \cdot Y \\
\text{pair}(a, X, b) \rightarrow a \cdot X \cdot b \text{ if } \text{basep}(a, b) \\
\text{mty} \rightarrow \varepsilon
\]

Joint folding:

\[
c \cdot X \leftarrow \text{single}(c, a, X) \rightarrow a \cdot X \\
X \cdot Y \leftarrow \text{split}(X, Y) \rightarrow X \cdot Y \\
c \cdot X \cdot d \leftarrow \text{pair}(c, a, X, d, b) \rightarrow a \cdot X \cdot b \text{ if } \text{basep}(a, b) \\
& \text{basep}(c, d)
\]

\[
\varepsilon \leftarrow \text{mty} \rightarrow \varepsilon
\]

Joint alignment and folding: the above plus

\[
a \cdot X \leftarrow \text{ins}(a, X) \rightarrow X \\
X \leftarrow \text{del}(c, X) \rightarrow c \cdot X
\]
The structural alignment tree $t$

\[
\begin{array}{c}
\text{split} \\
\text{pair} & \text{single} \\
G\ A & \text{ins} & U\ U & C\ G & \text{del} \\
U & \text{mty} & C & \text{mty} \\
\end{array}
\]

$G - U C C$  
$( . ) . .$  
$A U U G -$  

rewrites to sequences with ICORE Sankoff rules

\[
GUUC \leftrightarrow t \Rightarrow AUUG
\]
Where is the dynamic programming?
Where is the dynamic programming?

The ICORE gives a mathematical problem specification to be compiled automatically into DP recurrences such as ...
Let $C(i_1, j_1; i_2, j_2)$ be the minimum cost given that $(i_1, j_1) \in S_1$ and $(i_2, j_2) \in S_2$ without considering the costs of aligning $a_{i_1}^{(1)}$, $a_{j_1}^{(1)}$, $a_{i_2}^{(2)}$ and $a_{j_2}^{(2)}$. If no such pair of structures exists, set $C = \infty$. Then

\begin{align*}
(13) & \quad C(i_1, j_1; i_2, j_2) \\
& = \min \begin{cases} 
  e(s_1) + e(s_2) + D(i_1 + 1, j_1 - 1; i_2 + 1, j_2 - 1), & s_1, s_2 \text{ hairpins closed by} \\
  (i_1, j_1), (i_2, j_2) \text{ respectively}, \\
  \min \{ e(s_1) + e(s_2) + C(p_1, q_1; p_2, q_2) \\
  + D(i_1 + 1, p_1; i_2 + 1, p_2) + D(q_1, j_1 - 1; q_2, j_2 - 1) \}, & s_1, s_2 \text{ are 2-loops closed by} \\
  (i_1, j_1), (i_2, j_2) \text{ with} (p_1, q_1), (p_2, q_2) \text{ accessible}, \\
  p_1 - i_1 + j_1 - q_1 - 2 \leq U, p_2 - i_2 + j_2 - q_2 - 2 \leq U, \\
  \text{or one of} \\
  \begin{cases} 
  s_1 = \phi & \text{and} \ (p_1, q_1) = (i_1, j_1), \\
  s_2 = \phi & \text{and} \ (p_2, q_2) = (i_2, j_2), 
  \end{cases} \\
  \min \{ G(i_1 + 1, h_1; i_2 + 1, h_2) \\
  + G(h_1 + 1, j_1 - 1; h_2 + 1, j_2 - 1) + 2A \}, 
\end{cases}
\end{align*}
where

\[ G(i_1, j_1; i_2, j_2) = \min \left\{ \begin{array}{l}
\min_{i_2 < h_2 < j_2} \{ 
C(i_1, j_1; i_2, h_2) + 2P + D(i_1, i_1; i_2, i_2) + D(j_1, j_1; j_2, j_2), \\
G(i_1, h_1; i_2, h_2) + (j_1 - h_1 + j_2 - h_2)Q \\
+ D(h_1 + 1, j_1; h_2 + 1, j_2), \\
G(i_1, h_1; i_2, h_2) + G(h_1 + 1, j_1; h_2 + 1, j_2), \\
(h_1 - i_1 + 1 + h_2 - i_2 + 1)Q \\
+ G(h_1 + 1, j_1; h_2 + 1, j_2) + D(i_1, h_1; i_2, h_2) \\
\} \right\} \\
\]

