Temat:  
  
Optymalizacja procesu produkcyjnego metodą zachłanną (JW2).  
  
  
Zespół:  
Maciej Szerling  
Adrian Wiśniewski  
  
Prowadzący:  
prof. dr hab. Jacek Wojciechowski

## Zadanie projektowe.

Fabryka produkująca samochody oferuje n kolorów karoserii. Koszt wyczyszczenia i przestawienia linii do malowanie z koloru i-tego na j-ty wynosi cij (np. koszt przestawienia linii z koloru czarnego na biały może być wyższy od kosztu przestawienia z białego na czarny). Zakładamy, że możemy rozpocząć malowanie od dowolnego koloru i przechodząc przez wszystkie oferowane kolory wracamy do koloru startowego, a następnie powtarzamy cykl malowania. Kosztem pojedynczego cyklu jest suma kosztów cij przejścia przez kolejne kolory i powrotu do koloru startowego. Znaleźć najtańszą sekwencję malowania samochodów.   
  
Zastosować metodę zachłanną i jako referencję porównać z metodą pełnego przeglądu. Napisać odpowiedni program komputerowy (włączając w to generator danych). Porównać jakość rozwiązań otrzymanych obydwoma metodami. Zbadać zależność czasu rozwiązania od liczby kolorów n dla metody Greedy Edges, Greedy Vertices oraz pełnego przeglądu.

## Wybór i opis algorytmu.

Nasze zadanie jest w rzeczywistości problemem typu TSP. Możemy więc zastosować algorytmy używane przy rozwiązywaniu tego typu problemów. Konkretna konfiguracja linii produkcyjnej to wierzchołek grafu a ceny przestawienia z jednej konfiguracji na drugą to koszty krawędzi.   
W naszym projekcie mieliśmy z góry narzucony zbiór algorytmów które mieliśmy użyć.

### Algorytmy zachłanne

Algorytmu zachłanne są niezwykle proste. Ich działanie opiera się o zasadę: “Wybierz najlepsze rozwiązanie dostępne w tej chwili”.

#### Algorytm “Greedy Vertices”

Najbardziej intuicyjnym algorytmem jest algorytm bazujący na wybieraniu w kolejnym kroku następnego, najbliższego wierzchołka, który nie został jeszcze odwiedzony, do momentu aż odwiedzimy wszystkie wierzchołki.   
  
Algorytm “Greedy Edges”

Kolejnym algorytmem, który jest niezwykle intuicyjny i prosty. Polega na konstruowaniu cyklu w taki sposób, że wybieramy zawsze najkrótsze dostępne połączenie (krawędź) między dwoma wierzchołkami, unikając sytuacji w której wierzchołek jest połączony w więcej niż 2-ma innymi wierzchołkami. Robimy tak o momentu stworzenia pełnego cyklu.

### Algorytm pełnego przeglądu “Full Search”

Rozwiązanie to polega na przejrzeniu wszystkich dostępnych możliwości i wybraniu z nich najlepszego. Algorytm pozwala nam na znalezienie minimum globalnego, ale jego złożoność ogranicza jego zastosowanie jedynie do problemów bardzo małych rozmiarów.

## Program do analizy

### Wejście.

Na wejściu naszego programu podajemy:  
- plik w formacie testowym zawierający informacje o ilości kolorów oraz koszcie przestawienia lakierni z jednego koloru na drugi. Przykładowy plik ma strukturę, gdzie kolejne wiersze i kolumny   
oznaczają kolor a wartość w tabeli koszt przestawienia lakierni. (np C(3,1)=3)

0 1 3 4

1 0 3 4

3 2 0 1

2 2 1 0

- wykorzystywany algorytm  
- formatter uzyskanych wyników  
- ilość powtórzeń jaką ma się wykonać każdy algorytm  
- seed który posłuży do zainicjowania generatora pseudolosowego

### Wyjście.

Na wyjściu możemy otrzymać wyniki w trzech formatach:  
- tekstowym, w formacie:

*[Algorithm] ([loops] - [avg time]: [min time] : [max time]*

*[Solution cost]*

*[Solution path]*

- w postaci wykresu 3d  
- w prostej tekstowej postaci, format ten jest używany do późniejszego przetwarzania.

