Marcin Mrugas

Wydział Informatyki

Kierunek Informatyka

122580

**Problem komiwojażera**

1. Informacje podstawowe
2. Random
3. BruteForce
4. BoundAndBranch
5. ClosestNeighbour
6. SimulatedAnnealing
7. AntsColonyOptimization
8. GeneticAlgorithm
9. Porównanie algorytmów
10. Wnioski

* Informacje podstawowe

Definicja problemu:

Zadanie polega na wyznaczeniu najkrótszego pełnego cyklu w spójnym grafie.

Założenia:

Dla ułatwienia cykl jest szukany w grafie pełnym, nieskierowanym. Dodatkowo komiwojażer odwiedza każdy wierzchołek dokładnie raz.

Język: C++11

Środowisko: Clion JetBrains, Windows

Kompilator: MinGW 64

Pomiar czasu: QueryPerformanceFrequency()

Użyłem MinGW 64, ponieważ chciałem mieć możliwość wielowątkowego wykonywania algorytmów w Clion. Próbowałem używać jeszcze CygWin, ale nie dawało to dobrych rezulataów.

Zmieniłem także sposób pomiaru czasu z prostego clock() na QueryPerformanceFrequency(), ponieważ pierwszy nie działał dobrze przy wilowątkowości. Dodatkowo drugi zapewnia nam bardzo dokładne wyniki co do mikrosekundy.

* Rozwiązania problmu

1. Random

* Algorytm polega na wybraniu zupełnie losowej ścieżki. Nie znajduje on dobrych rozwiązań, ale daje dobre porównanie do otrzymanych rezultatów z innych algorytmów.
* Rozwiązanie jest generowane dzięki random\_shuffle ze standardowej biblioteki algorithm.
* Złożoność O(1)
* Wyniki: Na wykresie porównanie losowego rozwiązania z ClosoestNeighbour i AntsAlgorithm
* Wnioski: Znalezione rozwiązania nawet nie zbliżą się do optymalnych i dają złe rezultaty

1. BruteForce

* BruteForce jest rozwiązaniem które polega na znalezieniu wszystkich możliwych cykli Hamiltona w grafie i wybraniu najkrótszego. Jest to metoda siłowa i nie efektywna. Jednak w rezultacie mamy pewność że znalezione rozwiązanie jest najlepsze.
* Aby wygenerować wszystkie cykle wygenerowałem wszystkie możliwe permutacje wierzchołków dzięki standardowej funkcji next\_permutation z biblioteki algorithm wśród których znajduję najkrótszy cykl
* Złożoność: O(n!)
* Wyniki:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Il. Wierzchołków | 9 | 10 | 11 | 12 | 13 |
| Czas [s] | 0,29 | 2,19 | 18,66 | 224,432 | 2986,9 |

* Wnioski: Chociaż to rozwiązanie daje najlepsze wyniki to jego czas powyżej 12 wierzchołków jest zbyt długi (50 min).
  1. Branch and Bound
* Algorytm podobnie jak BruteForce znajduje najlepsze rozwiązanie. Jest on jednak usprawniony, ponieważ nie przegląda wszystkich możliwych rozwiązań. Odrzuca on rozwiązania, które są gorsze, niż najlepszy dotychczas znaleziony cykl.
* Rozwiązanie zostało zaimplementowane na bazie rekurencyjnego DFS’a. Algorytm przy pierwszym przejściu generuje pierwszy cykl. Jeżeli następna rozwijana ścieżka jest dłuższa, niż znalezione dotychczas rozwiązanie, to poszukiwanie w tej gałęzi grafu zostaje przerwane i rekurencja wycofuje się do poprzedniego wierzchołka w ścieżce.
* Złożoność O(n!)
* Wyniki:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Il. Wierzchołków | 9 | 10 | 12 | 14 | 15 | 16 | 17 | 18 |
| Czas [s] | 0,046 | 0,16 | 1,129 | 7,57 | 26,47 | 103,77 | 156,66 | 697,45 |

* Wnioski: Usprawnienie znacznie przyśpieszyło szukanie rozwiązania, jednak wciąż znalezienie najkrótszego cyklu dla większych grafów zajmuje dużo czasu. Sprawność algorytmu w dużej mierze zależy w od momentu, w którym znajdzie on pierwszy optymalne rozwiązanie, ponieważ dzięki temu szybciej zaczyna przerywać niewłaściwe poszukiwania.
  1. ClosestNeighbour
* Rozwiązanie które zaczynając z danego wierzchołka zawsze dodaje do rozwiązania najbliższego jeszcze nieodwiedzonego sąsiada.
* Implementacja jest także oparta na DFS’ie, który ze zbioru nieodwiedzonych sąsiadów wybiera najbliższego.
* Wyniki:
* Wnioski: Wybieranie najbliższego sąsiada jest szybkim rozwiązaniem i dobrze sprawdzi się jeżeli możemy dopuścić niedokładne rozwiązanie a zależy nam na krótkim czasie wykonania.
  1. SimulatedAnnealing
* Pomysł polega na tym by dla danego rozwiązania zamienić dwa losowe wierzchołki. Nowe rozwiązanie zostaje zachowane jeżeli jest lepsze lub gdy zostanie zaakceptowane z pewnym prawdopodobieństwem. Pozwala to „wyjść” algorytmowi z rozwiązań dobrych lokalnie, czyli takich wśród których zamiana dwóch wierzchołków nie prowadzi do najoptymalniejszego rozwiązania. Wspomniane prawdopodobieństwo zależy od temperatury, która jest ustalana na początku algorytmu i zmniejszana po każdej iteracji o procent ustalony w parametrze coolingFactor. Prawdopodobieństwo wybóru gorszego rozwiązania został policzony ze wzoru:

Gdzie ∆L to różnica długości między rozwiązaniami a T to temperatura w danej iteracji.

Wyniki

* Wnioski

Symulowane wyżarzanie daje gorsze wyniki niż algorytm genetyczny i mrówkowy. Może to być przez źle dobrane parametry. Przy zbyt małej temperaturze prawdopodobieństwo jest zbyt małe, przy zbyt szybkim stygnięciu algorytm za szybko blokuje się w lokalnym minimum. Ostatni parametr to ilość iteracji który ma główny wpływ na wyniki, gdyż wtedy jest więcej prób znalezienia lepszego rozwiązania.

* 1. AntsCollonyOptimalization
* Algorytm jest wzorowany na podstawie obserwacji mrowiska, w którym każda mrówka wyrusza na poszukiwanie pożywienia. Mrówki wychodząc z mrowiska zostawiają za sobą feromon, który wyparowuje po jakimś czasie. Dlatego im szybciej mrówka wróci z pożywieniem, tym samym posiłek jest bliżej, tym pozostawi za sobą bardziej nasyconą feromonem ścieżkę. Następna mrówka będzie mogła później zasugerować się jej wskazówkami lub wybrać nową drogę poszukiwania.  
  Zostało to wykorzystane w algorytmie mrówkowym, w którym mrówka idąc po grafie z wierzchołka x do y wybiera do go z prawdopodobieństwem:

Gdzie fxy to wartość pozostawionego feromonu, a lxy to długość krawędzi miedzy wierzchołkami x i y.

Po przejściu grafu przez N mrówek feromon jest aktualizowany o

Gdzie ∆f jest nowym zostawionym feromonem przez mrówkę, która przeszła krawędzią x-y i wynosi . Wartość L to długość znalezionego przez mrówkę cyklu.

* Wyniki

|  |  |  |  |
| --- | --- | --- | --- |
| alpha | beta | ρ | długość |
| 1 | 3 | 0,9 | 3388 |
| 0,5 | 1 | 0,9 | 13974 |
| 1 | 3 | 0,5 | 4474 |
| 1 | 3 | 0,2 | 4390 |
| 0,5 | 3 | 0,2 | 3619 |

* Wnioski

Wyniki powstały poprzez rozwiązywania problemu przy różnych parametrach alfa, beta i ρ, ale przy tej samej liczbie mrówek i iteracji. Tabela pokazuje że to rozwiązanie najlepiej działa przy parametrach α=1 β=3 rho=0.9. Wynika to z tego, że gdy ρ jest zbyt niskie to feromon za szybko wyparowuje, także gdy alfa jest zbyt mała a beta zbyt duża bardziej preferowany jest feromon niż faktyczna długość ścieżki, co daje gorsze wyniki.

* 1. GeneticAlgorithm
* Rozwiązanie jest zaprojektowanie na wzór ewolucji w przyrodzie. Na początku jest generowana populacja czyli zbiór losowych rozwiązań, które są analogiczne do organizmów w przyrodzie. W drugim etapie osobniki zostaną ze sobą krzyżowane, dzięki czemu powstają nowe rozwiązania po trochu dziedziczące cechy po każdym z rodziców. Następnie następuje selekcja w której populacja jest zmniejszana o najgorsze osobniki. Te kilka kroków jest powtarzanych przez zaplanowaną liczbę iteracji.
* W implementacji zastosowałem krzyżowanie poprzez order crossover, w którym fragment ścieżki jest przepisany od jednego z rodziców i uzupełniany w kolejności wystąpień od drugiego. Ścieżki samutowane poprzez swap mutation czyli zamianę miejscami dwóch losowych wierzchołków
* Wyniki

|  |  |  |
| --- | --- | --- |
| rozmiar populacji | l. iteracji | długość |
| 70 | 150 | 34890 |
| 50 | 150 | 36270 |
| 50 | 100 | 36449 |
| 70 | 100 | 40440 |
| 200 | 100 | 40447 |
| 20 | 100 | 43045 |

* Wnioski Im większa populacja i im więcej iteracji tym algorytm działa lepiej, więc zależnie od czasu jaki możemy poświęcić naszukanie rozwiązania możemy zmieniać te dwa parametry. Jednak przy bardzo dużej populacji widać, że po mutacji „świeże” osobniki, które mogą wprowadzić przełom są odrzucane gdyż są za długie. Przez to takie rozwiązanie daje gorsze wyniki.
  1. Porównanie wszystkich algorytmów
  2. Wnioski

Najlepszym algorytmem okazał się algorytm mrówkowy i daje lepsze wyniki niż algorytm najbliższego sąsiada powyżej 150 wierzchołków. Dla grafów o liczbie wierzchołków mniejszej niż 50 algorytm genetyczny i symulowanego wyżarzania dawały podobne wyniki co mrówkowy. Wyniki mogą w dużej mierze zależą od dobranych parametrów i ilości iteracji dla algorytmów heurystycznych.