Omówienie projektów

Mateusz Doliński, Katarzyna Głowacka, Michał Kozyra

Optymalizacja I - rok 2016/2017

21.06.2017

Pivot Rules - Projekt 1

Mateusz Doliński, Katarzyna Głowacka, Michał Kozyra

Optymalizacja I - rok 2016/2017

21.06.2017

Nasze zadanie

- zaimplementowanie 11 metod wyboru zmiennych (tj. wierzchołków) w algorytmie sympleks
- przetestowanie metod na 14 problemach testowych $LP0$-$LP13$: odpowiedź na pytanie, ile kroków potrzeba do znalezienia optymalnego wierzchołka dla każdej pary metoda-problem
- zbadanie, jak dana reguła wyboru zmiennych wchodzących/wychodzących wpływa na liczbę kroków algorytmu sympleks

Użytkowanie kodu

- kod programu uruchamiamy w Sage
- w linijce 1154 zmieniamy aktualny wybór zakresu metod (0-10) i poddawanych testom problemów LP (0-13)

Użyte funkcje pivot

- *smallest_coefficient* vs *largest_coefficient*
  - wybór zmiennej wchodzącej o najmniejszym/największym współczynniku funkcji celu;
- *smallest_increase* vs *largest_increase*
  - wybór zmiennej, który prowadzi do najmniejszego/największego wzrostu funkcji celu;
- *lexicographical_max* vs *lexicographical_min*
  - wybór zmiennej wchodzącej i wychodzącej pierwszej/ostatniej słownikowo;
- *steepest_edge* vs *flattest_edge*
  - wybór zmiennej, który prowadzi do wierzchołka w kierunku najbliższym/najdalszym wektorowi c (gradientowi funkcji celu);
- *random_edge* vs *uniform_random*
  - wybór losowy krawędzi/zmiennej (prawdopodobieństwo jednostajne)
- *poisson_modulo*

Analiza wyników

Zależności między kolejnymi parami metod porównane z metodami losowymi (w nawiasach podajemy średnią liczbę kroków; dla metod losowych **uniform_random** (8.7), **random_edge** (8.92), **poisson_modulo** (8.87) użyliśmy średniej z kilku prób):

- metoda **largest_coefficient** (5.36) wypadła lepiej niż metody losowe; metoda **smallest_coefficient** (9.71) radzi sobie gorzej
- metoda **largest_increase** (5.71) wypadła lepiej niż metody losowe; metoda **smallest_increase** (9.5) radzi sobie gorzej
- nie ma większych różnic między metodami **lexicographical_max** (8.64), **lexicographical_min** (8.64) a rezultatami metod losowych
- metoda **steepest_edge** (8.07) wypadła lepiej niż metody losowe; **flattest_edge** (10.57) wypadła gorzej

Wnioski

- metody **largest_coefficient**, **largest_increase** oraz **steepest_edge** dają lepsze rezultaty niż metody losowe
- metody **smallest_coefficient**, **smallest_increase** oraz **flattest_edge** dają rezultaty znacznie gorsze
- pozostałe metody dają podobne, średnio dobre wyniki

Podsumowanie

Najlepsze strategie: wybór zmiennej o największym współczynniku funkcji celu, wybór kierunku o największym jej wzroście lub wzdłuż jej gradientu.

Bluff - Projekt 2

Mateusz Doliński, Katarzyna Głowacka, Michał Kozyra

Optymalizacja I - rok 2016/2017

21.06.2017

Nasze zadanie

- znaleźć strategię optymalną w grze Bluff:
  - potraktować tę grę jako grę dwuosobową o sumie zerowej
  - naturalną reprezentacją jest drzewo gry
  - cel: uniknąć operowania macierzą gry w postaci normalnej

Podejście do problemu

Spojrzeliśmy na sekwencje ruchów obu graczy, które uwzględniliśmy w macierzy M, której wiersze i kolumny odpowiadają wspomnianym sekwencjom, a nie strategiom czystym zawodników.

