apto

Lab 6: Programowanie liniowe (Problem ILP)

W ramach laboratorium należy:

1. zapoznać się z biblioteką PuLP
2. zaimplementować programy rozwiązaujące problemy VertexCover oraz Weighted Vertex Cover przy pomocy solwera ILP
3. [Dodatkowe] zaimplementować program rozwiązaujący problem Graph Coloring przy pomocy solwera ILP

Szczegóły techniczne

Wszystkie programy powinny być implementowane w języku Python (3.x.y). Mogą Państwo (i powinni) korzystać z poniższych programów:

- dimacs.py – mikrobiblioteka pozwalająca na wczytywanie grafów, nagrywanie wyników, najbardziej podstawowe operacje na grafach, oraz nagrywanie formuł logicznych CNF w formacie DIMACS ascii (z którego korzystają solwery SAT)
- Biblioteka PuLP (należy ją sobie zainstalować)
- Solwer CBC (wbudowany w PuLP)
- Solwer GLPK (możliwe użycie przez PuLP; można zainstalować, ale nie jest to niezbędne)

Oprócz darmowych solwerów, na rynku dostępne są także solwery komercyjne (np. IBM ILOG CPLEX albo Gurobi), które są duże szybsze (często o wiele rzędów wielkości). Studenci i pracownicy naukowi mogą korzystać z nich za darmo przez licencje akademickie (konieczna rejestracja). W ramach laboratorium nie ma potrzeby korzystać z solwerów komercyjnych, ale jeśli ktoś z Państwa chciałby wykorzystać programowanie liniowe np. w pracy dyplomowej, to warto o nich pamiętać (zwłaszcza jeśli darmowe okażą się za wolne).

Uwaga Biblioteka PuLP jest bardzo wygodna w użyciu, ale wprowadza ogromny narzut wydajnościowy na etapie budowania modelu. W laboratorium nie będzie to przeszkadzać, ale przy większych modelach często korzysta się z innych rozwiązań.

Zadanie 1: Zapoznanie z biblioteką PuLP

W ramach tego zadania należy znaleźć rozwiązania (całkowitoliczbowe oraz rzeczywiste) dla następującego programu liniowego:

zminimalizuj: $x + y$

przy ograniczeniach:

$y \geq x - 1$

$y \geq -4x + 4$

$y \leq -0.5x + 3$

Podstawy PuLP. Z bilioteki PuLP korzysta się w następujący sposób:

- należy stworzyć obiekt "modelu" (czyli obiekt, który opisuje całe zadanie optymalizacyjne).
- dla każdej zmiennej należy stworzyć obiekt tej zmiennej (z odpowiednimi informacjami o niej, np. czy jest to zmienna całkowita, ciągła, czy zero-jedynkowa)
- do modelu należy dodać funkcję celu, która będzie optymalizowana (minimalizowana lub maksymalizowana, zależnie od modelu)
- do modelu należy dodać nierówności liniowe definiujące ograniczenia.
- następnie model można rozwiązać i odczytać wartości zmiennych oraz status rozwiązania (optymalne, nieograniczone itp.)

Poniższy kod pokazuje przykłady wykonania tych operacji:

```
from pulp import *

# stwórz model problemu o nazwie test, minimalizacja funkcji celu
model = LpProblem( "test", LpMinimize)   # LpMaximize dla maksymalizowania funkcji celu

# stwórz trzy zmienne, x, y, z
x = LpVariable( "x" , lowBound = 0, upBound = 5, cat = "Continuous" ) # zmienna ciągła z przedz
y = LpVariable( "y" , lowBound = 0, upBound = 3, cat = "Integer" )    # zmienna całkowita ze zł
z = LpVariable( "z" , cat = "Binary" )                                # zmienna binarna, wartos

# dodaj funkcję celu do modelu
model += 5-x+2*y-z

# dodaj ograniczenia (konieczne do spełnienia nierówności liniowe)
model += x >= y
model += y >= z

print( model )
```

W wyniku jego wykonania zostanie wypisany stworzony program liniowy:

```
test:
MINIMIZE
-1*x + 2*y + -1*z + 5
```

```
SUBJECT TO
_C1: x - y >= 0

_C2: y - z >= 0

VARIABLES
x <= 5 Continuous
0 <= y <= 3 Integer
0 <= z <= 1 Integer
```

Poniższy kod pozwala rozwiązać problem i wypisać wartości zmiennych.

```
model.solve()                    # rozwiąż używając CBC
print( LpStatus[model.status] )  # wypisz status rozwiązania

# wypisz zmienne
for var in model.variables():
  print( var.name, "=", var.varValue )

# wypisz wartość funkcji celu
print( value(model.objective) )
```

Zmienne można też wypisywać bezpośrednio:

```
print( x.value() )
```

Użycie GLPK. Zmiana solwera w PuLP jest bardzo łatwa. Wystarczy następująca linijka do rozwiązania danego problemu:

```
model.solve( GLPK() )
```

Podobnie można używać innych solwerów (nie zainstalowane). Bibliotego PuLP ma także wiele parametrów, którymi można doprecyzować proces optymalizacji i wykonania solwera (np. można wyłączyć wypisywanie komunikatów).

Zadanie 2: VertexCover

Dokładny opis zadania znajduje się w Laboratorium 1.

Redukcja ILP

Można wykorzystać następującą redukcję do ILP. Mamy graf $G = (V,E)$, gdzie $V = \{ v_1, \ldots, v_n \}$ oraz pytamy, czy istnieje pokrycie wierzchołkowe wykorzystujące k wierzchołków. Dla każdego wierzchołka v_i tworzymy zmienną x_i, której wartość interepretujemy następująco:

- $x_i = 1$ – wierzchołek v_i należy do pokrycia,

- $x_i = 0$ – wierzchołek v_i nie należy do pokrycia.

Tworzymy program liniowy, który minimalizuje wartość:

$x_1 + x_2 + \ldots + x_n$

Jednocześnie stawiamy warunek, że dla każdych dwóch wierzchołków v_i oraz v_j połączonych krawędzią, co najmniej jeden musi być w rozwiązaniu:

$x_i + x_j \geq 1$

Testowanie redukcji

Jako dane testowe proszę wykorzystać zestaw grafów z Laboratorium 1:

- graph.zip

Warto zwrócić uwagę na efektywność solwera CBC versus GLPK.

Weighted Vertex Cover

Proszę zmodyfikować program tak, by każdy wierzchołek v_i miał wagę równą swojemu stopniowi (albo jakiejś jego funkcji—na przykład potędze zadawanej jako parametr programu: dla wartości 0.0 każdy wierzchołek miałby koszt 1, a dla wartości 1.0 każdy wierzchołek miałby koszt równy swojemu stopniowi). Program powinien szukać pokrycia wierzchołkami o minimalnej sumie wag.

Proszę porównać liczby wierzchołków (oraz koszty rozwiązań w różnych metrykach) użyte w rozwiązaniu tak sformułowanego problemu.

Relaksacja LP

Proszę zmodyfikować program dla powyższego zadnia tak, żeby zmienne nie miały wartości całkowitych ale ciągłe (z przedziału [0,1]). Proszę w rozwiązaniu uwzględniać te wierzchołki, których zmienne osiągają wartość co najmniej 0.5. Proszę porównać jakość uzyskanego algorytmu aproksymacyjnego z algorytmem dokładnym (wyniki teoretyczne gwarantują, że wynik będzie najwyżej dwa razy gorszy). Proszę także porównać czas działania algorytmów.

Zadanie 3: Kolorowanie grafów

W tym zadaniu rozważamy problem kolorowania grafów (opisany w poprzednim laboratorium).

W ramach zadania proszę napisać program, który wczytuje graf oraz dostaje liczbę k dopuszczalnych kolorów, oblicza redukcję problemu do ILP oraz wyznacza rozwiązanie przy pomocy solwera.

Redukcja do ILP

Można wykorzystać następującą redukcję do ILP. Mamy graf $G = (V,E)$, gdzie $V = \{v_1, \ldots, v_n\}$ oraz k kolorów do wykorzystania. Dla każdego wierzchołka v_i oraz koloru j tworzymy zmienną $x_{i,j}$, której wartość interepretujemy następująco:

- $x_{i,j} = 1$ – wierzchołek v_i ma kolor j,
- $x_{i,j} = 0$ – wierzchołek v_i nie ma koloru j.

Dla każdego wierzchołka x_i wymuszamy posiadanie dokładnie jednego koloru:

$$x_{i,1} + x_{i,2} + \ldots + x_{i,k} = 1$$

Dla każdej krawędzi $\{ v_i, v_t \}$ i dla każdego koloru j wymuszamy, że oba wierzchołki nie mają tego koloru jednocześnie:

$$x_{i,j} + x_{t,j} \leq 1$$

Testowanie redukcji

Jako dane testowe proszę wykorzystać zestaw grafów dostępnych w zbiorze benchmarków Graph Coloring Benchmarks (zbiór grafów dostępny w sekcji `Files`). Liczba kolorów potrzebna dla danego grafu zapisana jest w kolumnie `X(G)` obok pliku z grafem. Proszę pamiętać o odczytaniu kolorów wierzchołków z rozwiązania ILP oraz zrobieniu wewnętrznego testu w programie, sprawdzającego czy kolorowanie jest poprawne.

Dodanie funkcji celu

Proszę zmodyfikować stworzony program tak, by tworzył egzemplarz ILP, który minimalizuje liczbę użytych kolorów (nadal należy przyjmować na wejściu maksymalną liczbę kolorów, które są do dyspozycji, ale jeśli ta liczba jest większa niż optymalna, to program ILP powinien i tak znaleźć rozwiązanie optymalne).