**WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI**

**POLITECHNIKA WROCŁAWSKA**

KIERUNEK: INFORMATYKA TECHNICZNA

**Projektowanie efektywnych algorytmów**

**PROJEKT** 2

**Autor**: Maksymilian Łukaszewski 272975

**Prowadzący**: dr inż. Dariusz Banasiak

Czwartek 7:30

1. **Cel Projektu**

Celem projektu było zaprojektowanie i zaimplementowanie 2 algorytmów dla asymetrycznego problemu komiwojażera. Dodatkowo należało zbadać i przeanalizować czas potrzebny na rozwiązanie problemu w zależności od rozmiaru instancji oraz zastosowanego algorytmu. Następujące algorytmy zostały zbadane w ramach projektu:

Algorytm Tabu Search

Algorytm symulowanego wyżarzania

**Przyjęte założenia**

Podczas projektowania i implementacji przyjęto następujące założenia:

* zadanie zrealizowane jest w ramach jednego programu konsolowego
* wszystkie struktury danych alokowane są dynamicznie
* odległości pomiędzy miastami są liczbami całkowitymi
* program umożliwia wczytanie danych wejściowych z pliku
* program jest stworzony zgodnie z zasadami programowania obiektowego
* zaimplementowano odcięcie „czasowe” górne
* zaimplementowano odcięcie dla TS wynikające z ilości iteracji bez zmiany najkrótszej ścieżki oraz dla SA dla Temperatury poniżej 1/100000. Potrzeba takich ograniczeń wynikła z początkowych badań i sprawiła że górne odcięcie czasowe było niepotrzebne.

**Język Programowania i środowisko**

Program, który stworzono w ramach projektu został napisany w języku C++. Badania przeprowadzono na komputerze lenovo Thinkpad T480 z systemem operacyjnym Windows10. Badania przeprowadzono po kompilacji i buildowaniu kodu w trybie *debug.* Podczas przeprowadzania testów wyłączono wszystkie procesy w tle, w celu uzyskania jak najbardziej precyzyjnych wyników.

1. **Wstęp teoretyczny: Symulowane Wyżarzanie i Tabu Search**

Problemy optymalizacyjne o wysokiej złożoności, takie jak problem komiwojażera, wymagają efektywnych metod poszukiwania rozwiązań. Algorytmy metaheurystyczne, takie jak Symulowane Wyżarzanie (*Simulated Annealing*) i Tabu Search, pozwalają na znalezienie rozwiązań bliskich optymalnym w akceptowalnym czasie.

**Symulowane Wyżarzanie** opiera się na analogii do procesu fizycznego wyżarzania, gdzie materiały są stopniowo schładzane. Algorytm iteracyjnie eksploruje przestrzeń rozwiązań, akceptując czasem gorsze rozwiązania z pewnym prawdopodobieństwem, co pozwala unikać lokalnych minimów. Proces kończy się, gdy temperatura spadnie poniżej ustalonego progu. Metoda jest prosta, ale wymaga precyzyjnego doboru parametrów, takich jak harmonogram chłodzenia.

**Tabu Search** wykorzystuje listę tabu do unikania powrotu do niedawno odwiedzonych rozwiązań, co sprzyja eksploracji nowych obszarów przestrzeni rozwiązań. Lista tabu oraz reguły aspiracji zapewniają balans między eksploracją a eksploatacją. Algorytm charakteryzuje się deterministycznym podejściem i skutecznością w unikania lokalnych minimów.

Oba algorytmy są szeroko stosowane w problemach kombinatorycznych, takich jak optymalizacja tras czy harmonogramowanie, różniąc się sposobem eksploracji przestrzeni i mechanizmami unikania pułapek lokalnych.

