PROGRAMMATION DYNAMIQUE POLYMORPHE, VÉRIFIABLE ET EFFICACE

Mattéo Delabre

Laboratoire de biologie informatique et théorique (LBIT)
Département d'informatique et de recherche opérationnelle (DIRO)
Université de Montréal

16 février 2024

INTRODUCTION

- **Programmation dynamique**
  - Technique de conception d'algorithmes
  - Problèmes d'*optimisation combinatoire*
  - Large spectre d'applications
- **Perspective *algébrique***
  - *Structures* sous-jacentes à la programmation dynamique
  - Conception d'algorithmes par *composition* de structures
- **Avantages**
  - Réutilisation des composants pour différents problèmes
  - Tests et vérifications unitaires de chaque composant
  - Développement et présentation des algorithmes simplifiés

SOMMAIRE

ALIGNEMENT DE SÉQUENCES

- Prenons deux **séquences de symboles** A et B
  - *Lignes* de deux fichiers texte
  - *Nucléotides* de deux génomes
  - *Lettres* de deux mots
  - *Échantillons* de deux fichiers audio

Aligner A avec B
Trouver une suite d'opérations plausible transformant A en B

- **Opérations:** *Insertion*, *délétion*, ou *substitution* de symboles

APPLICATIONS

- **Quelques applications**
  - *Identifier* similarités et différences dans du code, des génomes, …
  - *Corriger* l'orthographe de mots en recherchant des mots proches
  - *Reconnaître* la parole dans un fichier audio

- Par exemple, **`diff`** aligne les lignes de deux révisions d'un fichier

```
1121    {
1122            struct nft_pipapo *priv = nft_set_priv(set);
1123    -       unsigned long *res, *fill, *scratch;
1124            u8 genmask = nft_genmask_cur(net);
1125            const u8 *rp = (const u8 *)key;
1126            struct nft_pipapo_match *m;
1127            struct nft_pipapo_field *f;
1128            bool map_index;
1129            int i, ret = 0;
```

```
1118    {
1119            struct nft_pipapo *priv = nft_set_priv(set);
1120    +       struct nft_pipapo_scratch *scratch;
1121            u8 genmask = nft_genmask_cur(net);
1122            const u8 *rp = (const u8 *)key;
1123            struct nft_pipapo_match *m;
1124            struct nft_pipapo_field *f;
1125    +       unsigned long *res, *fill;
1126            bool map_index;
1127            int i, ret = 0;
```

EXEMPLE

- Trois façons d'aligner $A = \mathbf{patient}$ avec $B = \mathbf{content}$:

	p	a	t	i	e	n	t
c	o	n	t		e	n	t

1 insertion/délétion
2 substitutions
4 correspondances

p	a	t	i	e	n	t
c	o	n	t	e	n	t

4 substitutions
3 correspondances

p	a			t	i	e	n	t
		c	o	n	t		e	n	t

6 insertions/délétion
4 correspondances

- **Laquelle choisir?** *Min. d'opérations, max. de correspondances, ...*

REPRÉSENTATION

Un alignement se représente en faisant **correspondre** les séquences…

	p	a	t	i	e	n	t
c	o	n	t		e	n	t

…ou en donnant la **suite d'opérations** associée

$$\mathrm{Ins}(\mathrm{c}) \circ \mathrm{Sub}(\mathrm{p},\mathrm{o}) \circ \mathrm{Sub}(\mathrm{a},\mathrm{n}) \circ \mathrm{Cor}(\mathrm{t}) \circ \mathrm{Dél}(\mathrm{i}) \circ \mathrm{Cor}(\mathrm{e}) \circ \mathrm{Cor}(\mathrm{n}) \circ \mathrm{Cor}(\mathrm{t})$$

FORMALISATION

- $\mathbf{align}(A, B)$: Ensemble des suites d'opérations alignant A avec B
- $\mathbf{score}(x)$: Nombre d'opérations dans une suite d'opérations x

$$\mathrm{score}(x) = \sum_{\mathrm{op} \in x} \begin{cases} 1 \text{ si } \mathrm{op} \in \{\mathrm{Ins}(\cdot), \mathrm{Dél}(\cdot), \mathrm{Sub}(\cdot, \cdot)\} \\ 0 \text{ si } \mathrm{op} = \mathrm{Cor}(\cdot) \end{cases}$$

- $\mathbf{select}(L)$: Coût minimum parmi toutes les suites d'un ensemble L

$$\mathrm{select}(L) = \min_{x \in L} \mathrm{score}(x)$$

PROBLÈME D'ALIGNEMENT

Quel est le nombre minimum d'opérations permettant d'aligner A avec B?

- Équivalent à calculer $\mathbf{select}(\mathbf{align}(A, B))$
- L'**espace de recherche** $\mathbf{align}(A, B)$ est de taille exponentielle!

