

1 Немного про грамматики

Предполагаемую структуру свёртки можно описать КС-грамматикой. При этом можно использовать более выразительные конструкции, чем нормальная форма Хомского (НФХ). В результате можно пытаться описать именно вторичную структуру, а не содержимое конкретной цепочки. При этом возможно как уточнение содержимого, так и ослабление структуры.

Конструкции, расширяющие НФХ, доступные для описания вторичной структуры.

- **Регулярные выражения.** Доступны операции $\{+, *, |, ?\}$. Например, любой из символов $\{A, U, G, C\}$, повторённый не менее 1 раза.

`any : (A | U | G | C)+`

- **Повторения** позволяют ограничивать количество однотипных элементов (в отличие от $+$). Например, повторение `any` от 2 до 5 раз включительно:

`s : any*[2..5]`

- **Метаправила** позволяют описывать параметризуемые шаблоны вывода. Например, шаблон вывода списка, параметризованный элементом списка и разделителем.

`not_empty_list<item sep>: item (sep item)*`

Применение описанного выше шаблона. Фактическими аргументами метаправила могут быть произвольные конструкции: регулярные выражения, другие метаправила.

`s1: not_empty_list<NUM COLON>`
`s2: not_empty_list<s1 (DOT|COMMA)>`

Возможный пример описания свёртки для tRNA.

```
[<Start>]
folded: stem<(any*[1..3]
            stem<any*[4..10]>
            any*[1..3]
            stem<any*[4..7]>
            any*[3..5]
            stem<any*[4..7]>
            )>
```

```
stem<s>:
    A stem<s> U
    | U stem<s> A
    | C stem<s> G
    | G stem<s> C
    | G stem<s> U
    | U stem<s> G
    | s
```

```
any: A | U | G | C
```

Можно добавить вероятности для фильтрации деревьев. Применимы те же фильтры, что и в работах на основе CYK.

```
[<Start>]
folded: stem<(any*[1..3]
            stem<any*[4..10]>
            any*[1..3]
            stem<any*[4..7]>
            any*[3..5]
            stem<any*[4..7]>
            )>
```

```
stem<s>:
    [p1] (A stem<s> U)
    | [p1] (U stem<s> A)
    | [p1] (C stem<s> G)
    | [p1] (G stem<s> C)
    | [p2] (G stem<s> U)
    | [p2] (U stem<s> G)
    | [p3] (s)
```

```
a1: [p1] (A)
    | [p1] (U)
    | [p1] (G)
    | [p1] (C)
```

Грамматика может быть модульной. Это позволяет описать некоторые базовые элементы отдельно и далее использовать, подключая соответствующие модули/модуль. Например, в переиспользуемый модуль можно вынести определение для `stem`, `any`, `R`, `Y` и другие элементы.

Опсле работы алгоритма доступны все деревья разбора, соответствующие хаданной грамматике, а значит и вторичной структуре. Далее, эти деревья сожно обрабатывать отдельно. Например, отбирать более вероятные используя различные фильтры, метрики, шаблоны.

Что можно ещё интересного делать.

- Попытаться восстанавливать грамматики из известных структур свёрток, а не описывать руками [1].
- Восстанавливать деревья по известным свёрткам.
- Использовать структурные фильтры для деревьев.

2 Технические детали

Технически, задача достаточно хорошо масштабируемая. Возможно использование смешанных вычислений: CPU + GPGPU. В качестве алгоритма можно применять не только СУК, но и другие алгоритмы, которые могут оказаться более эффективными.

На Infernal я посмотрел. Внутри жуть что происходит. Код хоть и снабжён комментариями, но крайне плохо написан. Я не смог за 2 дня придумать, как быстро его передизайнить, чтобы он работал с графом, хотя для этого требуется просто другая раскладка матрицы, используемой в СУК. Что при аккуратном дизайне не должно вызывать проблем.

Список литературы

- [1] Dowell R. D., Eddy S. R. Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction //BMC bioinformatics. — 2004. — Т. 5. — №. 1. — С. 1.
- [2] Eddy S. R. A memory-efficient dynamic programming algorithm for optimal alignment of a sequence to an RNA secondary structure //BMC bioinformatics. — 2002. — Т. 3. — №. 1. — С. 1.