and

\[ F(i_1, j_1; i_2, j_2) = \min \left\{ \begin{array}{l}
\min_{i_1 < h_1 < j_1, i_2 < h_2 < j_2} \{ 
C(i_1, j_1; i_2, j_2) + D(i_1, i_1; i_2, i_2) + D(j_1, j_1; j_2, j_2), \\
F(i_1, h_1; i_2, h_2) + F(h_1 + 1, j_1; h_2 + 1, j_2), \\
D(i_1, j_1; i_2, j_2) \\
\} \right\} \\
\]

with initial conditions \( C(i_1, i_1; i_2, i_2) = \infty \) and \( G(i_1, i_1; i_2, j_2) = G(i_1, j_1; i_2, i_2) = \infty \).
Building up ICOREs

\[ \text{Sankoff} = \text{RNAfold} \times 2 + \text{EditDistance} \]
Building up ICOREs

\[ \text{Sankoff} = \text{RNAfold} \times 2 + \text{EditDistance} \]

\[ c \cdot X \leftarrow \text{single}(c, a, X) \rightarrow a \cdot X \]
\[ X \cdot Y \leftarrow \text{split}(X, Y) \rightarrow X \cdot Y \]
\[ c \cdot X \cdot d \leftarrow \text{pair}(c, a, X, d, b) \rightarrow a \cdot X \cdot b \quad \text{if } \text{basep}(a, b) \]
\[ \quad \& \text{basep}(c, d) \]
\[ \varepsilon \leftarrow \text{mty} \rightarrow \varepsilon \]
Building up ICOREs

**Sankoff = RNAfold * 2 + EditDistance**

\[
\begin{align*}
c \cdot X & \leftarrow \text{single}(c, a, X) \rightarrow a \cdot X \\
X \cdot Y & \leftarrow \text{split}(X, Y) \rightarrow X \cdot Y \\
c \cdot X \cdot d & \leftarrow \text{pair}(c, a, X, d, b) \rightarrow a \cdot X \cdot b \quad \text{if basep}(a, b) \\
& \quad \text{& basep}(c, d) \\
\varepsilon & \leftarrow \text{mty} \rightarrow \varepsilon \\
\end{align*}
\]

\[
\begin{align*}
c \cdot X & \leftarrow \text{del}(c, X) \rightarrow X \\
X & \leftarrow \text{ins}(a, X) \rightarrow a \cdot X
\end{align*}
\]

Note: Cases rep and mat already covered by the single-rule
A new problem with the SAME candidate space

rewrites to two structure trees with another rewrite system

which describes a structure comparison problem
Example 3: Tree comparison

This section

- raises a critique of the standard model of tree comparison
- suggest to model trees as terms from specific signatures
- introduces a level of specific tree edit operations
- formulates these as bi-directional tree rewrite rules
- shows how to turn such a model into an ICORE
Critique of standard model

The standard model of tree comparison used in algorithms for

- tree edit distance, a.k.a. largest common subtree
- tree alignment, a.k.a. smallest common supertree

... is too abstract
Classical methods of tree comparison

Classical model:
- rooted ordered labeled trees
- atomic edit operations: *relabel*, *delete*, *insert*

```
a(X) ↔ b(X)  -- node re-label
a(X) ↔ X    -- node indel
```

- Classical model:
  - rooted ordered labeled trees
  - atomic edit operations: *relabel*, *delete*, *insert*
This one uses associative rewriting wrto.
Rewriting ...