1. **Metodologia badania**

Eskperyment miał sprawdzić efektywność algorytmów w zależności od zadanych im parametrów. Badania przeprowadzono dla 3 reprezentatywnych ilości miast:

55 w pliku ftv55.atsp

170 w pliku ftv170.atsp

358 w pliku rbg358.atsp

Górna granica czasowa została ustalona w zależności od ilości miast w sposób nastepujący:

-2 minuty dla n=55

-3,5 minuty dla n=170

-5 minut dla n=358

W przypadku algorytmu Tabu Search sprawdzono działanie przy użyciu 2 rodzajów generowania sąsiedztw:

Zamiana (Swap) dwóch miast w rozwiązaniu

Wstawienie (Insert) miasta w inne miejsce, i przesunięcie wszystkich pozostałych „o 1 w prawo”

W przypadku symultanicznego wyżarzania sprawdzono efektywność przy 3 różnych sposobach chłodzenia

Liniowy: T(k)=T0\*(1/1+k)

Geometryczny:T(k) = T\*coolingrate

Logarytmiczny: T(k) = T0/( 1 + coolingRate · log(1 + k))

**4. Tabu Search (Przeszukiwanie Tabu)**

Tabu Search (przeszukiwanie tabu) to metaheurystyczna metoda optymalizacji, której celem jest efektywne przeszukiwanie przestrzeni rozwiązań problemu, unikając popadania w lokalne minima. W przeciwieństwie do algorytmów takich jak wspinaczka (hill climbing), Tabu Search zapamiętuje ostatnio odwiedzane rozwiązania, aby uniemożliwić algorytmowi powrót do nich w przyszłych iteracjach. Jest to realizowane za pomocą tzw. listy tabu, która pełni rolę pamięci krótkoterminowej.

Tabu Search działa iteracyjnie, przeszukując sąsiedztwo bieżącego rozwiązania w celu znalezienia rozwiązania o lepszej wartości funkcji celu. Jeżeli lepsze rozwiązanie nie istnieje w sąsiedztwie, algorytm może zaakceptować gorsze rozwiązanie, o ile nie jest ono zabronione przez listę tabu. W ten sposób metoda może unikać lokalnych minimów i poszukiwać rozwiązania globalnego.

**Podstawowe etapy Tabu Search:**

1. Inicjalizacja:

Wybierz rozwiązanie początkowe oraz ustal parametry, takie jak długość listy tabu i maksymalną liczbę iteracji.

1. Przeszukiwanie sąsiedztwa:

Wygeneruj zbiór sąsiadów bieżącego rozwiązania, które mogą być osiągnięte poprzez niewielkie zmiany (np. zamiana dwóch elementów w permutacji dla problemu TSP).

1. Aktualizacja listy tabu:

Dodaj do listy tabu niedozwolone kroki (np. zamiany elementów) i usuń najstarsze elementy z listy, aby kontrolować jej długość.

1. Kryteria akceptacji:

Wybierz najlepsze rozwiązanie z sąsiedztwa, nawet jeśli jest gorsze od bieżącego, pod warunkiem, że nie narusza zasad tabu (chyba że spełnia kryterium aspiracji, np. poprawia najlepsze znalezione rozwiązanie).

1. Aktualizacja najlepszego rozwiązania:

Zaktualizuj najlepsze znalezione rozwiązanie, jeśli jest lepsze od dotychczasowego.

Tabu Search jest stosowane w problemach takich jak problem komiwojażera (TSP), harmonogramowanie czy alokacja zasobów. Choć nie gwarantuje znalezienia rozwiązania optymalnego, algorytm często generuje bardzo dobre rozwiązania w rozsądnym czasie.

**5. Simulated Annealing (Wyżarzanie Symulacyjne)**

Wyżarzanie symulacyjne (ang. Simulated Annealing, SA) to metoda inspirowana procesami fizycznymi, takimi jak wyżarzanie metali, gdzie kontrolowane chłodzenie prowadzi do uzyskania stanu o minimalnej energii. W kontekście optymalizacji metoda ta znajduje rozwiązania problemów poprzez stopniowe zmniejszanie temperatury, co ogranicza losowość algorytmu i pozwala skoncentrować się na lepszych rozwiązaniach.