SOMMAIRE

TOUS LES ALIGNEMENTS

$$\mathrm{align}(\mathrm{ab}, \mathrm{bc}) =$$

$\mathrm{Ins}(b)$	$\circ$	$\mathrm{Ins}(c)$	$\circ$	$\mathrm{Dél}(a)$	$\circ$	$\mathrm{Dél}(b)$
$\mathrm{Ins}(b)$	$\circ$	$\mathrm{Dél}(a)$	$\circ$	$\mathrm{Ins}(c)$	$\circ$	$\mathrm{Dél}(b)$
$\mathrm{Dél}(a)$	$\circ$	$\mathrm{Ins}(b)$	$\circ$	$\mathrm{Ins}(c)$	$\circ$	$\mathrm{Dél}(b)$
		$\mathrm{Sub}(a, b)$	$\circ$	$\mathrm{Ins}(c)$	$\circ$	$\mathrm{Dél}(b)$
		$\mathrm{Ins}(b)$	$\circ$	$\mathrm{Sub}(a, c)$	$\circ$	$\mathrm{Dél}(b)$
$\mathrm{Ins}(b)$	$\circ$	$\mathrm{Dél}(a)$	$\circ$	$\mathrm{Dél}(b)$	$\circ$	$\mathrm{Ins}(c)$
$\mathrm{Dél}(a)$	$\circ$	$\mathrm{Ins}(b)$	$\circ$	$\mathrm{Dél}(b)$	$\circ$	$\mathrm{Ins}(c)$
		$\mathrm{Sub}(a, b)$	$\circ$	$\mathrm{Dél}(b)$	$\circ$	$\mathrm{Ins}(c)$
$\mathrm{Dél}(a)$	$\circ$	$\mathrm{Dél}(b)$	$\circ$	$\mathrm{Ins}(b)$	$\circ$	$\mathrm{Ins}(c)$
		$\mathrm{Dél}(a)$	$\circ$	$\mathrm{Cor}(b)$	$\circ$	$\mathrm{Ins}(c)$
		$\mathrm{Ins}(b)$	$\circ$	$\mathrm{Dél}(a)$	$\circ$	$\mathrm{Sub}(b, c)$
		$\mathrm{Dél}(a)$	$\circ$	$\mathrm{Ins}(b)$	$\circ$	$\mathrm{Sub}(b, c)$
				$\mathrm{Sub}(a, b)$	$\circ$	$\mathrm{Sub}(b, c)$

TOUS LES ALIGNEMENTS

$$\mathrm{align}(\mathrm{ab}, \mathrm{bc}) =$$

$$
\begin{array}{l|llll}
\mathrm{align}(\mathrm{a}, \mathrm{bc}) & \mathrm{Ins}(b) & \mathrm{Ins}(c) & \mathrm{Dél}(a) & \mathrm{Dél}(b) \\
& \mathrm{Ins}(b) & \mathrm{Dél}(a) & \mathrm{Ins}(c) & \mathrm{Dél}(b) \\
& \mathrm{Dél}(a) & \mathrm{Ins}(b) & \mathrm{Ins}(c) & \mathrm{Dél}(b) \\
& & \mathrm{Sub}(a, b) & \mathrm{Ins}(c) & \mathrm{Dél}(b) \\
& & \mathrm{Ins}(b) & \mathrm{Sub}(a, c) & \mathrm{Dél}(b) \\
\hline
\mathrm{align}(\mathrm{ab}, \mathrm{b}) & \mathrm{Ins}(b) & \mathrm{Dél}(a) & \mathrm{Dél}(b) & \mathrm{Ins}(c) \\
& \mathrm{Dél}(a) & \mathrm{Ins}(b) & \mathrm{Dél}(b) & \mathrm{Ins}(c) \\
& & \mathrm{Sub}(a, b) & \mathrm{Dél}(b) & \mathrm{Ins}(c) \\
& \mathrm{Dél}(a) & \mathrm{Dél}(b) & \mathrm{Ins}(b) & \mathrm{Ins}(c) \\
& & \mathrm{Dél}(a) & \mathrm{Cor}(b) & \mathrm{Ins}(c) \\
\hline
\mathrm{align}(\mathrm{a}, \mathrm{b}) & \mathrm{Ins}(b) & \mathrm{Dél}(a) & \mathrm{Sub}(b, c) & \\
& \mathrm{Dél}(a) & \mathrm{Ins}(b) & \mathrm{Sub}(b, c) & \\
& & & \mathrm{Sub}(a, b) & \mathrm{Sub}(b, c)
\end{array}
$$

TOUS LES ALIGNEMENTS

- **Factorisation** en étendant $\circ$ pour s'appliquer à des ensembles

$$X \circ Y = \{x \circ y \mid x \in X, y \in Y\}$$

$$\begin{aligned}\mathrm{align}(\mathrm{ab}, \mathrm{bc}) = \\ \mathrm{align}(\mathrm{a}, \mathrm{bc}) &\circ \{\mathrm{Dél}(\mathrm{b})\} \\ \cup\, \mathrm{align}(\mathrm{ab}, \mathrm{b}) &\circ \{\mathrm{Ins}(\mathrm{c})\} \\ \cup\, \mathrm{align}(\mathrm{a}, \mathrm{b}) &\circ \{\mathrm{Sub}(\mathrm{b}, \mathrm{c})\}\end{aligned}$$

RÉCURRENCE SUR L'ESPACE DE RECHERCHE

- **Généralisation** de la factorisation pour deux chaînes A et B

$$A = (a_i)_1^{n+1} \qquad A' = (a_i)_1^{n}$$
$$B = (b_i)_1^{m+1} \qquad B' = (b_i)_1^{m}$$

$$\begin{aligned}\mathrm{align}(A,B) = {} & \mathrm{align}(A',B) \circ \{\mathrm{Dél}(a_n)\} \\ \cup\, & \mathrm{align}(A,B') \circ \{\mathrm{Ins}(b_m)\} \\ \cup\, & \mathrm{align}(A',B') \circ \begin{cases} \mathrm{Cor}(a_n) & \text{si } a_n = b_m \\ \mathrm{Sub}(a_n, b_m) & \text{sinon} \end{cases}\end{aligned}$$

