Politechnika Wrocławska

Wydział Informatyki i Telekomunikacji

Projektowanie efektywnych algorytmów

Projekt nr 3 - Implementacja i analiza efektywności algorytmu Tabu Search i Symulowanego Wyżarzania dla problemu komiwojażera

semestr zimowy 2023/2024

Autor:
Eryk Mika 264451

Prowadzący:
Dr inż. Marcin Łopuszyński

Spis treści

1. WSTĘP TEORETYCZNY 1

1.1. *Tabu Search* 1
1.2. Symulowane wyżarzanie 1
1.3. Definicje sąsiedztwa 2

2. OPIS IMPLEMENTACJI ALGORYTMÓW 3

2.1. Klasa *Graph* 3
2.2. Klasa *Graph* – metoda *solveSimulatedAnnealing()* 3
2.1. Klasa *Graph* – metoda *generateInitialTemp()* 6

1. Wstęp teoretyczny

Zgodnie z informacjami przedstawionymi w poprzednich sprawozdaniach, problem komiwojażera jest problemem trudnym pod względem obliczeniowym. W tym opracowaniu zostanie omówione rozwiązanie tego problemu z wykorzystaniem algorytmów przeszukiwania z tabu (ang. *Tabu Search*, TS) oraz symulowanego wyżarzania (ang. *Simulated Annealing*, SA). Są to algorytmy przeszukiwania lokalnego (ang. *local search*) – **zazwyczaj** pozwalają one na znalezienie rozwiązania **zbliżonego** do optimum danego problemu optymalizacyjnego (globalnego maksimum lub minimum), jednak **nie muszą** zapewnić znalezienia tego optimum[1].

1.1. *Tabu Search*

Tabu Search (TS) to heurystyczna metoda optymalizacyjna używana do rozwiązywania problemów optymalizacji kombinatorycznej. Została wprowadzona przez Freda Glovera w latach 80. i opiera się na koncepcji przeszukiwania przestrzeni rozwiązań w sposób inteligentny, przy użyciu mechanizmu tabu (ruchów zakazanych), który zapobiega powtarzaniu tych samych ruchów lub rozwiązań w trakcie przeszukiwania.

Algorytm zaczyna od wygenerowania początkowego rozwiązania, które jest następnie oceniane pod kątem funkcji celu. Tworzone są sąsiednie rozwiązania poprzez wprowadzanie drobnych zmian do obecnie rozpatrywanego rozwiązania. To sąsiedztwo jest kluczowe dla algorytmu, ponieważ TS próbuje eksplorować przestrzeń rozwiązań, szukając lepszych wartości funkcji celu. Algorytm ocenia każde nowe rozwiązanie za pomocą funkcji celu, która przypisuje wartość jakości danego rozwiązania. Celem jest minimalizacja lub maksymalizacja tej funkcji, zależnie od charakterystyki problemu. Tabu Search utrzymuje listę tabu, która zawiera informacje na temat ostatnio odwiedzonych rozwiązań lub ruchów. Mechanizm tabu uniemożliwia powrót do niedawno odwiedzonych stanów, co pomaga uniknąć cykli i skuteczniej przeszukiwać przestrzeń rozwiązań. Pomimo tego, że ruch może być oznaczony jako zakazany na liście tabu, kryterium aspiracji może zezwalać na jego wykonanie, jeśli jest on korzystny w danej sytuacji. Algorytm składa się z faz intensyfikacji (skupiającej się na lokalnym przeszukiwaniu wokół obecnie najlepszego rozwiązania) i dywersyfikacji (poszerzającej przeszukiwanie w poszukiwaniu nowych obszarów rozwiązań). Algorytm działa do momentu spełnienia określonego warunku stopu, na przykład osiągnięcia maksymalnej liczby iteracji lub uzyskania satysfakcjonującego rozwiązania[2].

1.2. Symulowane wyżarzanie

Symulowane wyżarzanie metaheurystyka, której nazwa nawiązuje do procesu wyżarzania metali. Algorytm został zainspirowany procesem chłodzenia stopionego metalu, w którym stopiony metal jest stopniowo schładzany, co pozwala na osiągnięcie bardziej stabilnej struktury

[1] https://www.cs.cmu.edu/~15281/coursenotes/localsearch/index.html
[2] https://cs.pwr.edu.pl/zielinski/lectures/om/localsearch.pdf

krystalicznej. W przypadku symulowanego wyżarzania, proces chłodzenia jest odwzorowany w celu znalezienia globalnego optimum w przestrzeni rozwiązań.

