Algorytm rozwiązujący problem sekwencjonowania przez hybrydyzację

Marceli Jerzyński 136725
Korneliusz Szymański 136813

24 czerwca 2020

1 Przypomnienie

Problem sekwencjonowania przez hybrydyzację został sprowadzony do problemu komiwojażera. Postanowiliśmy rozwiązać go za pomocą algorytmu mrówkowego zaimplementowanego w języku Kotlin

2 Heurystyka pierwszego wierzchołka

W pierwszym sprawozdaniu założyliśmy, że heurystyka będąca podstawą do wyznaczenia prawdopodobieństwa pierwszego wierzchołka będzie równa różnicy sum krawędzi wchodzących i wychodzących do danego wierzchołka. W trakcie implementacji zauważyliśmy jednak, że krawędzie wchodzące do wierzchołka nie mają tak dużego znaczenia jak krawędzi wychodzące. Dlatego heurystyka obliczana jest w następujący sposób:

1. Przypisz wszystkim wierzchołkom heurystykę równą 0
2. Od heurystyki odejmij wszystkie koszty krawędzi wychodzące z danego wierzchołka
3. Znajdź najmniejszą wartość heurystyki w grafie
4. Od każdej heurystyki odejmij (ujemną wartość) wartość minimalną
5. Dodaj 1 (W ten sposób nawet wierzchołek, którego heurystyka była minimalna ma szanse stać się pierwszym wierzchołkiem danej mrówki)

Aby otrzymać prawdopodobieństwo, że dla danej mrówki dany wierzchołek stanie się pierwszym, należy wartość heurystyki wierzchołka podzielić przez sumę heurystyk w całym grafie

3 Wzór na pozostawianą ilość feromonów

W dokumencie opisującym algorytm przed jego implementacją podaliśmy wzór na ilość pozostawianych feromonów przez każdą mrówkę:

$$\forall_{ij \in P} \tau ij = \tau ij + \frac{zysk_P^2}{koszt_P}$$

Okazało się jednak, że iloraz powoduje zbyt dużą ilość feromonów na krawędziach, dlatego postanowiliśmy współczynnik 2 oznaczyć jako $importanceOfProfit$ i sprawdzić jak będą wyglądały wartości po jego zmianie. Wartość **1.5** okazała się być najlepiej zbilansowaną i to dla niej uruchamialiśmy nasz algorytm na podanych instancjach

4 Parametry

Parametry w stosunku do pierwszego sprawozdania uległy zmianie. Wynika to z faktu, że przy algorytmie mrówkowym parametry można dostosować tylko empirycznie i poprzednie wartości były tylko przypuszczeniami wynikającymi z poprzednich doświadczeń z implementacją tego algorytmu.

- Ilość mrówek w jednej iteracji - 1000 / 5000
- Ilość iteracji - 10
- ρ: 0.7
- τ_{ij} początkowe : 1
- α : 1
- β: 5

Dla podanych instancji wykonaliśmy obliczenia dla 1000 oraz 5000 mrówek aby pokazać jak 5-krotna większość ilość mrówek przekłada się na jakość oraz czas rozwiązań

5 Potencjalne udoskonalenia algorytmu

Poprzednio wyróżniliśmy dwa potencjalne udoskonalenia algorytmu:

- Założyć maksymalny koszt pomiędzy wierzchołkami by przyspieszyć rozwiązanie
- Zmiana wartości heurystyki pierwszego wierzchołka co iterację

Założenie maksymalnego kosztu pomiędzy wierzchołkami zgodnie z przewidywaniami przyspieszyło czas otrzymania rozwiązania. Okazało się jednak, że taka zmiana pogarsza jakość wyników oraz nie jest tak wydajna jak zakładaliśmy, ponieważ nawet bez niej prawdopodobieństwa wybrania krawędzi o tak dużym koszcie bardzo szybko zmierza do 0, co przyspiesza obliczenia. Dlatego wersja ostateczna algorytmu nie zawiera tej modyfikacji

Natomiast zmiana wartości heurystyki co iterację nie została przez nas zaimplementowana w ogóle. Wynika to z faktu że ta modyfikacja nie ma wpływu na prędkość obliczeń tylko na ich jakość. Sprawiłoby to, że więcej mrówek rozpocznie z dobrego wierzchołka, jednak nie jest to potrzebne, wystarczy że jedna mrówka rozpocznie w odpowiednim a reszta będzie szła tylko "mniej więcej"w dobrym kierunku. Uzyskana jakość wyników była dla nas wystarczająca i nie czuliśmy potrzeby jej poprawiania.

