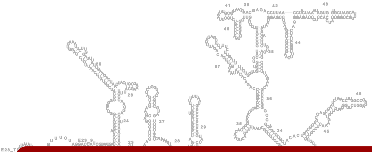
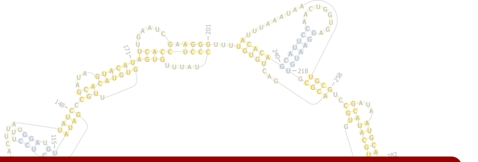
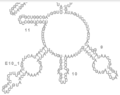
Some little-known tricks in ensemble algebraic dynamic programming and enumerative combinatorics

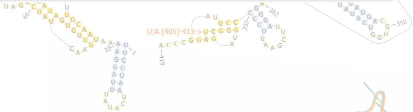
Yann Ponty

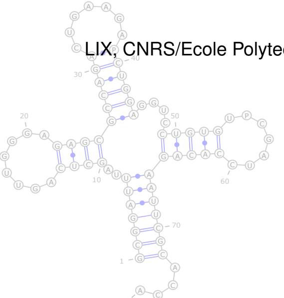
LIX, CNRS/Ecole Polytechnique

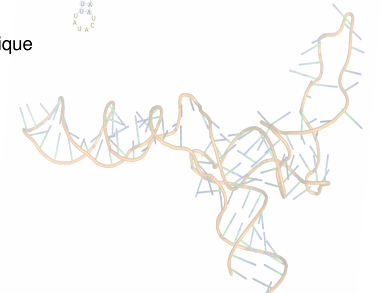


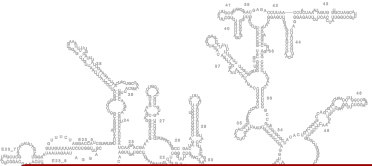


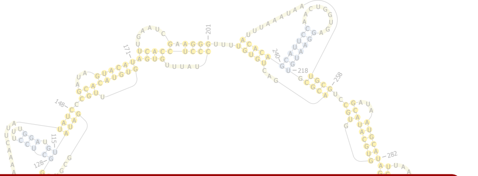




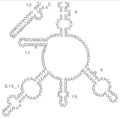


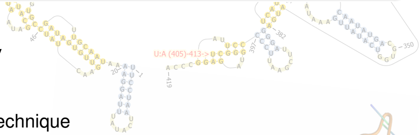






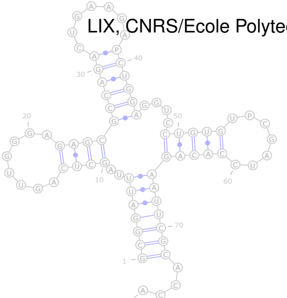
Enumerative combinatorics is taking ADP by storm, don't be left out!

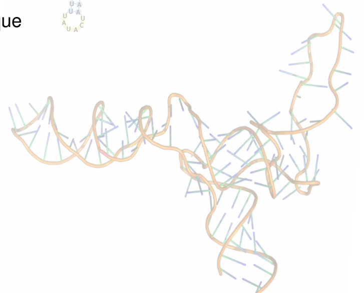




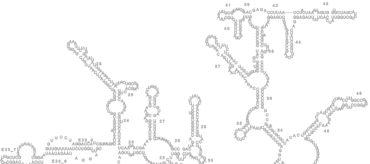
Yann Ponty

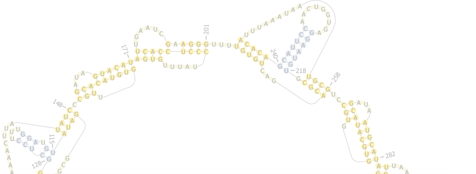
LIX, CNRS/Ecole Polytechnique





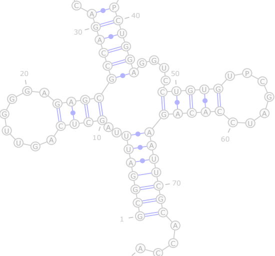
One weird trick algorithm designers don’t want you to know!!

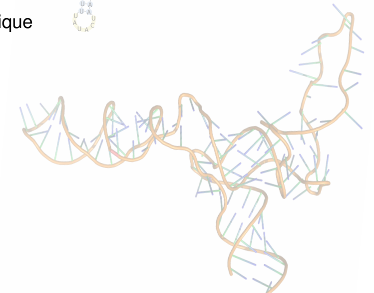


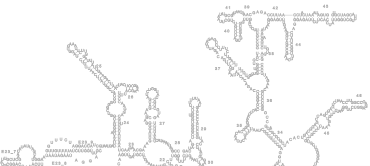
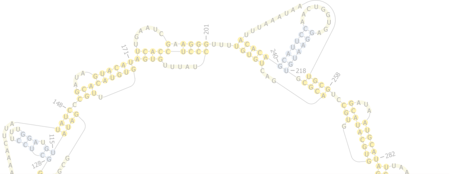


Yann Ponty

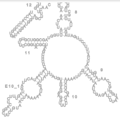
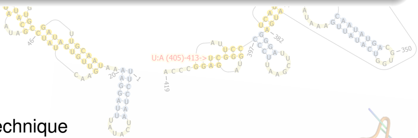
LIX, CNRS/Ecole Polytechnique





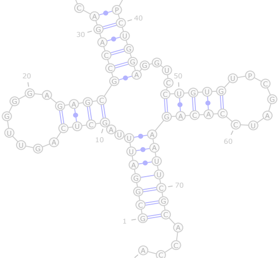
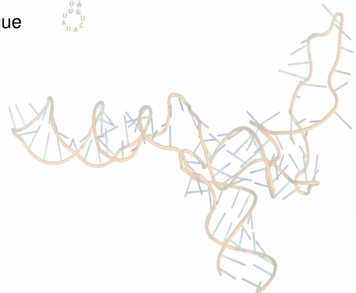



Closeted combinatorist found this old algebra in the attic, watch what happened next!!!

Yann Ponty

LIX, CNRS/Ecole Polytechnique

Optimality and beyond

Why do we optimize?

- **For fun:** Because, who doesn't just ***love*** algorithms?
- **For money:** Operations research, network design. . .
- **For love** (of exhaustivity): Correct negative results (vs heuristics)
- **To predict** the unobservable: RNAs, phylogenies. . .

But optima may be poorly representative of search space

$\Rightarrow$ **Ensemble dynamic programming**

Optimality and beyond

Why do we optimize?

- **For fun:** Because, who doesn't just ***love*** algorithms?
- **For money:** Operations research, network design. . .
- **For love** (of exhaustivity): Correct negative results (vs heuristics)
- **To predict** the unobservable: RNAs, phylogenies. . .

But optima may be poorly representative of search space

⇒ **Ensemble dynamic programming**

Optimality and beyond

Why do we optimize?

- **For fun:** Because, who doesn't just ***love*** algorithms?
- **For money:** Operations research, network design…
- **For love** (of exhaustivity): Correct negative results (vs heuristics)
- **To predict** the unobservable: RNAs, phylogenies…

But optima may be poorly representative of search space

⇒ **Ensemble dynamic programming**

Optimality and beyond

Why do we optimize?

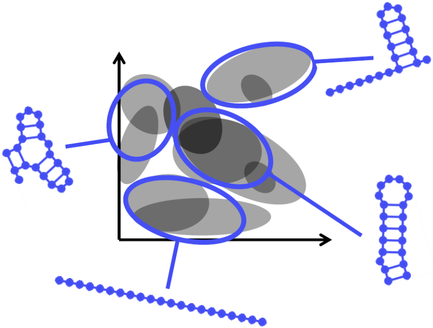
- **For fun:** Because, who doesn't just ***love*** algorithms?
- **For money:** Operations research, network design...
- **For love** (of exhaustivity): Correct negative results (vs heuristics)
- **To predict** the unobservable: RNAs, phylogenies...

