Zastosowanie UCT w problemach transportowych ze zmiennym zakorkowaniem ulic

MACIEJ ŚWIECHOWSKI

PROMOTOR: PROF. JACEK MAŃDZIUK

TEMAT W RAMACH GRANTU NCN NA PODSTAWIE DECYZJI DEC-2012/07/B/ST6/01527

Plan prezentacji

- Definicja problemu CVRP
- Obszerna definicja DCVRP
- Zakorkowanie

- Proponowane rozwiązanie
- Metody porównawcze

- Wyniki
- Podsumowanie

Capacitated Vehicle Routing Problem (CVRP)

CVRP (wersja bazowa)

- Pełny graf **N** lokalizacji z wyszczególnioną lokalizacją **zajezdni**
- Każda lokalizacja z wyjątkiem zajezdni to klient, który wymaga dostarczenia **K > 0**, **K**$\in \mathbb{N}$ jednostek pewnego towaru (jeden typ towaru)
- Ciężarówki mają ustaloną globalną pojemność $\boldsymbol{C} \geq \mathbf{K}, \mathbf{C} \in \mathbb{N}$
- Dostarczenie towaru to odwiedziny przez **ciężarówkę**
- Zamówienia sią **niepodzielne**, czyli muszą być zrealizowane w jednej wizycie
- Każdy klient może zostać odwiedzony tylko raz
- Ciężarówki startują z **zajezdni** i po zrealizowaniu zamówień muszą wrócić do zajezdni
- Zwykle zdefiniowana jest maksymalna liczba dostępnych ciężarówek

CVRP

Cel:

Zrealizowanie **wszystkich zamówień** przy jednoczesnej **minimalizacji** sumarycznej **długości** tras ciężarówek.

CVRP to znany problem optymalizacji kombinatorycznej leżący na przecięciu dyskretnego problemu **plecakowego** oraz **TSP** (Travelling Salesman Problem).

- Posiada liczne warianty, jeden z nich jest tematem seminarium
- Sam jest wariantem **VRP**

DCVRP → VRPwT

DYNAMIC **C**APACITATED **V**EHICLE ROUTING **P**ROBLEM

VEHICLE **R**OUTING **P**ROBLEM **W**ITH **T**RAFFIC

Zakorkowanie ulic

Problem przestaje być statyczny

- Wymaga symulacji do zrealizowania i oceny wyniku
- Statyczny problem często można ocenić wizualnie

Problem przestaje być deterministyczny

- Nie można ocenić jakości rozwiązania bez znajomości realizacji wszelkich zmiennych losowych
- Na tej samej mapie, identyczne trasy mogą być raz dobre a raz złe

Jaką dokładność symulacji przyjąć?

Jak wykorzystać istniejące benchmarki, jeżeli wprowadzimy za dużo własnych zmian?

Założenia

Rozwiązanie jest tworzone i ewaluowane podczas symulacji krokowej.

Stan przed aktualizacją:

- Pewien stan zakorkowania
- Każda ciężarówka, która nie jest w zajezdni musi posiadać **kurs** (plan na najbliższy ruch)
  (rozwiązanie bez tej własności jest nielegalne lub można losować jej legalny kurs)
- Ciężarówka, która nie jest w zajezdni, nie może pozostać w miejscu

Aktualizacja:

- Ciężarówki są przesuwane zgodnie ze swoim kursem
- Przejazd między lokalizacjami to operacja atomowa
- Co najmniej jedna ciężarówka musi się ruszyć (**ang**. *net change*)
  (przeciwna sytuacja oznacza koniec symulacji)
- Aktualizacja zakorkowania

Założenia

Rozwiązanie jest tworzone i ewaluowane podczas symulacji krokowej.

Stan przed aktualizacją:

- Pewny stan zakorkowania
- Każda ciężarówka, która nie jest w zajezdni musi posiadać **kurs** (plan na najbliższy ruch)
  (rozwiązanie bez tej własności jest nielegalne lub można losować jej legalny kurs)
- Ciężarówka, która nie jest w zajezdni, nie może pozostać w miejscu

Aktualizacja:

- Ciężarówki są przesuwane zgodnie ze swoim kursem
- Przejazd między lokalizacjami to operacja atomowa
- Co najmniej jedna ciężarówka musi się ruszyć (**ang**. *net change*)
  (przeciwna sytuacja oznacza koniec symulacji)
- Aktualizacja zakorkowania

Zakorkowanie ulic

Funkcja zakorkowania:

- ***P*** - prawdopodobieństwo wystąpienia korka na krawędzi
- ***L*** - długość trwania korka w krokach z rozkładu jednostajnego dyskretnego
- ***I*** - intensywność trwania korka z rozkładu jednostajnego dyskretnego