6 Wyniki - 5000 mrówek

6.1 Instancje z błędami negatywnymi losowymi

Instancja	n	Wartość optymalna	Rezultat algorytmu	Jakość (%)	Czas (s)
35200-40	209	160	159	99.375	47
18200-80	209	120	119	99.1666	20
18200-40	209	160	159	99.375	45
35200-80	209	120	120	**100**	20
9200-40	209	160	160	**100**	43
9200-80	209	120	120	**100**	19
20300-60	309	240	239	99.583	123
20300-120	309	180	177	98.33	54
55300-60	309	240	239	99.583	125
55300-120	309	180	180	**100**	56
58300-120	309	180	177	98.33	55
58300-60	309	240	239	99.583	124
55400-160	409	240	227	**94.583**	114
62400-160	409	240	235	97.916	112
55400-80	409	320	315	98.437	272
68400-160	409	240	232	96.666	125
62400-80	409	320	318	99.375	285
68400-80	409	320	317	99.06	266
53500-100	509	400	392	98	473
25500-200	509	300	285	95	207
53500-200	509	300	292	97.33	206
25500-100	509	400	395	98.785	493
10500-100	509	400	397	99.257	492
10500-200	509	300	289	96.333	206

6.2 Wybrane instancje z błędami negatywnymi wynikającymi z powtórzeń

Instancja	n	Wartość optymalna	Rezultat algorytmu	Jakość (%)	Czas (s)
/59500-2	509	498	495	99.397	932
144500-12	509	492	487	98.98	894
34500-32	509	477	467	97.90	786

6.3 Wybrane instancje z błędami pozytywnymi losowymi

Instancja	n	Wartość optymalna	Rezultat algorytmu	Jakość (%)	Czas (s)
18200+80	209	200	200	100	119
55300+120	309	300	299	99.66	383
68400+160	409	400	399	99.75	786

6.4 Wybrane instancje z błędami pozytywnymi wynikającymi z przekłamań na końcach oligonukleotydów

Instancja	n	Wartość optymalna	Rezultat algorytmu	Jakość (%)	Czas (s)
35200+20	209	200	196	98	96
55300+30	309	300	289	96.33	250
55400+40	409	400	354	88.5	497

7 Wyniki - 1000 mrówek

7.1 Instancje z błędami negatywnymi losowymi

Instancja	n	Wartość optymalna	Rezultat algorytmu	Jakość (%)	Czas (s)
35200-40	209	160	158	98.75	12
18200-80	209	120	116	98.66	5
18200-40	209	160	157	98.125	12
35200-80	209	120	118	98.33	5
9200-40	209	160	159	**99.375**	12
9200-80	209	120	117	97.5	5
20300-60	309	240	236	98.33	33
20300-120	309	180	173	96.11	14
55300-60	309	240	235	98.91	34
55300-120	309	180	178	98.88	16
58300-120	309	180	176	97.77	16
58300-60	309	240	236	98.33	34
55400-160	409	240	222	**92.5**	33
62400-160	409	240	226	94.166	32
55400-80	409	320	313	98.81	77
68400-160	409	240	223	92.91	34
62400-80	409	320	310	96.875	76
68400-80	409	320	315	98.4375	75
53500-100	509	400	387	96.75	143
25500-200	509	300	280	93.333	66
53500-200	509	300	283	94.33	66
25500-100	509	400	388	97.00	140
10500-100	509	400	393	98.25	147
10500-200	509	300	285	95	62

7.2 Wybrane instancje z błędami negatywnymi wynikającymi z powtórzeń

Instancja	n	Wartość optymalna	Rezultat algorytmu	Jakość (%)	Czas (s)
59500-2	509	498	490	98.39	266
144500-12	509	492	486	98.78	270
34500-32	509	477	460	96.43	211

7.3 Wybrane instancje z błędami pozytywnymi losowymi

Instancja	n	Wartość optymalna	Rezultat algorytmu	Jakość (%)	Czas (s)
18200+80	209	200	200	100	55
55300+120	309	300	299	99.66	132
68400+160	409	400	396	99	273

7.4 Wybrane instancje z błędami pozytywnymi wynikającymi z przekłamań na końcach oligonukleotydów

Instancja	n	Wartość optymalna	Rezultat algorytmu	Jakość (%)	Czas (s)
35200+20	209	200	191	95.55	26
55300+30.	309	300	266	88.66	73
55400+40	409	400	362	90.5	159
25500+50	509	500	445	89	272

8 Wnioski

Powyższe tabele pokazują, że pięciokrotne zwiększenie ilości mrówek nieznacznie podnosi jakość otrzymywanych wyników, jednak wiąże się to ze zwiększeniem czasu trwania obliczeń około 3-krotnie. Pokazuje to, że w zależności od tego, co jest ważniejsze dla osoby korzystającej z programu - dokładność bądź prędkość

- może ona sterować tymi wartościami parametrem "ilość mrówek".

W podanych wynikach można zauważyć również spadek jakości rozwiązań powiązany ze zwiększoną ilością oligonukleotydów w spektrum. Najniższą jakość rozwiązań badany algorytm zwraca dla instancji z błędami pozytywnymi wynikającymi z przekłamań na końcach oligonukleotydów. Związane jest to z faktem, że takie rodzaje błędów są najtrudniejsze do wykrycia dla naszego algorytmu. Oligonukleotyd będący takim błedem ma bardzo mały "koszt"dla mrówki, przez co jest on bardziej prawdopodobnym ruchem. W związku z tym większość mrówek w kolejnej iteracji również podąży w błędnym kierunku powodując efekt "śnieżnej kuli".

Rozwiązanie tego problemu algorytmem mrówkowym spełniło swoje zadanie jako heurystyki. Użytkownik otrzymuje dokładne dane (często powyżej 95%) w krótkim okresie czasu (często poniżej 5 minut)

Aby przyspieszyć działanie algorytmu wszystkie ruchy mrówek wykonywane są wielowątkowo. Punktem synchronizacji jest kolejna iteracja. Prawdopodobnie długie czasy spowodowane są również językiem implementacji (Kotlin)