But optima may be poorly representative of search space

$\Rightarrow$ **Ensemble dynamic programming**

Ensemble analysis in RNA research

Paradigm shifts: Energy-minimization $\rightarrow$ **Thermodynamics** $\rightarrow$ Kinetics



- Partition function
- Inside/Outside
- Max Expected Accuracy
- Partitioned approaches
- Sampling/clustering

Most likely features rather than most likely solution
Distributions more meaningful than objects

Minimal Algebraic DP framework

In this talk, ADP instance =

- **Yield Grammar** (Context-Free)
  - Each derivation *decorated* with a **constructor** (function)
  - Implicitly generates search space for instance

    (no filtering beyond size of terminal $\texttt{char}/\varepsilon$)
  - Derivations implicitly combined by a **choice** function $\oplus$
- **Evaluation Algebra**, defining semantics of functions
- Homogenous **domain**/data type Δ

Example: Nussinov-style RNA folding, input sequence w, $|w| = n$

$$S \to \mathrm{up}(\texttt{char}, S) \mid \mathrm{pair}(\texttt{char}, S, \texttt{char}, S) \mid \mathrm{nil}(\varepsilon) \qquad (\text{choice } \oplus)$$

MFE algebra: $(\Delta, \oplus) := (\mathbb{R}, \min)$

- $\mathrm{up}(i \in [1, n], v \in \Delta) \to v$
- $\mathrm{pair}(i, v_1, j, v_2) \to v_1 + v_2 + E(w_i, w_j)$
- $\mathrm{nil}(i) \to 0$

Partition function: $(\Delta, \oplus) := (\mathbb{R}, +)$

- $\mathrm{up}(i, v) \to v$
- $\mathrm{pair}(i, v_1, j, v_2) \to e^{-E(w_i, w_j)\cdot\beta} \times v_1 \times v_2$
- $\mathrm{nil}(i) \to 0$

with $E(x, y) = -1$ if $\{x, y\} \in \{\{\mathrm{G}, \mathrm{C}\}, \{\mathrm{A}, \mathrm{U}\}, \{\mathrm{G}, \mathrm{U}\}\}$, or $+\infty$ otherwise

Additive features of a DP scheme

Additive feature F: Associate atomic contributions to the constructor terms, and accumulate them over parse tree/term.

Example: #Base-pairs feature F_{bp}

$$S \rightarrow \text{up}(\texttt{char}, S) \mid \text{pair}(\texttt{char}, S, \texttt{char}, S) \mid \text{nil}(\varepsilon) \qquad (\text{choice } \oplus)$$

$F_{bp}(\text{pair}) := 1$, $F_{bp}(-) := 0$

Typical candidates for F:

- #Base-pairs, #Unpaired
- 5'/3' distance
- Free-Energy (requires arguments)
- #Helices, #Hairpins, #Multiloops, #Bulges... (refined grammar)
- Distance to reference structure

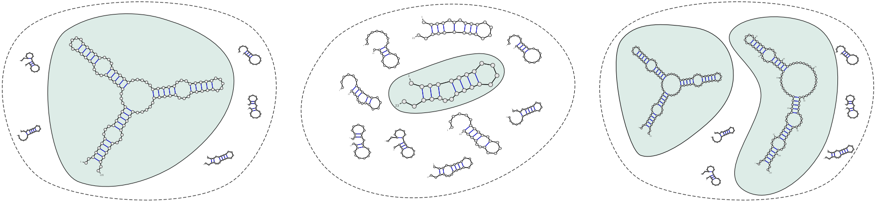
Part I: Moments of additive features in Boltzmann-like distributions

Motivation

Thermodynamic equilibrium, Boltzmann-distributed sec. struct. for RNA w:

$$\mathbb{P}(S \mid w) = \frac{\mathcal{B}(w,S)}{\mathcal{Z}_w} \text{ with } \mathcal{Z}_w = \sum_{S'} \mathcal{B}(w,S') \text{ and } \mathcal{B}(w,S) := e^{-E(w,S)\cdot\beta}$$

General belief: Thermodynamics more accurate than energy minimization
$\Rightarrow$ Functional evolutionary pressure on Boltzmann ensemble?



Functional folding?

Ill-defined folding: mRNA?

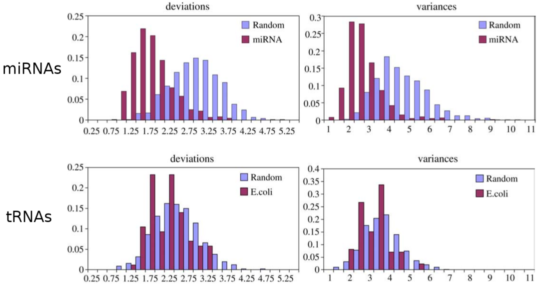
Bistable RNA Kinetics?

Function-specific nature of ensemble features – 1

Consider **moments** μ_1 (expect.), μ_2 ($\approx$var.)... of free-energy distribution

$$\mu_p = \mathbb{E}(E(S)^p \mid w) = \sum_{S'} E(w, S')^p \cdot \mathbb{P}(S' \mid w) = \frac{\mathcal{Y}_p}{\mathcal{Y}_0}$$

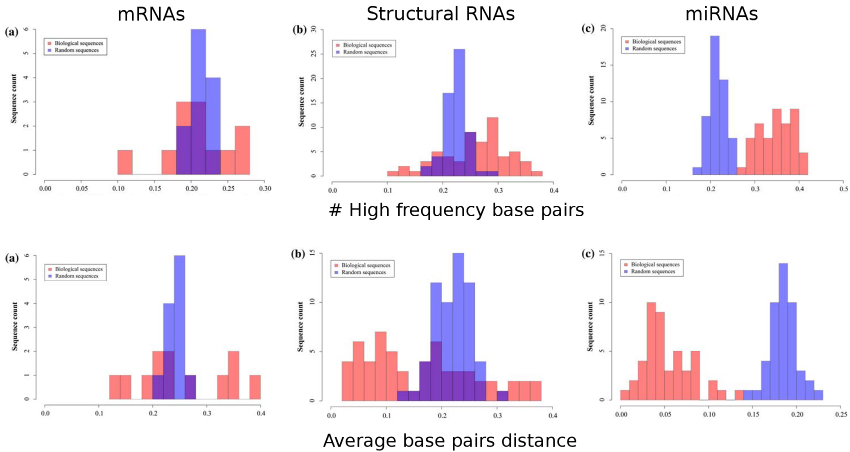
with $\mathcal{Y}_p = \sum_{S'} E(w, S')^p \cdot \mathcal{B}(w, S')$ (Note: $\mathcal{Z}_w = \mathcal{Y}_0$)



Pre-ML (*i.e.* modest) conclusion: *None of the statistics has sufficient separating power to distinguish individual biological and random sequences*

[Miklos, Meyer & Nagy, Bul Mat Bio 2005]

Function-specific nature of ensemble features – 2



Boltzmann sampling shows that miRNAs have distinct base-pair distributions than dinucleotide shuffles. [Chan & Ding, J Mat Bio 2008]

Other features/moments? (at which cost?)

Objective

Problem: Given ADP scheme for partition function $\mathcal{Z}_w$, compute[1]

$$\mathcal{Y}_p = \sum_{S'} F(w, S')^p \cdot \mathcal{Z}_{w,S'}$$

where F is an additive feature.