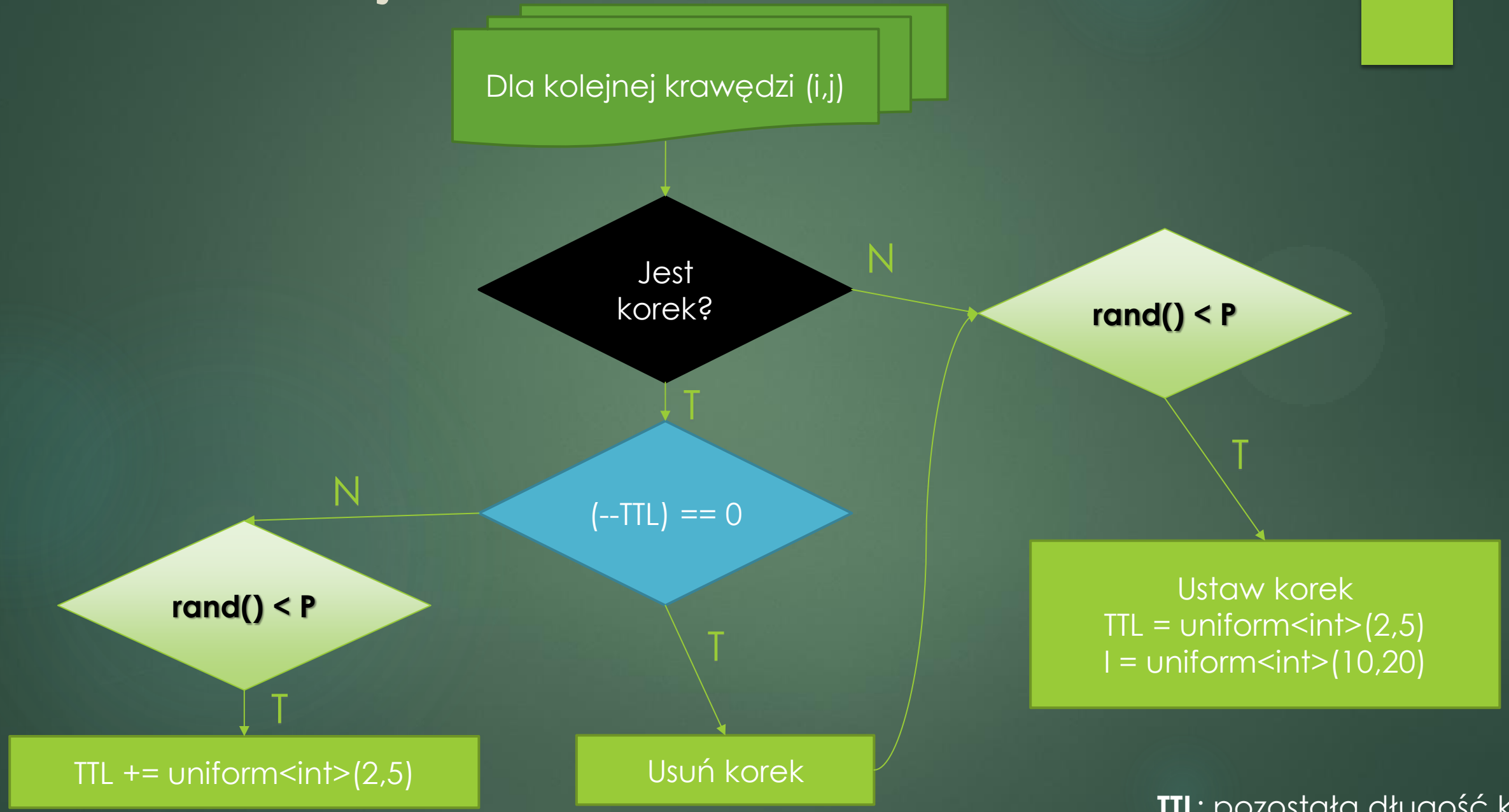
Efekt zakorkowania:

Przez ***TTL*** = ***L*** kroków efektywna odległość między miastami wynosi:

I * odległość_euklidesowa

taka odległość liczy się do jakości rozwiązania

Aktualizacja zakorkowania – ver 1.0



TTL: pozostała długość korka

Aktualizacja zakorkowania – ver 2.0

Przechowujemy indeksy zakorkowanych krawędzi w kubełkach etykietowanych **TTL**

Zmniejszenie czasu trwania to tylko zmiana funkcji poszczególnych kubełków i wyzerowanie kubełka z etykietą TTL = 1.

Nie iterujemy po krawędziach, a generujemy indeksy zakorkowanych krawędzi w liczbie:

$|E|(P + rand(-0.005, 0.005))$

Zamiana:

TTL += uniform<int>(2,5) na TTL = min(5, TTL + uniform<int>(2,5))

Proponowana Metoda UCT

Początkowy sceptycyzm wobec UCT w VRP

- Zbyt duży wymiar problemu, aby UCT zastosować bezpośrednio do konstrukcji rozwiązania

- UCT znam z podejść, gdzie celem jest wybranie akcji == lokalna optymalizacja w danym momencie np. wybór najbardziej opłacalnego jednorękiego bandyty lub **wybór akcji w grze**
- W problemach VRP mamy globalną optymalizację – to nie jest sekwencyjny wybór pewnych atomowych akcji
  - mamy wiele tras
- Problem wydaje się być na tyle mocno zdefiniowany, żeby stosować metody wykorzystujące wiedzę domenową

Rozwiązanie początkowe

- Generujemy **rozwiązanie początkowe** dla statycznego problemu, które będzie bazą do modyfikacji przez algorytm UCT
- Można wybrać dowolny algorytm (!), który da rozwiązanie szybko (chcemy wykorzystać czas na symulacje MCTS)
- Stosujemy **Parallel Clarke & Wrights Savings** [1]

[1] ***An improved Clarke and Wright savings algorithm for the capacitated vehicle routing problem***, T. Pichpibul and R. Kawtummachai, Science Asia, 38 (2012), 307-318, 2012