Wyżarzanie symulacyjne opiera się na dwóch kluczowych mechanizmach:

* Losowa eksploracja: Algorytm przeszukuje przestrzeń rozwiązań, przechodząc do losowo wybranego sąsiada bieżącego rozwiązania.
* Kryterium akceptacji: Nowe rozwiązanie jest akceptowane z pewnym prawdopodobieństwem, które maleje wraz z temperaturą. Na początku algorytm jest skłonny zaakceptować gorsze rozwiązania, co pozwala unikać lokalnych minimów, ale w miarę chłodzenia akceptowane są tylko lepsze lub minimalnie gorsze rozwiązania.

**Podstawowe etapy Simulated Annealing:**

1. Inicjalizacja:

Wybierz rozwiązanie początkowe, ustal temperaturę początkową, współczynnik chłodzenia oraz maksymalną liczbę iteracji.

1. Generowanie sąsiedztwa:

Wygeneruj losowego sąsiada bieżącego rozwiązania (np. poprzez zamianę dwóch elementów w permutacji w problemie TSP).

1. Kryterium akceptacji:

Jeśli sąsiad jest lepszy od bieżącego rozwiązania, zaakceptuj go.

Jeśli sąsiad jest gorszy, zaakceptuj go z prawdopodobieństwem P=e^(−ΔE/T) , gdzie ΔE to różnica między wartościami funkcji celu sąsiada i bieżącego rozwiązania, a T to bieżąca temperatura.

1. Chłodzenie:

Zmniejsz temperaturę według przyjętego schematu (np. chłodzenie geometryczne T=T⋅α, gdzie α to współczynnik chłodzenia).

1. Zakończenie:

Powtarzaj kroki 2–4 do momentu osiągnięcia maksymalnej liczby iteracji lub wystarczająco niskiej temperatury.

Wyżarzanie symulacyjne jest szeroko stosowane w problemach kombinatorycznych, takich jak problem komiwojażera, harmonogramowanie czy optymalizacja tras. Jest cenione za prostotę i zdolność do unikania lokalnych minimów, choć jego wydajność zależy od odpowiedniego dostrojenia parametrów, takich jak temperatura początkowa i współczynnik chłodzenia.

1. **Metodologia badania**

**Klasy Pomocnicze**

**7.1. Klasa Matrix**

**Konstruktor i Destruktor**

* **Matrix(int size)**: Inicjalizuje macierz o zadanym rozmiarze.
* **~Matrix()**: Zwalnia pamięć zaalokowaną dla macierzy.

**Operacje na macierzy**

* **freeMatrix()**: Zwalnia pamięć zaalokowaną dla macierzy.
* **loadFromFile(const string& filename)**: Ładuje macierz z pliku. Odczytuje rozmiar macierzy i jej elementy.
* **generateRandom(int newSize, int min, int max)**: Generuje losową macierz o określonym rozmiarze, z wartościami w podanym zakresie (min, max).
* **print()**: Wypisuje macierz na ekran z sformatowanym wyjściem.
* **getSize()**: Zwraca rozmiar macierzy.
* **getMatrix()**: Zwraca wskaźnik na surową tablicę macierzy.
* **getValue(int row, int col)**: Zwraca wartość z macierzy na podanych (wiersz, kolumna).

**7.2. Klasa Timer**

**Operacje na timerze**

* **Timer()**: Konstruktor inicjalizuje elapsedTime na 0.
* **start()**: Rozpoczyna pomiar czasu, zapisując czas początkowy.
* **stop()**: Zatrzymuje pomiar czasu, zapisując czas końcowy i obliczając czas wykonania w
* milisekundach.
* **getElapsed()**: Zwraca czas wykonania.
* \**printElapsed(const char* label)\*\*: Wypisuje czas wykonania z etykietą.