Typical candidates for F:

- Energy
- Distance to reference structure
- #Helices, #Hairpins, #Multiloops, #Bulges...
- 5'/3' distance

Expected value of F = $\mathcal{Y}_1 / \mathcal{Y}_0$; Std dev of F = $\sqrt{\mathcal{Y}_2 / \mathcal{Y}_0 - {\mathcal{Y}_1}^2 / {\mathcal{Y}_0}^2}$

[1] as lazily as possible

Objective

Problem: Given ADP scheme for partition function $\mathcal{Z}_w$, compute[2]

$$\mathcal{Y}_{p_1\cdots p_k} = \sum_{S'} F_1(w,S')^{p_1}\cdots F_k(w,S')^{p_k}\cdot \mathcal{Z}_{w,S'}$$

where $F_1,\cdots,F_k$ are additive features.

Typical candidates features:

- Energy
- Distance to reference structure
- #Helices, #Hairpins, #Multiloops, #Bulges…
- 3'/5' distance

Expected value of F = $\mathcal{Y}_1/\mathcal{Y}_0$; Std dev of F = $\sqrt{\mathcal{Y}_2/\mathcal{Y}_0 - {\mathcal{Y}_1}^2/{\mathcal{Y}_0}^2}$

Pearson Correlation of F and $F' = \dfrac{\mathcal{Y}_{1,1}-\mathcal{Y}_{0,1}\cdot\mathcal{Y}_{1,0}/\mathcal{Y}_{0,0}}{\sqrt{\mathcal{Y}_{2,0}-{\mathcal{Y}_{1,0}}^2/\mathcal{Y}_{0,0}}\cdot\sqrt{\mathcal{Y}_{0,2}-{\mathcal{Y}_{0,1}}^2/\mathcal{Y}_{0,0}}}$

[2] as lazily as possible

Moments through controlled ambiguity

Idea: Introduce controlled ambiguity within part. fun. grammar

[Ponty, Saule WABI 2011]

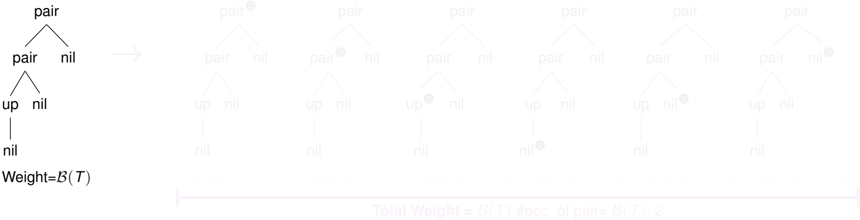
For each parse tree T generated with weight $\mathcal{B}(T)$ by original DP scheme:

- Duplicate tree $|T|$ times, **pointing** a different node in each copy;
- Multiply **weight** of copies by feature value of pointed node.

Overall weight of T-duplicates now becomes $\sum_{v\in T} F(v) \times \mathcal{B}(T) = F(T) \times \mathcal{B}(T)$

Example: Structure `((.))`, Feature = #base pairs

Original



Property: Computing partition function over duplicates yields $\sum_T F(T) \times \mathcal{B}(T) = \mathcal{Y}_1$.

Remark: Similar to **(partial) pointing** in enumerative combinatorics [Denise, P, Termier TCS 2010]

Moments through controlled ambiguity

Idea: Introduce controlled ambiguity within part. fun. grammar

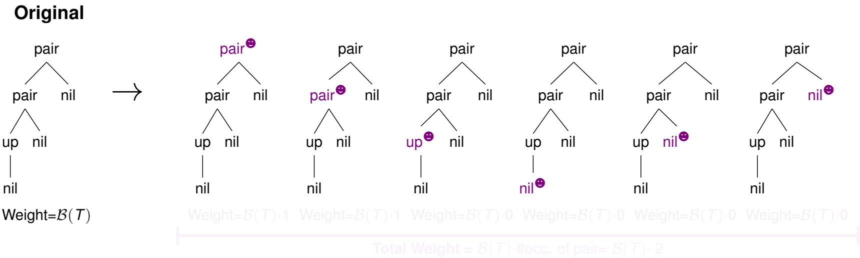
[Ponty, Saule WABI 2011]

For each parse tree T generated with weight $\mathcal{B}(T)$ by original DP scheme:

- Duplicate tree $|T|$ times, **pointing** a different node in each copy;
- Multiply **weight** of copies by feature value of pointed node.

Overall weight of T-duplicates now becomes $\sum_{v\in T} F(v) \times \mathcal{B}(T) = F(T) \times \mathcal{B}(T)$

Example: Structure `((.))`, Feature = #base pairs



Property: Computing partition function over duplicates yields $\sum_T F(T) \times \mathcal{B}(T) = \mathcal{Y}_1$.

Remark: Similar to **(partial) pointing** in enumerative combinatorics [Denise, P, Termier TCS 2010]

Moments through controlled ambiguity

Idea: Introduce controlled ambiguity within part. fun. grammar

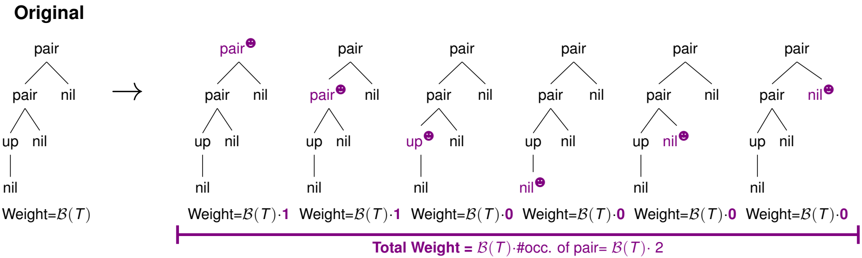
[Ponty, Saule WABI 2011]

For each parse tree T generated with weight $\mathcal{B}(T)$ by original DP scheme:

- Duplicate tree $|T|$ times, **pointing** a different node in each copy;
- Multiply **weight** of copies by feature value of pointed node.

Overall weight of T-duplicates now becomes $\sum_{v\in T} F(v) \times \mathcal{B}(T) = F(T) \times \mathcal{B}(T)$

Example: Structure `((.))`, Feature = #base pairs



Property: Computing partition function over duplicates yields $\sum_T F(T) \times \mathcal{B}(T) = \mathcal{Y}_1$.

Remark: Similar to **(partial) pointing** in enumerative combinatorics [Denise, P, Termier TCS 2010]

Moments through controlled ambiguity

Idea: Introduce controlled ambiguity within part. fun. grammar

[Ponty, Saule WABI 2011]

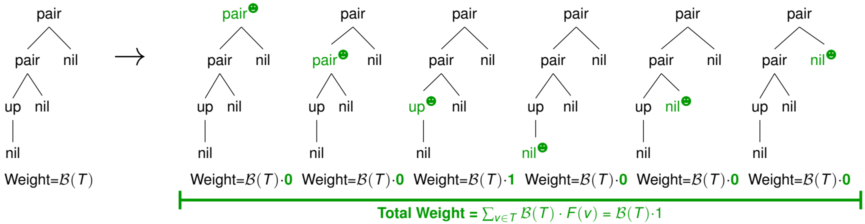
For each parse tree T generated with weight $\mathcal{B}(T)$ by original DP scheme:

- Duplicate tree $|T|$ times, **pointing** a different node in each copy;
- Multiply **weight** of copies by feature value of pointed node.

