Kacper Tomczykowski
Nr albumu: 318736

Problem najmniejszego pokrycia wierzchołków w grafie- algorytm ewolucyjny

Wersja języka

- Python - 3.9.7

Używane biblioteki

- Random
- Timeit

Dane wejściowe

Wszystkie testy odbywały się na grafie pełnym z którego losowo usunąłem 50% krawędzi. Graf ma 25 wierzchołków, czyli posiada 25*24/2=300 krawędzi. Po usunięciu 300-150=150 krawędzi. Graf reprezentowany jest jako lista par wierzchołków które się ze sobą łączą.
Osobnik reprezentowany jest przez 25 elementowa listę składającą się z 0 i 1 (0 oznacza zgaszoną latarnie, 1 oznacza zapaloną latarnie) reprezentujących latarnie znajdującą się na wierzchołku o numerze indeksu bitu. Osobnicy w bazowej populacji są tworzeni losowo.

Selekcja turniejowa

W trakcie każdej iteracji programu tworzonych jest tyle turniejów, ile obiektów jest w generacji. Do turnieju losowo wybierane jest 2 osobników bez zwracania. Z każdego turnieju do nowej populacji przechodzi ten osobnik, który osiągnął mniejszy wynik w trakcie oceny. Ocena zachodzi według wzoru ocena= x + 1000 * y, gdzie x oznacza ilość zapalonych latarni a y ilość nierozświetlonych dróg. Funkcja została określona w taki sposób, aby algorytmowi nie opłacało się pozostawiać nierozświetlonych dróg. Dużym plusem powyższej funkcji heurystycznej jest łatwe określenie jak zachował się nasz program. W wyniku liczba tysięcy oznacza ilość nierozświetlonych dróg a liczba mniejsza niż tysiąc oznacza ilość zapalonych latarni.

Mutacja

W trakcie fazy mutacji każda jednostka ma szanse 100% na zmutowanie. Jest ona taka, aby jednostka już wcześniej oceniona i porównana z najlepszą nie marnowała miejsca. Dzięki temu algorytm zamiast kilkukrotnie sprawdzać to samo rozwiązanie rusza się w przestrzeni rozwiązań. Każdy bit jednostki ma 5% na zajście mutacji co daje każdorazową zmianę ok. 1 bitu. Jakby szansa na mutację bitu była mniejsza, wiązałoby się to z kilkukrotnym sprawdzaniem tego samego rozwiązania co spowolniłoby działanie programu. Gdyby szansa mutacji była większa, algorytm tworzyłby losowo nowe obiekty które byłyby mało podobne do rodzica.

Działanie programu

W tabeli odpowiednio średnie oceny wyników \ średni czas funkcji

Ilość iteracji / Wielkość populacji	100	1000	2000	5000
5	8219|0,07	22616|0,70	8419|1,40	11917|3,47
25	4620|0,34	2221|3,34	3219|6,67	1221|16,65
100	5019|1,35	3220|13,28	2620|26,46	2220|66,57
250	4219|3,57	3020|33,10	2325|68,34	1221|163,89

Wnioski

Algorytm radzi sobie dużo lepiej dla większej ilości iteracji. Znacząco zwiększa to szanse na znalezienie minimum globalnego. Przy małej ilości iteracji a nawet bardzo dużej populacji algorytm zatrzymywał się na dostatecznie dobrym wyniku i miał problemy ze znalezieniem lepszego rozwiązania. Duża liczba iteracji znacząco spowalnia program co jest nie wprost proporcjonalne z jakością znalezionego wyniku.