Algorytm rozpoczyna od losowego rozwiązania początkowego. Podobnie jak w przypadku Tabu Search, symulowane wyżarzanie korzysta z funkcji celu do oceny jakości rozwiązania. Algorytm wprowadza pojęcie temperatury, która kontroluje prawdopodobieństwo akceptacji gorszych rozwiązań. W początkowej fazie temperatura jest wysoka, co pozwala na akceptowanie większej liczby gorszych rozwiązań. W miarę postępu algorytmu temperatura maleje, co sprawia, że akceptacja gorszych rozwiązań staje się coraz mniej prawdopodobna. Prawdopodobieństwo zaakceptowania gorszego rozwiązania jest związane z różnicą wartości funkcji celu oraz aktualnej temperatury. Im wyższa temperatura, tym większa szansa na akceptację gorszego rozwiązania. Algorytm działa do momentu spełnienia określonego warunku stopu, na przykład osiągnięcia maksymalnej liczby iteracji lub uzyskania satysfakcjonującego rozwiązania. Symulowane wyżarzanie jest stosowane w przypadkach, gdzie przestrzeń rozwiązań jest duża i skomplikowana, a funkcja celu może zawierać wiele lokalnych optimum. Algorytm pozwala na unikanie zatrzymywania się w lokalnych optimum poprzez akceptację czasami gorszych rozwiązań na początku procesu optymalizacji, a następnie stopniowe zaostrzanie kryteriów akceptacji w miarę postępu algorytmu. To podejście pozwala na przeszukiwanie przestrzeni rozwiązań w sposób bardziej elastyczny i bardziej zrównoważony[3].

1.3. Definicje sąsiedztwa[4]

Sąsiedztwo dla *tabu search* i symulowanego wyżarzania jest kluczowym elementem, ponieważ wpływa na sposób generowania sąsiednich rozwiązań podczas przeszukiwania przestrzeni rozwiązań. Dla wielu problemów optymalizacyjnych, takich jak problem komiwojażera, stosuje się trzy popularne operatory sąsiedztwa: *swap* (zamiana miejscami), *insert* (wstawianie) i *inverse* (odwracanie).

- *swap* - polega na zamianie dwóch elementów w rozwiązaniu. Dla problemu trasowania, swap może oznaczać zamianę dwóch miast w trasie.

- i*nsert* - polega na przeniesieniu jednego elementu z jednego miejsca w rozwiązaniu i wstawieniu go w inne miejsce. Dla problemu trasowania, insert może oznaczać przeniesienie miasta z jednej pozycji w trasie i wstawienie go w inne miejsce.

- *inverse* - polega na odwróceniu kolejności pewnego fragmentu rozwiązania. Dla problemu trasowania, inverse może oznaczać odwrócenie kolejności miast pomiędzy dwoma wybranymi punktami w trasie.

[3] http://www.pi.zarz.agh.edu.pl/intObl/notes/IntObl_w2.pdf
[4] https://www.researchgate.net/publication/266033160_13091453v1#pf11

2. Opis implementacji algorytmów

W celu analizy efektywności omawianych algorytmów został napisany program w języku C++ z wykorzystaniem obiektowego paradygmatu programowania. Najistotniejszymi komponentami aplikacji są klasy *Graph* oraz *Route*, których pola (struktury danych) oraz metody są odpowiedzialne za realizację algorytmu. Wiele istotnych kwestii związanych z implementacją zostało wyjaśnionych w komentarzach w plikach źródłowych.

2.1. Klasa *Graph*

Klasa Graph jest główną klasą programu, która jest odpowiedzialna za przechowywanie struktury i metod grafu, na którym wykonywane są badane algorytmy. Pola prywatne klasy – dwuwymiarowa tablica std::vector matrix oraz size są użyte do przechowywania długości krawędzi w postaci macierzy kwadratowej – kosztów - matrix stopnia size. Oba pola przechowują liczby stałoprzecinkowe typu int. W macierzy komórka o współrzędnych i, j zawiera odległość pomiędzy wierzchołkami i i j.

Zaimplementowano konstruktory (domyślny, generujący losową instancję problemu o rozmiarze N, wczytujący instancję z pliku tekstowego, przeładowany operator przypisania oraz metodę wypisującą graf (macierz) na ekran.

2.2. Klasa *Graph* – metoda *solveSimulatedAnnealing()*

Algorytm zaczyna się od inicjalizacji zmiennych, takich jak czas, liczba elementów trasy, i temperatura początkowa – wykorzystana jest w tym celu metoda *generateInitialTemp()*. Następnie początkowa trasa jest generowana w sposób zachłanny i obliczany jest jej koszt. Algorytm przechodzi do głównej pętli, która będzie trwała do momentu spełnienia warunku stopu. W każdej iteracji pętli następują następujące operacje:

1. Losowane są dwie różne pozycje w trasie, a następnie tworzona jest nowa trasa poprzez zastosowanie operatora *swap*.
2. Obliczane są koszty obecnej trasy i nowo wygenerowanej trasy. Następnie obliczana jest różnica kosztów między nimi.
3. Jeśli nowa trasa jest lepsza (o niższym koszcie), to zostaje zaakceptowana jako obecna trasa. W przeciwnym razie, jest ona akceptowana z pewnym prawdopodobieństwem, zależnym od różnicy kosztów i temperatury. Wyraża się ono wzorem (
4.
5. Rysunek 2.1), gdzie lewa strona nierówności to losowa liczba z przedziału [0,1), *diff* to różnica kosztów pomiędzy nową a obecną trasą, natomiast *T* to temperatura w obecnej iteracji. Prawdopodobieństwo akceptacji gorszego rozwiązania maleje wraz ze spadkiem temperatury.