**8. Algorytm Przeszukiwania Tabu (Tabu Search)**

**8.1 Opis teoretyczny**

Przeszukiwanie tabu (Tabu Search, TS) jest metaheurystycznym algorytmem optymalizacji, stosowanym do rozwiązywania złożonych problemów kombinatorycznych, takich jak problem komiwojażera (TSP). Algorytm opiera się na iteracyjnym przeszukiwaniu przestrzeni rozwiązań i unikaniu lokalnych minimów poprzez wprowadzenie mechanizmu pamięci krótkoterminowej – listy tabu, która blokuje ponowne odwiedzanie niedawno rozważanych rozwiązań.

**Kluczowe elementy algorytmu Tabu Search:**

* Pamięć tabu: Lista tabu przechowuje pewną liczbę ostatnich rozwiązań lub ruchów (np. zamian elementów), co pozwala algorytmowi unikać powracania do niedawno odwiedzanych stanów.
* Przeszukiwanie sąsiedztwa: Dla bieżącego rozwiązania generowane są potencjalne sąsiednie rozwiązania, które różnią się od bieżącego jednym lub kilkoma ruchami (np. zamianą dwóch miast w ścieżce).
* Kryterium akceptacji: Algorytm wybiera najlepsze rozwiązanie z sąsiedztwa, które nie jest tabu (chyba że spełnia kryterium aspiracji, np. prowadzi do najlepszego znanego rozwiązania).
* Aktualizacja listy tabu: Po każdej iteracji najnowszy ruch (lub rozwiązanie) zostaje dodany do listy tabu, a najstarszy ruch jest usuwany.

Przeszukiwanie tabu to skuteczna metoda pozwalająca uniknąć zakleszczenia w lokalnych minimach. Dzięki elastycznemu mechanizmowi pamięci może eksplorować przestrzeń rozwiązań w sposób bardziej globalny.

**8.2 Złożoność obliczeniowa**

Złożoność obliczeniowa algorytmu zależy od kilku czynników:

* Rozmiar przestrzeni sąsiedztwa: Dla problemu z nnn miastami, rozmiar sąsiedztwa zależy od liczby generowanych sąsiadów (np. zamian, insertów), co może wynosić O(n^2).
* Długość listy tabu: Lista tabu ogranicza przestrzeń rozwiązań poprzez pamiętanie ostatnich t rozwiązań, gdzie ttt to ustalony rozmiar listy tabu (np. O(t⋅n)).
* Czas przeszukiwania: Liczba iteracji k oraz liczba sąsiadów analizowanych w każdej iteracji wpływają na czas działania algorytmu.

W najgorszym przypadku złożoność algorytmu może wynosić O(k⋅n^2), gdzie k to liczba iteracji. Efektywność w praktyce zależy jednak od parametrów takich jak rozmiar listy tabu, liczba generowanych sąsiadów i strategia przeszukiwania.

**8.3 Opis implementacji**

Poniżej przedstawiono działanie algorytmu Tabu Search implementowanego w kodzie:

1. Inicjalizacja:
   * Algorytm rozpoczyna od rozwiązania wygenerowanego metodą zachłanną, które odwiedza miasta w kolejności minimalizującej koszt lokalny.
   * Lista tabu jest zainicjowana jako pusta i ma określoną długość tenure.
2. Przeszukiwanie sąsiedztwa:
   * Generowane są sąsiednie rozwiązania poprzez dwie strategie:
     + Swap: Zamiana dwóch miast w ścieżce.
     + Insert: Przeniesienie jednego miasta w inne miejsce w ścieżce.
   * Dla każdego sąsiada obliczany jest koszt, a algorytm wybiera najlepsze rozwiązanie, które nie jest tabu.
3. Aktualizacja listy tabu:
   * Po każdej iteracji lista tabu jest aktualizowana (updateTabuList) poprzez dodanie nowego rozwiązania, a najstarszy element jest usuwany.
4. Kryterium zatrzymania:
   * Algorytm zatrzymuje się, gdy osiągnięty zostanie maksymalny czas działania lub maksymalna liczba iteracji.
5. Wynik:
   * Najlepsze znalezione rozwiązanie jest zapisywane jako wynik globalny, a jego koszt jest przechowywany w zmiennej optimalCost.