\[
\text{rep}(a, a, \text{del}(e, \text{rep}(b, b, \text{mt})), \text{ins}(f, \text{rep}(c, c, \text{mt})), \text{rep}(d, d, \text{mty}))) \\
\rightarrow
\text{a(del}(e, b\text{~}\text{ins}(f, c\text{~}d))\text{))}
\]

\[
\text{a(del}(e, b\text{~}c\text{~}d)) \\
\rightarrow
\text{a}(e(b\text{~}c))\text{~}d
\]

\[
\text{X} \cdot \text{Y} \leftarrow \text{ins}(a, \text{X} \cdot \text{Y}) \rightarrow \text{a(X)} \cdot \text{Y}
\]

\[
\text{a(X)} \cdot \text{Y} \leftarrow \text{del}(a, \text{X} \cdot \text{Y}) \rightarrow \text{X} \cdot \text{Y}
\]
This combines with four different grammars describing different gap models ...
Rewriting ...

Let $t =$

\[
    \text{rep}(a, a, (\text{del}(e, \text{rep}(b, b, \text{mt}) \sim \text{del}(c, \text{mt})) \sim \text{ins}(f, (\text{ins}(c, \text{mt}) \sim \text{rep}(d, d, \text{mt})))))
\]

We rewrite

\[
    a(e(b, c), d) = a((e((b \sim c) \sim d))) \xrightarrow{*} t \rightarrow^* a(b \sim f(c \sim d)) = a(b, f(c, d))
\]

\[
    \text{TreeSimilarity}(t) = 2\delta + 2\gamma + w(a, a) + w(b, b) + w(d, d)
\]
Critique of “atomic” edit model

The general model allows for

- relabeling anything into anything
- deleting/inserting anything in any context

This does not allow to forbid or to enforce combinations of edits.
Tree alignment in the real world

I Can Only Read Equations
Robert Giegerich
Introduction
String edit distance
Framework
ICORE language
RNA Structure
Tree comparison
Implementation
Acknowledgement
Overflow Theory
Tree alignment in the real world

Generalize: domain specific tree edit models where

- trees are terms from some signature: richer semantics!
- edit operations are rewrite rules: non-atomic edits!
Search space refinement and clean-up

Positive

Negative

b_del b_match b_match b_match b_del
A C C C C C C U

b_match
A A

bp_ins
A U

bp_match
A U

bp_match
A A

bp_match
U U

(A,P)

(P,P)

(A,-) (U,-)

(P,P)

(A,-) (U,-)

(−,U) (U,A)

(−,A) (U,-)

(−,A) (U,−)
Non-atomic tree edits

Generalized tree edit operations are modeled as bidirectional
term rewrite rules such as

\[ \text{shuffle} : \quad f(g(x, y), h(z)) \leftrightarrow g(h(y), g(z, h(x))) \]

(which may also rewrite well-formed terms between different
signatures)
Non-atomic tree edits

Generalized tree edit operations are modeled as bidirectional term rewrite rules such as

$$\text{shuffle} : \quad f(g(x, y), h(z)) \leftrightarrow g(h(y), g(z, h(x)))$$

(which may also rewrite well-formed terms between different signatures)

ICORE formulation:

$$\text{shuffle} : S \times S \times S \rightarrow S$$

and

$$f(g(X, Y), h(Z) \leftarrow \text{shuffle}(X, Y, Z) \rightarrow g(h(Y), g(Z, h(X)))$$
Benefits

The more expressive model

- implements a *specific* set of tree edit operations
- avoids clutter in the search space
- allows for situation specific scoring

Example: see bp_match below
Domain specific edit model for RNA structure trees

\[
\begin{align*}
    a & \leftrightarrow b & \text{-- base match} \\
    a & \leftrightarrow \varepsilon & \text{-- base indel} \\
    P(a \cdot X \cdot b) & \leftrightarrow P(c \cdot X \cdot d) & \text{-- base pair match} \\
    P(a \cdot X \cdot b) & \leftrightarrow c \cdot X \cdot d & \text{-- bond indel} \\
    P(a \cdot X \cdot b) & \leftrightarrow X & \text{-- base pair indel} \\
    P(a \cdot X \cdot b) & \leftrightarrow c \cdot X & \text{-- base pair loss_intro right} \\
    P(a \cdot X \cdot b) & \leftrightarrow X \cdot d & \text{-- base pair loss_intro left}
\end{align*}
\]

for \( a, b, c, d \in \mathcal{R} = \{A, C, G, U\} \).