RÉCURRENCE POUR « SELECT »

- **Remplacement** de $\cup$ et $\circ$ par $\min$ et $+$

$$\mathrm{select}(\mathrm{align}(A,B)) = \min \begin{cases} \mathrm{select}(\mathrm{align}(A',B)) & +1 \\ \mathrm{select}(\mathrm{align}(A,B')) & +1 \\ \mathrm{select}(\mathrm{align}(A',B')) + \begin{cases} 0 \text{ si } a_n = b_m \\ 1 \text{ sinon} \end{cases} \end{cases}$$

- Peut être calculée **efficacement** en $\Theta(nm)$
  - **Mémoïsation** des valeurs intermédiaires de $\mathrm{select}(\mathrm{align}(\cdot,\cdot))$

ALIGNEMENT EN PYTHON

```python
@cache
def align(a, b):
  if a == "" and b == "": return 0

  return min(
    align(a[:-1], b) + 1 if a else inf,
    align(a, b[:-1]) + 1 if b else inf,
    align(a[:-1], b[:-1]) + int(a[-1] != b[-1])
                          if a and b else inf,
  )

>>> align("patient", "content")
4
```

RECETTE DE PROGRAMMATION DYNAMIQUE

- **Espace de recherche récursif**
  - Division du problème en sous-problèmes (diviser pour régner)
  - Chevauchement des sous-problèmes
  - *Exemple:* Ensemble des alignements possibles
- **Fonction d'évaluation des candidats**
  - *Exemple:* $\mathbf{score}(x)$ donne une valeur à chaque alignement
- **Fonction de sélection des candidats**
  - *Exemple:* $\mathbf{select}(L)$ choisit la valeur minimale
- **Mémoïsation**
  - Conservation des calculs intermédiaires en mémoire

SOMMAIRE

MOTIVATION

Pourquoi est-il valide de remplacer
$\cup$ et $\circ$ par $\mathbf{min}$ et $+$
dans la récurrence?

DEMI-ANNEAUX

- **Demi-anneau:** Ensemble A muni de deux opérations $\oplus$ et $\otimes$ avec
  - $\oplus$ **associative et commutative avec élément neutre** 0_A

    $x \oplus (y \oplus z) = (x \oplus y) \oplus z$

    $x \oplus y = y \oplus x$

    $x \oplus 0_A = 0_A \oplus x = x$

DEMI-ANNEAUX

- **Demi-anneau:** Ensemble A muni de deux opérations $\oplus$ et $\otimes$ avec
  - $\oplus$ associative et commutative avec élément neutre 0_A
  - $\otimes$ **associative avec élément neutre 1_A et élément absorbant 0_A**

    $x \otimes (y \otimes z) = (x \otimes y) \otimes z$

    $x \otimes 1_A = 1_A \otimes x = x$

    $x \otimes 0_A = 0_A \otimes x = 0_A$

DEMI-ANNEAUX

- **Demi-anneau:** Ensemble A muni de deux opérations $\oplus$ et $\otimes$ avec
  - $\oplus$ associative et commutative avec élément neutre 0_A
  - $\otimes$ associative avec élément neutre 1_A et élément absorbant 0_A
  - $\otimes$ **distributive sur** $\oplus$

    $$x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$$
    $$(x \oplus y) \otimes z = (x \otimes z) \oplus (y \otimes z)$$

DEMI-ANNEAUX

- **Demi-anneau:** Ensemble A muni de deux opérations $\oplus$ et $\otimes$ avec
  - $\oplus$ associative et commutative avec élément neutre 0_A
  - $\otimes$ associative avec élément neutre 1_A et élément absorbant 0_A
  - $\otimes$ distributive sur $\oplus$
- Noté par $\langle A, \oplus, \otimes, 0_A, 1_A \rangle$

EXEMPLES DE DEMI-ANNEAUX

- **Naturels** avec addition et multiplication
  $\mathcal{C} = \langle \mathbb{N}, +, \times, 0, 1 \rangle$

EXEMPLES DE DEMI-ANNEAUX

- **Naturels** avec addition et multiplication
  $\mathcal{C} = \langle \mathbb{N}, +, \times, 0, 1 \rangle$
- **Tropical** avec minimum et addition
  $\mathcal{T}^{\min} = \langle \mathbb{N} \cup \{\infty\}, \min, +, \infty, 0 \rangle$

$$\min(x, \infty) = x \qquad \text{Neutralité de } \infty \text{ pour } \min$$

$$x + 0 = x \qquad \text{Neutralité de } 0 \text{ pour } +$$

$$x + \infty = \infty \qquad \text{Absorption de } + \text{ par } \infty$$

$$\min(x + y, x + z) = x + \min(y, z) \qquad \text{Distributivité de } + \text{ sur } \min$$