Overall weight of T-duplicates now becomes $\sum_{v\in T} F(v) \times \mathcal{B}(T) = F(T) \times \mathcal{B}(T)$

Example: Structure `((.))`, Feature = #unpair bases

Original



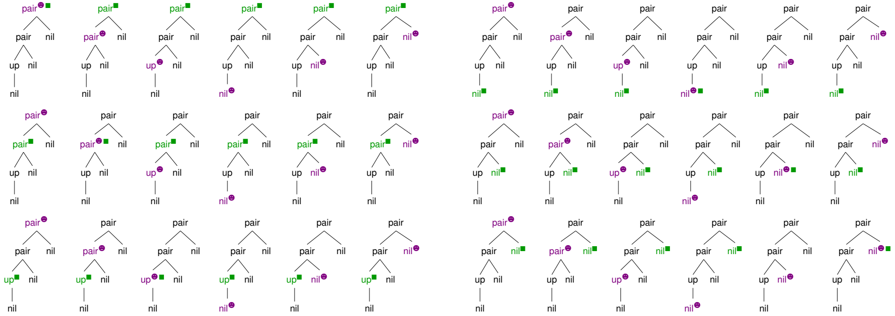
Property: Computing partition function over duplicates yields $\sum_T F(T) \times \mathcal{B}(T) = \mathcal{Y}_1$.

Remark: Similar to **(partial) pointing** in enumerative combinatorics [Denise, P, Termier TCS 2010]

Higher-order moments

Pointing can be generalized using multiple (types of) points.
Any ☻$_i$-pointing of node $v \in T$ contributes factor $F_i(v)$ to weight

Example: Two (possibly identical) features F (☻ point) and F' (■ point)



Overall weight of T-duplicates: $\sum_{v \in T} \sum_{v' \in T} F(v) \cdot F'(v') = F(T) \cdot F'(T)$

Property: Computing the partition function on a grammar $\mathcal{G}_{p_1 \cdots p_k}$ generating $(p_1 \cdots p_k)$-pointed versions of original trees yields $\mathcal{Y}_{p_1 \cdots p_k}$

$\rightarrow$ **How to construct $\mathcal{G}_{p_1 \cdots p_k}$?**

Constructing a single-pointed grammar

Pointing derivation as a **propagation/dropping** scheme for $\mathcal{G}_1$:

- For each non-terminal $S \in NT$, create new NT $S^{\blacksmiley}$ responsible for placing a point in the sub-parsetrees generated from S;
- Each derivation $S \to \text{fun}(arg_1, \ldots, arg_m)$, $arg_i \in NT \cup T \cup \{\varepsilon\}$, leads to:

$$\begin{aligned} S^{\blacksmiley} \to\ & \text{fun}^{\blacksmiley}(arg_1, \ldots, arg_m) && \text{(Dropping point)} \\ \mid\ & \text{fun}(arg_1{}^{\blacksmiley}, \ldots, arg_m) && \text{(Propagation, only if } arg_1 \in NT\text{)} \\ & \vdots \\ \mid\ & \text{fun}(arg_1, \ldots, arg_m{}^{\blacksmiley}) && \text{(Propagation, only if } arg_m \in NT\text{)} \end{aligned}$$

- Enrich algebra with $\text{fun}^{\blacksmiley}(arg_1, \ldots, arg_m) \to F(\text{fun}) \times \text{fun}(arg_1, \ldots, arg_m)$

Example:

$$S \to \text{pair}(\texttt{char}, S, \texttt{char}, S) \mid \text{up}(\texttt{char}, S) \mid \text{nil}(\varepsilon)$$

$$\begin{aligned} S^{\blacksmiley} \to\ & \text{pair}^{\blacksmiley}(\texttt{char}, S, \texttt{char}, S) \mid \text{pair}(\texttt{char}, S^{\blacksmiley}, \texttt{char}, S) \mid \text{pair}(\texttt{char}, S, \texttt{char}, S^{\blacksmiley}) \\ \mid\ & \text{up}^{\blacksmiley}(\texttt{char}, S) \mid \text{up}(\texttt{char}, S^{\blacksmiley}) \mid \text{nil}^{\blacksmiley}(\varepsilon) \end{aligned}$$

Complexity: Similar as initial grammar, up to constants

Memory$\times 2$, Time$\times(m^\star + 1)$ $\qquad (m^\star := \max_f \#\text{NTargs}(f))$

Constructing a single-pointed grammar

Pointing derivation as a **propagation/dropping** scheme for $\mathcal{G}_1$:

- For each non-terminal $S \in NT$, create new NT $S^{\smiley}$ responsible for placing a point in the sub-parsetrees generated from S;
- Each derivation $S \to \mathrm{fun}(arg_1, \ldots, arg_m)$, $arg_i \in NT \cup T \cup \{\varepsilon\}$, leads to:

$$
\begin{aligned}
S^{\smiley} &\to \mathrm{fun}^{\smiley}(arg_1, \ldots, arg_m) && \text{(Dropping point)}\\
&\mid \mathrm{fun}(arg_1{}^{\smiley}, \ldots, arg_m) && \text{(Propagation, only if } arg_1 \in NT\text{)}\\
&\qquad \vdots \\
&\mid \mathrm{fun}(arg_1, \ldots, arg_m{}^{\smiley}) && \text{(Propagation, only if } arg_m \in NT\text{)}
\end{aligned}
$$

- Enrich algebra with $\mathrm{fun}^{\smiley}(arg_1, \ldots, arg_m) \to F(\mathrm{fun}) \times \mathrm{fun}(arg_1, \ldots, arg_m)$

Example:

$$
\begin{aligned}
S &\to \mathrm{pair}(\mathtt{char}, S, \mathtt{char}, S) \mid \mathrm{up}(\mathtt{char}, S) \mid \mathrm{nil}(\varepsilon)\\
S^{\smiley} &\to \mathrm{pair}^{\smiley}(\mathtt{char}, S, \mathtt{char}, S) \mid \mathrm{pair}(\mathtt{char}, S^{\smiley}, \mathtt{char}, S) \mid \mathrm{pair}(\mathtt{char}, S, \mathtt{char}, S^{\smiley})\\
&\mid \mathrm{up}^{\smiley}(\mathtt{char}, S) \mid \mathrm{up}(\mathtt{char}, S^{\smiley}) \mid \mathrm{nil}^{\smiley}(\varepsilon)
\end{aligned}
$$

Complexity: Similar as initial grammar, up to constants

Memory$\times 2$, Time$\times(m^\star + 1)$ $\quad (m^\star := \max_f \#\mathrm{NTargs}(f))$

Constructing a single-pointed grammar

Pointing derivation as a **propagation/dropping** scheme for $\mathcal{G}_1$:

- For each non-terminal $S \in NT$, create new NT $S^{☻}$ responsible for placing a point in the sub-parsetrees generated from S;
- Each derivation $S \to \mathrm{fun}(arg_1, \ldots, arg_m)$, $arg_i \in NT \cup T \cup \{\varepsilon\}$, leads to:

$$
\begin{aligned}
S^{☻} &\to \mathrm{fun}^{☻}(arg_1, \ldots, arg_m) && \text{(Dropping point)}\\
&\mid \mathrm{fun}(arg_1{}^{☻}, \ldots, arg_m) && \text{(Propagation, only if } arg_1 \in NT\text{)}\\
&\quad\vdots\\
&\mid \mathrm{fun}(arg_1, \ldots, arg_m{}^{☻}) && \text{(Propagation, only if } arg_m \in NT\text{)}
\end{aligned}
$$

- Enrich algebra with $\mathrm{fun}^{☻}(arg_1, \ldots, arg_m) \to F(\mathrm{fun}) \times \mathrm{fun}(arg_1, \ldots, arg_m)$