For each edit operation, we need a scoring function, depending on the changes made ...
The core signature has one function per edit operation ... ... and represents tree alignments under this specific model

```
bp_match
A A bp_ins
A bp_match
A U b_del
A
b_match
C C
b_match
C C
b_match
C C
b_del
U
U A
U
U U
```
Domain specific ICORE rewrite rules

Rules of this ICORE:

\[
\begin{align*}
    a & \leftarrow \text{b\_match}(a, b) \quad \rightarrow b \\
    \varepsilon & \leftarrow \text{b\_ins}(b) \quad \rightarrow b \\
    a & \leftarrow \text{b\_del}(a) \quad \rightarrow \varepsilon \\
    P(a \cdot X \cdot c) & \leftarrow \text{bp\_match}(a, b, X, c, d) \quad \rightarrow P(b \cdot X \cdot d) \\
    X & \leftarrow \text{bp\_ins}(b, X, d) \quad \rightarrow P(b \cdot X \cdot d) \\
    P(a \cdot X \cdot c) & \leftarrow \text{bp\_del}(a, X, c) \quad \rightarrow X \\
    a \cdot X \cdot c & \leftarrow \text{bond\_ins}(a, b, X, c, d) \quad \rightarrow P(b, X, d) \\
    P(a \cdot X \cdot c) & \leftarrow \text{bond\_del}(a, b, X, c, d) \quad \rightarrow b \cdot X \cdot d \\
    X \cdot c & \leftarrow \text{bp\_introL}(b, X, c, d) \quad \rightarrow P(b \cdot X \cdot d) \\
    a \cdot X & \leftarrow \text{bp\_introR}(a, b, X, d) \quad \rightarrow P(b \cdot X \cdot d) \\
    P(a \cdot X \cdot c) & \leftarrow \text{bp\_lossL}(a, X, c, d) \quad \rightarrow X \cdot d \\
    P(a \cdot X \cdot c) & \leftarrow \text{bp\_lossR}(a, b, X, c) \quad \rightarrow b \cdot X
\end{align*}
\]
Score derived from core term

\[ t = \text{bp\_match}(A, A, \text{bp\_ins}(A, \text{bp\_match}(A, U, t', U, A), U), U, U) \]

where \( t' = \)

\[ \text{b\_del}(A) \cdot \text{b\_match}(C, C) \cdot \text{b\_match}(C, C) \cdot \text{b\_match}(C, C) \cdot \text{b\_del}(U) \]
The generalized model awaits implementation ... 

... though several ad-hoc specializations exist
Implementation status

The ICORE framework has not been implemented

One ICORE, as a proof of principle, was coded in the functional logic language Curry (M. Hanus, U. Kiel)
The ICORE framework has not been implemented

One ICORE, as a proof of principle, was coded in the functional logic language Curry (M. Hanus, U. Kiel)

But:
ICOREs properly subsume the framework of Algebraic Dynamic Programming, with semantics preserved
ICOREs versus ADP

Compared to ICOREs, ADP is

- based not on rewrite systems, but on yield grammars
- Computational engine: tabulating yield parser
- less general: restricted to sequence input
- implemented by several compilers

**Theorem:** All ADP programs can be rewritten as ICOREs.
ADP implementations

- ADP as a Haskell-embedded DSL (2000ff)
- ADP to C Compiler (P. Steffen, 2004 ff)
- ADPfusion (Haskell using stream fusion) (C. Höner zu Siederdissen, 2012 ff)
- Bellman’s GAP language and compiler: (G. Sauthoff, 2013)


Practical experience with ADP

- **RNAshapes**: RNA structure prediction
- **RNAhybrid**: miRNA target prediction
- **RNAforester**: RNA structure comparison
- **Locomotif**: Thermodynamic matchers