## Rozwiązanie przykładowego zadania optymalizacyjnego.

Korzystając z naszego programu rozwiążmy problem rozmiaru 10 dla kosztów danych tabelą:

['x', 8.4, 7.6, 2.6, 5.0, 4.5, 6.5, 7.9, 1.0, 0.3]

[8.3, 'x', 7.6, 0.1, 4.5, 7.2, 2.3, 9.4, 9.0, 0.4]

[0.3, 5.4, 'x', 3.8, 2.2, 4.2, 0.3, 2.2, 4.4, 5.0]

[2.4, 2.3, 2.2, 'x', 2.9, 0.3, 8.3, 5.6, 6.4, 1.9]

[9.9, 8.6, 1.2, 3.3, 'x', 7.1, 9.3, 4.2, 8.3, 6.7]

[3.1, 5.9, 8.8, 8.4, 5.1, 'x', 0.4, 2.5, 7.9, 4.2]

[1.8, 5.5, 7.0, 6.7, 3.8, 4.4, 'x', 7.8, 5.2, 3.9]

[4.9, 0.3, 0.5, 7.0, 9.8, 5.9, 3.9, 'x', 5.0, 9.8]

[7.7, 5.4, 8.6, 2.3, 5.1, 9.5, 5.8, 4.6, 'x', 5.5]

[9.5, 0.1, 7.8, 8.2, 8.8, 7.4, 8.1, 5.2, 5.6, 'x']

Otrzymane rozwiązania (uśrednione dla 10 pętli):

|  |  |  |  |
| --- | --- | --- | --- |
|  | Greedy Edges | Greedy Vertices | Full Search |
| Rozwiązanie problemu | [8, 7, 2, 0, 9, 1, 3, 5, 6, 4] | [7, 1, 3, 5, 6, 0, 9, 8, 4, 2] | [0, 8, 9, 1, 3, 5, 6, 4, 7, 2] |
| Czas rozwiązania problemu [ms] | 0.115262768189 | 0.573561398662 | 25997.1340619 |
| Koszt rozwiązania | 18.7 | 17.3 | 16.2 |
| O ile gorsze od optymalnego | 15,4% | 10% | 0% |

Analizując powyższą tabelę możemy dojść do następujących wniosków:

* Najlepsze rozwiązanie otrzymaliśmy przy wykorzystaniu metody Full Search. Jest to rozwiązanie optymalne. Obliczenie tego rozwiązanie trwało jednak dziesiątki tysięcy razy dłużej niż w przypadku pozostałych metod.
* Drugie w kolejności pod względem optymalności było rozwiązanie otrzymane za pomocą algorytmu Greedy Vertices. Było o 10% gorsze od rozwiązania optymalnego.
* Najgorsze pod względem optymalności, ale za to najszybciej obliczone jest rozwiązanie uzyskane przy użyciu algorytmu Greedy Edges.

#### Rozwiązanie przedstawione w postaci 3d:

#### 

Rysunek : Greedy Edges

#### 

Rysunek : Greedy Vertices

#### 

Rysunek : Full Search

## Badania

### Platforma testowa

Wszystkie badania zostały wykonane na komputerze klasy PC o następującej konfiguracji:  
Procesor: AMD Athlon II X4 640 3.00Ghz  
Pamięć: 4.0 GB  
System: Windows 7 Professional 64bit  
Do badania została użyta aplikacja napisana w języku Python.

Badania zostały przeprowadzone badając 10 różnych przypadków (10 różnych seedów generatora liczb losowych) dla każdej liczby kolorów karoserii. Następnie wyniki scalono i uśredniono.

### Porównanie jakości rozwiązań dla badanych algorytmów

### 

Analizując powyższą tabelę możemy dojść do następujących wniosków:

* Najlepsze rozwiązania daje zawsze algorytm FullSearch , lecz ze względu na jego złożoność nie jesteśmy w stanie osiągnąć wyników w zadowalającym czasie dla problemu o rozmiarze większym niż 11 kolorów
* Porównując algorytmy heurystyczne to dla naszych danych to pod względem optymalności rozwiązania zawsze lepszy okazuje się być algorytm Greedy Vertices

### Badanie czasu rozwiązania w zależności od liczby kolorów karoserii.

### 

Rysunek : Wyniki w postaci wykresu w skali logarytmicznej

Rysunek : Wynik w postaci wykresu w skali liniowej

Wnioski:

Analizując dane zebrane w tabeli oraz wykresy możemy dojść do następujących wniosków:

* Dla wyszukiwania „FullSearch” nie jesteśmy uzyskać wyników w zadowalającym czasie już dla problemów większych niż 11 kolorów
* Algorytm FullSearch jest bardzo wolny (złożoność n!)
* Czas rozwiązania przy użyciu algorytmu Greedy Edges rośnie liniowo wraz ze wzrostem ilości wierzchołków
* Czas rozwiązania zadania przy użyciu algorytmu Greedy Vertices rośnie proporcjonalnie do kwadratu liczby kolorów

## Wnioski końcowe

Z przeprowadzonego przez nas wynika, że z pośród badanych algorytmów nie możemy wybrać jednego „najlepszego”. Wszystko zależy od kilku czynników: jak duży problem chcemy rozwiązać, jak bliskich optymalnemu rozwiązań oczekujemy, jakimi mocami obliczeniowymi i czasem na rozwiązanie zadania dysponujemy. Dla małych problemów wybierzemy FullSearch dla większych Greedy Vertices a dla ogromnych Greedy Edges.

## Wykaz literatury.

Z.Michalewicz, D. Fogel, Jak to rozwiązać, czyli nowoczesna heurystyka