.

**9. Algorytm Symulowanego Wyżarzania (Simulated Annealing)**

**9.1 Opis teoretyczny**

Symulowane wyżarzanie (Simulated Annealing, SA) to probabilistyczny algorytm optymalizacji inspirowany procesami fizycznymi wyżarzania metali. Metoda ta pozwala na eksplorację przestrzeni rozwiązań i unikanie zakleszczenia w lokalnych minimach poprzez kontrolowane "rozgrzewanie" i "chłodzenie" systemu, czyli stopniowe zmniejszanie temperatury w miarę przebiegu iteracji.

Kluczowe elementy algorytmu Symulowanego Wyżarzania:

1. Inicjalizacja temperatury: Algorytm rozpoczyna z wysoką temperaturą, co pozwala na akceptację także gorszych rozwiązań w początkowych etapach.
2. Sąsiedztwo rozwiązania: W każdej iteracji algorytm generuje sąsiada bieżącego rozwiązania poprzez losowy ruch, np. zamianę dwóch miast w ścieżce.
3. Prawdopodobieństwo akceptacji: Rozwiązanie gorsze od aktualnego może zostać zaakceptowane z pewnym prawdopodobieństwem: P=exp^(−ΔE/T), gdzie ΔE to różnica kosztu między rozwiązaniem sąsiada a bieżącym, a T to temperatura.
4. Harmonogram chłodzenia: Temperatura jest zmniejszana zgodnie z przyjętą strategią chłodzenia:
   * Logarytmiczne chłodzenie
   * Liniowe chłodzenie
   * Geometryczne chłodzenie
5. Kryterium zatrzymania: Algorytm kończy działanie, gdy temperatura osiągnie zbyt niską wartość, czas wyczerpie się lub zostanie osiągnięta maksymalna liczba iteracji.

**9.2 Złożoność obliczeniowa**

Złożoność algorytmu zależy od:

* Liczby iteracji: O(k), gdzie k to maksymalna liczba iteracji.
* Generowania sąsiadów: Koszt każdej iteracji zależy od czasu generowania sąsiada O(n) oraz obliczania kosztu rozwiązania O(n). W praktyce całkowita złożoność wynosi O(k⋅n), gdzie k to liczba iteracji, a nnn to liczba miast.

**9.3 Opis implementacji**

Kod implementuje algorytm Symulowanego Wyżarzania z uwzględnieniem różnych harmonogramów chłodzenia oraz metod generowania sąsiadów.

1. Inicjalizacja:
   * Rozwiązanie początkowe to losowa permutacja miast.
   * Temperatura początkowa T0​ jest ustawiana jako parametr wejściowy.
2. Generowanie sąsiada:
   * Sąsiad jest generowany przez losową zamianę dwóch miast w ścieżce. Operacja ta jest szybka i efektywna.
3. Obliczanie kosztu:
   * Funkcja calculateCost oblicza koszt ścieżki jako sumę kosztów krawędzi pomiędzy kolejnymi miastami oraz powrót do miasta startowego.
4. Akceptacja sąsiada:
   * Jeśli rozwiązanie sąsiada ma mniejszy koszt, zostaje zaakceptowane.
   * Jeśli rozwiązanie sąsiada ma większy koszt, zostaje zaakceptowane z prawdopodobieństwem: P=exp^(−ΔE/T)
5. Harmonogram chłodzenia:
   * Logarytmiczne chłodzenie: Temperatura zmniejszana zgodnie z formułą: T=T0/(1+coolingRate⋅log(1+iteration))​​
   * Liniowe chłodzenie: Temperatura zmniejszana w sposób liniowy: T=T0−coolingRate⋅iteration
   * Geometryczne chłodzenie: Temperatura zmniejszana w sposób wykładniczy: T=T0⋅coolingRate
6. Kryterium zatrzymania:
   * Algorytm kończy się, gdy osiągnięty zostanie maksymalny czas działania maxTimeSA lub maksymalna liczba iteracji maxIterations.
7. Wynik:
   * Najlepsze rozwiązanie znalezione podczas działania algorytmu jest zapisane w tablicy bestSolution, a jego koszt w zmiennej bestCost.
8. **WYNIKI**