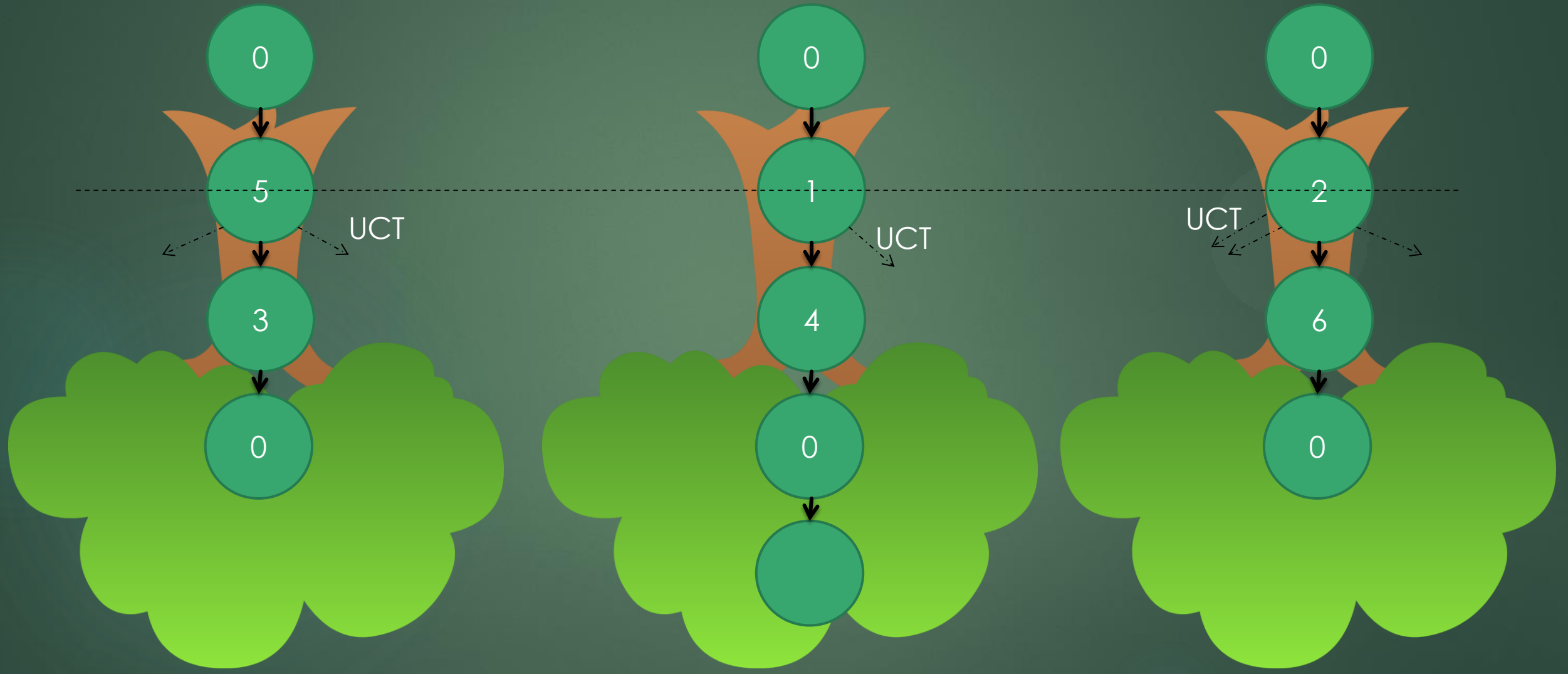
Synchronizowany las UCT

- Każda wygenerowana trasa stanowi bazę do stworzenia **drzewa UCT**
- **Wierzchołek:** lokalizacja
- **Korzeń**: początek trasy (t_0 = zajezdnia)
- **Liść**: koniec trasy – zajezdnia
- Wierzchołki są połączone krawędzią, jeżeli jest bezpośredni przejazd między lokalizacjami

Początkowo drzewo dla każdej ciężarówki to ścieżka.

Synchronizowany – wykorzystujemy fakt, że symulacja jest dyskretna i zsynchronizowana. Wierzchołki na odpowiadających poziomach w drzewach dotyczą tej samej sytuacji w czasie (tego samego kroku symulacji).

Synchronizowany Las UCT



Symulacja MC/UCT

- Wykonywana poziomami w Synchronizowanym Lesie UCT
- Przeprowadzana na wszystkich drzewach równocześnie, stan symulacji wynika z wszystkich drzew
- Las jest rozbudowywany w każdym dużym kroku głównej symulacji (dostaje budżet określony przez max liczbę symulacji)
- Krok symulacji pobiera listę wierzchołków **wejściowych (IN)** i zwraca **wyjściowych (OUT)**
- Jeżeli **OUT =** Ø, to koniec symulacji

Symulacja MC/UCT

- Ma dostęp do rozkładów prawdopodobieństwa korków, lecz nie ostatecznych realizacji korków wykorzystanych w weryfikacji (grze)
- Rozpoczyna od ostatnio obserwowanego korka głównej symulacji

- Dla każdego wierzchołka drzewa wyznaczane są legalne akcje bazując na bieżącym zakorkowaniu (wyjaśnienie później)
- Dla każdej akcji obliczana jest wartość **UCT**
- Drzewa sortowane są względem malejącej wartości **UCT** ich bieżących wierzchołków

- Akcje są realizowane – co prowadzi do przejścia do nowych wierzchołków

```
TreeSimulationStep()
{
    traffic = ResolveTraffic();
    foreach(UctNode node in inNodes)
    {
        legalActions = actionPreparer.ComputeLegal(node, inNodes, traffic);
        node.SelectedAction = UCT(legalActions);
    }
    inNodes.Sort(UCTComparer);
    outNodes.Clear();

    foreach(UctNode node in inNodes.Where(node => node.NonTerminal)
    {
        if(node.SelectedAction.LinkedAction != null) //akcja ma wpływ na inny węzeł
            SynchronizeActions(node, node.SelectedAction.LinkedAction.ToNode);
        ApplyAction(node.SelectedAction);
        if(node.SelectedAction.ResultNode.CityID != 0)
            outNodes.Add(node.SelectedAction.ResultNode);
        TotalCost += GetDistance(node.CityID, node.SelectedAction.ResultNode.CityID);
    }
    if(outNodes.Count == 0)
     return;
    swap(inNodes, outNodes);
}
```

Wzór UCT

- ***Q(s,a)*** **–** średnia sumaryczna długość tras przy wybraniu akcji ***a*** w stanie ***s***
- ***A(s)*** **–** akcje legalne w stanie ***s***
- **Q** aktualizowane po zakończonej symulacji (pamiętane są ścieżki)
- **C** ustawione po krótkim testowaniu na długość początkowego rozwiązania

$$a^* = arg \max_{a \in A(s)} \left\{ C \sqrt{\frac{\ln N(s)}{N(s,a)}} - Q(s,a) \right\}$$

Ten wzór pochodzi od wzoru ***UCB1*** i degeneruje się do niego w przypadku drzewa o głębokości **1** (z jednym poziomem wyboru).

Warianty UCB

$$index^{\text{UCB1}}(H_{t-1}^k, t) = \overline{r}_k + \sqrt{\frac{C \ln t}{t_k}} \tag{2}$$

$$index^{\text{UCB1-Tuned}}(H_{t-1}^k, t) = \overline{r}_k + \sqrt{\frac{\ln t}{t_k} \min\left(1/4, \overline{\sigma}_k + \sqrt{\frac{2 \ln t}{t_k}}\right)} \tag{3}$$

$$index^{\text{UCB1-Normal}}(H_{t-1}^k, t) = \overline{r}_k + \sqrt{16 \frac{t_k \overline{\sigma}_k^2}{t_k - 1} \frac{\ln(t-1)}{t_k}} \tag{4}$$

$$index^{\text{UCB-V}}(H_{t-1}^k, t) = \overline{r}_k + \sqrt{\frac{2 \overline{\sigma}_k^2 \zeta \ln t}{t_k}} + c \frac{3 \zeta \ln t}{t_k} \tag{5}$$

where $\overline{r}_k$ and $\overline{\sigma}_k$ are the mean and standard deviation of the rewards so far obtained from arm k and t_k is the number of times it has been played.