EXEMPLES DE DEMI-ANNEAUX

- **Naturels** avec addition et multiplication
  $\mathcal{C} = \langle \mathbb{N}, +, \times, 0, 1 \rangle$
- **Tropical** avec minimum et addition
  $\mathcal{T}^{\min} = \langle \mathbb{N} \cup \{\infty\}, \min, +, \infty, 0 \rangle$
- **Générateur** sur l'ensemble Λ des alignements possibles
  $\mathcal{G} = \langle \mathcal{P}(\Lambda), \cup, \circ, \emptyset, \{()\} \rangle$

$$x \cup \emptyset = x \qquad \text{Neutralité de } \emptyset \text{ pour } \cup$$

$$x \circ \{()\} = x \qquad \text{Neutralité de } \{()\} \text{ pour } \circ$$

$$x \circ \emptyset = \emptyset \qquad \text{Absorption de } \circ \text{ par } \emptyset$$

$$(x \circ y) \cup (x \circ z) = x \circ (y \cup z) \qquad \text{Distributivité de } \circ \text{ sur } \cup$$

HOMOMORPHISMES

- **Homomorphisme:** Fonction d'un demi-anneau $\mathcal{A}$ vers un autre $\mathcal{B}$ *préservant les opérations* de la structure

$$h : A \to B$$
$$h(x \oplus_A y) = h(x) \oplus_B h(y)$$
$$h(x \otimes_A y) = h(x) \otimes_B h(y)$$

- $\mathbf{select}$ est un homomorphisme de $\mathcal{G}$ vers $\mathcal{T}^{\min}$

$$\mathbf{select}(X \cup Y) = \mathbf{min}(\mathbf{select}(X), \mathbf{select}(Y))$$
$$\mathbf{select}(X \circ Y) = \mathbf{select}(X) + \mathbf{select}(Y)$$

DÉRIVATION DE LA RÉCURRENCE DE « SELECT »

$$
\begin{aligned}
\mathrm{align}(A, B) = \\
&\mathrm{align}(A', B) \circ \{\mathrm{Dél}(a_n)\} \\
\cup\ &\mathrm{align}(A, B') \circ \{\mathrm{Ins}(b_m)\} \\
\cup\ &\mathrm{align}(A', B') \circ \begin{Bmatrix} \mathrm{Cor}(a_n) & \text{si } a_n = b_m \\ \mathrm{Sub}(a_n, b_m) & \text{sinon} \end{Bmatrix}
\end{aligned}
$$

DÉRIVATION DE LA RÉCURRENCE DE « SELECT »

$$
\begin{aligned}
&\mathrm{select}(\mathrm{align}(A, B)) = \\
&\qquad \mathrm{select}\Bigg(\mathrm{align}(A', B) \ \circ \{\mathrm{Dél}(a_n)\} \\
&\qquad\qquad \cup \mathrm{align}(A, B') \ \circ \{\mathrm{Ins}(b_m)\} \\
&\qquad\qquad \cup \mathrm{align}(A', B') \circ \begin{Bmatrix} \mathrm{Cor}(a_n) & \text{si } a_n = b_m \\ \mathrm{Sub}(a_n, b_m) & \text{sinon} \end{Bmatrix} \Bigg)
\end{aligned}
$$

DÉRIVATION DE LA RÉCURRENCE DE « SELECT »

$$\mathrm{select}(\mathrm{align}(A,B)) =$$

$$\min\begin{cases}
\mathrm{select}(\mathrm{align}(A',B) & \circ \{\mathrm{Dél}(a_n)\}) \\
\mathrm{select}(\mathrm{align}(A,B') & \circ \{\mathrm{Ins}(b_m)\}) \\
\mathrm{select}\left(\mathrm{align}(A',B') \circ \begin{cases} \mathrm{Cor}(a_n) & \text{si } a_n = b_m \\ \mathrm{Sub}(a_n,b_m) & \text{sinon} \end{cases}\right)
\end{cases}$$

DÉRIVATION DE LA RÉCURRENCE DE « SELECT »

$$\mathrm{select}(\mathrm{align}(A,B)) =$$

$$\min\begin{cases} \mathrm{select}(\mathrm{align}(A',B)) & + \mathrm{score}(\mathrm{Dél}(a_n)) \\ \mathrm{select}(\mathrm{align}(A,B')) & + \mathrm{score}(\mathrm{Ins}(b_m)) \\ \mathrm{select}(\mathrm{align}(A',B')) + \mathrm{score}\begin{pmatrix} \mathrm{Cor}(a_n) & \text{si } a_n{=}b_m \\ \mathrm{Sub}(a_n,b_m) & \text{sinon} \end{pmatrix} \end{cases}$$

DÉRIVATION DE LA RÉCURRENCE DE « SELECT »

$$\mathrm{select}(\mathrm{align}(A,B)) =$$

$$\min \begin{cases} \mathrm{score}(\mathrm{align}(A',B)) & +1 \\ \mathrm{score}(\mathrm{align}(A,B')) & +1 \\ \mathrm{score}(\mathrm{align}(A',B')) + \begin{cases} 0 \text{ si } a_n = b_m \\ 1 \text{ sinon} \end{cases} & \end{cases}$$

POLYMORPHISME

- N'importe quel demi-anneau peut-être utilisé dans la récurrence à la place du générateur $\mathcal{G}$ → **polymorphisme**
- Chaque demi-anneau permet de résoudre **un problème différent** sur le même espace de recherche décrit par la récurrence

