1 E.A.3.3 (ProteinFolding)

1.1 Modello

Una proteina è una sequenza di amminoacidi $a_1, ..., a_n \;\; t.c. \;\; a_i \in \{H, P\}$.

Def. 1 conformazione (stato)

Una conformazione è un assegnamento $p_1, .., p_k$ con $1 \leq k \leq n \;\; t.c.$

- $p_i \in \mathbb{Z}^2$
- $\text{dist}(p_i, p_{i+1}) = 1$ *(adiacenza)*
- $\nexists p_i, p_j \;\; i \neq j \wedge p_i = p_j$ *(non sovrapposizione)*

Una conformazione è detta *parziale* se $k < n$, *completa* altrimenti (una conformazione completa è considerata uno stato obiettivo per la ricerca).

Def. 2 modello di transizione

Data una conformazione $p_1, ..., p_k$, l'insieme delle azioni A_k^p è definito come segue: $A_k^p = \varnothing$ se $k = n$, altrimenti se $k < n$

$$
\begin{aligned}
A_k^p = \{&(x, y) \mid \\
&(x, y) \in \mathbb{Z}^2 \wedge \\
&\exists x_k, y_k \\
&\quad p_k = (x_k, y_k) \wedge \\
&\quad (\\
&\qquad ((x = x_k + 1 \vee x = x_k - 1) \wedge y = y_k) \vee \\
&\qquad ((y = y_k + 1 \vee y = y_k - 1) \wedge x = x_k) \\
&\quad) \wedge \\
&\quad \nexists p_i \;\; p_i = (x, y) \\
\}&
\end{aligned}
\tag{1}
$$

Dato uno stato $p_1, ..., p_k$ e data un'azione $\alpha_{k+1} \in A_k^p$, lo stato generato è $p'_1, ..., p'_{k+1}$ t.c.

$$
p'_i = \begin{cases} p_i & \text{se } i \leq k \\ \alpha_{k+1} & \text{se } i = k + 1 \end{cases} \tag{2}
$$

(Tecnicamente dovrei dimostrare che $p'_1, ..., p'_{k+1}$ rispetta l'adiacenza e la non sovrapposizione. L'adiacenza vale dato che una sola delle due coordiante cambia, e cambia esattamente di 1, quindi la distanza euclidea fra p_{k+1} e p_k è esattamente 1. La non sovrapposizione è garantita dal fatto che non esiste nessun $i \neq k + 1$ t.c. $p_i = (x, y)$)

Si considera che, indipendentemente dalla percezione, lo stato iniziale è $p_1 = (0, 0)$ (per evitare conformazioni equivalenti ma traslate).

Da questa modellazione del problema deriva un'importante osservazione, che permette di ottimizzare gli algoritmi di ricerca:

Oss. 1

Il grafo di ricerca per **ProteinFolding** è un **albero**.

Dim. Per dimostrarlo bisogna far vedere che non è possibile raggiungere lo stesso stato con sequenze di azioni diverse. Per assurdo, si supponga che lo stato $p_1, ..., p_k$ sia stato raggiunto con due sequenze di azioni $\alpha_1, ..., \alpha_k$ e $\beta_1, ..., \beta_k$ diverse, per cui esiste i t.c. $\alpha_i \neq \beta_i$, quindi, per il modello di transizione, $p_i = \alpha_i \neq \beta_i = p_i \Rightarrow p_i \neq p_i$, contraddizione. *(forse viene meglio per induzione su k, ma devo impostare meglio cosa voglio dimostrare, inoltre credo che dovrei usare in qualche modo la definizione di azione)*

L'**Oss. 1** permette di ottimizzare la ricerca, perché se dopo una transizione ogni stato è garantito nuovo, non serve controllare se questo è già stato esplorato o sta già in frontiera.

Dato che il problema presenta forti **simmetrie**, si può ridurre lo spazio di ricerca evitando le azioni che generano conformazioni simmetriche o ruotate, permettendo solo le seguenti azioni:

1. $A_1^p = \{(0, 1)\}$ (rotazioni)
2. per $p_1, ..., p_k$ con $k > 1$
   1. se $\forall j \;\; j \leq k \;\; \exists y_j \;\; p_j = \big(0, y_j\big)$ (la conformazione non ha nessuna piega), allora $A_k^p = \{(0, y_k + 1), (1, y_k)\}$
   2. altrimenti A_k^p è la stessa in **Def. 2**

Oss. 2

Nel caso 2.1 le nuove azioni garantiscono la non sovrapposizione. In particolare:

- per l'azione $(1, y_k)$ la garanzia deriva dal fatto che la conformazione ha tutte $x = 0$
- per l'azione $(0, y_k + 1)$ l'argomentazione è un po' più complicata, ma sostanzialmente bisogna usare l'*adiacenza* della conformazione per far vedere che non è possibile che ci sia un amminoacido in quella posizione (se tutte le x sono 0), perché altrimenti si avrebbero due amminoacidi adiacenti nella sequenza ma con distanza 2 nella conformazione (bisognerebbe anche far vedere che $y_{(k+1}$ non torna indietro, quindi che la conformazione è formata da $y_i \geq 0$