Stosowane są również metody: UCB2, Exp3, PBBM oraz ε-Greedy.

[***Meta-Learning of Exploration/Exploitation Strategies: The Multi-Armed Bandit Case***, F. Maes, D. Ernst and L. Wehenkel, CoRR, Springer Selection of papers of ICAART'12, 2012]

Dostępne akcje

Istotny jest **warunek legalności**. Akcja możliwa jest do wybrania w UCT oraz podczas prawdziwej (głównej) symulacji rozwiązania tylko spełnieniu tego samego warunku.

Klasyfikacja akcji:

- Modyfikujące 0 tras
- Modyfikujące 1 trasę
- Modyfikujące 2 trasy

Akcje 0

Jedź dalej po zaplanowanej (niezakorkowanej) trasie	**[ID=0]**
• Trasa nie rozpoczyna się zakorkowaną krawędzią	

Jedź dalej po zaplanowanej (zakorkowanej) trasie	**[ID=1]**
• Trasa rozpoczyna się zakorkowaną krawędzią	

Fizyczna realizacja jest taka sama, ale akcje są **rozróżniane**.

Warunki na legalność są wykluczające i dopełniające się – dzięki temu zawsze istnieje co najmniej 1 legalna akcja jak w dobrze zdefiniowanej grze.

Akcje 1

Przesuń pierwszego klienta na koniec, tuż przed zajezdnię **[ID=2]**

- Trasa rozpoczyna się korkiem
- Nowa trasa nie rozpoczyna się korkiem (z bieżącego miejsca do uprzednio ostatniego klienta)

Przesuń kolejnego klienta w miejsce chwilowo optymalne **[ID=3]**

- Trasa rozpoczyna się zakorkowaną krawędzią

Niech początek trasy to klient A
Szukamy takich sąsiadujących na trasie klientów B i C (B≠A oraz C≠A), aby minimalizować odległość:
|BA| + |AC| - |BC|

Wstawiamy A między B i C.

Akcje 1

Jako następnego klienta wstaw pierwszego z trasy, na drodze do którego nie ma korka **[ID=4]**

- Trasa rozpoczyna się korkiem
- Z bieżącego miejsca do znalezionego klienta korka nie ma

Wstawiamy nowego klienta przed najbliższego

Jako następnego klienta wstaw najbliższego licząc dynamiczną odległość **[ID=5]**

- Trasa rozpoczyna się korkiem
- Trasa nie jest oryginalnie zaplanowaną, czyli rozstała co najmniej raz zmodyfikowana

Niech bieżące miejsce to O. Szukamy takiego A, które minimalizuje $|OA|$

Odwróć trasę pozostawiając na końcu zajezdnię **[ID=6]**

- Trasa rozpoczyna się korkiem
- Po odwróceniu, trasa nie rozpoczyna się korkiem

Akcje 2

Zakończ bieżącą trasę i utwórz nową z pozostałych klientów **[ID=7]**

- Z bieżącej pozycji ciężarówki istnieje korek do każdego klienta na trasie
- Droga z bieżącej pozycji do zajezdni jest niezakorkowana
- Droga z zajezdni do pierwszego klienta jest niezakorkowana
- Jeżeli limit ciężarówek nie został przekroczony, to nowa ciężarówka startuje od razu – jeżeli został, to bieżąca trasa jest sztucznie przedłużana przez zajezdnię po drodze.

Wymień klientów między dwiema trasami w wersji MIN **[ID=8]**

- Jedna, bieżąca rozpatrywana, trasa rozpoczyna się korkiem
- Po wymianie obie trasy nie rozpoczynają się korkiem
- Zachowane są warunki na ładowność

Wymień klientów między dwiema trasami w wersji MAX **[ID=9]**

- Jedna, bieżąca rozpatrywana, trasa rozpoczyna się korkiem
- Po wymianie obie trasy nie rozpoczynają się korkiem
- Zachowane są warunki na ładowność

Algorytm wymiany tras

Załóżmy, że rozpatrywana trasa **R1** a kandydat do wymiany to **R2**:

R1 = [20, 31, 7, 29, 4, 0] R2 = [11, 12, 25, 35, 0]

Rozpatrujemy 4 możliwości i wybieramy najkrótszy legalny przejazd:

Algorytm wymiany tras

MIN

- Wystartuj od oddania 1 wierzchołka z zakorkowanej trasy
- Jeżeli nie są przekroczone pojemności ciężarówek, to: **ZAKOŃCZ**
- Kontynuuj balansując pojemności – oddawaj wierzchołki z bardziej przepełnionej

MAX

- Wystartuj od oddania 1 wierzchołka z zakorkowanej trasy
- Jeżeli nie są przekroczone pojemności ciężarówek, to: **ZAPAMIĘTAJ ROZWIĄZANIE**
- Kontynuuj balansując pojemności – oddawaj wierzchołki z bardziej przepełnionej

Wyniki eksperymentalne

Dostępnych 5 podejść:

1. Proponowany algorytm UCT
2. Rozwiązanie statyczne (początkowe)
3. Metoda Heurystyczna 1
4. Metoda Heurystyczna 2
5. Algorytm Mrówkowy

Testy (900 gier każdego gracza):

- 6 benchmarków
- 3 prawdopodobieństwa korków
- 50 powtórzeń dla każdej instancji

Podejście Heurystyczne 1 (H1)

Realizujemy rozwiązanie statyczne krok po kroku, z zastrzeżeniem, ze jeżeli w kolejnym kroku trafiamy na korek, to weryfikujemy 2-OPTem możliwość zamiany par krawędzi (w ekstremalnej sytuacji - gdy korek jest mały albo inne miasta są daleko położone - możemy tez przejść po krawędzi zakorkowanej).

Po zamianie krawędzi musimy odwrócić kolejność w części rozwiązania, tzn. jeżeli mamy ponumerowane miasta 1, 2, ..., N w danym cyklu ciężarówki (N=depot) to rozwiązanie składa się z N odcinków N-1,1-2, 2-3, ..., N-1 - N. Odcinek k-ty prowadzi do miasta o numerze k.

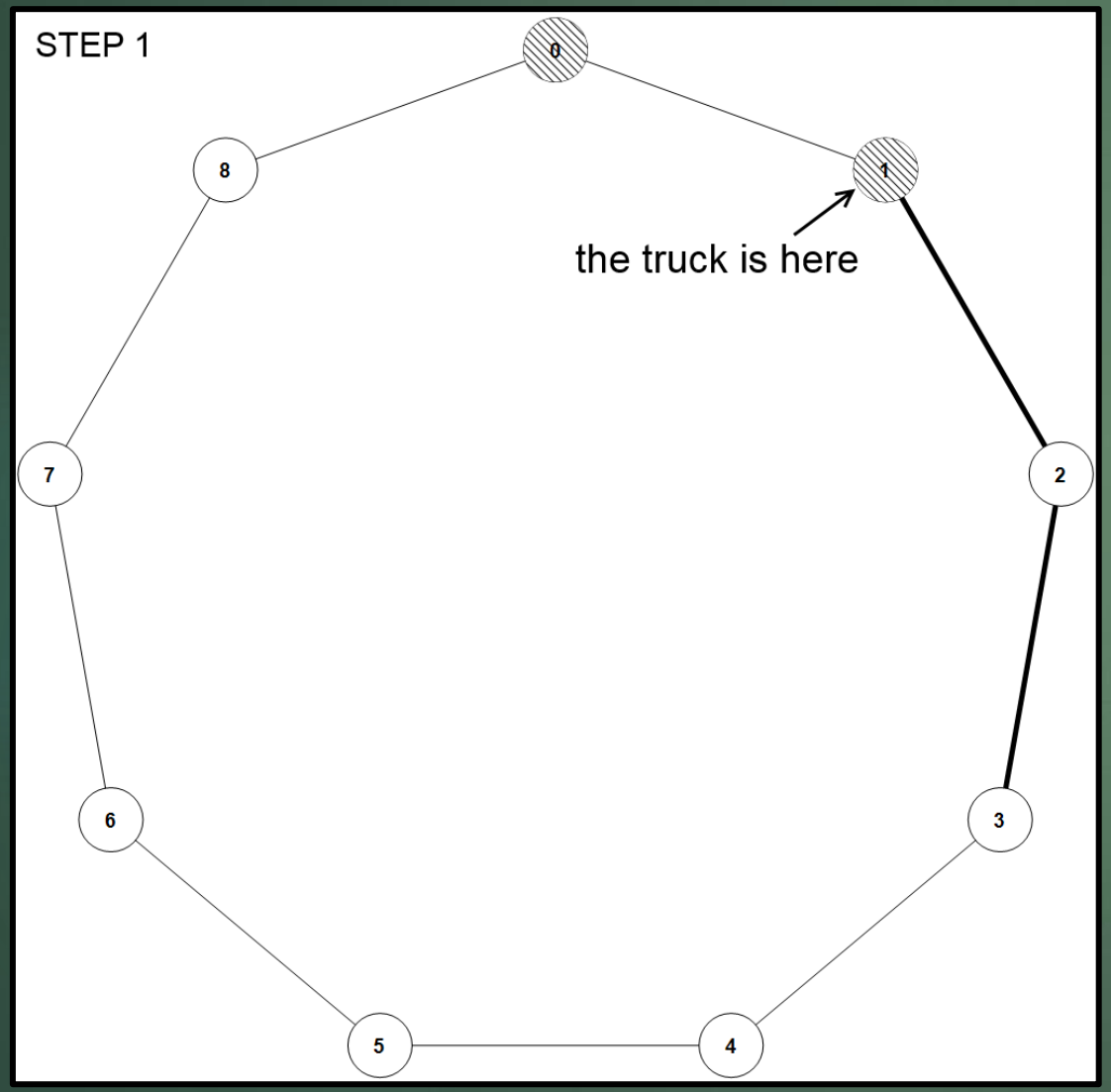
Załóżmy, ze wykonaliśmy dwa kroki i teraz w trzecim (2-3) mamy korek (i inne korki oczywiście także). Szukamy takiego miast m, dla którego wartość "odległości 2-m + odległości 3- m+1" jest minimalna (w szczególności m=3 czyli przejście po zakorkowanej krawędzi jest dopuszczalne).

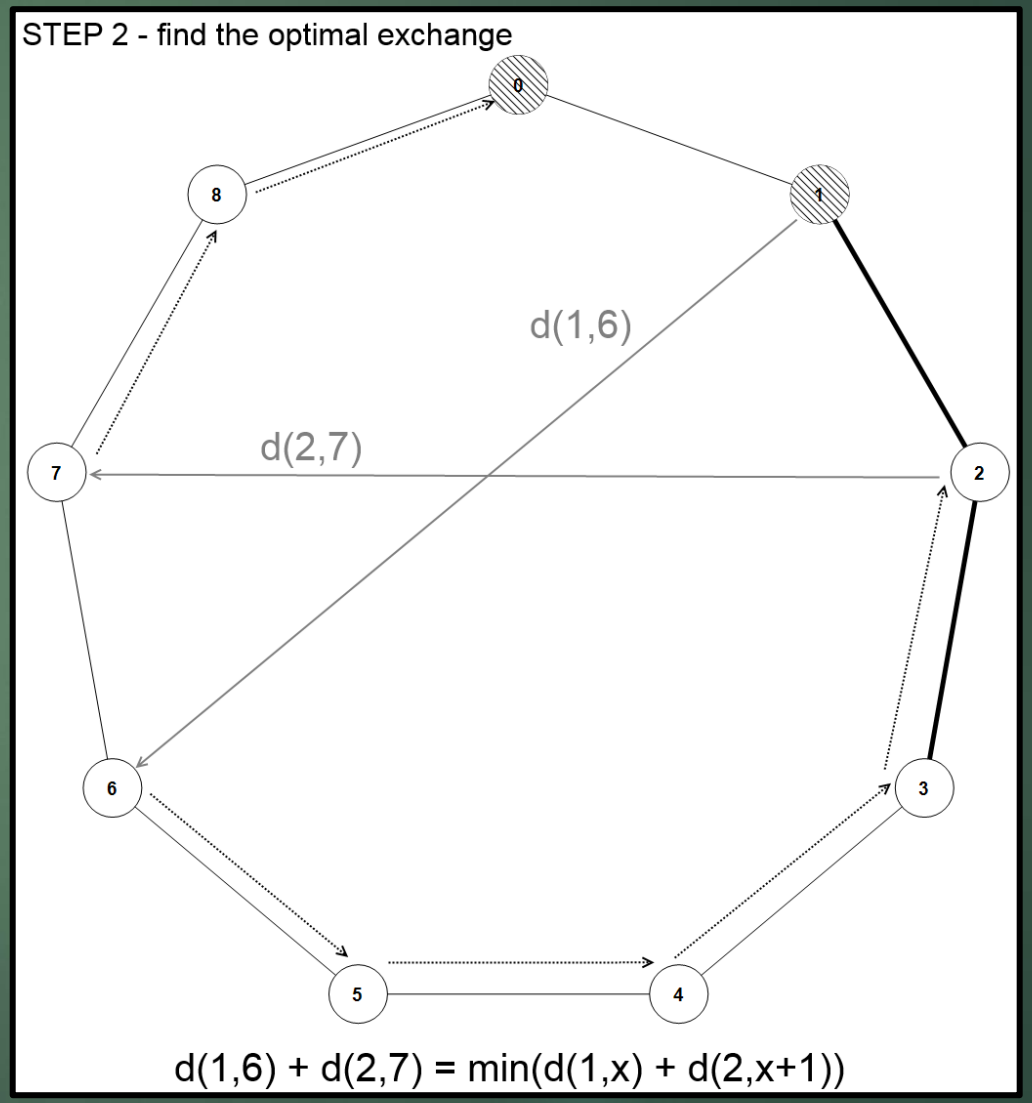
Nasze nowe rozwianie ma następująca kolejność miast

(N)-1-2- m - m-1 - m-2 ... - 3 - m+1 - m+2 ... N

czyli we fragmencie rozwiązania od m do 3 odwracamy kolejność (tak jak w klasycznym 2-OPT).

Podejście Heurystyczne 1 (H1)





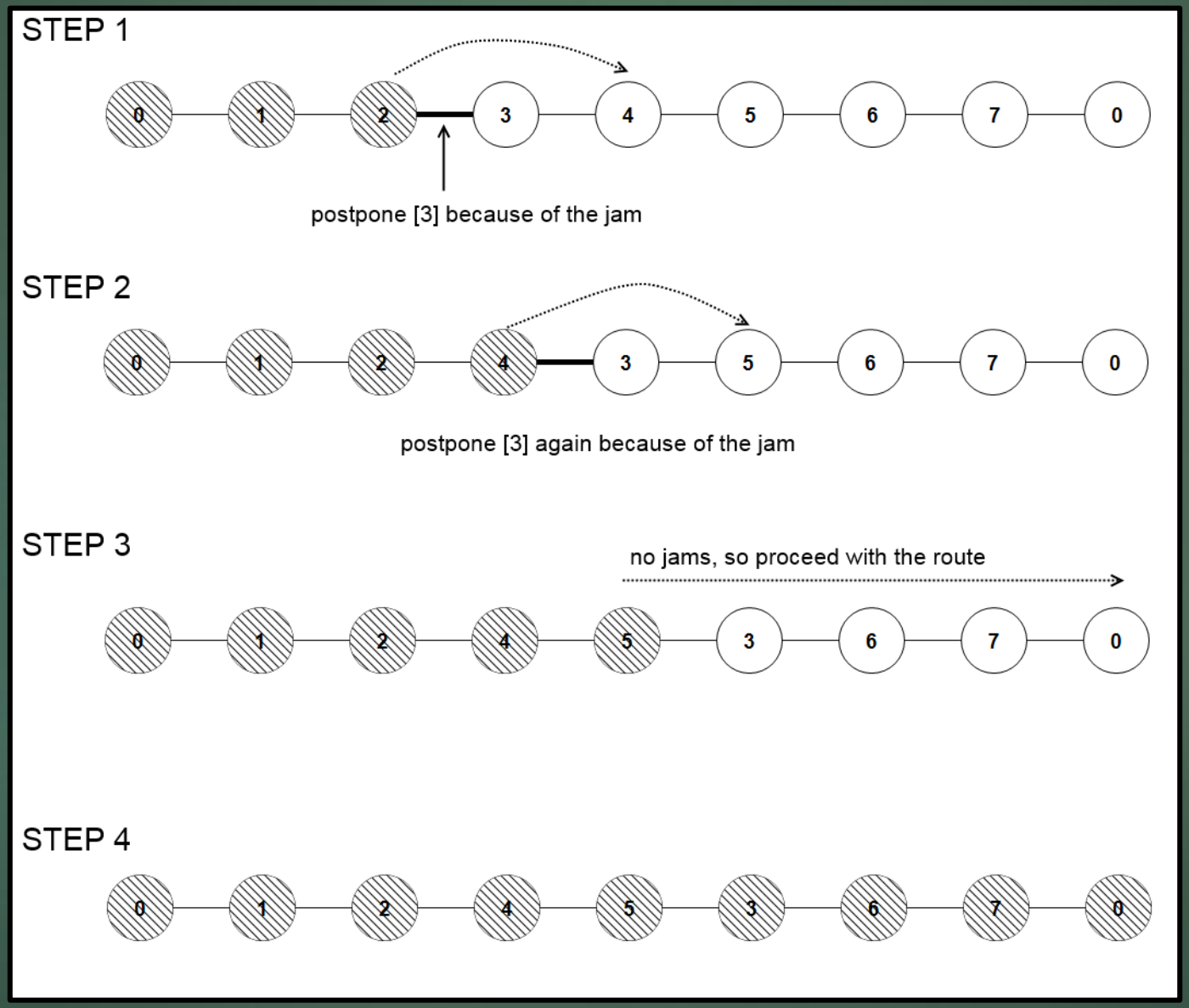
Podejście Heurystyczne 2 (H2)

Jeżeli mamy korek 2-3 to próbujemy przejść do następnego miasta (2-4) - o ile nie ma tam korka - natomiast do 3 wracamy w kolejnym kroku (najszybciej jak to będzie możliwe). W tym wypadku byłaby to trasa

(N)-1-2-4-3-5-...-N

przy założeniu, ze nie ma korków 2-4, 4-3 i 3-5. Technicznie: ominięte miasta (w tym wypadku 3) wstawiamy do kolejki i zanim przejdziemy do kolejnego miasta próbujemy po kolei czy nie da się obsłużyć, któregoś z pominiętych miast (sprawdzamy je kolejno zgodnie z kolejnością włożenia do "poczekalni").

Podejście Heurystyczne 2 (H2)



Benchmarki

Dodaliśmy dynamiczne zakorkowanie wygenerowane funkcjami:
P = {0.15, 0.05, 0.02} (prawdopodobieństwo wystąpienia)
L = uniform_distribution<int>(2,5) (długość trwania)
I = uniform_distribution<int>(10,20) (intensywność)

Nazwa	Liczba miast	Skrót
P-n101-681-8000	101	P101
A-n80-k10	80	A80
A-n69-k9	69	A69
A-n54-k7	54	A54
P-n45-k5-510-6000	45	P45
P-n19-212	19	P19

Liczba razy bycia nie gorszym niż najlepszy

Benchmark	%P	Statyczne	H1	H2	UCT
P101	15	0	0	6	**44**
P101	5	0	12	**21**	20
P101	2	0	**25**	24	10
A80	15	0	0	0	**50**
A80	5	0	1	10	**39**
A80	2	0	13	17	**27**
A69	15	0	0	0	**50**
A69	5	0	3	5	**42**
A69	2	0	11	20	**31**

Średnie długości rozwiązań 1/2

Benchmark	%P	Statyczne (σ)	H1 (σ)	H2 (σ)	UCT (σ)
P101	15	6465 (606,9)	2750,9 (689,6)	1758,0 (319,9)	**1379,2 (209,9)**
P101	5	2337 (439,4)	1035,3 (156,0)	972,3 (128,4)	**963,7 (73,8)**
P101	2	1538 (283,6)	861,1 (85,9)	**848,0** (84,3)	863,2 **(51,0)**
A80	15	9073 (1527,8)	5765,3 (851,4)	4549,0 (816,7)	**2594,3 (218,3)**
A80	5	4434,0 (976,7)	2865,7 (498,6)	2543,6 (391,4)	**2137,1 (133,4)**
A80	2	2937,6 (689,7)	2231,1 (357,8)	2151,9 (307,2)	**1967,0 (86,1)**
A69	15	6937,0 (1048,7)	4401,1 (945,8)	3394,4 (806,2)	**1815,8 (222,9)**
A69	5	3275,9 (791,6)	2018,1 (415,1)	1849,3 (423,9)	**1436,6 (136,6)**
A69	2	2144,5 (490,3)	1646,8 (406,4)	1505,9 (373,7)	**1284,0 (52,1)**

Liczba razy bycia nie gorszym niż najlepszy

Benchmark	%P	Statyczne	H1	H2	UCT
A54	15	0	0	0	**50**
A54	5	0	2	9	**39**
A54	2	1	11	20	**35**
P45	15	0	0	0	**50**
P45	5	1	9	13	**38**
P45	2	5	23	26	**28**
P19	15	0	9	**26**	20
P19	5	9	27	34	**38**
P19	2	21	30	38	**41**

Średnie długości rozwiązań 2/2

Benchmark	%P	Statyczne	H1	H2	UCT
A54	15	6793,7 (1518,6)	4181,8 (888,4)	3253,4 (823,4)	**1738,2 (220,8)**
A54	5	3218,6 (928,8)	2025,2 (422,9)	1798,2 (369,8)	**1401,8 (125,8)**
A54	2	2006,6 (519,7)	1626,2 (483,0)	1441,4 (345,7)	**1265,3 (66,7)**
P45	15	3650,4 (553,2)	2256,7 (544,9)	1597,5 (402,0)	**800,1 (125,7)**
P45	5	1372,6 (375,4)	835,1 (221,9)	739,9 (144,1)	**623,2 (38,4)**
P45	2	1016,1 (309,2)	736,6 (207,0)	667,6 (147,6)	**607,1 (29,9)**
P19	15	1407,4 (468,4)	869,5 (399,2)	**685,8 (211,5)**	748,2 (381,0)
P19	5	582,9 (258,8)	345,2 (126,8)	307,34 (92,4)	**283,9 (62,0)**
P19	2	461,2 (247,0)	308,7 (91,5)	**285,9 (79,1)**	303,6 (105,7)

Przed kolejnymi testami nastąpiła zmiana liczenia korków (wersja 2)

$|E|(P + rand(-0.005, 0.005))$

TTL = MIN(5, TTL + UNIFORM<INT>(2,5))

Algorytm Mrówkowy

- Zrealizowany w myśl klasycznego algorytmu rozwiązującego Problem Komiwojażera

- Każda mrówka znajduje pełne rozwiązanie problemu, ale w główne symulacji bierzemy jedynie kolejne przejazdy z najlepszego rozwiązania *(potem jest szansa na przeliczenie)*

- Feromon odkładany na krawędziach
- Minimalna wartość feromonu:

  f_min = 0.01 * maksymalna odległość między dwoma miastami

Algorytm Mrówkowy

Dla iteracji i=0…. MAX_ITERATIONS

Dla mrówki m=0….MAX_ANTS

Inicjalizuj mrówkę

Iteracja mrówki

Sprawdź czy rozwiązanie jest najlepsze

Uaktualnij feromon

Wybierz najlepsze rozwiązanie

Algorytm Mrówkowy

Inicjalizacja na samym początku:

- Jeśli krawędź należy do rozwiązania początkowego (z którego zaczyna również UCT), to otrzymuje początkowy feromon = **3*f_min**

Inicjalizacja przed każdym krokiem głównej symulacji (weryfikującej):

- Zerowane jest najlepsze znalezione rozwiązanie
- Równolegle uruchamiane są dwa algorytmy mrówkowe. W jednym, feromon resetowany jest do wartości początkowej.

Inicjalizacja przed każdą iteracją mrówki:

- Pełny bieżacy stan (pozycje ciężarówek, pozostałe pojemności, nieodwiedzone miasta, zakorkowanie) jest synchronizowane ze stanem głównej symulacji.

Algorytm Mrówkowy

Iteracja pojedynczej mrówki:

---------- *ALGORYTM*----------

Pobieraj ciężarówki w losowej kolejności

IF (liczba legalnych miast jest równa zero)

Ustaw zajezdnie jako następny punkt trasy.

ELSE IF (liczba legalnych miast = liczba korków)

Ustaw zajezdnie jako następny punkt trasy.

Utwórz nową ciężarówkę z zajezdni i urucham dla niej metodę kontynuacji trasy

ELSE

Uruchom dla niej metodę kontynuacji dla bieżącej trasy

Algorytm Mrówkowy

Metoda kontynuacji trasy:

Z prawdopodobieństwem równym **0.75** stosujemy ruletkę, w przeciwnym przypadku wybieramy najkrótszą krawędź zachłannie.

Ruletka:

Wybiera spośród legalnych miast (brak wizyty, nieprzekroczona ładowność):

$$p_{ij} = \frac{\tau_{\mathrm{ij}}^{\alpha} * \eta_{\mathrm{ij}}^{\beta}}{\sum\left(\tau_{\mathrm{ij}}^{\alpha} * \eta_{\mathrm{ij}}^{\beta}\right)}, \eta_{ij} = \frac{10f\_min}{d_{ij}}$$

d_{ij}- bieżący koszt krawędzi

$\alpha = \beta = 1$

Algorytm Mrówkowy

Uaktualnienie feromonu następuje po zakończonej iteracji:

$$\tau_{ij}(t+1) = \max(f_{min}, \frac{1}{2}\tau(t)_{ij} + \sum_{m \in Mrówki} (\delta_{ij} * Q_m))$$

δ_{ij} = *0 – krawędź nie należy do rozwiązania danej mrówki*

δ_{ij} = *1 – krawędź należy do rozwiązania, ale nie było ono globalnie najlepsze*

δ_{ij} = *10 – krawędź należy do globalnie najlepszego rozwiązania (mrówka elitarna)*

$Q_m = \frac{10 * f\_min}{D_m}$, D_m - oznacza sumę długości tras rozwiązania znalezionego przez mrówkę

Benchmark	Liczba mrówek (MAX_ANTS)	Liczba iteracji mrówki (MAX_ITERATIONS)
P101	80	50
A80	80	50
A69	200	130
A54	200	130
P45	200	130
P19	400	200

- Wartości dla jednego kroku głównej symulacji (PlayClock w GGP)
- Zgranie liczby symulacji UCT = MAX_ANTS * MAX_ITERATIONS
- Wstępne testy pokazały, że wyniki są lepsze przy faworyzowaniu MAX_ANTS w tym iloczynie

Liczba razy bycia nie gorszym niż najlepszy

Benchmark	%P	Statyczne	Mrówki	UCT
P101	15	0	17	**33**
P101	5	0	15	**35**
P101	2	4	1	**45**
A80	15	0	12	**38**
A80	5	1	8	**41**
A80	2	7	1	**45**
A69	15	0	10	**40**
A69	5	0	8	**42**
A69	2	4	1	**47**

Średnie długości rozwiązań 1/2

Benchmark	%P	Statyczne	Mrówki	UCT
P101	15	5705,7 (726,7)	3201,6 (802,8)	**2923,2 (705,4)**
P101	5	2646,7 (508,2)	1851,0 (529,6)	**1526,2 (312,7)**
P101	2	1565,9 (336,2)	2029,9 (306,5)	**1129,4 (231,2)**
A80	15	8702,0 (1590,5)	4885,6 (1103,5)	**3977,3 (794,1)**
A80	5	4370,4 (1168,9)	3493,5 (637,5)	**2685,4 (556,5)**
A80	2	2840,6 (676,7)	3026,7 (271,8)	**2155,2 (361,0)**
A69	15	6583,1 (1118,4)	3880,9 (1091,2)	**2743,7 (639,2)**
A69	5	3499,8 (731,8)	2592,2 (539,9)	**1932,8 (473,9)**
A69	2	2086,1 (508,0)	2314,8 (281,4)	**1504,7 (239,7)**

Liczba razy bycia nie gorszym niż najlepszy

Benchmark	%P	Statyczne	Mrówki	UCT
A54	15	0	9	**41**
A54	5	0	6	**44**
A54	2	8	1	**47**
P45	15	0	21	**29**
P45	5	0	20	**30**
P45	2	8	6	**43**
P19	15	1	**35**	15
P19	5	6	**25**	**25**
P19	2	18	13	**37**

Średnie długości rozwiązań 2/2

Benchmark	%P	Statyczne	Mrówki	UCT
A54	15	6093,2 (1216,4)	3673,5 (1036,5)	**2795,2 (741,8)**
A54	5	3174,7 (834,2)	2347,8 (509,1)	**1791,3 (458,7)**
A54	2	2045,6 (695,6)	2107,4 (254,5)	**1441,4 (249,1)**
P45	15	3551,0 (759,3)	1636,0 (541,0)	**1568,0 (507,3)**
P45	5	1528,1 (392,2)	1043,7 **(166,6)**	**998,6** (260,2)
P45	2	1085,6 (326,3)	968,5 **(138,5)**	**725,8** (144,8)
P19	15	1407,4 (468,4)	**542,4 (303,8)**	748,2 (381,0)
P19	5	573,8 (240,8)	**356,8 (96,3)**	365,1 (122,1)
P19	2	461,2 (247,0)	310,9 **(38,9)**	**303,6** (105,7)

Podsumowanie

Pozytywy:

- Coś działa...
- Zaskakująco dobre wyniki w porównaniu do algorytmu mrówkowego
- Nie mamy póki co lepszej metody do porównania
- Metoda na pewno reaguje na korki (wyraźnie krótsze trasy niż przesymulowanie rozwiązania początkowego)
- Metoda może być stosowana z dowolną metodą generowania rozwiązania początkowego

Podsumowanie

Negatywy:

- Czasami przegrywa z bardzo prostą metodą heurystyczną (H1 lub H2) – powinna wygrywać w każdej instancji testu
- Algorytm UCT nie ma dużych szans na poprawę rozwiązania statycznego
- Propagowane **Q** dla każdej trasy z jednej symulacji jest wspólne. Trasa może ucierpieć z powodu innych / *brak prostego rozwiązania – bo używanie odległości tras oddzielnie powoduje bardzo złe wyniki globalne*
- Szybkość algorytmu jest wrażliwa na długości tras, co pośrednio wynika z proporcji wielkości zamówień do pojemności ciężarówki. Algorytm lepiej działa dla większej liczby krótszych tras.