Wrocław, 16 marca 2021

Zaawansowane **T**echniki **O**ptymalizacji

Laboratorium 2

Rozwiązywanie wybranych zadań optymalizacji w środowisku aplikacji ILOG

prowadzący: *dr inż. Jarosław Rudy*

1 Cel laboratorium

Celem laboratorium jest zapoznanie się z metodami modelowania i rozwiązywania problemów z wykorzystaniem interfejsu graficznego aplikacji ILOG CPLEX Optimization Studio firmy IBM. Temat obejmuje instalację i obsługę aplikacji ILOG CPLEX Optimization Studio, modelowanie problemu i jego ograniczeń, zapis instancji testowej oraz wybór metody rozwiązania, uruchomienie optymalizatora oraz analizę otrzymanych wyników.

2 Przebieg zajęć

Laboratorium obejmuje zajęcia nr 3 i 4 (4 godziny zajęć). Praca odbywa się w ramach grup dwuosobowych. Każda grupa otrzymuje do zrealizowania dwa problemy optymalizacji dyskretnej lub ciągłej z poniższych list.

Zagadnienia dyskretne:

1. Problem sumy podzbioru (problem 2.1).
2. Zagadnienie transportowe (problem 2.2).
3. Zagadnienie przydziału (problem 2.3).
4. Kwadratowe zagadnienie przydziału (problem 2.4).
5. Dyskretny problem plecakowy (problem 2.5).

Zagadnienia ciągłe (jako modyfikacje problemów dyskretnych):

1. Problem inspekcji (problem 5.1).
2. Kwadratowe zagadnienie przydziału (problem 5.2).
3. Problem dwuplecakowy (problem 5.3).
4. Szeregowanie zadań na równoległych maszynach (problem 5.4).

Modele zadanych problemów należy zapisać używając ILOG CPLEX Optimization Studio oraz określić metodę optymalizacji. Następnie należy uruchomić optymalizator, dokonać analizy wynik (w tym logów) pracy optymalizatora. Należy też oszacować zakres stosowalności i/lub złożoność optymalizatora dla danego problemu.

Uwaga: dane wejściowe należy wygenerować korzystając z dostępnych generatorów liczb pseudolosowych (na stronie kursu). Ze względu na fakt, że ILOG CPLEX Optimization Studio posiada własny język modelowania, generowanie danych najłatwiej zrealizować w osobnym języku (np. `python`), najlepiej od razu tworząc plik w formacie odpowiednim dla ILOG-a.

Naliczanie oceny zaczynamy od 1. Za poprawne przedstawienie (model, generacja danych wejściowych, analiza wyników) każdego z dwóch problemów grupa otrzymuje +2 do oceny. Na ocenę wpływa również czas oddania zadania oraz wiedza studenta.

3 Praca z ILOG CPLEX Optimization Studio

3.1 Dostęp i instalacja

Do pracy niezbędny jest dostęp do zainstalowanej aplikacji ILOG CPLEX Optimization Studio. Preferowanym sposobem jest praca z wykorzystaniem maszyny wirtualnej (np. `VirtualBox`). Prowadzący może dostarczyć wymagane środowisko. Jeśli to nie nastąpiło (lub student preferuje samodzielną instalację), pakiet ILOG CPLEX Optimization Studio należy zainstalować samodzielnie korzystając z poniższych instrukcji.

Poniżej przedstawiono sposób instalacji pakietu optymalizacyjny `IBM ILOG CPLEX Optimization` w wersji `Studio Free Edition`. Wersja darmowa jest wersją w pełni funkcjonalną, posiada jednakże limitem na liczbę zmiennych oraz liczbę ograniczeń w modelowanym problemie. Oba limity są ustawione na 1000 co powinno w zupełności wystarczyć do celów przeprowadzenia zajęć dla małych lub średnich instancji dla wszystkich omawianych problemów.

Pierwszym krokiem jest rejestracja konta IBM (`IBMid`). Konto należy założyć na stronie ibm.com przy użyciu poczty studenckiej. W trakcie procesu rejestracji należy podać rzeczywiste dane osobowe: imię, nazwisko oraz kraj. Jako województwo należy wskazać dolnośląskie. W drugim kroku należy potwierdzić, że jest się studentem. Należy pamiętać o zweryfikowaniu adresu email. Po udanej rejestracji i zalogowaniu się na konto `IBMid` należy pobrać odpowiednią wersje narzędzie ILOG CPLEX Optimization Studio ze strony imb.com. Dostępne są wersje na `MS Windows`, `Linux` oraz `MacOS` (rozmiar pliku instalacyjnego zależnie od wersji mieści się w granicach 635,7–849,5 MB). Program należy pobrać oraz zainstalować przed zajęciami. W uzasadnionych przypadkach dopuszcza się jedną instalacje na grupę.

3.2 Modelowanie problemu

Aplikacja ILOG CPLEX Optimization Studio wykorzystuje do opisu modelu wykorzystuje Optimization Programming Language (OPL). Poniżej opiszemy jego najważniejsze cechy. Dokładniejsze informacje można znaleźć na stronie producenta.

Rozpatrzmy optymalizację dyskretną na przykładzie omawianego wcześniej problemu maksymalizacji wyników testu jednokrotnego wyboru. Dla przypomnienia, dany jest test n pytań, gdzie dla każdego pytania podane jest k wariantów odpowiedzi, wraz z wartością punktową. Zadanie polega na wyborze dla każdego pytania wariantu tak, aby zmaksymalizować sumę punktów uzyskanych za testu. Zauważmy, że w każdym pytaniu można zaznaczyć co najwyżej jeden wariant, co oznacza że można nie zaznaczać żadnego wariantu.

Zaczniemy od utworzenia nowego projektu OPL. W tym celu z menu `Plik` → `Nowy` wybieramy opcję „Projekt OPL”. Pojawi się formularz, w którym wybieramy nazwę i miejsce projektu oraz zaznaczamy opcję utworzenia pliku modelu (innymi plikami zajmiemy się później).

Problem modelujemy w pliku o rozszerzeniu `.mod`. Zacznijmy od przedstawienia zmiennych decyzyjnych. Pojedynczą zmienną decyzyjną definiujemy słowem kluczowym `dvar` po którym podajemy jej typ (dostępne są np. `boolean`, `int` lub `float`) oraz nazwę podobnie jak w standardowych językach programowania. Linię z definicją zmiennej należy zakończyć średnikiem. Przykładowo instruk-

cja `dvar int x;` utworzy pojedynczą całkowitą zmienną decyzyjną o nazwie x. Oprócz zmiennych decyzyjnych możemy tworzyć zwykłe zmienne w sposób znany z języka `C` np. `int n = 5;`.

Oprócz pojedynczych zmiennych decyzyjnych możemy tworzyć ich wektory. W tym celu należy wykorzystać nawiasy kwadratowe, w których umieszczamy zakres np. `[1..4]` dla utworzenia wektora czterech zmiennych. Podczas podawania zakresu można posłużyć się inną zmienną np. `[1..n]` lub wyrażeniem np. `[n+1..2*n+1]`. Zakresy możemy również przechowywać w zmiennych specjalnego typu `range` np. `range numbers = 1..n;`. Zwykle tworząc zakresy mamy na myśli zakresy dla liczb całkowitych. Możliwe jest jednak tworzenie zakresów jako przedziałów liczb rzeczywistych po dodaniu słowa kluczowego `float`. Przykładowo aby zdefiniować zmienną `fr` przechowującą przedział liczb rzeczywistych od 1.2 do 2.2 możemy napisać `range float fr = 1.2..2.2;`

Wróćmy do wektora zmiennych decyzyjnych. Aby zdefiniować wektor `x` zawierający N zmiennych decyzyjnych typu `int` używając zdefiniowanego powyżej zakresu `numbers` należy napisać `dvar int x[numbers];`. Zauważmy, że w nawiasach podajemy zakres, a nie pojedynczą liczbę jak w języku `C`! Gdy zamiast powyższej instrukcji napiszemy `dvar int x[n];`, to w wyniku powstanie wektor zawierający tylko jeden element a nie N elementów! Inicjalizacja wektora odbywa się w następujący sposób. Załóżmy, że `numbers` reprezentuje zakres 5 elementów, wtedy zapis `int values [numbers] = [2, 3, 1, 2, 6];` utworzy 5-elementowy wektor o nazwie `values` z indeksami opisanymi przez zakres `numbers` i wartosciami kolejno 2, 3, 1, 2 i 6. W przypadku wektora dwuwymiarowego zapis jest analogiczny jak w języku `python` np.:

```
int values [1..2][numbers] = [[2, 3, 1, 2, 6], [1, 2, 1, 5, 2]];.
```

Wróćmy do naszego testu jednokrotnego wyboru. Aby utworzyć macierz `values` zawierającą wartości punktowe poszczególnych wariantów pytań możemy zapisać:

```
int values [questions][variants] = [
    [7,2,9,3],
    [0,6,7,1],
    [4,1,6,3],
    [1,4,8,5],
    [3,3,9,8],
    [3,2,2,2]
    ];
```

gdzie `questions` oraz `variants` to zdefiniowane zakresy odpowiednio dla pytań oraz wariantów. Oczywiście macierz można zapisać w jednej linii, ale powyższy zapis zwiększa czytelność. Zauważmy że macierz `values` wraz z wartościami `n` (liczba pytań) oraz `k` (liczba wariantów na pytanie) stanowi dane wejściowe (instancję) problemu. Analogicznie macierz zmiennych decyzyjnych definiujemy poprzez:

```
dvar boolean x [questions][variants];
```

Mając zdefiniowane dane wejściowe i zmienne decyzyjne, należy określić żądany sposób optymalizacji (`maximize` lub `minimize`) oraz określić funkcję celu do optymalizacji. Najprostsza postać funkcji celu to maksymalizacją wartości jednej zmiennej: `maximize x`. W przypadku naszego testu funkcja celu jest nieco bardziej skomplikowana: chcemy maksymalizować sumę po wszystkich zmiennych decyzyjnych przemnożonych przez wartość punktową danego wariantu danego pytania. W celu zapisu tej funkcji celu ponownie posłuży się pojęciem zakresu, a ściślej możliwości iterowania po danym zakresie, podobnie jak w wielu językach programowania ogólnego przeznaczenia. Nasza funkcja celu będzie miała postać:

```
maximize sum(q in questions, v in variants ) x[q][v]*values[q][v];
```

W powyższym zapisie `q in questions` oznacza że zmienna `q` będzie iterowała po zakresie `questions`, zaś `x[q][v]` oznacza element wektora `x` o indeksach `q` oraz `v`.

Kolejnym krokiem jest wskazanie ograniczeń problemu. Ograniczenia wprowadzamy podobnie jak funkcje celu, przy pomocy instrukcji `subject to`. Mamy trzy typy wprowadzanych ograniczeń: (1) `==`, (2) `<=` lub (3) `>=`. Zauważmy, że nie ma ograniczeń `<` ani `>`!

Ograniczenia umieszczamy w nawiasach klamrowych. Dla problemu testu jednokrotnego wyboru ograniczenia mogą przyjąć formę:

```
subject to {
     sum( v in variants )  x[1][v] <= 1;
     sum( v in variants )  x[2][v] <= 1;
     sum( v in variants )  x[3][v] <= 1;
     sum( v in variants )  x[4][v] <= 1;
     sum( v in variants )  x[5][v] <= 1;
     sum( v in variants )  x[6][v] <= 1;
}
```

Ponownie korzystamy z zakresów, by zdefiniować sumę zaznaczonych wariantów w każdym pytaniu. Każda taka suma musi być mniejsza lub równa 1. Zauważmy, że liczba ograniczeń zależy od n i musielibyśmy ją ręcznie zmieniać przy każdej zmianie n. Aby tego uniknąć możemy skorzystać z instrukcji `forall`. Zastosowanie jej wprowadzi zapis ogólny, niezależny od liczby n:

```
subject to {
   forall(q in questions) sum( v in variants ) x[q][v] <= 1;
}
```

Po wprowadzeniu modelu mamy jeszcze możliwość wykonania dodatkowego kodu w bloku `execute`. Optymalizator na koniec zwróci wartość funkcji celu. Będziemy wtedy mieli też możliwość podejrzenia wartości zmiennych decyzyjnych oraz danych wejściowych. Blok `execute` możemy wykorzystać do wyświetlenia rozwiązania w wygodnym dla nas formacie. Składnia języka używana w tym bloku jest bliska do tej znanej z języków programowania ogólnego przeznaczenia.

```
execute {
    for (var i = 1; i <= nr; i++)
        for (var j = 1; j <= k; j++)
            if(x[i][j]==1)
                writeln("q: ", i, " v: ", j);
}
```

Powyższy fragment kodu wypisze dla każdego pytania ten wariant, który został wybrany.

3.3 Dane wejściowe

Podczas modelowania możemy posłużyć się danymi wejściowymi wczytanymi z odpowiednio przygotowanego pliku. Należy dodać nowy plik *dane* (lub zaznaczyć odpowiednią opcję przy tworzeniu projektu OPL). Domyślnie dane są wczytywane z pliku o rozszerzeniu `.dat`. Aby zaznaczyć, że wartości danej zmiennej określone są w osobnym pliku należy skorzystać z wielokropka. Przykładowo zapis `int n = ...;` oznacza że liczba pytań n będzie podana w osobnym pliku. Analogicznie macierz wartości punktowych przyjmuje wtedy postać:

```
int values [questions][variants] = ...;
```

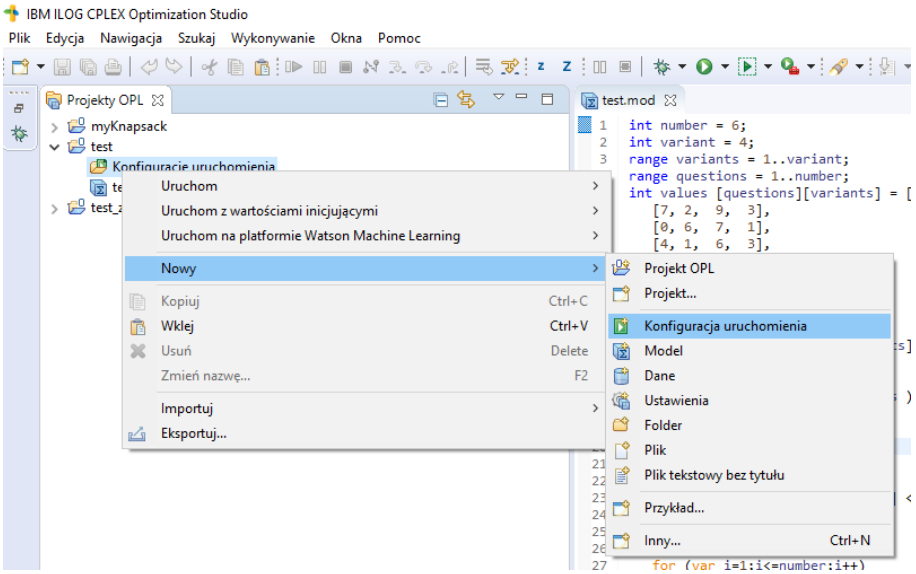
Należy pamiętać, że kolejność danych w pliku z danymi musi być taka sama jak w pliku z modelem. W pliku z danymi trzeba oznaczyć do jakiej zmiennej mają być przypisane wartości, bez ponownego podawania typu. Wielkość liter ma znaczenie. W naszym przypadku plik z danymi ma postać:

```
n = 6;
k = 4;
values = [
    [7,2,9,3],
    [0,6,7,1],
    [4,1,6,3],
    [1,4,8,5],
    [3,3,9,8],
    [3,2,2,2]
    ];
```

W jednym projekcie OPL może być więcej niż jeden plik z danymi. Aby optymalizator wiedział których danych ma użyć, należy dodać plik z danymi do odpowiedniej *konfiguracji uruchomienia*, co omówimy w następnym podrozdziale.

3.4 Uruchomienie i wyniki

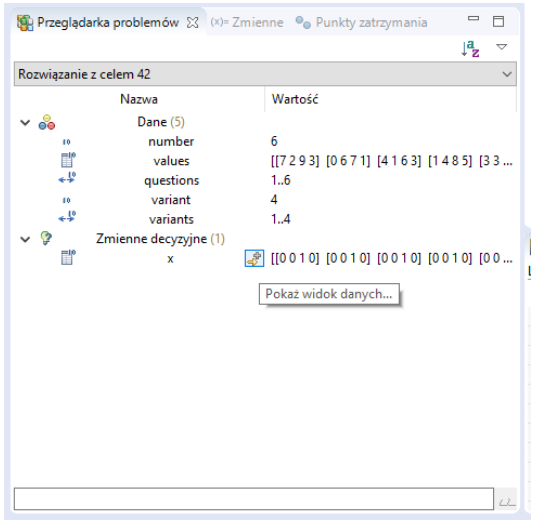
W celu uruchomienia optymalizatora dla wybranego modelu należy najpierw utworzyć dla niego plik *konfiguracji uruchomienia*. Należy w eksploratorze projektów kliknąć prawym klawiszem myszy w naszym projekcie, następnie wybrać z menu `Nowy` → `Konfiguracja uruchomienia`, tak jak pokazano to na Rysunku 1. Warto zaznaczyć opcję, aby nowa konfiguracja była konfiguracją domyślną.



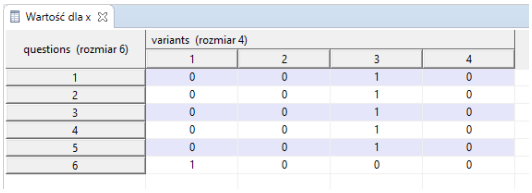
Rysunek 1: Dodanie nowej konfiguracji do projektu.

Następnie należy dodać nasz model do nowo utworzonej konfiguracji, poprzez przeciągnięcie pliku z modelem na nazwę konfiguracji wewnątrz eksploratora projektów. Następnie trzeba potwierdzić dodanie w nowym oknie. W przypadku modeli ze stałymi danymi wejściowym (wpisanymi na sztywno w kodzie), tak przygotowany projekt jest gotowy do uruchomienia poprzez kliknięcie na konfigurację prawym klawiszem myszy i wybranie opcji `Uruchom ten`. W przypadku korzystania z danych wejściowych zdefiniowanych w osobnym pliku `.dat` należy utworzyć osobną konfigurację dla każdego pliku z danymi. Pliki dołączamy do konfiguracji tak samo jako model poprzez przeciągnięcie i opuszczenie na konkretnej konfiguracji.

Po skończeniu pracy optymalizatora mamy dostęp do różnych danych. W przeglądarce problemów (Rysunek 2a) mamy widok na wszystkie dane wejściowe, końcowe wartości zmiennych decyzyjnych oraz wartość funkcji celu. Dla zmiennych, które są wektorami jedno- lub dwuwymiarowymi mamy możliwość szczegółowego podglądu (Rysunek 2b) przy użyciu opcji `Pokaż widok danych`, którą można wybrać z poziomu Przeglądarki problemów.



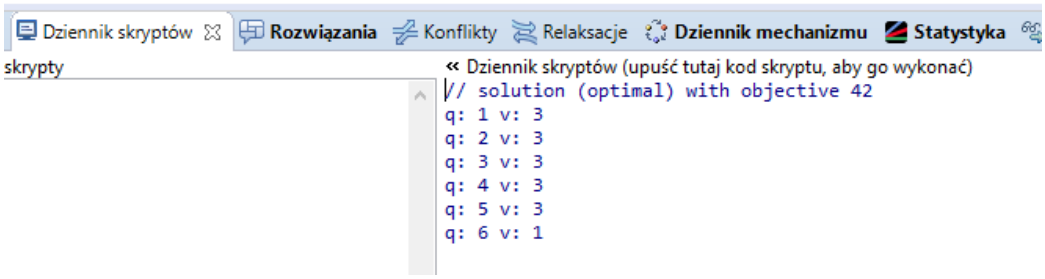
(a) podgląd ogólny



(b) szczegółowy podgląd zmiennej wektorowej

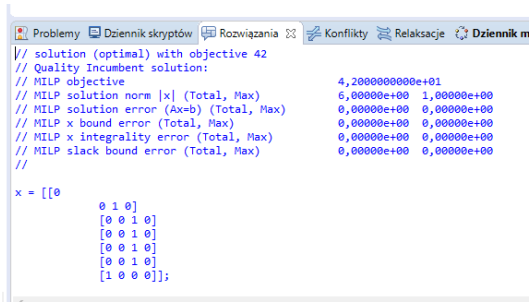
Rysunek 2: Podgląd końcowych wartości zmiennych

W Dzienniku skryptów przedstawionym na Rysunku 3 optymalizator pokazuje wartość funkcji celu dla znalezionego rozwiązania oraz efekt wykonania kodu z bloku `execute`.

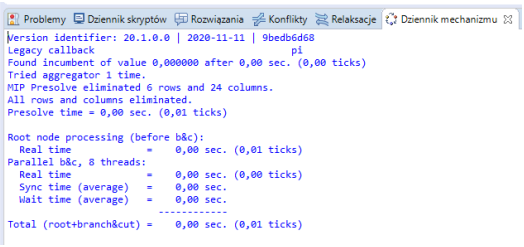


Rysunek 3: Dziennik skryptów

W karcie Rozwiązania widocznej na Rysunku 4a możemy podejrzeć szczegółowe informacje dotyczące znalezionego rozwiązania w tym: (1) wartość funkcji celu dla znalezionego rozwiązania, (2) szacowana odległość od rozwiązania optymalnego, (3) rozwiązanie. Szczegółowy czas działania optymalizatora znajdziemy w Dzienniku mechanizmu pokazanym na Rysunku 4b. Innych istotnych informacji możemy dowiedzieć się między innymi z karty Statystyka.



(a) podgląd szczegółowy rozwiązania



(b) szczegóły i czas działania optymalizatora

Rysunek 4: Podgląd innych zakładek