A table with numbers and a few black text

Description automatically generated

Wyniki Tabu Searcha dla odpowiednich algorytmów generowania sąsiedztw dla n=55 A table with numbers and letters

Description automatically generated

Wyniki Tabu Searcha dla n=170.

A table with numbers and letters

Description automatically generated

Wyniki dla n=358.

A graph with lines and numbers

Description automatically generated

A graph with lines and numbers

Description automatically generated

A graph with lines and dots

Description automatically generated

A graph with blue lines and numbers

Description automatically generated

A graph with blue lines and dots

Description automatically generated

A table with numbers and a few black text

Description automatically generated

Wyniki wyżarzania symultanicznego dla n=55

A table of numbers and a few black text

Description automatically generated with medium confidence

Wyniki wyżarzania symultanicznego dla n =170

A table with numbers and a percentage

Description automatically generated

Wyniki wyżarzania symultanicznego dla n=358.

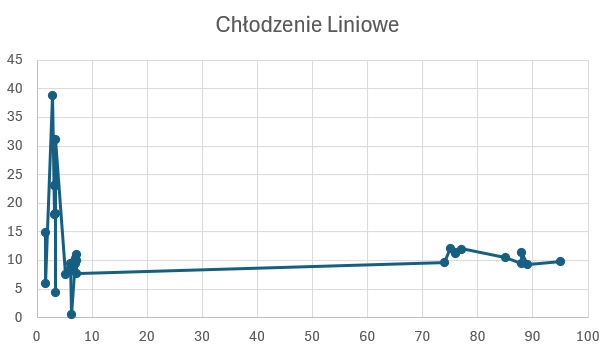
A graph with lines and dots

Description automatically generated

A graph with lines and dots

Description automatically generatedA graph with numbers and lines

Description automatically generated

A graph with blue lines and numbers

Description automatically generated

A graph with blue lines and numbers

Description automatically generated

**10.WNIOSKI**

Swap w 2 przypadkach znalazł krótszą ścieżkę od inserta, dopiero dla n = 358 insert znalazł krótszą ścieżkę, lecz jest to w pewnym stopniu zależne od przypadku.

Swap w każdym wypadku jest szybszy, co jest zgodne z teorią, ponieważ swap ma złożonośc O(1), a insert O(n), co powoduje zauważalne różnice.

W przypadku symultanicznego wyżarzania algorytm logarytmiczny działa najszybciej, ponieważ najszybciej zbija temperaturę (warunek graniczny to temperatura poniżej 1/100000 T. Pomimo najkrótszego czasu działania generuje on też najdokładniejsze rozwiązania.

Dla wszystkich kombinacji, zgodnie z teorią im czas bliższy nieskończoności, tym mniejsze wachania wyników.

**DODATEK**

**Przykład działania algorytmu symulowanego wyżarzania (Simulated Annealing) z ochładzaniem geometrycznym dla problemu komiwojażera**

**Dane wejściowe:**

| **Miasto** | **A** | **B** | **C** | **D** |
| --- | --- | --- | --- | --- |
| A | 0 | 10 | 15 | 20 |
| B | 10 | 0 | 30 | 25 |
| C | 15 | 30 | 0 | 35 |
| D | 20 | 25 | 35 | 0 |

**Parametry algorytmu:**

* **Temperatura początkowa (T₀)**: 100
* **Współczynnik ochładzania (α)**: 0.9
* **Minimalna temperatura (Tₘ)**: 0.1
* **Liczba iteracji na temperaturę**: 100

**Krok 1: Wybór początkowego rozwiązania**

Rozpoczynamy od losowego rozwiązania, np. **C → A → D → B → C**.  
Koszt początkowy trasy wynosi:  
15+20+25+30=90

* **Początkowe rozwiązanie (currentSolution)**: **C → A → D → B → C**
* **Koszt początkowy (currentCost)**: 90
* **Najlepsze rozwiązanie (bestSolution)**: **C → A → D → B → C**
* **Koszt najlepszego rozwiązania (bestCost)**: 90

**Krok 2: Wybór sąsiedniego rozwiązania**

Generujemy sąsiednie rozwiązanie, zmieniając kolejność dwóch losowych miast w trasie. Na przykład, zamieńmy pozycje A i D w trasie:

**Nowe rozwiązanie (neighborSolution):**  
**C → D → A → B → C**.  
Koszt nowej trasy wynosi:  
35+20+10+30=95.

**Krok 3: Akceptacja sąsiada**

* Różnica kosztów: ΔE=95−90=5.
* Warunek akceptacji: Akceptujemy nowe rozwiązanie, jeśli:
  + ΔE<0 (koszt maleje), lub
  + e^(−ΔE/T)>losowa liczba z [0,1]

Dla T=100:  
e^(−5/100)≈0.951.  
Jeśli wylosowana liczba wynosi np. 0.800, akceptujemy nowe rozwiązanie.

* **Zaktualizowane currentSolution:** **C → D → A → B → C**
* **Zaktualizowany currentCost:** 95.

**Krok 4: Aktualizacja najlepszej trasy i temperatury**

* Ponieważ 95>90, najlepsze rozwiązanie (bestSolution) nie zmienia się.
* Aktualizujemy temperaturę według wzoru T=T⋅α:  
  T=100⋅0.9=90.

**Krok 5: Kolejne iteracje**

W kolejnych iteracjach algorytm generuje nowe rozwiązania i decyduje o ich akceptacji na podstawie zmniejszającej się temperatury. Przykładowe kroki:

**Iteracja 2:**  
**Nowe rozwiązanie:**  
**C → B → A → D → C**  
Koszt: 30+10+20+35=95.  
ΔE=95−90=5.  
e^−5/90≈0.945.  
Jeśli wylosowana liczba wynosi 0.700, akceptujemy.

* **currentSolution:** **C → B → A → D → C**
* **currentCost:** 95.
* Temperatura: T=90⋅0.9=81.

**Iteracja 3:**  
**Nowe rozwiązanie:**  
**C → A → B → D → C**  
Koszt: 15+30+25+35=105.  
ΔE=105−95=10.  
e−10/81≈0.878.  
Jeśli wylosowana liczba wynosi 0.850. akceptujemy.

* **currentSolution:** **C → A → B → D → C**
* **currentCost:** 105.
* Temperatura: T=81⋅0.9=72.9.

**Krok 6: Osiągnięcie najlepszego rozwiązania**

W miarę ochładzania algorytm coraz rzadziej akceptuje gorsze rozwiązania. Po wielu iteracjach (np. 200) może osiągnąć najlepsze rozwiązanie:

**Ostateczne rozwiązanie:**  
**C → A → B → D → C**  
Koszt: 15+10+25+35=85.

**Wynik końcowy:**

* **Najlepsza trasa (bestSolution):** **C → A → B → D → C**
* **Najlepszy koszt (bestCost):** 85.
* **Liczba iteracji:** 200.
* **Temperatura końcowa:** 0.10.