- **aCMs**: Ambivalent covariance models (*BMC Bioinformatics* 2015)
ICORE implementation challenges (1)

Implementation issues:

- Computational engine for coupled inverse rewriting
- Generation of tight recurrences
- Graceful adaptation to special cases
- Automated asymptotics
- DP table design (NP-hard)
- Table dimension analysis
- Automated backtrace
ICORE implementation challenges (2)

Theory

- computational power of ICOREs
- ICORE equivalence and optimization
- abstract operations on rewrite systems
- properties of algebra products – from $G(A, x)$ and $G(B, x)$ to $G(A \ast B, x)$
ICOREs are joint work with Hélène Touzet of Lille University

Robert Giegerich and Hélène Touzet:

Aside from background and theory, the above article provides about 30 DP problems in bioinformatics, formulated as ICOREs
ICOREs are joint work with Hélène Touzet of Lille University

Robert Giegerich and Hélène Touzet:

Aside from background and theory, the above article provides about 30 DP problems in bioinformatics, formulated as ICOREs

Thanks for your attention!
The ICORE vision

Towards an ICORE language and programming system

Repositories/Sublanguages
ICORE design
Instance compilation
Executable instances

Signatures
Rewrite Systems
Grammars
Algebras

Products & Instances
ICORE compiler

optimization
classification
stochastic sampling
# Problems on Sequences

<table>
<thead>
<tr>
<th>ICORE/Grammar</th>
<th>Dim.</th>
<th>Problem addressed</th>
<th>In Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EDITDISTANCE</strong></td>
<td>2</td>
<td>simple edit distance/alignment</td>
<td>4.2</td>
<td>77</td>
</tr>
<tr>
<td><strong>AFFINE</strong></td>
<td>2</td>
<td>edit distance, affine gaps</td>
<td>5.1</td>
<td>82</td>
</tr>
<tr>
<td><strong>AFFIOSCI</strong></td>
<td>2</td>
<td>oscillating affine gaps</td>
<td>5.1</td>
<td>84</td>
</tr>
<tr>
<td><strong>AFFITRACE</strong></td>
<td>2</td>
<td>sequence traces, affine gaps</td>
<td>5.1</td>
<td>84</td>
</tr>
<tr>
<td><strong>LOCALSEARCH</strong></td>
<td>2</td>
<td>generic local alignment</td>
<td>5.2</td>
<td>85</td>
</tr>
<tr>
<td><strong>MOTIFSEARCH</strong></td>
<td>2</td>
<td>short in long alignment</td>
<td>5.2.1</td>
<td>86</td>
</tr>
<tr>
<td><strong>SEMILOBALIGNMENT</strong></td>
<td>2</td>
<td>semi-global alignment</td>
<td>5.2.2</td>
<td>86</td>
</tr>
<tr>
<td><strong>LOCALALIGNMENT</strong></td>
<td>2</td>
<td>local alignment</td>
<td>5.2.3</td>
<td>86</td>
</tr>
<tr>
<td><strong>MATCHSEQ_S</strong></td>
<td>1</td>
<td>hardwired sequence matching</td>
<td>5.3.1</td>
<td>88</td>
</tr>
<tr>
<td><strong>MATCHAFFLI_S</strong></td>
<td>1</td>
<td>same with affine gap model</td>
<td>5.3.1</td>
<td>88</td>
</tr>
<tr>
<td><strong>MATCHSEQ_S</strong></td>
<td>1</td>
<td>profile HMM</td>
<td>5.3.2</td>
<td>88</td>
</tr>
</tbody>
</table>

with position-specific scores
### Problems on Trees (Structures)

<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
<th>Description</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNAFOLD</td>
<td>1</td>
<td>RNA folding</td>
<td>6.3</td>
<td>93</td>
</tr>
<tr>
<td>STRUCTALI</td>
<td>2</td>
<td>struct. Alignment prototype</td>
<td>6.4</td>
<td>97</td>
</tr>
<tr>
<td>SIMULTANEOUSFOLDING</td>
<td>2</td>
<td>generalized fold and align</td>
<td>6.5</td>
<td>98</td>
</tr>
<tr>
<td>EXACTCONSENSUSSTRUCTURE</td>
<td>2</td>
<td>exact consensus structure for two RNA sequences</td>
<td>6.5.2.</td>
<td>98</td>
</tr>
<tr>
<td>SANKOFF</td>
<td>2</td>
<td>simultaneous fold and align</td>
<td>6.5.3.</td>
<td>99</td>
</tr>
<tr>
<td>S2SGENERIC</td>
<td>2</td>
<td>covariance model, generic</td>
<td>6.6.1.</td>
<td>101</td>
</tr>
<tr>
<td>S2SEXACT</td>
<td>2</td>
<td>match RNA sequence to target structure</td>
<td>6.6.2.</td>
<td>100</td>
</tr>
<tr>
<td>MATCH_S2S_R</td>
<td>2</td>
<td>exact local motif matcher</td>
<td>6.6.4.</td>
<td>103</td>
</tr>
<tr>
<td>MATCH_S2S_R'</td>
<td>1</td>
<td>motif matcher, hard coded</td>
<td>6.6.4.</td>
<td>104</td>
</tr>
<tr>
<td>SCFG</td>
<td>1</td>
<td>stochastic context free grammar</td>
<td>6.3</td>
<td>93</td>
</tr>
<tr>
<td>COVARIANCEMODEL_R</td>
<td>1</td>
<td>covariance model, hard coded</td>
<td>6.6.5.</td>
<td>106</td>
</tr>
</tbody>
</table>
The ICORE framework – some technicalities

ICOREs are built from

- signatures: represent input(s) and solution candidates
- tree grammars: define candidate space
- algebras: evaluate solution candidates under objective function $\phi$
- term rewrite rules define input–output relation

Tree grammar example
ICORE definition

Given
- $k$ “satellite” (input) signatures $\Sigma_i$,
- “core” signature $\zeta$,
- tree grammar $\mathcal{G}$ over $\zeta$,
- $\zeta$-algebra $A$ with objective $\phi$
- ICORE rewrite rules $T_\zeta \rightarrow T_{\Sigma_i}$
- $k$ inputs $t_i \in T(\Sigma_i)$,

compute the multiset

$$\phi([A(c) \mid c \in L(\mathcal{G}), c \overset{\hat{i}}{\Rightarrow} t_i])$$
ICORE definition

Given

– $k$ “satellite” (input) signatures $\Sigma_i$,
– “core” signature $\zeta$,
– tree grammar $G$ over $\zeta$,
– $\zeta$-algebra $A$ with objective $\phi$
– ICORE rewrite rules $T_\zeta \rightarrow T_{\Sigma_i}$
– $k$ inputs $t_i \in T(\Sigma_i)$,

compute the multiset

$$\phi([A(c) \mid c \in L(G), c \overset{\hat{\Rightarrow}}{\Rightarrow} i, t_i])$$

Note: $\overset{\hat{\Rightarrow}}{\Rightarrow}$ is coupled rewriting

identical lefthand sides for all $i$

redex position synchronized for all $i$
ICORE computability

Under mild conditions, the relevant part of $L(\mathcal{G})$ is finite. Inputs are homeomorphically embedded in the core terms.

Brute force strategy for computing

$$\phi([A(c) \mid c \in L(\mathcal{G}), c \xrightarrow{i} t_i])$$

in three phases:

- enumerate (relevant part of) $L(\mathcal{G})$
- check coupled rewrite to inputs
- compute $[A(c)\mid\ldots]$  
- apply $\phi(\ldots)$

Candidate multiset is exponential in the size of the input(s): Phases must be amalgamated in execution.
 Rewriting with $\Rightarrow$

$\Rightarrow$ implements a special rewrite relation:

- disjoint core/satellite signatures
- coupled rewrites from core term positions to $k$ inputs: same lefthand sides applied in the same positions

When needed: associative rewriting for $\cdot$ allowed