Macierz sekwencji M

- elementy to wartości oczekiwane wypłaty w momencie wykonywania ruchów przez graczy lub zera
- macierz rzadka
- ma 2^{20} elementów, a jedynie 2^8 z nich jest odpowiednikiem liści w drzewie gry

Przez x oznaczyliśmy wektor sekwencji ruchów gracza pierwszego, a przez y - drugiego. Przedmiotem naszego zainteresowania była zatem funkcja celu postaci $x^T My$.

Kodowanie wierzchołków

Dany wierzchołek kodowaliśmy stringiem wyznaczającym ciąg sekwencji przebiegu gry do tego momentu.
W naszym słowniku: $a = (1, 1)$, $b = (1, 2)$, ..., $h = (2, 4)$, $i = blef$.

Zmiennymi decyzyjnymi określiliśmy prawdopodobieństwo bycia w danym wierzchołku. Suma prawdopodobieństw w k-tym ruchu danego gracza jest równa prawdopodobieństwu znalezienia się w wierzchołku będącym w ostatnim ruchu.

Problem liniowy

- funkcja celu jest postaci $x^T My$
- rozwiązujemy problem z perspektywy gracza drugiego
- traktujemy wektor x jako ustalony
- minimalizacja oczekiwanej wypłaty gracza 1 - funkcji celu przy warunkach określających zmienną y
- rozwiązujemy równoważny problem dualny, w którym zmienne y zostały zastąpione przez sztuczne zmienne z, a współrzędne wektora x traktujemy jako parametry
- znalezienie optymalnego rozwiązania polega na uzmiennieniu wektora x.

Spy Union - Projekt 3

Mateusz Doliński, Katarzyna Głowacka, Michał Kozyra

Optymalizacja I - rok 2016/2017

21.06.2017

Nasze zadanie

- rozwiązanie problemu Spy Union przy dwóch hierarchiach pracowników zadanych w postaci drzew (WSA i Union)
- określenie, ilu i których pracowników można zwolnić przy danych minimalnych liczbach pracowników w poszczególnych departamentach
- celem jest uzyskanie jak najmniejszej możliwej wciąż funkcjonującej struktury organizacji i związku zawodowego

Dane

- **INPUT**
  Przykładowe dane wejściowe:

```
5
1 0 1 2
2 0 1 2
2 1 2 0
2 1 0 1
1 3 0 0
```

Liczba w pierwszym wierszu (5) oznacza całkowitą liczbę pracowników.

Interpretacja kolejnych danych

Table: Przykładowy input

ID pracownika	ID przełożonego WSA	ID przełożonego Union	Minimalna liczba podwładnych WSA	Minimalna liczba podwładnych Union
0	1	0	1	2
1	2	0	1	2
2	2	1	2	0
3	2	1	0	1
4	1	3	0	0

- **OUTPUT**

  Dane wyjściowe to maksymalna liczba pracowników, których można zwolnić, aby podane struktury nie zostały zaburzone oraz ich numery. Dla wyżej podanych danych wejściowych dostajemy:

  2

  4 2

3-etapowe podejście do problemu

- funkcja **fill** - parsowanie danych wejściowych i konwertowanie ich na dwa drzewa implementowane przy pomocy słowników oraz dwie listy z minimalną liczbą pracowników w każdym departamencie w obu strukturach
- funkcja **department** - dla zadanego drzewa tworzy słownik niezbędny do utworzenia warunków ograniczających dla zadanego problemu liniowego; zaimplementowany algorytm pozwala na otrzymanie wszystkich warunków przy jednokrotnym przeszukaniu całej struktury danych
- funkcja **solve_problem** - wykorzystanie przygotowanych słowników do stworzenia oraz rozwiązania zadanego problemu przy pomocy klasy MixedIntegerLinearProgram

Testy

Table: Zwolnieni pracownicy

Numer testu	**Liczba pracowników**	**Liczba zwolnionych**
0	5	2
1	20	9
2	100	53
3	200	115
4	500	302
5	1000	585
6	2000	1174
7	5000	2754
8	10000	5761
9	20000	11492
10	50000	26428