---

M. A. Little et U. Kayas, *Polymorphic dynamic programming by algebraic shortcut fusion,* juillet 2021 (arXiv).

ALIGNEMENT GÉNÉRIQUE EN PYTHON

```python
@cache
def align(a, b, struct, h):
  if a == "" and b == "": return struct.one()

  return sum((
    align(a[:-1], b, struct, h) * h(("del", a[-1]))
      if a else struct.zero(),

    align(a, b[:-1], struct, h) * h(("ins", b[-1]))
      if b else struct.zero(),

    align(a[:-1], b[:-1], struct, h) *
      (h(("match", a[-1])) if a[-1] == b[-1]
       else h(("sub", a[-1], b[-1])))
      if a and b else struct.zero(),
  ), start=struct.zero())
```

SOMMAIRE

TROPICAUX

- **Utilisation:** Calculer le coût d'une solution optimale

$$\mathcal{T}^{\min} = \langle \mathbb{R} \cup \{+\infty\}, \min, +, +\infty, 0 \rangle$$
$$\mathcal{T}^{\max} = \langle \mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0 \rangle$$

TROPICAL EN PYTHON

```python
@dataclass
class MinPlus:
  value: int

  @classmethod
  def zero(cls): return MinPlus(inf)

  @classmethod
  def one(cls): return MinPlus(0)

  def __add__(self, other):
    return MinPlus(min(self.value, other.value))

  def __mul__(self, other):
    return MinPlus(self.value + other.value)
```

Coût minimum d'un alignement

- Il suffit de donner le **demi-anneau** et l'**homomorphisme** appropriés à la fonction d'alignement générique

```
1  def hom_minplus(op):
2    # Chaque opération coûte 1, sauf les matchs
3    if op[0] == "match": return MinPlus(0)
4    else: return MinPlus(1)
5
6  >>> align("patient", "content", MinPlus, hom_minplus)
7  MinPlus(value=4)
```

COMPTEUR

- **Utilisation:** Compter le nombre d'éléments dans l'espace

$$\mathcal{C} = \langle \mathbb{N}, +, \times, 0, 1 \rangle$$

COMPTEUR EN PYTHON

```python
@dataclass
class Counter:
  value: int

  @classmethod
  def zero(cls): return Counter(0)

  @classmethod
  def one(cls): return Counter(1)

  def __add__(self, other):
    return Counter(self.value + other.value)

  def __mul__(self, other):
    return Counter(self.value * other.value)
```

Nombre d'alignements

- **Pas de changement dans l'algorithme**, il suffit de remplacer le demi-anneau et l'homomorphisme

```
1  def hom_counter(_): return Counter(1)
2
3  >>> align("ab", "bc", Counter, hom_counter)
4  Counter(value=13)
```

- Compte le nombre (exponentiel) d'alignements, sans les lister (fonctionne en **temps polynomial**!)

GÉNÉRATEUR

- **Utilisation:** Lister tous les éléments de l'espace de recherche

$$\mathcal{G} = \langle \mathcal{P}(\Lambda), \cup, \circ, \emptyset, \{()\} \rangle$$

GÉNÉRATEUR EN PYTHON

```python
@dataclass
class Generator:
  value: set[tuple]

  @classmethod
  def zero(cls): return Generator(set())

  @classmethod
  def one(cls): return Generator({()})

  def __add__(self, other):
    return Generator(self.value | other.value)

  def __mul__(self, other):
    return Generator({x + y for x in self.value
                            for y in other.value})
```

TOUS LES ALIGNEMENTS

```
def hom_generator(op):
  return Generator({(op,)})

>>> align("ab", "bc", Generator, hom_generator).value
{(('ins', 'b'), ('ins', 'c'), ('del', 'a'), ('del', 'b')),
 (('ins', 'b'), ('del', 'a'), ('ins', 'c'), ('del', 'b')),
 (('del', 'a'), ('ins', 'b'), ('ins', 'c'), ('del', 'b')),
 (('sub', 'a', 'b'), ('ins', 'c'), ('del', 'b')),
 (('ins', 'b'), ('sub', 'a', 'c'), ('del', 'b')),
 (('ins', 'b'), ('del', 'a'), ('del', 'b'), ('ins', 'c')),
 (('del', 'a'), ('ins', 'b'), ('del', 'b'), ('ins', 'c')),
 (('sub', 'a', 'b'), ('del', 'b'), ('ins', 'c')),
 (('del', 'a'), ('del', 'b'), ('ins', 'b'), ('ins', 'c')),
 (('del', 'a'), ('match', 'b'), ('ins', 'c')),
 (('ins', 'b'), ('del', 'a'), ('sub', 'b', 'c')),
 (('del', 'a'), ('ins', 'b'), ('sub', 'b', 'c')),
 (('sub', 'a', 'b'), ('sub', 'b', 'c'))}
```

k-OPTIMAUX

- **Utilisation:** Calculer les k meilleures solutions

$$\mathcal{B}^k = \langle \mathbb{R}^k, \oplus_B, \otimes_B, 0_B, 1_B \rangle$$

$$\begin{aligned} 0_B &= (\infty, \infty, \infty, ...) \\ 1_B &= (0, \infty, \infty, ...) \\ X \oplus_N Y &= \min^k(X \cup Y) \\ X \otimes_N Y &= \min^k\{x + y \mid x \in X, y \in Y\} \end{aligned}$$