Example:

$$
\begin{aligned}
S &\to \mathrm{pair}(\mathtt{char}, S, \mathtt{char}, S) \mid \mathrm{up}(\mathtt{char}, S) \mid \mathrm{nil}(\varepsilon)\\
S^{☻} &\to \mathrm{pair}^{☻}(\mathtt{char}, S, \mathtt{char}, S) \mid \mathrm{pair}(\mathtt{char}, S^{☻}, \mathtt{char}, S) \mid \mathrm{pair}(\mathtt{char}, S, \mathtt{char}, S^{☻})\\
&\mid \mathrm{up}^{☻}(\mathtt{char}, S) \mid \mathrm{up}(\mathtt{char}, S^{☻}) \mid \mathrm{nil}^{☻}(\varepsilon)
\end{aligned}
$$

Complexity: Similar as initial grammar, up to constants
Memory $\times 2$, Time $\times (m^\star + 1)$ $\quad$ ($m^\star := \max_f \#\mathrm{NTargs}(f)$)

Constructing a multiple-pointed grammar

Iterated pointing : $\mathcal{Z}_{\mathcal{G}_1} = \mathcal{Y}_1, \mathcal{Z}_{(\mathcal{G}_1)_1} = \mathcal{Y}_2, \mathcal{Z}_{((\mathcal{G}_1)_1)_1} = \mathcal{Y}_3 \ldots$
Complexity overhead for computing $\mathcal{Y}_p$: Memory$\times 2^p$, Time$\times(m+1)^p$

Multiply-pointed derivations: [Ponty, Saule WABI 2011]

- $\forall S \in NT$, create new NTs $S^{☻1}, \cdots, S^{☻p}$ responsible for placing 1 to p points in the sub-parsetrees generated from S;
- Each derivation $S \to \text{fun}(arg_1, \ldots, arg_m)$, leads to:

$$S^{☻q} \to \bigcup_{\substack{q_0+q_1+\cdots+q_m=q \\ \text{s.t. } q_i=0 \text{ if } arg_i \notin NT}} \binom{q}{q_1, q_2, \cdots, q_m} \text{fun}^{☻q_0}(arg_1{}^{☻q_1}, \ldots, arg_m{}^{☻q_m})$$

- Add to algebra $\text{fun}^{☻q}(arg_1, \ldots, arg_m) \to F(\text{fun})^q \times \text{fun}(arg_1, \ldots, arg_m)$

Complexity: Memory$\times p$, Time$\times\binom{p+m^\star}{m^\star} \sim (p/m^\star+1)^{m^\star}$ when $k \gg m$

Bellman's GAP implementation **in progress** (M. Pommeret's PhD)
Applications: Machine learning, RNA evolution, Approx. of density of states…

Constructing a multiple-pointed grammar

Iterated pointing : $\mathcal{Z}_{\mathcal{G}_1} = \mathcal{Y}_1, \mathcal{Z}_{(\mathcal{G}_1)_1} = \mathcal{Y}_2, \mathcal{Z}_{((\mathcal{G}_1)_1)_1} = \mathcal{Y}_3 \ldots$
Complexity overhead for computing $\mathcal{Y}_p$: Memory$\times 2^p$, Time$\times(m+1)^p$

Multiply-pointed derivations: [Ponty, Saule WABI 2011]

- $\forall S \in NT$, create new NTs $S^{☻1}, \cdots, S^{☻p}$ responsible for placing 1 to p points in the sub-parsetrees generated from S;
- Each derivation $S \to \mathrm{fun}(arg_1, \ldots, arg_m)$, leads to:

$$S^{☻q} \to \bigcup_{\substack{q_0+q_1+\cdots+q_m=q \\ \text{s.t. } q_i=0 \text{ if } arg_i \notin NT}} \binom{q}{q_1, q_2, \cdots, q_m} \mathrm{fun}^{☻q_0}(arg_1{}^{☻q_1}, \ldots, arg_m{}^{☻q_m})$$

- Add to algebra $\mathrm{fun}^{☻q}(arg_1, \ldots, arg_m) \to F(\mathrm{fun})^q \times \mathrm{fun}(arg_1, \ldots, arg_m)$

Complexity: Memory$\times p$, Time$\times\binom{p+m^\star}{m^\star} \sim (p/m^\star+1)^{m^\star}$ when $k \gg m$

Bellman's GAP implementation **in progress** (M. Pommeret's PhD)
Applications: Machine learning, RNA evolution, Approx. of density of states...

Constructing a multiple-pointed grammar

Iterated pointing : $\mathcal{Z}_{\mathcal{G}_1} = \mathcal{Y}_1, \mathcal{Z}_{(\mathcal{G}_1)_1} = \mathcal{Y}_2, \mathcal{Z}_{((\mathcal{G}_1)_1)_1} = \mathcal{Y}_3 \ldots$
Complexity overhead for computing $\mathcal{Y}_p$: Memory$\times 2^p$, Time$\times(m+1)^p$

Multiply-pointed derivations: [Ponty, Saule WABI 2011]

- $\forall S \in NT$, create new NTs $S^{\text{☻}1}, \cdots, S^{\text{☻}p}$ responsible for placing 1 to p points in the sub-parsetrees generated from S;
- Each derivation $S \to \text{fun}(arg_1, \ldots, arg_m)$, leads to:

$$S^{\text{☻}q} \to \bigcup_{\substack{q_0+q_1+\cdots+q_m=q \\ \text{s.t. } q_i=0 \text{ if } arg_i \notin NT}} \binom{q}{q_1, q_2, \cdots, q_m} \text{fun}^{\text{☻}q_0}(arg_1{}^{\text{☻}q_1}, \ldots, arg_m{}^{\text{☻}q_m})$$

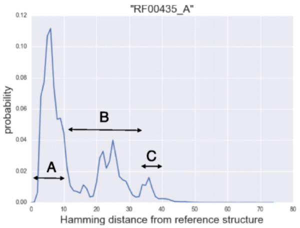
- Add to algebra $\text{fun}^{\text{☻}q}(arg_1, \ldots, arg_m) \to F(\text{fun})^q \times \text{fun}(arg_1, \ldots, arg_m)$

Complexity: Memory$\times p$, Time$\times\binom{p+m^\star}{m^\star} \sim (p/m^\star + 1)^{m^\star}$ when $k \gg m$

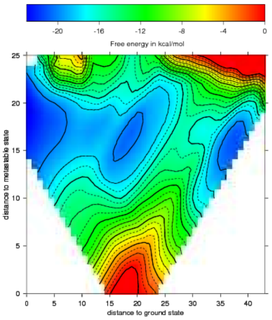
Bellman's GAP implementation **in progress** (M. Pommeret's PhD)
Applications: Machine learning, RNA evolution, Approx. of density of states...

Part II: Discrete Fourier Transform for classified ensemble DP

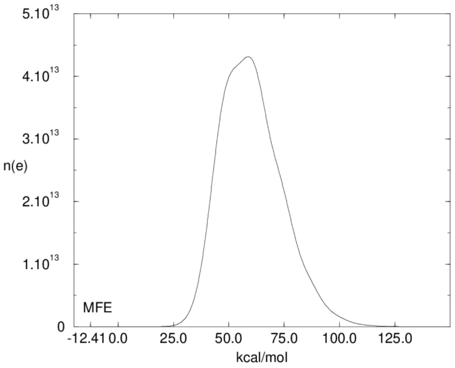
Classified ensemble dynamic programming



[Hagio *et al*, Bioinf. 2018]



[Lorenz, Flamm, Hofacker, GCB 2009]



[Cupal, Hofacker, Stadler GCB 1996]

- Measure prevalence of sub-class
- Projection of search space
- Combinatorics vs Optimality
- Computationally challenging

Explicit/implicit convolution products

Objective: For all value v in the co-domain of feature F, compute

$$\mathcal{Z}^{(v)} = \sum_{T \in \mathcal{T}_w \text{ s.t. } F(T)=v} \mathcal{B}(T)$$

Remark: Co-domain D can be **exponentially large**, *e.g.* when values taken by F are exponentially far apart (on $|w|$).

