Teoria i metody optymalizacji, AREU00003P

Opis projektu

Programowanie liniowe całkowitoliczbowe metodą podziału i ograniczeń

autor: **Krzysztof Rewak**, numer indeksu: #192935
termin zajęć: wtorki nieparzyste, 07^{30}-09^{00}
prowadząca: dr inż. Ewa Szlachcic

1 Zadanie optymalizacji

Zadanie programowania liniowego przedstawionego w opisywanym projekcie polega na znalezieniu zbioru wartości zmiennych maksymalizujących liniową funkcję celu i spełniających zadane ograniczenia [1]. Można przedstawić standardową postać programu liniowego:

$$\max_{x \in X} x_0 = c^T x$$

1.1 Ograniczenia

Zgodnie z poleceniem, przedstawione zadanie programowania liniowego przyjmuje zarówno ograniczenia dolne, jak i górne. Współczynniki ograniczeń w macierzach A_1 i A_2 oraz wektory wartości b_1 i b_2 można przedstawić następująco:

$$X = \left\{ x : \begin{array}{l} A_1 x \leq b_1 \\ A_2 x \geq b_2 \end{array} \right.$$

Ponadto do każdego zadania powinno zostać dołączone ograniczenie obligatoryjne, które wymaga, aby wszystkie współczynniki x_i^* dla $i = 1, 2, ..., n$ były nieujemne:

$$x_i^* \geq 0$$

Rozwiązanie problemu programowania całkowitoliczbowego należy ograniczyć do przypadku, gdzie wszystkie otrzymane wartości x_i^* dla $i = 0, 1, 2, ..., n$ należą do zbioru liczb całkowitych:

$$x^* \in \mathbb{Z}^n$$

Nałożono również ograniczenia na wymiar zadania: liczba zmiennych w równaniach $n \leq 10$ oraz liczba ograniczeń $m \leq 10$, a także limit iteracji w algrytmie podziału i ograniczeń $L \leq 100$.

1.2 Przyjęta metoda rozwiązania

Zgodnie z poleceniem do rozwiązania zadania programowania liniowego całkowitoliczbowego wykorzystano metodę podziału i ograniczeń (ang. *Bound and Branch*).

Postać zadania programowania całkowitoliczbowego PCL można matematycznie przedstawić jako:

$$S_j = \{x^* | A^j x = b^j,\ x \geq 0 \text{ i } x \in \mathbb{Z}^n\}$$

Należy zastosować osłabienie prowadzące do zadania programowania liczbowego PL:

$$T_j = \{x^* | A^j x = b^j,\ x \geq 0 \text{ i}\}$$

$$T_j \supseteq S_j$$

W tym celu uruchamiany jest dwufazowy algorytm simpleks, który zwraca wynik optymalizacji dla zadanych parametrów. Jeżeli wynik spełnia warunek $x^* \in \mathbb{Z}^n$, algorytm zostaje zakończony ze względu na znalezienie optymalnego rozwiązania; w przeciwnym wypadku zostaje uruchomiona metoda podziału i ograniczeń.

Pierwszym krokiem metody B&B jest odnalezienie górnego ograniczenia, które wynosi $x^*_{0z} \leq \lfloor x^*_0 \rfloor$. Następnie należy znaleźć niecałkowitoliczbowe rozwiązanie:

$$x_{Bi} = [y_{i0}] + f_{i0}$$

$$0 < f_{i0} < 1$$

Następnie dokonać podziału zbioru S_j:

$$S^*_j = \left\{ S_j \cap \{x | x_{Bi} \leq \hat{y}_{i0}\}, S_j \cap \{x | x_{Bi} \geq \check{y}_{i0}\} \right\}, \text{ gdzie}$$

- $\check{a}$ oznacza najmniejszą liczbę całkowitą większą lub równą a
- $\hat{a}$ oznacza największą liczbę całkowitą mniejszą lub równą a

Następnie należy dodać ograniczenia na zakres dopuszczalnych wartości rozwiązań kolejnym zmiennym, które nie spełniają warunku całkowitoliczbowości. W geometrycznym sensie wycinane jest pasmo rozwiązań, które prowadzi do podziału na dwa zbiory:

$$\lfloor x_i \rfloor < x_i < \lfloor x_i \rfloor + 1$$

Przykładowo dla zadania $\max x_0 = 6x_1 + 5x_2$ dla $x_i \geq 0$ oraz dla następujących ograniczeń:

$$X = \left\{ x : \begin{array}{rcl} 9x_1 + 7x_2 & \leq & 63 \\ x_1 + x_2 & \leq & 8 \\ 3x_1 + 2x_2 & \geq & 6 \end{array} \right.$$

metoda simpleks zwróci wartość optymalną funkcji celu $x^*_0 = 43.5$ dla $x^* = [3.5,\ 4.5]^T$. Wynik nie spełnia warunku całkowitoliczbowości, więc uruchamiana jest metoda podziału i ograniczeń. Górne ograniczenie wyniesie $x^*_{0z} \leq 43$ i pod uwagę brana jest zmienna x^*_1 (wybór jest heurystyczny). Do drzewa rozwiązań dodane są dwa nowe zadania, do których dodano nowe ograniczenia pochodzące z metody podziału:

(a) ograniczenie dolne:

$$X = \left\{ x : \begin{array}{rcl} 9x_1 + 7x_2 & \leq & 63 \\ x_1 + x_2 & \leq & 8 \\ 3x_1 + 2x_2 & \geq & 6 \\ x_1 & \leq & 3 \end{array} \right.$$

(b) ograniczenie górne:

$$X = \left\{ x : \begin{array}{rcl} 9x_1 + 7x_2 & \leq & 63 \\ x_1 + x_2 & \leq & 8 \\ 3x_1 + 2x_2 & \geq & 6 \\ x_1 & \geq & 4 \end{array} \right.$$

Każdy przypadek należy rozwiązać ponownie metodą simpleks i przeanalizować uzyskane wyniki pod względem całkowitoliczbowości. Problem można przedstawić w postaci drzewa binarnego oraz zaimplementować jako stos, na który dodaje się kolejno tworzone przypadki ograniczeń i analizuje się pierwszy na stosie; w taki sposób drzewo przeszukiwane jest w głąb.

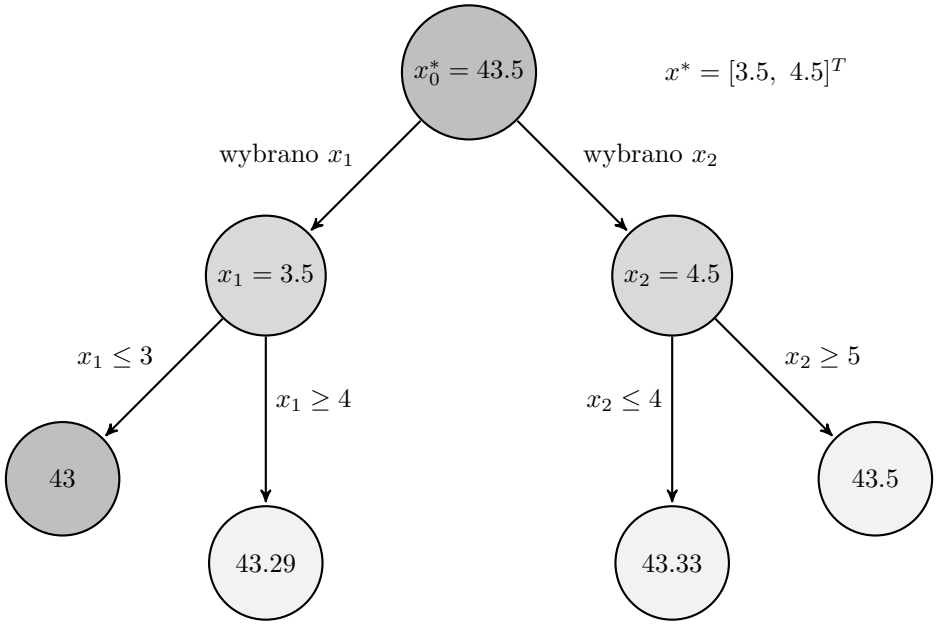
Stos został zaimplementowany jako lista zbudowana na zwykłej tablicy, do której dodawane są JSON-owe obiekty przechowujące wartości x^*_0, tablicę wyników x^*_i oraz wynikowe macierz A oraz wektory b i c.

Na początku algorytmu na stos wrzucane jest rozwiązanie zadania progamowania liniowego wykonane przez algorytm simpleks. Następnie uruchamiana jest pętla z trzema warunkami stopu:

- znalezienie całkowitoliczbowego wyniku równego górnemu oszacowaniu; poprzez osłabienie zadania PCL do PL wiemy, że szukana wartość optymalna nie może przekroczyć wartości x^*_{0z}
- pusty stos; nie można utworzyć dzieci danego przypadku, gdy analizowane zadanie jest sprzeczne lub nieograniczone
- przekroczenie odgórnie ustalonego limitu iteracji L zwiększanego przy każdym następnym wywołaniu algorytmu simpleks dla każdego nowego zestawu ograniczeń

Z racji przekazywania parametrów funkcji poprzez referencję w języku Javascript, przy każdym podziale węzła następuje klonowanie (*deep copy*) macierzy A oraz wektorów b i c, aby nie zachodziły żadne interferencje z wielokrotnie wykonywanego algorytmu simpleks dla różnych zestawów ograniczeń.

Schemat drzewa binarnego należałoby przedstawić następująco:



Schemat 1: Schemat metody podziału i ograniczeń

Jak widać na załączonym schemacie, wynik zadania po dodaniu ograniczenia $x_1 \leq 3$ okazuje się być całkowitoliczbowy. Z racji znalezienia wyniku równego górnemu oszacowaniu, główna pętla programu zostaje przerwana i zostaje zwrócony wynik; pozostałe gałęzie drzewa zostały wyrysowane w celach poglądowych. Algorytm metody podziału i ograniczeń przedstawił wynik: $x^* = [3,\ 5]^T$.

2 Algorytm optymalizacji

2.1 Opis algorytmu simpleks

Algorytm simpleks przede wszystkim pozwala na ograniczenie nakładu obliczeń koniecznych do rozwiązania zadania programowania liniowego [3], które polega na sprawdzeniu wartości badanej funkcji we wszystkich znalezionych wierzchołkach wielościanu wypukłego Φ. Zadanie należy na początku algorytmu przkształcić do postaci kanonicznej [1]:

$$[A]x = b \Leftrightarrow [N]x_N + [B]x_B = b \Leftrightarrow [B, N]\begin{bmatrix} x_B \\ x_N \end{bmatrix} = b$$

Skrócony opis przebiegu algorytmu można przedstawić następująco:

- zbudowanie tablicy simpleksowej z macierzy A oraz wektorów b i c
- znalezienie bazowego rozwiązania dopuszczalnego poprzez eliminację Gaussa dla bazowych kolumn tablicy simpleksowej; jeżeli któryś z elementów wynikowego wektora jest mniejszy od zera, należy poszerzyć tablicę simpleksową o rozszerzoną bazę i znaleźć właściwe rozwiązanie właściwą metodą simpleks opisaną w kolejnych krokach
- właściwy algorytm simpleks (w nieskończonej pętli):
  - wyliczenie kosztów zredukowanych
  - wybór kolumny opuszczającej bazę (najmniejsza wartość wektora kosztów); jeżeli $p_{max} \leq 0$ to znaleziono optymalne rozwiązanie, a w przeciwnym wypadku należy kontynuować
  - jeżeli w wybranej kolumnie znajdują się elementy większe od zera, wtedy zadanie jest nieograniczone i należy przerwać pętlę
  - wyliczenie wartości θ [1] i na jej podstawie wyznaczenie wiersza, który zostanie usunięty z bazy
  - operacje algebraiczne na tablicy simpleksowej: odejmowanie wiersza (odpowiadającego zmiennej opuszczającej bazę) od wszystkich pozostałych w tablicy przemnożonych przez wartość z przecięcia wychodzącej kolumny i wiersza

– powrót do początku pętli

- poszukiwania wartość celu x_0^* to ujemna wartość x_0 z tablicy simpleksowej

2.2 Przyjęte tablice simpleksowe

Do rozwiązania zadania programowania liczbowego przyjęto następującą budowę tablic simpleksowych [1]:

$$\begin{bmatrix} A & E & b \\ c^T & Z & x_0 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} A_{00} & A_{01} & \dots & A_{0(n-1)} \\ A_{10} & A_{11} & \dots & A_{1(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m0} & A_{m1} & \dots & A_{m(n-1)} \end{bmatrix} & \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} & \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \\ \begin{bmatrix} c_1 & c_2 & \dots & c_{(n-1)} \end{bmatrix} & \begin{bmatrix} 0 & 0 & \dots & 0 \end{bmatrix} & x_0 \end{bmatrix}$$

Macierz A zawiera współczynniki parametrów x_i dla $i = 0, 1, 2, ..., n-1$ z zadanych m ograniczeń. Wektor b składa się z wyrazów wolnych ograniczeń, a c to transponowane współczynniki funkcji celu. Macierz oznaczona symbolem E to jednostkowa macierz rozmiaru $m \times m$, podczas gdy wektor Z rozmiaru m wypełniony jest samymi zerami.

W przypadku, gdy nie wszystkie współczynniki wektora b są nieujemne, budowana jest tymczasowa rozszerzona macierz simpleksowa, dzięki której można znaleźć rozwiązanie bazowe [2]. Po przekształceniach (wykonanych w celu ułatwienia poruszania się po tablicy dla iteratorów) wygląda ona następująco:

$$\begin{bmatrix} A & E & E & b \\ c^T & Z & Z & x_0 \\ Z_m & Z_m & M & 0 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} A_{00} & A_{01} & \dots & A_{0(n-1)} \\ A_{10} & A_{11} & \dots & A_{1(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m0} & A_{m1} & \dots & A_{m(n-1)} \end{bmatrix} & \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} & \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \\ \begin{bmatrix} c_1 & c_2 & \dots & c_{(n-1)} \end{bmatrix} & \begin{bmatrix} 0 & 0 & \dots & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & \dots & 0 \end{bmatrix} & x_0 \\ \begin{bmatrix} 0 & 0 & \dots & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & \dots & 0 \end{bmatrix} & \begin{bmatrix} -1 & -1 & \dots & -1 \end{bmatrix} & 0 \end{bmatrix}$$

Z_m to wektor wypełniony zerami rozmiaru $n-1$, a wektor M to ujemny wektor jednostkowy rozmiaru m.

2.3 Przypadki rozwiązań

Należy wyróżnić cztery podstawowe przypadki rozwiązania zadania programowania liniowego, również całkowitoliczbowego. Podstawowym oraz oczekiwanym jest odnalezienie jednego rozwiązania optymalnego dla niepustego zbioru rozwiązań dopuszczalnych X. Rozszerzenie problemu do poziomu całkowitoliczbowego nakłada dodatkowy warunek $x \in \mathbb{Z}^n$.

W szczególnych przypadkach mogą zaistnieć trzy inne rozwiązania:

- zbiór rozwiązań dopuszczalnych X nie jest zbiorem pustym i istnieje nieskończona liczba rozwiązań optymalnych na odcinku lub półprostej,
- zbiór rozwiązań dopuszczalnych X jest zbiorem pustym, czyli zadanie jest sprzeczne,
- zadanie PL jest zadaniem nieograniczonym.

Zadanie PL ma nieskończoną liczbę rozwiązań na odcinku lub półprostej jeżeli któreś z ograniczeń jest kombinacją liniową funkcji celu. Przykładowo będzie to zadanie $\max_{x \in X} x_0 = x_1 + x_2$ z m ograniczeniami, spośród których jedno będzie zapisane jako $x_1 + x_2 \leq 20$.

Zadanie PL jest sprzeczne, gdy nie istnieje żadne rozwiązanie spełniające wszystkie zadane ograniczenia, czyli zbiór X jest zbiorem pustym. Podstawowym sposobem wykrywania takiej sytuacji jest sprawdzenie czy wynik x spełnia obligatoryjne założenie $x \geq 0$.

Zadanie PL może być zadaniem nieograniczonym, gdy istnieje ciąg [2]:

$$\left\{x^i\right\}_{i=1}^{\infty} \subset X : \text{ taki, że } f(x^i) \xrightarrow[i \to \infty]{} \infty$$

Zadanie jest nieograniczone, jeżeli dla dowolnej wartości $M < 0$ istnieje dopuszczalny punkt x spełniający nierówność $f(x) \leq M$. Sprawdzane jest to w metodzie simpleks dla wartości kolumny opuszczającej bazę.

3 Informacje o programie

Program można uruchomić na większości przeglądarek internetowych, choć polecane są ich najnowsze wersje. Przetestowano działanie na Mozilla Firefox 37.0.2, Internet Explorer 10.0.9 oraz Google Chrome 42.0.2. Plik otwierający główne okno programu to `index.html` w głównym katalogu. Z racji wykorzystania rozwiązań webowych, program można również uruchomić zdalnie pod adresem `http://pwr.rewak.eu/timo/`

Interfejs jest podzielony zasadniczo na dwa podstawowe tryby: podawanie parametrów zadania oraz wyświetlanie wyników. Po uruchomieniu ładowane są domyślne dane, które pozwalają na rozpoczęcie obliczeń.

Zrzut ekranu 1: Program w trybie podawania parametrów

Zrzut ekranu 2: Program w trybie wyświetlania wyników

Szkielet strony został zbudowany w języku znaczników HTML5 z wykorzystaniem kaskadowych arkuszów stylów CSS3. Struktura blokowa została oparta na kontenerach zamkniętych tagami `<div></div>`, pola wpisywania danych wejściowych zbudowano znacznikami `<input>`, a komunikacja z użytkownikiem została zrealizowana poprzez okna modalowe będące podstawowym komponentem biblioteki Bootstrap.

Aplikacja trzyma się konwencji *single-page website*, czyli wszystkie możliwe do obejrzenia widoki generowane są na bieżąco i nie wymagają odświeżenia strony. Dynamizacja została zaimplementowana dzięki bibliotece jQuery i manipulacji drzewem DOM, które pozwala na pokazywanie i ukrywanie komponentów o danym identyfikatorze oraz zmianie klas powiązanych z arkuszem stylów.

3.1 Wykorzystane środowisko i biblioteki

Program do optymalizacji został w całości napisany w języku **JavaScript**, który jest wieloparadygmatowym (obiektowym, funkcyjnym oraz imperatywnym) językiem programowania wykorzystywanym najczęściej do obliczeń oraz wykonywania akcji po stronie klienta na stronach internetowych.

Podstawową biblioteką wykorzystaną w projekcie jest **jQuery**[1] - służy do manipulacji drzewem DOM, czyli bezpośrednio łączy interfejs z zaimplementowanym algorytmem. Do przedstawienia oraz obsługi elementów interfejsu użyto klas i metod popularnego dla rozwiązań webowych frameworka **Bootstrap**[2]. Wykorzystane ikony pochodzą z biblioteki **Font Awesome**[3].

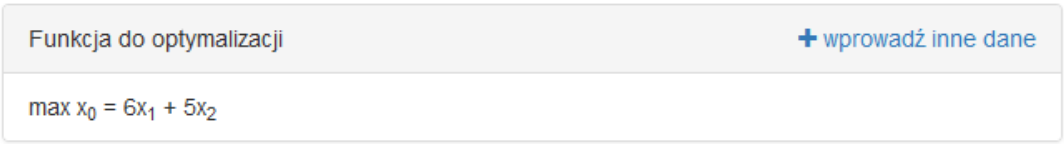
Sam interfejs został stworzony we frameworku **Backbone.js**[4] wykorzystującym wzorzec architektoniczny MVC; wymaga on dołączenia biblioteki **underscore.js**[5]. Funkcje rozwiązujące zagadnienia programowania liczbowego oraz metody podziału i ograniczeń dla PCL zostały opracowane w JavaScriptcie w wersji 1.5.

3.2 Zasady wprowadzania danych początkowych

Program przy starcie automatycznie ładuje przykładowy problem do rozwiązania z plików `target.json` oraz `boundaries.json`. Jest to przykład przedstawiony na wykładzie *Teorii i metod optymalizacji*, w którym należy znaleźć rozwiązanie $\max x_0 = 6x_1 + 5x_2$ dla $x_i \geq 0$ oraz dla następujących ograniczeń:

$$X = \left\{ x : \begin{array}{rcl} 9x_1 + 7x_2 & \leq & 63 \\ x_1 + x_2 & \leq & 8 \\ 3x_1 + 2x_2 & \geq & 6 \end{array} \right.$$

Dane funkcji celu można zmienić poprzez kliknięcie etykiety *+wprowadź inne dane* w prawym górnym rogu panelu oznaczonego jako *Funkcja optymalizacji*. W oknie dialogowym należy wpisać wartości parametrów dla x_i dla których będzie przeprowadzana optymalizacja.



Zrzut ekranu 3: Panel funkcji do optymalizacji

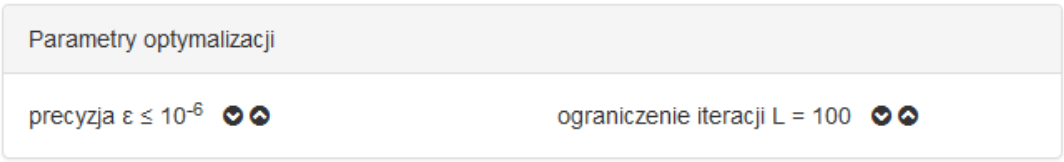
Ograniczenia można dodawać poprzez kliknięcie etykiety *+dodaj ograniczenie* w prawym górnym rogu panelu oznaczonego jako *Dobrane ograniczenia (n)*, gdzie n to liczba ograniczeń. W oknie dialogowym należy wpisać wartości parametrów dla x_i, wybrać rodzaj ograniczenia (domyślne mniejszościowe lub większościowe) oraz podać wartość b.

Ograniczenia można usuwać poprzez kliknięcie etykiety *-usuń*, która pojawia się po najechaniu kursora na wiersz z danym ograniczeniem.

Zrzut ekranu 4: Panel ograniczeń

[1] dokumentacja dostępna pod adresem `http://api.jquery.com/`
[2] dokumentacje dostępne pod adresami `http://getbootstrap.com/css` i `http://getbootstrap.com/javascript`
[3] opis elementów dostępny pod adresem `http://fortawesome.github.io/Font-Awesome/`
[4] dokumentacja dostępna pod adresem `http://backbonejs.org/`
[5] dokumentacja dostępna pod adresem `http://underscorejs.org/`

Żądaną precyzję obliczeń ε oraz limit iteracji L można określić w panelu oznaczonym jako *Parametry optymalizacji*. Klikając w ikony strzałek *w dół* lub *w górę* można odpowiednio zmniejszyć lub zwiększyć precyzję ustalania czy dany wynik jest liczbą całkowitą oraz ograniczenie iteracji algorytmu podziału i ograniczeń.



Zrzut ekranu 5: Panel parametrów optymalizacji

Ponadto można zmienić automatycznie załadowywane parametry z poziomu kodu, bez ingerencji w warstwę logiki. Należy wstawić zgodne z formatem i standardem JSON dane do plików `boundaries.json` i `target.json` - odpowiednio dla zestawu ograniczeń oraz postaci funkcji celu. Z racji, iż przykładowe dane zostały już wprowadzone, do zmiany parametrów wystarczy zmienić wartości w zapisanych już tablicach w wymienionych plikach.

Aby rozpocząć rozwiązywanie zadania należy kliknąć w zielony przycisk *Znajdź maksimum funkcji*. Jeżeli nie zadano żadnych ograniczeń, będzie on nieaktywny. Po uruchomieniu algorytmów możliwość zmiany funkcji celu, ograniczeń oraz parametrów zostaje zablokowana.

4 Przykłady testowe

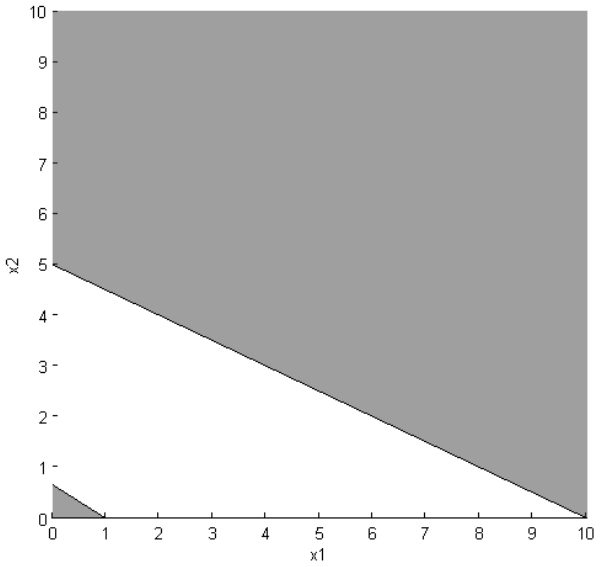
4.1 Brak rozwiązań

Sprzeczne rozwiązanie programowania liniowego można przedstawić jako zadanie:

$$\max x_0 = x_1 + x_2$$

dla następujących ograniczeń:

$$X = \left\{ x : \begin{array}{rcl} x_1 + 2x_2 & \geq & 10 \\ 2x_1 + 3x_2 & \leq & 2 \end{array} \right.$$



(a) warstwica funkcji celu

Tablica simpleksowa dla takiego zadanie będzie przedstawiała się następująco:

	x_1	x_2	x_3	x_4	
x_3	-1	-2	1	0	-10
x_4	2	3	0	1	2
x_0	1	1	0	0	0

Z racji występowania ujemnych wartości w wektorze rozwiązań dopuszczalnych utworzonym w wyniku eliminacji Gaussa, należy rozszerzyć bazę tablicy simpleksowej, a następnie znaleźć dopuszczalne rozwiązanie bazowe, dla którego uruchomiony zostanie algorytm simpleks:

	x_1	x_2	x_3	x_4	x_5	x_6	
x_3	-1	-2	1	0	1	0	-10
x_4	2	3	0	1	0	1	2
x_5	1	1	0	0	0	0	0
x_6	0	0	0	0	-1	-1	0

$\Rightarrow$

	x_1	x_2	x_3	x_4	
x_3	-1	-2	1	0	-10
x_1	2	3	0	1	2
x_0	1	1	0	0	0

Po zmianie bazy uruchamiany jest właściwy algorytm poszukiwania optimum:

	x_1	x_2	x_3	x_4	
x_3	-1	-2	1	0	-10
x_1	2	3	0	1	2
x_0	1	1	0	0	0

$\Rightarrow$

	x_1	x_2	x_3	x_4	
x_3	0	-0.5	1	0.5	-9
x_1	1	1.5	0	0.5	1
x_0	0	-0.5	0	-0.5	-1

Algorytm rozwiązywania zadania programowania liniowego zwraca wynik $x_0^* = [-9,\ 1]$. Zmienna x_1^* nie spełnia warunku $x^* \geq 0$, więc zbiór jest pusty, więc [2] zadanie jest sprzeczne

4.2 Zadanie nieograniczone

Nieograniczone rozwiązanie programowania liniowego można przedstawić jako zadanie:

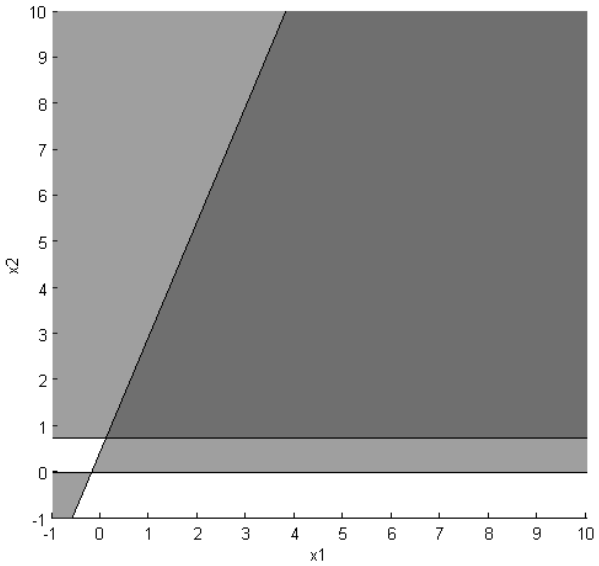
$$\max x_0 = 5x_1 + 2x_2$$

dla następujących ograniczeń:

$$X = \left\{ x : \begin{array}{lcl} 20x_2 & \geq & 15 \\ -5x_1 + 2x_2 & \leq & 1 \end{array} \right.$$

Tablica simpleksowa dla takiego zadania będzie przedstawiała się następująco:

	x_1	x_2	x_3	x_4	
x_3	0	-20	1	0	-15
x_4	-5	2	0	1	1
x_0	5	2	0	0	0



(b) warstwica funkcji celu

Należy rozszerzyć tablice simpleksową w celu znalezienia dopuszczalnego rozwiązania bazowego:

	x_1	x_2	x_3	x_4	x_5	x_6	
x_3	0	-20	1	0	1	0	-15
x_4	-5	2	0	1	0	1	1
x_5	5	2	0	0	0	0	0
x_6	0	0	0	0	-1	-1	0

Nawet po rozszerzeniu tablicy w celu znalezienia dopuszczalnego rozwiązania bazowego w wektorze kosztów zredukowanych znajdują się wartości większe od zera, zatem zadanie jest nieograniczone.

4.3 Jedno optymalne rozwiązanie PL

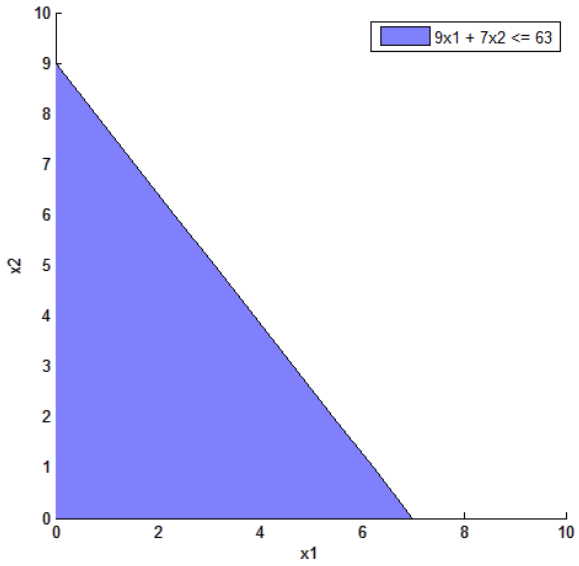
Optymalne rozwiązanie programowania liniowego można przedstawić na przykładzie wcześniej już przytaczanego zadania:

$$\max x_0 = 6x_1 + 5x_2$$

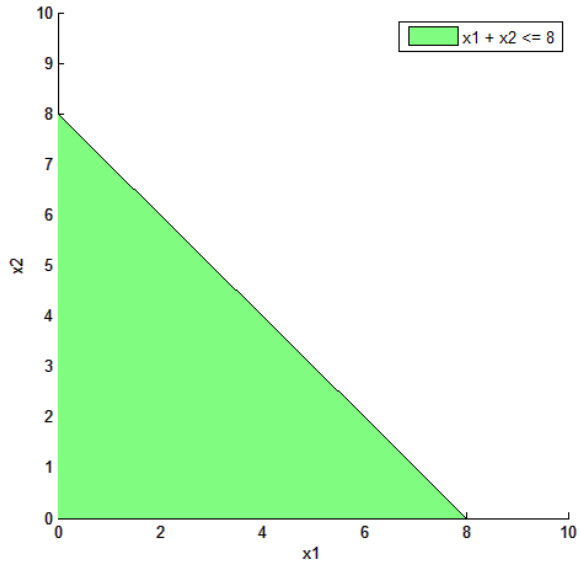
dla następujących ograniczeń:

$$X = \left\{ x : \begin{array}{lcl} 9x_1 + 7x_2 & \leq & 63 \\ x_1 + x_2 & \leq & 8 \\ 3x_1 + 2x_2 & \geq & 6 \end{array} \right.$$

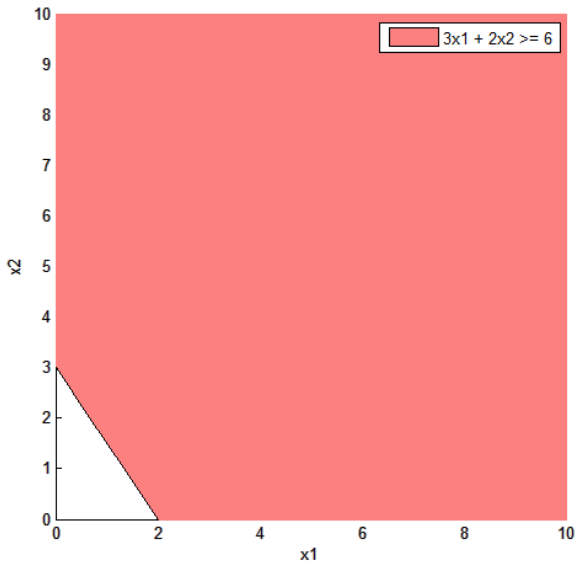
Geometryczne przedstawienie problemu należy przedstawić następująco (kolejne ograniczenia w punktach (c-e) oraz rozwiązanie w punkcie (f)):



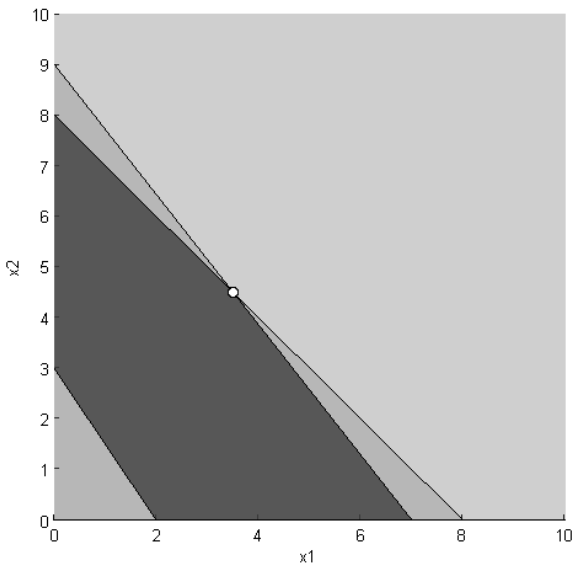
(c) ograniczenie mniejszościowe $9x_1 + 7x_2 \leq 63$



(d) ograniczenie mniejszościowe $x_1 + x_2 \leq 8$



(e) ograniczenie większościowe $3x_1 + 2x_2 \geq 6$



(f) najciemniejszy obszar to suma ograniczeń

Białą kropką na rysunku (d) zaznaczono znalezione geometrycznie maksimum funkcji celu dla zadanych ograniczeń. Z osi OX i OY można odczytać, że $x_1^* = 3.5$ i $x_2^* = 4.5$, zatem:

$$x_0^* = 6x_1 + 5x_2 = 6 \cdot 3.5 + 5 \cdot 4.5 = 21 + 22.5 = 43.5$$

Wynik interpretacji geometrycznej zostanie porównany z wynikiem zaimplementowanego algorytmu. Z macierzy A oraz wektorów b i c zostaje zbudowana tablica simpleksowa:

	x_1	x_2	x_3	x_4	x_5	
x_3	9	7	1	0	0	63
x_4	1	1	0	1	0	8
x_5	-3	-2	0	0	1	-6
x_0	6	5	0	0	0	0

Niestety nie wszystkie wartości w ostatniej kolumnie są nieujemne, więc należy rozszerzyć tablicę simpleksową w celu przeprowadzenia pierwszej fazy algorytmu i eliminacji ujemnych wartości. Po znalezieniu dopuszczalnego rozwiązania bazowego i wyeliminowaniu z macierzy zbędnych wierszy i kolumn, tablica simpleksowa wygląda następująco:

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	
x_3	9	7	1	0	0	1	0	0	63
x_4	1	1	0	1	0	0	1	0	8
x_5	-3	-2	0	0	1	0	0	1	-6
x_6	6	5	0	0	0	0	0	0	0
x_7	0	0	0	0	0	-1	-1	-1	0

$\Rightarrow$

	x_1	x_2	x_3	x_4	x_5	x_6
x_6	9	7	1	0	0	63
x_1	1	1	0	1	0	8
x_5	-3	-2	0	0	1	-6
x_1	6	5	0	0	0	0

$\Rightarrow$

W tym momencie zaczyna się druga faza algorytmu simpleks:

	x_1	x_2	x_3	x_4	x_5	
x_6	9	7	1	0	0	63
x_1	1	1	0	1	0	8
x_5	-3	-2	0	0	1	-6
x_1	6	5	0	0	0	0

$\Rightarrow$

	x_1	x_2	x_3	x_4	x_5	
x_1	1	0.778	0.111	0	0	7
x_1	0	0.222	-0.111	1	0	1
x_5	0	0.333	0.333	0	1	15
x_1	0	0.333	-0.667	0	0	-42

$\Rightarrow$ $\Rightarrow$

	x_1	x_2	x_3	x_4	x_5	
x_1	1	0	0.5	-3.5	0	3.5
x_2	0	1	-0.5	4.5	0	4.5
x_5	0	0	0.5	-1.5	1	13.5
x_1	0	0	-0.5	-1.5	0	-43.5

$\Rightarrow$

Wynik to ujemna wartość ostatniej kolumny i wiersza tablicy, czyli $x_0^* = 43.5$. Spełnia on założenia zadania PCL osłabionego do zadania PL, jednak aby uzyskać wynik całkowitoliczbowy, należy uruchomić procedurę podziału i ograniczeń, która ponownie wykona wielokrotnie algorytm simpleks. Przykład rozwiązania moetdy *Bound and Branch* można zobaczyć w sekcji 1.2 tego dokumentu).

Wynik pokrywa się z wcześniej znalezionym wynikiem wyliczonym za pomocą metody geometrycznej.

4.4 Wiele rozwiązań PL na odcinku lub półprostej

Wiele rozwiązań programowania liniowego można przedstawić na przykładzie zadania:

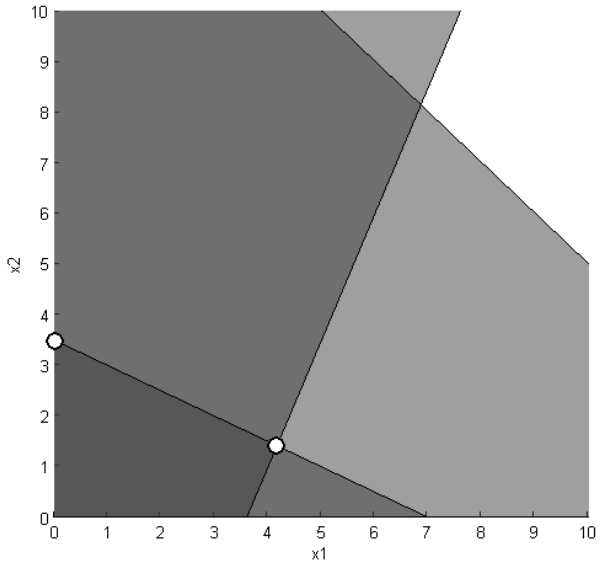
$$\max x_0 = 2x_1 + 4x_2$$

dla następujących ograniczeń:

$$X = \left\{ x : \begin{array}{lcl} x_1 + x_2 & \leq & 15 \\ x_1 + 2x_2 & \leq & 7 \\ 5x_1 - 2x_2 & \leq & 18 \end{array} \right.$$

Należy zbudować tablicę simpleksową:

	x_1	x_2	x_3	x_4	x_5	
x_3	1	1	1	0	0	15
x_4	1	2	0	1	0	7
x_5	5	-2	0	0	1	18
x_0	2	4	0	0	0	0



(g) warstwica funkcji celu

Należy powiększyć bazę tablicy simpleksowej, aby znaleźć nową bazę:

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	
x_3	1	1	1	0	0	1	0	0	15
x_4	1	2	0	1	0	0	1	0	7
x_5	5	-2	0	0	1	0	0	1	18
x_6	2	4	0	0	0	0	0	0	0
x_7	0	0	0	0	0	-1	-1	-1	0

$\Rightarrow$

		x_1	x_2	x_3	x_4	x_5	x_6
	x_6	0.5	0	1	-0.5	0	11.5
$\Rightarrow$	x_2	0.5	1	0	0.5	0	3.5
	x_2	6	0	0	1	1	25
	x_1	0	0	0	-2	0	-14

Otrzymany wynik to: $x_0^* = 14$ dla $x^* = [0,\ 3.5]$. Ograniczenie $x_1 + 2x_2 \leq 7$ jest liniową kombinacją funkcji celu, więc optymalne wyniki znajdują się na odcinku $[0,\ 3.5] - [4.16,\ 1.416]$. Uruchomiona metoda podziału i ograniczeń powinna znaleźć rozwiązanie całkowitoliczbowe:

$$X = \left\{ x : \begin{array}{lcl} x_1 + x_2 & \leq & 15 \\ x_1 + 2x_2 & \leq & 7 \\ 5x_1 - 2x_2 & \leq & 18 \\ x_1 & \leq & 3 \end{array} \right.$$

Wynik całkowitoliczbowy został znaleziony w pierwszej iteracji: $x_0^* = 14$ (czyli równa się nie tylko górnemu oszacowaniu, ale również rozwiązaniu PL), a $x^* = [1, 3]$.

4.5 Interpretacja praktyczna

Przykład interpretacji praktycznej zadania programowania liniowego można zapożyczyć z książki *Badania operacyjne w przykładach i zadaniach* [5]:

> *Przedsiębiorstwo produkuje cztery wyroby: W_1, W_2, W_3, W_4. Dwa spośród wielu środków używanych w procesie produkcji są limitowane. Limity te wynoszą: środek I - 90000 j., środek II - 120000 j. Nakłady limitowanych środków na jednostkę produkcji zawarto w tablicy:*

Środki produkcji	Jednostkowe nakłady			
	W_1	W_2	W_3	W_4
I	1	2	1.5	6
II	2	2	1.5	4

> *Zysk osiągany na jednostce produkcji kształtuje się odpowiednio: 400, 600, 300, 1200 zł. Ustalić optymalne rozmiary produkcji poszczególnych wyrobów. Podać łączny zysk zrealizowany przy optymalnym asortymencie produkcji.*

Aby rozwiązać dane zadanie, należy przyjąć funkcję celu:

$$\max x_0 = 400x_1 + 600x_2 + 300x_3 + 1200x_4$$

dla następujących ograniczeń:

$$X = \left\{ x : \begin{array}{rcl} x_1 + 2x_2 + 1.5x_3 + 6x_4 & \leq & 90000 \\ 2x_1 + 2x_2 + 1.5x_3 + 4x_4 & \leq & 120000 \end{array} \right.$$

Z racji wymiaru problemu R^4 nie można wykreślić warstwicy funkcji celu.

Tablica simpleksowa złożona z A, b i c zawiera w sobie dopuszczalne rozwiązanie bazowe, więc faza pierwsza dwufazowego algorytmu simpleks zostaje pominięta. Zadanie wymaga aż trzech przekształceń tablicy simpleksowej:

	x_1	x_2	x_3	x_4	x_5	x_6	
x_5	1	2	1.5	6	1	0	90000
x_6	2	2	1.5	4	0	1	120000
x_0	400	600	300	1200	0	0	0

$\Rightarrow$

	x_1	x_2	x_3	x_4	x_5	x_6	
x_4	0.167	0.333	0.25	1	0.167	0	15000
x_6	1.333	0.667	0.50	0	-0.667	1	60000
x_0	200	200	0	0	-200	0	-18000000

$\Rightarrow$

	x_1	x_2	x_3	x_4	x_5	x_6	
x_4	0	0.25	0.188	1	0.25	-0.125	7500
x_1	1	0.5	0.375	0	-0.5	0.75	45000
x_0	0	100	-75	0	-100	-150	-27000000

$\Rightarrow$

	x_1	x_2	x_3	x_4	x_5	x_6	
x_2	0	1	0.75	4	1	-0.5	30000
x_1	1	0	-0	-2	-1	1	30000
x_0	0	0	-150	-400	-200	-100	-30000000

Wynik $x_0^* = 30000000$ dla $x^* = [30000,\ 30000]$ jest tożsamy z wynikiem przedstawionym w książce. Z racji całkowitoliczbowego rozwiązania, wynik zadania programowania liniowego jest również wynikiem programowania liniowego całkowitoliczbowego.

5 Powstawanie błędów w PCL

5.1 Sprawdzanie warunku całkowitoliczbowości

Dla każdej liczby zmiennoprzecinkowej k można znaleźć wartość f_r, która spełni założenie:

$$k = \lfloor k \rfloor + f_r$$

Aby sprawdzić czy daną liczbę należy traktować jako liczbę całkowitą w odniesieniu do zadanej precyzji ε, należy sprawdzić logiczny wynik nierówności:

$$\min\{f_r, 1 - f_r\} \leq \varepsilon$$

Przykładowo dla $\varepsilon = 10^{-4}$ oraz $k = 0.9$ zapytanie zwróci fałsz, gdyż:

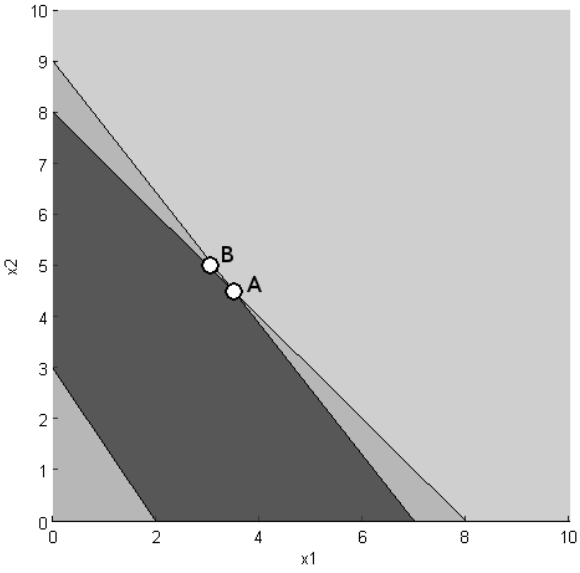
$$\min\{0.9, 0.1\} = 0.1 \not\leq 0.0001 \Rightarrow 0$$

Natomiast już dla $k = 0.9999$ można mówić o spełnieniu warunku na liczbę całkowitą:

$$\min\{0.9999, 0.0001\} = 0.0001 \leq 0.0001 \Rightarrow 1$$

5.2 Dyskusja błędów i konsekwencje niedokładności obliczeń numerycznych

Prawidłowe rozpoznawanie całkowitoliczbowości ma krytyczny wpływ na prawidłowe rozwiązanie zadania programowania liniowego całkowitoliczbowego. Zgodnie z przedstawionym powyżej sposobem sprawdzania warunku całkowitoliczbowości, dokładność oraz czas obliczeń zależą od wielkości ε - im mniejszy parametr precyzji, tym dłuższy czas wykonywania się algorytmu, ale także, do pewnego stopnia, lepszy wynik.



Przykładowo dla abstrakcyjnie wysokiej wartości $\varepsilon = 10^{-1}$ nawet liczba 10.9 zostanie zinterpretowana jako całkowitoliczbowa. Po kilkudziesięciu próbach zaobserwowano, że zmniejszanie precyzji poniżej 10^{-4} nie przynosi już żadnych pozytywnych rezultatów - algorytm podziału i ograniczeń albo znajduje optymalny wynik, który faktycznie jest całkowitoliczbowy, albo przekracza zadaną liczbę iteracji i nie znajduje nic.

Javascript, zgodnie ze swoją dokumentacją[6], wszystkie zmiennoprzecinkowe liczby rozumie jako *float*, czyli 64-bitowe liczby podwójnej precyzji. Błędy numeryczne pojawiające się z racji skończonej liczby pamięci przeznaczonej na pojedynczą zmienną można zaobserwować nawet przy najprostszych działaniach: przykładowo `0.1 * 0.1 = 0.010000000000000002`. Im więcej dzielenia liczb, tym więcej błędów się napiętrza, a eliminacja Gaussa oraz usuwanie wierszy z tablicy simpleksowej wymagają dzielenia prawie wszystkich elementów tablicy.

Jednakże przy 64 bitach dla jednej liczby trudno doprowadzić do tak potężnego błędu numerycznego, który zmieniłby oczekiwaną liczbę na poziomie 10^{-9} czy nawet 10^{-4}. Teoretycznie nierozpoznanie całkowitoliczbowości skutkowałoby dłuższą pracą algorytmu (wykonanie niepotrzebnych iteracji) lub nawet zafałszowałoby wynik (dodanie nieprawidłowych odcięć, pominięcie optymalnego rozwiązania). Z kolei zbyt mało restrykcyjne sprawdzanie warunku całkowitoliczbowości mogłoby przedwcześnie zakończyć algorytm ze złym wynikiem.

Przeprowadzono testy dla kilku różnych zestawów ograniczeń, aby sprawdzić doświadczalnie jak bardzo wyniki PL różnią się od PCL. Zebrane oraz przeliczone dane zebrano w tabeli:

Wynik PL			Wynik PCL			różnica [%]		
x_0	x_1	x_2	x_0	x_1	x_2	Δx_0	Δx_1	Δx_2
43.5	3.5	4.5	43	3	5	1.15	15.38	11.76
7.75	2.75	2.25	6	2	2	25.45	31.58	11.76
41.25	2.25	3.75	40	0	5	3.08	-	28.57

[6]dokumentacja dostępna pod adresem `http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf`

Nawet dla tak małej próbki danych widać, że wartości wyliczone w ramach poszukiwania całkowitoliczbowego wyniku znacznie odbiegają od wartości wyliczonych pierwotnie. Wbrew intuicyjnym przewidywaniom optymalny całkowitoliczbowy wynik bardzo rzadko będzie równy górnemu ograniczeniu - o wiele częściej algorytm podziału i ograniczeń musi przeiterować L razy, a znaleziony wynik jest mniejszy od oczekiwanego.

6 Wnioski

6.1 Algorytm optymalizacji

Algorytm simpleks został opracowany na podstawie wykładu oraz slajdów kursu *Teoria i metody optymalizacji* oraz literatury przedstawionej w sekcji *Literatura*. Wykorzystany został dwufazowy algorytm simpleks, który pozwala na obliczenie optimum dla funkcji celu ograniczonej zarówno mniejszościowymi, jak i większościowymi ograniczeniami. W przypadku braku ograniczeń większościowych, pierwsza faza algorytmu nie jest uruchamiana.

Sam algorytm odnajdowania rozwiązania optymalnego nie był trudnym zagadnieniem jeżeli chodzi o implementację. Podstawą było zrozumienie całego procesu na prostych przykładach analizowanych "na kartce", gdzie wyniki można było znaleźć metodą geometryczną, a później sprawdzić czy przyjęty sposób rozumowania prowadzi do tego samego rezultatu.

Przełożenie algorytmu na program komputerowy nie obyło się bez problemów, a największą trudność sprawiła analiza wszystkich możliwych przypadków. Znajdowanie jednego rozwiązania optymalnego funkcjonowało poprawnie w zasadzie już w dniu zero, jednak odnalezienie warunków na sprzeczność i nieograniczoność zajęło trochę czasu. Choć przykłady znajdowały się w literaturze naukowej, należało przekonwertować matematyczny zapis macierzowy na kod komputerowy dopasowany do zaimplementowanego już trzonu algorytmu. Błędem koncepcyjnym było późniejsze dopisywanie kolejnych warunków dla różnych przypadków - ten krok powinien być zaplanowanym na samym początku projektowania algorytmu, aby kod wyglądał jak najbardziej naturalnie. Najtrudniejszym do zrealizowania okazało się znajdowanie wielu rozwiązań na półprostej lub odcinku.

Samo zagadnienie większej niż jedno liczby rozwiązań optymalnych nie pojawiło się w dostępnej literaturze, więc wykorzystano pomocniczą wiedzę pozyskaną z opracowań kursów optymalizacji z różnych uczelni wyższych. Niestety prawie żadne źródła nie mówiły o takim przypadku dla dwafazowego algorytmu simpleks, więc została zaimplementowana dodatkowa metoda, która jeszcze przed pętlą algorytmu sprawdza czy któreś z ograniczeń nie jest kombinacją liniową funkcji celu. Sprawdzene jest to w dwóch pętlach, które iterują najpierw po wierszach, a potem po kolejnych elementach wiersza w macierzy A. Należy sprawdzić czy wynik dzielenia i-tego wyrazu ograniczenia przez i-ty wyraz funkcji celu daje ten sam wynik - jeżeli tak, jest to kombinacja liniowa i można wnioskować o większej niż jeden liczbie optymalnych rozwiązań.

W przypadku zaistnienia kombinacji liniowej któregoś z ograniczeń z funkcją celu, zmieniana jest procedura zatrzymywania pętli, w której poszukiwane jest optimum. Przede wszystkim znalezione rozwiązanie zostaje zapamiętane w pomocniczej strukturze danych, a algorytm działa dalej. Jeżeli następny wynik jest tożsamy z zapamiętanym - wtedy rozwiązanie optymalne znajduje się na półprostej zaczynającej się w punkcie przedstawionym jako x^*; jeżeli znaleziono inne rozwiązane, oznacza to, iż rozwiązania znajdują się na odcinku ograniczonym dwoma punktami. Do dalszej części algorytmu (również do znajdowania rozwiązania zadania programowania liniowego całkowitoliczbowego) wykorzystywane jest pierwsze znalezione optimum, jako wynik brzegowy.

Metoda podziału i ograniczeń okazała się być mniej kłopotliwym algorytmem. Podstawowym zagadnieniem było sprawdzanie czy wynik jest całkowitoliczbowy, jednak zostało to zaimplementowane zgodnie z teorią przedstawioną w sekcji 5.1 i nie sprawiło żadnych problemów.

Oprócz wymaganego warunku za zatrzymanie algorytmu - czyli liczby iteracji L zwiększającej się z każdym uruchomieniem kolejnej instancji problemu rozwiązywanego simpleksem - dodano dwa dodatkowe. