Optymalizacja I

Omówienie projektów

Jakub Dudziński
Natalia Kwiecień

Wydział Matematyki, Informatyki i Mechaniki

Projekt 1 - Pivot Rules

Omówienie reguł I

Porządek leksykograficzny, minimum

Wybiera minimalny indeks z możliwych do wejścia/wyjścia wierzchołków.

Porządek leksykograficzny, maksimum

Wybiera maksymalny indeks z możliwych do wejścia i wyjścia wierzchołków.

Największy wzrost

Metoda zwraca parę wierzchołków (wierzchołek wejściowy i wyjściowy) w postaci listy, która prowadzi do największego wzrostu funkcji celu.

Najmniejszy wzrost

Funkcja zwraca nam listę dwóch wierzchołków, dla których wzrost funkcji celu jest najmniejszy.

Omówienie reguł II

Największy współczynnik

Metoda wybiera zmienną wejściową, przy której stoi największy współczynnik w funkcji celu. Przy wyciąganiu wierzchołków korzystamy w naszym programie z metody *lexicographical_min_leaving(self)*.

Losowy wybór wierzchołka

Jest to metoda wybierająca losowy wierzchołek wejściowy/wyjściowy ze wszystkich możliwych wejść/wyjść.

Średnia ważona

Metoda, która determinuje wybór zmiennych wejściowych i wyjściowych w oparciu o przypisane każdej z możliwości prawdopodobieństwo.

Najbardziej stroma krawędź

Metoda wybierająca wierzchołek minimalizujący kąt między gradientem funkcji celu a krawędzią łączącą stary wierzchołek z nowym.

Omówienie reguł III

Pierwszy od lewej

Metoda wybierająca ze wszystkich możliwych wejść/wyjść wierzchołek z początku listy.

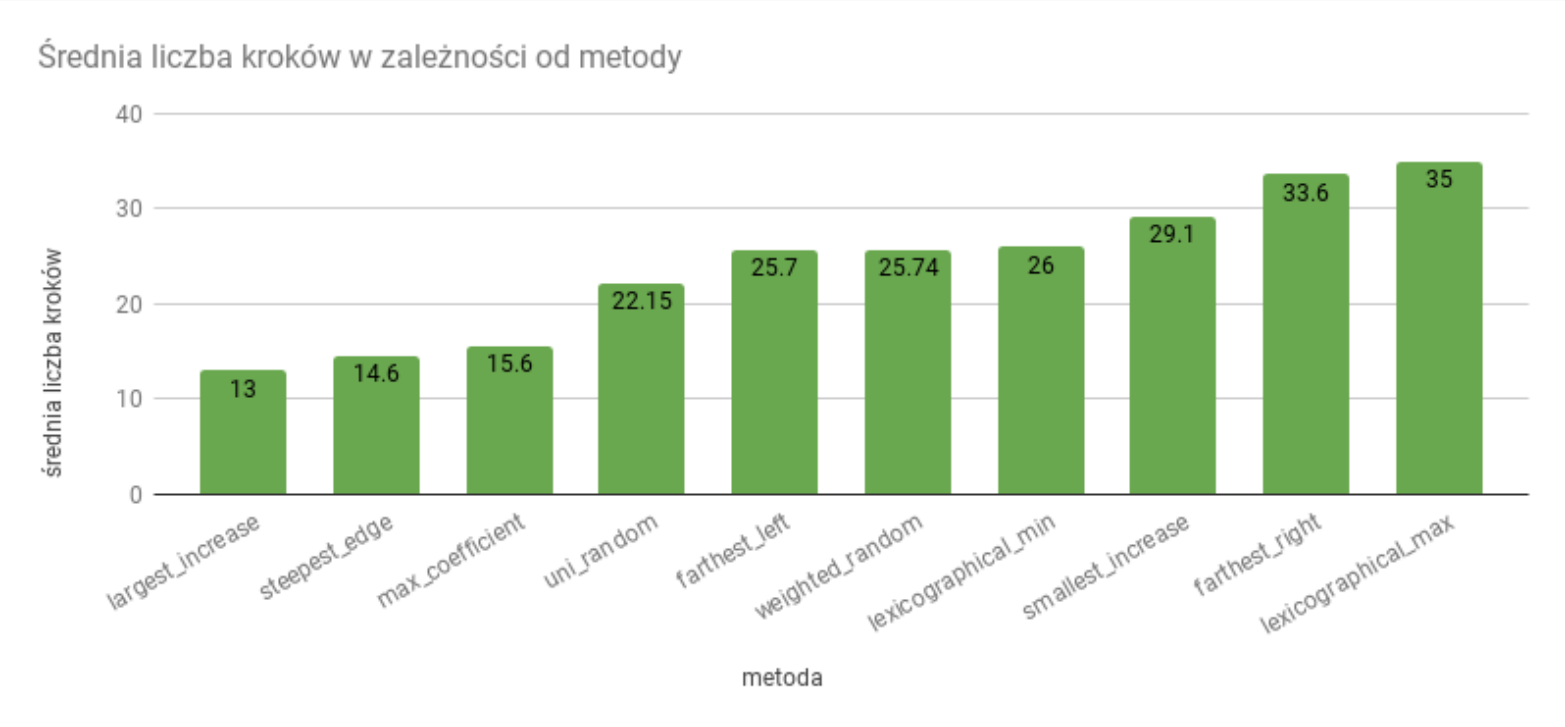
Pierwszy od prawej

Metoda wybierająca ze wszystkich możliwych wejść/wyjść wierzchołek z końca listy.

Użytkowanie kodu

Poszczególne funkcje dla programu wybieramy w linijkach *#222 - #244*, poprzez odkomentowanie odpowiedniej metody wybierania wierzchołka wejścia/wyjścia wewnątrz funkcji *my_entering(self)* oraz *my_leaving(self)*.

Wyniki


Średnia liczba kroków w zależności od metody
40
30
20
10
0
średnia liczba kroków
13
14.6
15.6
22.15
25.7
25.74
26
29.1
33.6
35
largest_increase
steepest_edge
max_coefficient
uni_random
farthest_left
weighted_random
lexicographical_min
smallest_increase
farthest_right
lexicographical_max
metoda

Omówienie wyników

Jak widać wyróżniają się (pozytywnie) trzy metody: *largest_increase*, *steepest_edge* i *max_coefficient*. W prawie wszystkich testach które wykonaliśmy *largest_increase* zanotowało najlepszy lub drugi najlepszy wynik. Uzyskało także średnio najniższą liczbę kroków we wszystkich testach. Szczególnie dobrze wypadło w teście (10) i (8). Metoda *steepest_edge* wypadła niewiele gorzej. Wykonała średnio 1,8 kroków więcej. Uzyskała bardzo dobre wyniki w testach (3) i (9). W teście (10) uzyskała niezadowalający wynik. Metoda *max_coefficient* uzyskała niezłe wyniki we wszystkich testach. Trzy pierwsze metody uzyskały dobre (średnio mniej niż 16 kroków) wyniki.

Wnioski

Reguły można podzielić na grupy:

- **Reguły o niskiej liczbie kroków**. Do tej grupy należą: *largest_increase*, *steepest_edge* i *max_coefficient*.
- **Reguły wydajne obliczeniowo**. Do tej grupy należą: *farthest_left*, *farthest_right*, *lexicographical_min*, *lexicographical_max*.
- **Reguły zapobiegające pętlom**. Do tej grupy należy: *lexicographical_min*
- **Reguły niedeterministyczne**.Metody niepraktyczne. Do tej grupy należą: *uni_random*, *weighted_random*.
- **Reguły 'złośliwe'**. Bez praktycznego zastosowania. Do tej grupy należy: *smallest_increase*.

Do każdego problemu możemy dobrać pewną, odpowiednią regułę w zależności od jego specyfiki (np. pętle) i tego czy zależy nam na wydajności obliczeniowej czy też zminimalizowaniu liczby kroków algorytmu sympleks.

Projekt 2 - Bluff

Opis gry

W tym wariancie gra Bluff jest grą dwuosobową o sumie zerowej (suma wypłat jest równa zero). Jest to gra sekwencyjna, więc jej naturalną reprezentacją jest drzewo gry. To również gra o niepełnej informacji (na początku gracze rzucają kostką i nie znają wyniku drugiej osoby). W każdej turze gracz musi dokonywać decyzji czy przebić przeciwnika (i o ile przebić) czy zawołać *blef*.

Próba rozwiązania

Na początku zdefiniowaliśmy funkcję *match*, która jako argument bierze dwie strategie czyste i rozgrywa je między sobą. Funkcja zwraca *1*, jeśli wygrał gracz 1 i *-1*, jeśli wygrał gracz 2. W dalszej części kodu tworzymy wszystkie strategie czyste obu graczy. A następnie tworzymy macierz wypłat *M* w oparciu o strategie czyste i funkcję *match*. Następnie, podobnie jak na ćwiczeniach, tworzymy problem dualny i rozwiązujemy go, uzyskując równowagę Nasha w strategiach mieszanych.

WYNIKI

Niestety brak. Metoda okazałą się zbyt pamięcożerna i nie uzyskaliśmy wyniku. Ale gdybyśmy dysponowali większą pamięcią obliczeniową, to uzyskalibyśmy poprawny wynik.

Projekt 3 - Spy Agency

WSTĘP

Projekt ma na celu ustalenie ilu pracowników możemy zwolnić, by uzyskać jak najmniejszą możliwą, wciąż funkcjonującą organizację o zadanej strukturze. Mamy dane dwie hierarchie pracowników w postaci drzew: jedną organizacyjną (WSA - World Spy Agency), drugą związkową (Union). Mamy również dane ograniczenia na minimalną liczbę pracowników w poszczególnych departamentach. Naszym zadaniem jest ustalić, których pracowników możemy zwolnić bez szkody dla organizacji i związku zawodowego jednocześnie.

Algorytm

Na samym początku definiujemy funkcję *inf*, która zwraca nam wszystkich podwładnych pracownika o ID *w* w postaci listy *E* w hierarchii WSA dla $k = 0$ i w hierarchii Union dla $k = 1$. Zmienna globalna *n* jest całkowitą liczbą pracowników, a zmienna *A* to macierz wczytywana z pliku. W dalszej części kodu dla każdego pracownika tworzymy zmienną zero - jedynkową, która determinuje czy dany pracownik zostanie zwolniony z pracy (wartość 0) czy zostanie w pracy (wartość 1). Problem rozwiązujemy przy pomocy *MixedIntegerLinearProgram*. Funkcja celu jest minimalizacją sumy wszystkich pracowników, a ograniczenia są nałożone na minimalną liczbę osób w każdym departamencie.