First idea: Duplicate NTs, distribute targeted value

$$S \to \text{fun}(arg_1, \ldots, arg_m) \longrightarrow S^{(v)} \to \bigcup_{\substack{v_1^{v_1} + \cdot + v_m^{v_m} + F(\text{fun}) = v \\ \text{s.t. } v_i = 0 \text{ if } arg_i \notin NT}} \text{fun}(arg_1, \ldots, arg_m)$$

Complexity: Memory$\times|D|$, Time$\times|D|^{m^\star}$
Alt.: Use polynomial data-type $\to$ complexity unchanged yet better constants

Impractical for typical applications ($|D| = n, n^2 \ldots, m^* = 2$)

Explicit/implicit convolution products

Objective: For all value v in the co-domain of feature F, compute

$$\mathcal{Z}^{(v)} = \sum_{T\in\mathcal{T}_w \text{ s.t. } F(T)=v} \mathcal{B}(T)$$

Remark: Co-domain D can be **exponentially large**, *e.g.* when values taken by F are exponentially far apart (on $|w|$).

First idea: Duplicate NTs, distribute targeted value

$$S \to \text{fun}(arg_1, \ldots, arg_m) \longrightarrow S^{(v)} \to \bigcup_{\substack{v_1^{v_1}+\cdot+v_m^{v_m}+F(\text{fun})=v \\ \text{s.t. } v_i=0 \text{ if } arg_i \notin NT}} \text{fun}(arg_1, \ldots, arg_m)$$

Complexity: Memory$\times|D|$, Time$\times|D|^{m^\star}$
Alt.: Use polynomial data-type $\to$ complexity unchanged yet better constants

Impractical for typical applications ($|D| = n, n^2 \ldots, m^* = 2$)

Interpolation

Second idea: Lagrange interpolation [Waldispühl & P, RECOMB 2011]

- Add monomial, based on local feature contribution, to evaluation algebra

$$\mathrm{fun}(arg_1, \ldots, arg_m) \longrightarrow x^{F(\mathrm{fun})} \times \mathrm{fun}(arg_1, \ldots, arg_m)$$

  for x a formal variable;
- For any concrete value of x, grammar/New algebra pair now computes

$$\mathcal{Z}(x) := \sum_{T \in \mathcal{T}_w \text{ s.t. } F(T)=v} \mathcal{B}(T) \cdot x^{F(T)} = \sum_{v \in D} \mathcal{Z}^v \cdot x^v$$

- $\mathcal{Z}(x)$ is a polynomial whose coeffs are interpolated from $|D|$ evaluations

Complexity: Time$\times|D|$ +$|D|^2$; Memory $+|D|$

Final idea: Discrete Fourier Transform (DFT) [Senter *et al*, RECOMB'13 & Plos One]

- Algebra change to evaluate $\mathcal{Z}(\omega)$ on (complex) $|D|$-th roots of -1;
- Inverse DFT gives coeffs in $\mathcal{O}(|D| \log |D|)$ after $|D|$ evaluations.

Complexity: Time$\times|D|$ + $|D|$ log; Memory $+|D|$

Interpolation

Second idea: Lagrange interpolation [Waldispühl & P, RECOMB 2011]

- Add monomial, based on local feature contribution, to evaluation algebra

$$\mathrm{fun}(arg_1, \ldots, arg_m) \longrightarrow x^{F(\mathrm{fun})} \times \mathrm{fun}(arg_1, \ldots, arg_m)$$

  for x a formal variable;
- For any concrete value of x, grammar/New algebra pair now computes

$$\mathcal{Z}(x) := \sum_{T \in \mathcal{T}_w \text{ s.t. } F(T)=v} \mathcal{B}(T) \cdot x^{F(T)} = \sum_{v \in D} \mathcal{Z}^v \cdot x^v$$

- $\mathcal{Z}(x)$ is a polynomial whose coeffs are interpolated from $|D|$ evaluations

Complexity: Time$\times|D|$ $+|D|^2$; Memory $+|D|$

Final idea: Discrete Fourier Transform (DFT) [Senter *et al*, RECOMB'13 & Plos One]

- Algebra change to evaluate $\mathcal{Z}(\omega)$ on (complex) $|D|$-th roots of -1;
- Inverse DFT gives coeffs in $\mathcal{O}(|D| \log |D|)$ after $|D|$ evaluations.

Complexity: Time$\times|D|$ + $|D| \log$; Memory $+|D|$

Conclusion on classified ensemble DP

- Inverse DFT allows an implicit computation of classified partition function/counting
- Stable numerically & amenable to interval arithmetics (Sato *et al*)
- Bottleneck (evaluation on D points) embarassingly parallelizable

- Drastic asymptotic speed-up, some examples:
  - **Density of states**: Feature = Free-energy
    Time/Memory: $\Theta(n^5)/\Theta(n^3) \xrightarrow{\text{DFT}} \Theta(n^4)/\Theta(n^2)$ ($\Theta(n^3)$ on n cores)
  - **RNAbor**: Feature = Base-pair distance to reference structure
    Time/Memory: $\Theta(n^5)/\Theta(n^3) \xrightarrow{\text{DFT}} \Theta(n^4)/\Theta(n^2)$ ($\Theta(n^3)$ on n cores)
  - **RNA2DFold**: Features = distances to two reference structures
    Time/Memory: $\Theta(n^7)/\Theta(n^4) \xrightarrow{\text{DFT}} \Theta(n^5)/\Theta(n^2)$ ($\Theta(n^3)$ on n^2 cores)

- **Caveat 1:** Sensitive to #classes (Moments computations are not!)
- **Caveat 2:** Stochastic backtrack/sampling not readily available
  $\rightarrow$ Multidimensional Boltzmann sampling [Bodini & P, DMTCS & AOFA 2010]

Conclusion on classified ensemble DP

- Inverse DFT allows an implicit computation of classified partition function/counting
- Stable numerically & amenable to interval arithmetics (Sato *et al*)
- Bottleneck (evaluation on D points) embarassingly parallelizable

- Drastic asymptotic speed-up, some examples:
  - **Density of states**: Feature = Free-energy
    Time/Memory: $\Theta(n^5)/\Theta(n^3) \xrightarrow{\text{DFT}} \Theta(n^4)/\Theta(n^2)$ ($\Theta(n^3)$ on n cores)
  - **RNAbor**: Feature = Base-pair distance to reference structure
    Time/Memory: $\Theta(n^5)/\Theta(n^3) \xrightarrow{\text{DFT}} \Theta(n^4)/\Theta(n^2)$ ($\Theta(n^3)$ on n cores)
  - **RNA2DFold**: Features = distances to two reference structures
    Time/Memory: $\Theta(n^7)/\Theta(n^4) \xrightarrow{\text{DFT}} \Theta(n^5)/\Theta(n^2)$ ($\Theta(n^3)$ on n^2 cores)

- **Caveat 1:** Sensitive to #classes (Moments computations are not!)
- **Caveat 2:** Stochastic backtrack/sampling not readily available
  $\rightarrow$ Multidimensional Boltzmann sampling [Bodini & P, DMTCS & AOFA 2010]

Part III: Parametric optimization and ADP

Motivation

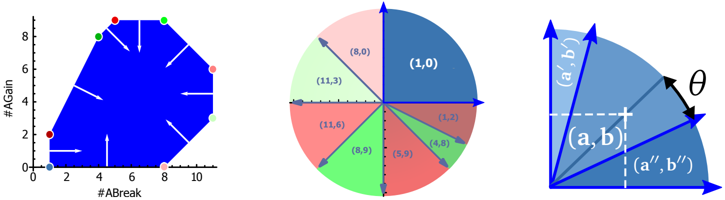
Meanwhile, in bioinformatics, you sometimes need **a single solution**:

- Objective function parameters are guess-timated
- Subject to experimental noise
- Inferred from partial data

How to measure the impact of parameters perturbations on predictions?
How to assess the validity domain, in the parameter space, of a prediction?