L. Huang, *Advanced dynamic programming in semiring and hypergraph frameworks,* août 2008 (Coling)

k-OPTIMAUX EN PYTHON

```python
k = 5

def select(items):
  return sorted(items)[:k] + [inf] * (k - len(items))

@dataclass
class KBest:
  value: list[int]

  @classmethod
  def zero(cls): return KBest([inf] * k)

  @classmethod
  def one(cls): return KBest([0] + [inf] * (k - 1))

  def __add__(self, other):
    return KBest(select(self.value + other.value))

  def __mul__(self, other):
    return KBest(select([x + y for x in self.value
                               for y in other.value]))
```

k-MEILLEURS ALIGNEMENTS

```
1  def hom_kbest(op):
2    if op[0] == "match": return KBest([0])
3    else: return KBest([1])
4
5  >>> align("ab", "bc", KBest, hom_kbest)
6  KBest(value=[2, 2, 3, 3, 3]))
```

MULTI-OBJECTIFS (PARETO)

- **Utilisation:** Optimiser simultanément plusieurs critères
  - Représentés par les entrées d'un vecteur à k dimensions
- Un vecteur $a \in X \subseteq \mathbb{R}^k$ est **Pareto-optimal** si aucun autre $b \in X$ n'est inférieur ou égal à lui sur toutes ses coordonnées
  - $X = \{(3, 1), (2, 3), (1, 10)\}$ sont tous Pareto-optimaux
  - $(3, 2)$ n'est pas Pareto-optimal par rapport à X
- $\mathrm{par}(X)$: Sous-ensemble Pareto-optimal maximal de X

MULTI-OBJECTIFS (DEMI-ANNEAU)

$$\mathcal{Q}_k^{\min} = \langle \mathbb{R}^k, \oplus_Q, \otimes_Q, 0_Q, 1_Q \} \rangle$$

$$0_Q = \emptyset$$

$$1_Q = \{(0, 0, 0, ...)\}$$

$$X \oplus_Q Y = \mathrm{par}(X \cup Y)$$

$$X \otimes_Q Y = \mathrm{par}(\{x + y \mid x \in X, y \in Y\})$$

C. Saule et R. Giegerich, *Pareto optimization in algebraic dynamic programming,* décembre 2015 (Algorithms for Molecular Biology)

DEMI-ANNEAU DE PARETO EN PYTHON

```python
k = 3

@dataclass
class Pareto:
  value: set[tuple[int]]

  @classmethod
  def zero(cls): return Pareto(set())

  @classmethod
  def one(cls): return Pareto({(0,) * k})

  def __add__(self, other):
    return Pareto(select(self.value | other.value))

  def __mul__(self, other):
    return Pareto(select({add(x, y) for x in self.value
                                    for y in other.value}))
```

ALIGNEMENTS PARETO-OPTIMAUX

- Précédemment, les alignements étaient évalués sur le **nombre total** d'insertions, délétions et substitutions
- Ces trois critères sont maintenant optimisés **séparément**

```
1  def hom_pareto(op):
2    if op[0] == "ins": return Pareto({(1, 0, 0)})
3    elif op[0] == "del": return Pareto({(0, 1, 0)})
4    elif op[0] == "sub": return Pareto({(0, 0, 1)})
5    else: return Pareto({(0, 0, 0)})
6
7  >>> align("ab", "bc", Pareto, hom_pareto)
8  Pareto(value={(1, 1, 0), (0, 0, 2)})
```

COMBINAISON DE DEMI-ANNEAUX

- Comment obtenir la **suite d'opérations** des meilleurs alignements, et pas uniquement leur score?
- On veut **combiner** l'effet des demi-anneaux suivants
  - *Tropical* $\mathcal{T}^{\min}$: Calcule le meilleur score d'un alignement
  - *Générateur* $\mathcal{G}$: Donne les suites d'opérations des alignements

PRODUIT-SÉLECTION

- Le **produit-sélection** de $\mathcal{A} = \langle A, \oplus_A, \otimes_A, 0_A, 1_A \rangle$ par $\mathcal{B} = \langle B, \oplus_B, \otimes_B, 0_B, 1_B \rangle$ est

$$\mathcal{A} * \mathcal{B} = \langle A \times B, \oplus_{A*B}, \otimes_{A*B}, (0_A, 0_B), (1_A, 1_B) \rangle$$

$$(a, b) \otimes_{A*B} (a', b') = (a \otimes_A a', b \otimes_B b')$$

$$(a, b) \oplus_{A*B} (a', b') = \begin{cases} (a, b \oplus_B b') & \text{si } a = a' \\ (a, b) & \text{si } a < a' \\ (a', b') & \text{sinon} \end{cases}$$

(En supposant une relation d'ordre $\leq$ sur $\mathcal{A}$)

P. Steffen et R. Giegerich, *Versatile and declarative dynamic programming using pair algebras,* décembre 2005 (BMC Bioinformatics)

PRODUIT-SÉLECTION EN PYTHON

- **Exemple:** $\mathcal{T}^{\min} * \mathcal{G}$ donne une paire (c, x) où
  - c est le coût d'un alignement optimal
  - x contient les suites d'opérations des alignements **ayant ce coût**
- Le produit-sélection peut être implémenté de façon **générique** pour combiner n'importe quels demi-anneaux

```
>>> align("patient", "content", MinPlus * Generator,
           hom_minplus * hom_generator).value
(2,
 {(('del', 'a'), ('match', 'b'), ('ins' , 'c')),
  (('sub', 'a', 'b'), ('sub', 'b', 'c'))})
```

PRODUIT DIRECT

- Le **produit direct** de $\mathcal{A}$ et $\mathcal{B}$ est

$$\mathcal{A} \times \mathcal{B} = \langle A \times B, \oplus_{A\times B}, \otimes_{A\times B}, (0_A, 0_B), (1_A, 1_B)\rangle$$
$$(a, b) \oplus_{A\times B} (a', b') = (a \oplus_A a', b \oplus_B b')$$
$$(a, b) \otimes_{A\times B} (a', b') = (a \otimes_A a', b \otimes_B b')$$

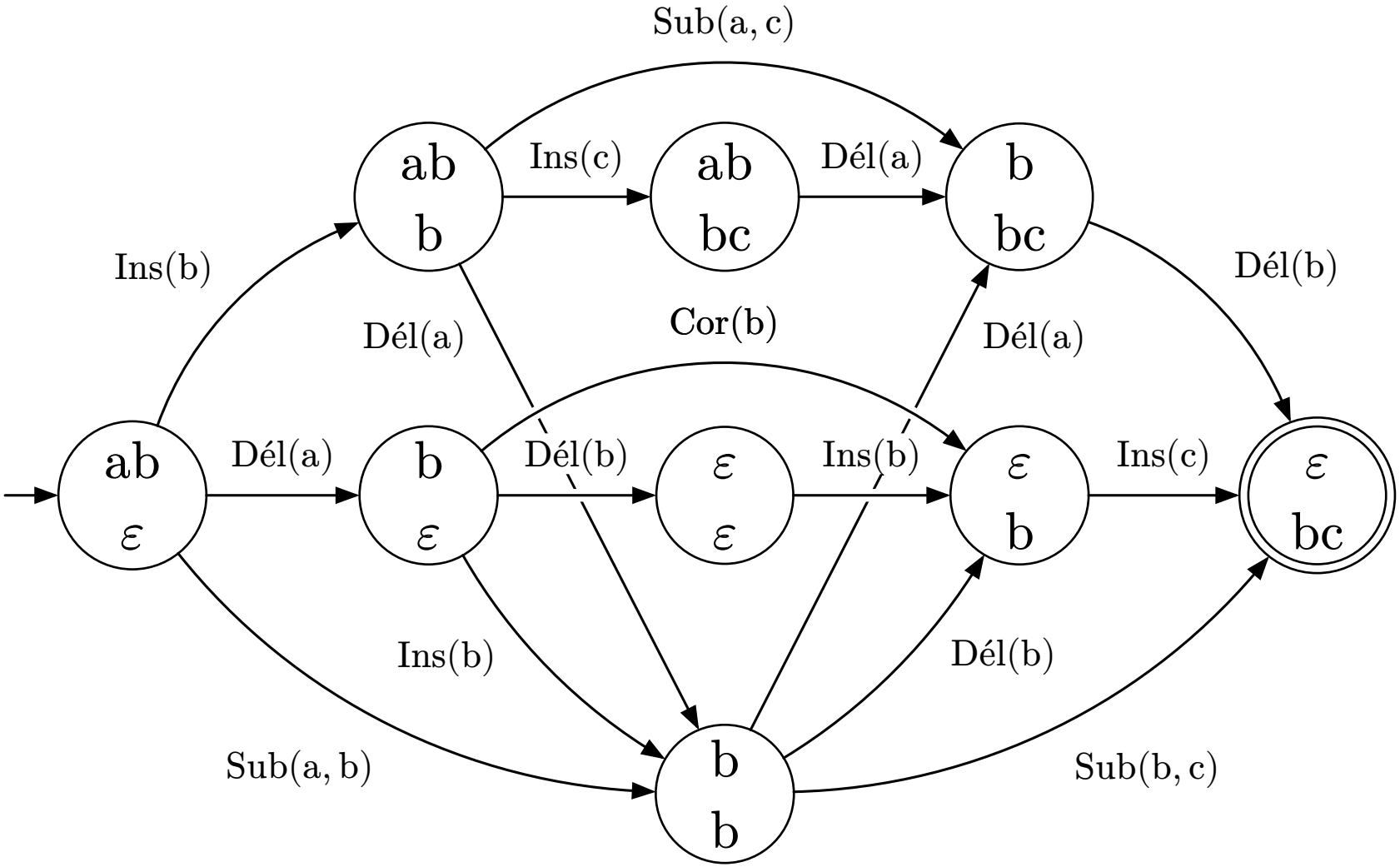
- **Exemple:** $\mathcal{T}^{\min} * (\mathcal{G} \times \mathcal{C})$ trouve un alignement de coût minimum et le nombre d'alignements de coût minimum

SOMMAIRE

GRAPHE DES ALIGNEMENTS

- L'ensemble des alignements possibles de A avec B correspond aux chemins dans un **graphe dirigé acyclique**
  - **Sommets:** États de l'alignement (X, Y)
  - **Arcs:** Opérations $\mathrm{Dél}(\cdot)$, $\mathrm{Ins}(\cdot)$, $\mathrm{Sub}(\cdot, \cdot)$ ou $\mathrm{Cor}(\cdot)$
- Meilleur alignement $\equiv$ **Plus court chemin** dans le graphe