Questo è particolarmente vantaggioso, perché, se la non sovrapposizione è garantita di base, non serve controllare se le azioni la rispettano (un controllo che normalmente è in $O(k)$, qui sto provando a spremere la performance in tutti i modi possibili)

Tecnicamente bisognerebbe dimostrare che le nuove definizioni per A_k^p rispettano tutte il modello di transizione, quindi che **Oss. 1** è ancora valida:

- per il caso 1., dato che $p_1 = (0, 0)$ per **Def. 2**, l'azione $(0, 1)$ è proprio una delle 4 azioni possibili
- per il caso 2.1 la non sovrapposizione è stata dimostrata sopra, e vale anche l'adiacenza (al massimo cambia una sola delle due coordinate, e cambia esattamente di 1)
- per il caso 2.2. si usa proprio **Def. 2**, quindi la situazione non cambia

Oss. 3

Usando le regole sopra non si generano conformazioni simmetriche o ruotate *(intuitivamente regge, ma è un po' più ostica da dimostrare).*

Dim. Per induzione su k

- $k = 1$: ...
- $k = 2$: ...

Passo induttivo

1. se non c'è una piega
   - allora la conformazione è dritta, dato che la conformazione precedente non ha simmetrie o rotazioni, non si può generare roba simmetrica andando avanti, e facendo il giro a destra non c'è roba ruotata o simmetrica
2. se c'è una piega
   - usare in qualche modo modo l'ipotesi induttiva per far vedere che partendo da una sequenza senza simmetrie le nuove azioni non possono generare stati simmetrici

1.2 Euristica

Due parole veloci sul come l'idea di fondo è che proteine H vicine nella sequenza devono stare vicine nella proteina finale, o, in generale gli stati più «promettenti» sono quelli che hanno «meno contatti non realizzati» per ora.

Def. 3 Costo

Tocca dare ad un certo punto la definizione di costo. Ad alto livello il costo è dato dal «numero di contatti H non realizzati», e l'obiettivo è minimizzarlo.

Per ora ho provato diverse euristiche, ma per nessuna era perfettamente chiaro il funzionamento (anche se i risultati erano molto promettenti, queste euristiche misuravano parametri che non erano legati

a come viene effettivamente misurato il costo), e non era chiaro se fosse garantita la consistenza.

Dopo un po' di riflessione queste sono alcune idee per un'euristica che sulla carta dovrebbe funzionare benissimo (modulo i dettagli implementativi)

- solo l'amminoacido finale e quello iniziale possono avere 3 contatti, un amminoacido H in mezzo ne può avere al massimo 2 (devo capire bene perché, ma questa osservazione velocizza i tempi in un modo assurdo, forse per come vengono influenzati i costi… devo sempre verificare se è corretta; inoltre questa osservazione non è legata all'euristica, influenza direttamente il costo)
- data una conformazione serve un modo veloce per poter calcolare, per tutti gli amminoacidi H non assegnati, il numero di contatti che non verranno sicuramente realizzati, ma che altrimenti sarebbero stati possibili (non quelli dovuti a causa delle distanze, ad esempio per la proteina H P P P H non è possibile avere un contatto a prescindere… dovrei dimostrare che per poter avere un contatto il numero di amminoacidi in mezzo deve essere pari)
- sarebbe interessante vedere se questo numero è memorizzabile all'interno dello stato, e se c'è un modo per aggiornarlo ad ogni transizione, senza doverlo ricalcolare da capo (velocizzerebbe di molto i calcoli)

Oss. 4 Ammissibilità

Oss. 5 Consistenza

Oss. 6 Critical ratio

Per la critical ration il discorso è ancora un po' fumoso: ho provato diverse esecuzioni sul cluster, in un caso fissando n e generando le H in modo casuale (anche il numero di H è deciso in modo uniforme)… ma i risultati non erano consistenti (c'erano diversi picchi e ogni volta cambiavano per numero di H diversi). Quindi ipotizzo che la critical ratio non dipenda solo dal rapporto fra n e il numero di H.

Provando con H tutte adiacenti, invece, si vede come per poche H c'è un solo picco, ed è sempre nello stesso posto. Ma il problema è che queste H sono tutte adiacenti, e non sono rappresentative del caso medio.

Servirebbe provare a trovare un valore che dipenda non solo dal numero di H, ma anche dalla loro disposizione.

1.3 Possibili miglioramenti

- levare l'`Rc<AminoAcid>` (il reference counter), ma servirebbe modificare l'interfaccia per il `Problem`. L'obiettivo sarebbe possibilmente usare un `usize` per riferirsi allo stato.

```
pub struct AminoAcid {
    pos: Pos,
    prev: Option<Rc<AminoAcid>>,
    depth: usize,
    first_turn: bool,
}
```

- verificare in modo più intelligente la non sovrapposizione di un'azione senza dover scorrere tutta la proteina
- alternativamente, trovare una conformazione in memoria per gli stati in modo da tenere vicini gli stati visti più di frequente, e quelli «inutili» lasciarli in fondo
- trovare un modo per eliminare dalla memoria gli stati una volta che sono stati esplorati: ad ogni esplorazione verrebbe eliminato uno stato, e, nel caso peggiore, ne verrebbero aggiunti 3