Rysunek 2.1

$$random[0,1) < e^{-diff/T}$$

Źródło: opracowanie własne na podstawie[5]

6. Jeśli obecna trasa jest lepsza niż trasa optymalna, to aktualizowana jest trasa optymalna oraz jej koszt.

7. Temperatura jest zmniejszana, co odpowiada procesowi schładzania. W algorytmie Symulowanego Wyżarzania schładzanie ma na celu zwiększenie prawdopodobieństwa akceptacji lepszych rozwiązań na początku, a następnie stopniowe zaostrzanie kryteriów akceptacji. Schładzanie odbywa się według wzoru Rysunek 2.2. We wzorze tym *T(i)* to temperatura w obecnej iteracji, *α* to współczynnik schładzania – w implementacji określony przez parametr *delta*, natomiast *T(i-1)* to temperatura w poprzedniej iteracji.

Rysunek 2.2

$$T(i) = \alpha * T(i-1)$$

Źródło: opracowanie na podstawie założeń dot. projektu

8. Po każdej 100000. iteracji pętli sprawdzane jest, czy upłynął określony limit czasu. Jeśli tak, to wykonywanie pętli jest przerywane. Sprawdzanie czasu co określoną liczbę iteracji ma na celu ograniczenie wpływu pomiaru czasu na czas wykonywania właściwego algorytmu.

Po zakończeniu głównej pętli algorytm następuje wypisanie czasu wykonania, optymalnego kosztu oraz ścieżki. Zwracany jest koszt trasy. Według przyjętych założeń projektowych główna pętla algorytmu jest wykonywana przez sztywnie narzucony limit czasu określony przez parametr *timeLimit*. Omówiony algorytm przedstawia Rysunek 2.3.

[5] https://cs.pwr.edu.pl/zielinski/lectures/om/localsearch.pdf

Rysunek 2.3 Główna pętla algorytmu symulowanego wyżarzania

```
while(warunek stopu) {
        // Losowanie dwóch różnych pozycji w trasie
         int pos1 = rand() % routeElements;
         int pos2 = rand() % routeElements;
        if(pos1 == pos2) pos2 = (pos2 + 1) % routeElements;
        // Tworzenie nowej trasy poprzez zastosowanie operatora swap
        Route newRoute = currentRoute;
        newRoute.procedureSwap(pos1, pos2);
        // Obliczenie kosztów obecnej i nowej trasy
        int currentCost = calculateRouteCost(currentRoute);
        int newCost = calculateRouteCost(newRoute);
        // Obliczenie różnicy kosztów między trasami
        int costDiff =  newCost - currentCost;
        // Sprawdzenie czy nowa trasa jest lepsza lub czy zaakceptować gorsze rozwiązanie
        if(costDiff < 0)
        {
            currentRoute = newRoute;
         }
        else if(((double)rand()/(double)RAND_MAX)<=exp(-(double)costDiff/T))
        {
            currentRoute = newRoute;
            currentCost = newCost;
         }
         // Aktualizacja trasy optymalnej, jeśli znaleziono lepsze rozwiązanie
         if(currentCost < optimalCost)
          {
              optimalRoute = currentRoute;
               optimalCost = currentCost;
              // Aktualizacja czasu
              auto end = std::chrono::steady_clock::now();
          }
        // Zmniejszanie temperatury
        T *= delta;
        timeCheck++;
}
```

Źródło: opracowanie własne

2.1. Klasa *Graph* – metoda *generateInitialTemp()*

Metoda ta użyta jest do generowania temperatury początkowej dla danego problemu w oparciu o przetwarzane dane. W tym celu oblicza się **odchylenie przeciętne**[6] dla długości krawędzi w grafie reprezentującym dany problem. Następuje sumowanie wartości krawędzi dla wszystkich par różnych wierzchołków grafu, pomijając krawędzie pętlowe (krawędzie prowadzące do tego samego wierzchołka). Oblicza się średnią wartości krawędzi jako ilorazu sumy wartości krawędzi przez liczbę wszystkich krawędzi. Szukane odchylenie wyznacza się jako stosunek sumy wartości bezwzględnej różnicy między średnią a wartością krawędzi dla wszystkich par różnych wierzchołków i liczby krawędzi niebędących pętlami. Temperaturę początkową stanowi wartość odchylenia pomnożona przez 10^5.

2.2. Klasa Graph – metoda *solveTabuSearch()*

[6] http://home.agh.edu.pl/~bartus/index.php?action=dydaktyka&subaction=statystyka&item=miary_zmiennosci