GRAPHE DES ALIGNEMENTS



PROBLÈMES SÉQUENTIELS

- Beaucoup de problèmes de programmation dynamique sont des calculs sur les **chemins** d'un graphe → **problèmes « séquentiels »**
  - *Alignement de séquences*
  - *Plus longue sous-séquence commune*
  - *Plus courts chemins*
  - *Rendu de monnaie*
  - *Décodage d'un modèle de Markov caché*
- **Problème du chemin algébrique:** Calcul d'un *homomorphisme* h vers un *demi-anneau* $\mathcal{A}$ sur l'ensemble des *chemins* d'un graphe G

ALGORITHMES D'EXPLORATION

- Si G est **acyclique** → parcours topologique
  - Généralisation de l'alignement de séquences
  - Temps de calcul $\mathcal{O}(V + E)$

L. Huang, *Advanced dynamic programming in semiring and hypergraph frameworks,* août 2008 (Coling)

ALGORITHMES D'EXPLORATION

- Si G est **acyclique** → parcours topologique
  - Généralisation de l'alignement de séquences
  - Temps de calcul $\mathcal{O}(V + E)$
- Si G n'est **pas acyclique** → algorithme de Dijkstra
  - Requiert que $\mathcal{A}$ soit *supérieur* (~ pas d'arcs négatifs)

    $\forall a, b \in A, a \leq a \otimes b$
  - Temps de calcul $\mathcal{O}(V \log V + E)$

L. Huang, *Advanced dynamic programming in semiring and hypergraph frameworks*, août 2008 (Coling)

ALGORITHMES D'EXPLORATION

- Si G est **acyclique** → parcours topologique
  - Généralisation de l'alignement de séquences
  - Temps de calcul $\mathcal{O}(V + E)$
- Si G n'est **pas acyclique** → algorithme de Dijkstra
  - Requiert que $\mathcal{A}$ soit *supérieur* (~ pas d'arcs négatifs)

    $\forall a, b \in A, a \leq a \otimes b$
  - Temps de calcul $\mathcal{O}(V \log V + E)$
- Si $\mathcal{A}$ n'est **pas supérieur** → algorithme de Bellman-Ford
  - Temps de calcul $\mathcal{O}(VE)$

L. Huang, *Advanced dynamic programming in semiring and hypergraph frameworks,* août 2008 (Coling)

PROBLÈMES ARBORESCENTS

- Dans certains problèmes, une action (arc) peut provenir de plusieurs états (sommets) → **problèmes « arborescents »**
  - *Analyse syntaxique avec grammaire hors-contexte (CYK)*
  - *Produit chaîné de matrices*
  - *Repliement de l'ARN en structures secondaires*
  - *Petite parsimonie de Sankoff*
- Le graphe devient alors un **hypergraphe** H, dont on fait des calculs sur les **hyperchemins** (qui sont des arbres)

ALGORITHMES D'EXPLORATION (HYPERGRAPHES)

- Si H est **acyclique** → parcours topologique
  - Généralisation de l'algorithme CYK
  - Temps de calcul $\mathcal{O}(V + E)$
- Si H n'est **pas acyclique** → algorithme de Knuth
  - Requiert que $\mathcal{A}$ soit *supérieur*
  - Temps de calcul $\mathcal{O}(V \log V + E)$

D. E. Knuth, *A generalization of Dijkstra's algorithm,* février 1977 (Information Processing Letters)

DESCRIPTION DE L'ESPACE DE RECHERCHE

- Le graphe implicite de l'espace de recherche peut être **spécifié manuellement** dans l'algorithme
- Demande d'adapter l'algorithme d'exploration à chaque problème
- **Génération automatique** du graphe à partir de règles
  - Grammaires à récolte (*yield grammar*)

- R. Giegerich, G. Sauthoff, *Yield grammar analysis in the Bellman's GAP compiler,* mars 2011 (LDTA)
- S. Berkemer, C. H. Siederdissen et P. Stadler, *Algebraic dynamic programming on trees,* décembre 2017 (Algorithms)

SOMMAIRE

EN BREF

- La **programmation dynamique** permet de résoudre efficacement certains problèmes d'**optimisation combinatoire**
- L'approche algébrique permet une séparation de:
  - l'algorithme d'**exploration** de l'espace de recherche
    → *topologique, Dijkstra, Bellman-Ford, Knuth*
  - la **spécification** de l'espace de recherche
    → *via des grammaires*
  - l'**évaluation** de ses éléments
    → *demi-anneau et homomorphisme*

AVANTAGES ET LIMITES

- Les trois composants sont interchangeables (**polymorphisme**)
  - Tests et vérification individuels
  - Réutilisation dans plusieurs problèmes
- Certaines combinaisons d'espaces de recherche et de demi-anneaux donnent lieu à des **optimisations** manuelles
  - Reste à déterminer comment automatiser ces optimisations

Z. Galil et K. Park, *A linear-time algorithm for concave one-dimensional dynamic programming,* février 1990 (Information Processing Letters)

IMPLÉMENTATIONS

- ADP Compiler
- ADPfusion
- adp-multi
- Bellman's GAP