Grid search wasteful and often incorrect → **Parametric optimization**

[Gusfield *et al*, SODA 1994] [Pachter & Strumfels, 2005] [Forouzmand & Citsaz, Bioinf. 2013]



a. Polytope computation b. Cone segmentation c. Robustness analysis

Motivation

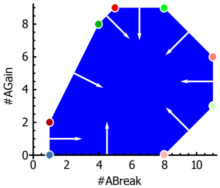
Meanwhile, in bioinformatics, you sometimes need **a single solution**:

- Objective function parameters are guess-timated
- Subject to experimental noise
- Inferred from partial data

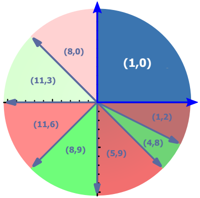
How to measure the impact of parameters perturbations on predictions?
How to assess the validity domain, in the parameter space, of a prediction?

Grid search wasteful and often incorrect $\rightarrow$ **Parametric optimization**

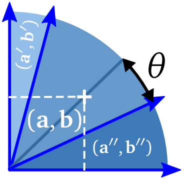
[Gusfield *et al*, SODA 1994] [Pachter & Strumfels, 2005] [Forouzmand & Citsaz, Bioinf. 2013]



a. Polytope computation



b. Cone segmentation



c. Robustness analysis

The Newton polytope

Consider **additive features** $F_1, \cdots, F_k$ + **objective function** $f^\star$ such that:

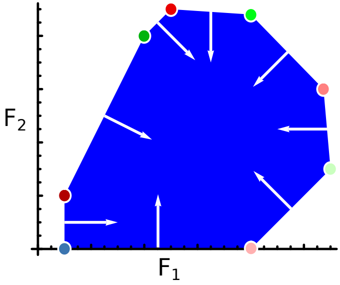
$$f^\star : T \rightarrow \alpha_1 \cdot F_1(T) + \alpha_2 \cdot F_2(T) + \cdots + \alpha_k \cdot F_k(T)$$

where $(\alpha_1, \cdots, \alpha_k)$ are the parameters of the optimization.

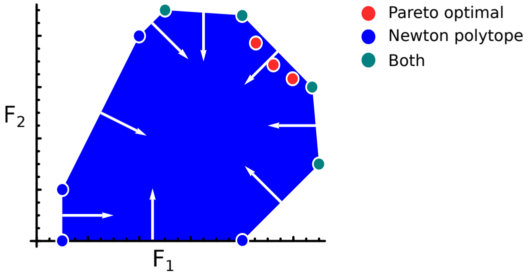
We call $(F_1(T), \cdots, F_k(T))$ the **signature** of T.

Definition (Newton polytope)

Newton polytope = **Convex hull** of signatures reached in the search space of a **given instance**



Some properties of the Polytope



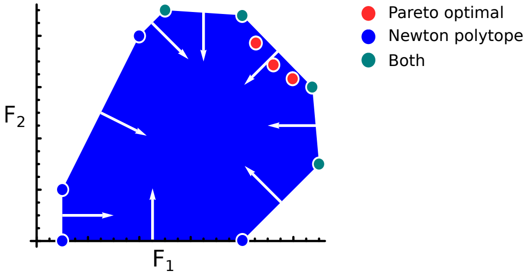
Property 1: The Newton polytope **only** contains signatures that are **(co)optimal** for some parameters vector

Property 2: The Newton polytope overlaps with the **Pareto front**, yet the two are incomparable

Property 3: **Normal vectors** of facets (lines) represent parameters such that all signatures in the facet are co-optimals (here, for minimization)

Property 4: Any param. vector which **falls *between*** the normal vectors adjacent to a vertex admits this vertex as co-optimal

Some properties of the Polytope



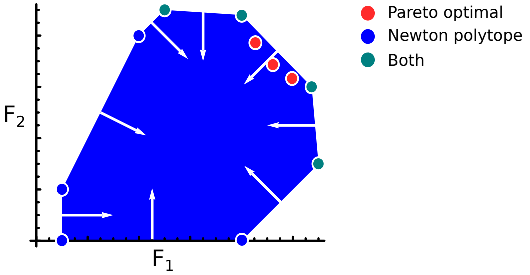
Property 1: The Newton polytope **only** contains signatures that are **(co)optimal** for some parameters vector

Property 2: The Newton polytope overlaps with the **Pareto front**, yet the two are incomparable

Property 3: **Normal vectors** of facets (lines) represent parameters such that all signatures in the facet are co-optimals (here, for minimization)

Property 4: Any param. vector which **falls *between*** the normal vectors adjacent to a vertex admits this vertex as co-optimal

Some properties of the Polytope



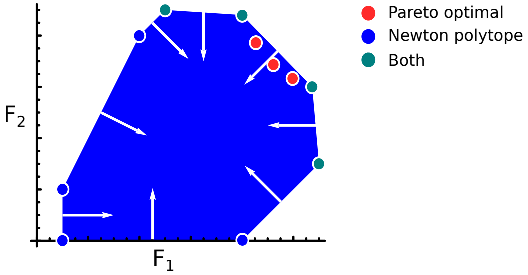
Property 1: The Newton polytope **only** contains signatures that are **(co)optimal** for some parameters vector

Property 2: The Newton polytope overlaps with the **Pareto front**, yet the two are incomparable

Property 3: **Normal vectors** of facets (lines) represent parameters such that all signatures in the facet are co-optimals (here, for minimization)

Property 4: Any param. vector which **falls *between*** the normal vectors adjacent to a vertex admits this vertex as co-optimal

Some properties of the Polytope



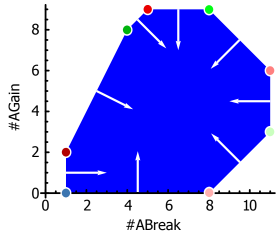
Property 1: The Newton polytope **only** contains signatures that are **(co)optimal** for some parameters vector

Property 2: The Newton polytope overlaps with the **Pareto front**, yet the two are incomparable

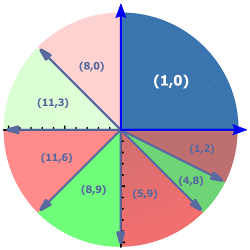
Property 3: **Normal vectors** of facets (lines) represent parameters such that all signatures in the facet are co-optimals (here, for minimization)

Property 4: Any param. vector which ***falls between*** the normal vectors adjacent to a vertex admits this vertex as co-optimal

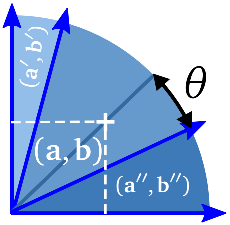
Normal fan



a. Polytope computation



b. Cone segmentation

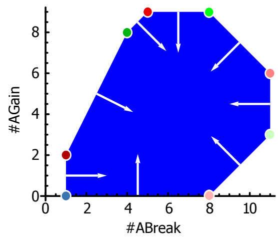


c. Robustness analysis

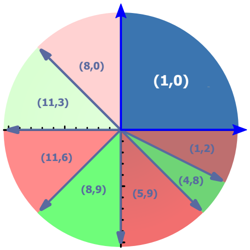
Computing the normal fan:

1. Compute Polytope (but how?)
2. Derive the normals of the polytope facets
3. Project them back onto the origin (dual parameter space)
4. Segment param. space into (hyper)cones where a single signature rules

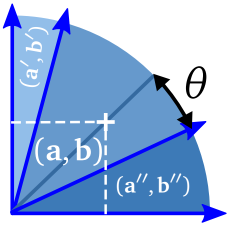
Normal fan



a. Polytope computation



b. Cone segmentation



c. Robustness analysis

Computing the normal fan:

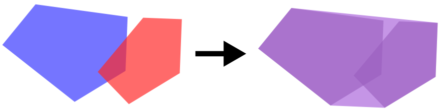
1. Compute Polytope **(but how?)**
2. Derive the normals of the polytope facets
3. Project them back onto the origin (dual parameter space)
4. Segment param. space into (hyper)cones where a single signature rules

Computing the polytope

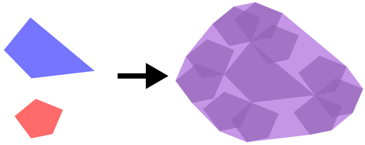
Thanks to ADP, simple algebraic substitution:

$$\begin{aligned} \mathsf{min} &\longrightarrow \text{Union + Convex Hull} \\ + &\longrightarrow \text{Minkowski sum + Convex Hull} \\ \alpha_i \cdot F_i(\text{fun}) &\longrightarrow (0, \ldots, F_i(\text{fun}), \ldots, 0) \end{aligned}$$

Union



Minkowski sum



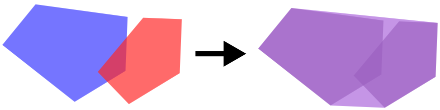
- Works even for ambiguous DP schemes (but completeness matters!)
- In practice, much fewer points that the worst-case $\mathcal{O}(D^{m-1})$ bound
- Implementation aspects a bit tricky (`qhull`, double representation...)

Computing the polytope

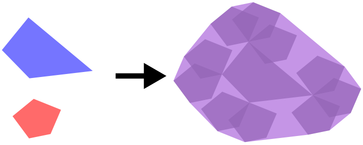
Thanks to ADP, simple algebraic substitution:

$$\begin{aligned} \min &\longrightarrow \text{Union + Convex Hull} \\ + &\longrightarrow \text{Minkowski sum + Convex Hull} \\ \alpha_i \cdot F_i(\text{fun}) &\longrightarrow (0, \ldots, F_i(\text{fun}), \ldots, 0) \end{aligned}$$

Union



Minkowski sum



- Works even for ambiguous DP schemes (but completeness matters!)
- In practice, much fewer points that the worst-case $\mathcal{O}(D^{m-1})$ bound
- Implementation aspects a bit tricky (`qhull`, double representation...)

Conclusion

Still many **generic applications** of ADP to explore:

- Exotic semi-rings + post-treatments (DFT, Normal Fan analysis)
- Grammar rewriting (Moments)
- Variations on stochastic backtrack (Boltzmann multidim., Boustrophedon, Non-redundant sampling)
- Generic optimizations (Sparsification?)

and low-level implementation for free!

Enumerative combinatorics **helps** by providing principled ways to:

- Transform grammars to do one's bidding (cf Labelle's species theory)
- Shorten unambiguity/completeness proofs through generating functions

But grammars (even multiple) are **not** co-substantial to ADP!

$\rightarrow$ Extend on the generative formalism? (while remaining effective)

Conclusion

Still many **generic applications** of ADP to explore:

- Exotic semi-rings + post-treatments (DFT, Normal Fan analysis)
- Grammar rewriting (Moments)
- Variations on stochastic backtrack (Boltzmann multidim., Boustrophedon, Non-redundant sampling)
- Generic optimizations (Sparsification?)

and low-level implementation for free!

Enumerative combinatorics **helps** by providing principled ways to:

- Transform grammars to do one's bidding (cf Labelle's species theory)
- Shorten unambiguity/completeness proofs through generating functions

But grammars (even multiple) are **not** co-substantial to ADP!

$\rightarrow$ Extend on the generative formalism? (while remaining effective)

Conclusion

Still many **generic applications** of ADP to explore:

- Exotic semi-rings + post-treatments (DFT, Normal Fan analysis)
- Grammar rewriting (Moments)
- Variations on stochastic backtrack (Boltzmann multidim., Boustrophedon, Non-redundant sampling)
- Generic optimizations (Sparsification?)

and low-level implementation for free!

Enumerative combinatorics **helps** by providing principled ways to:

- Transform grammars to do one's bidding (cf Labelle's species theory)
- Shorten unambiguity/completeness proofs through generating functions

But grammars (even multiple) are **not** co-substantial to ADP!

$\rightarrow$ Extend on the generative formalism? (while remaining effective)

That's all folks!

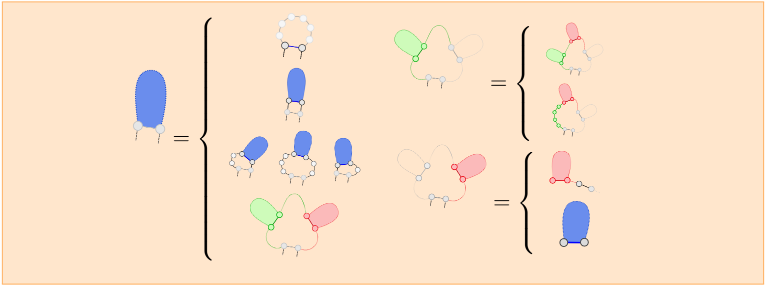
...or is it?

Combinatorics help in the design of DP schemes

Reminder: Generating function of secondary structures [Waterman1978]

$$S(z) := \sum_{n\geq 0} s_n\, z^n = \frac{1 - z + z^2 - \sqrt{1 - 2z - z^2 - 2z^3 + z^4}}{2z^2}$$

- DP scheme **unambiguous**;
- **Completeness** can be established by cardinality argument



Combinatorics help in the design of DP schemes

Reminder: Generating function of secondary structures [Waterman1978]

$$S(z) := \sum_{n\geq 0} s_n\, z^n = \frac{1 - z + z^2 - \sqrt{1 - 2z - z^2 - 2z^3 + z^4}}{2z^2}$$

- DP scheme **unambiguous**;
- **Completeness** can be established by cardinality argument

$$A(z) = \begin{cases} \mathrm{Seq}(z) \\ z^2\, A(z) \\ z\,\mathrm{Seq}(z)\, z^2\, A(z) + z^2\, A(z)\,\mathrm{Seq}(z)\, z \\ +z\,\mathrm{Seq}(z)\, z^2\, A(z)\,\mathrm{Seq}(z) z \\ B(z)\, C(z) \end{cases} \qquad B(z) = \begin{cases} B(z)\, C(z) \\ \mathrm{Seq}(z)\, B(z) \end{cases} \qquad C(z) = \begin{cases} C(z)\, z \\ z^2\, A(z) \end{cases}$$

$$\mathrm{Seq}(z) = 1 + z\,\mathrm{Seq}(z)$$

$$A(z) = \frac{1 - z - z^2 - \sqrt{1 - 2z - z^2 - 2z^3 + z^4}}{2z^2}$$

$$= W(z) - 1 \quad \text{(OMG! The \emph{empty} secondary structure is missing...)}$$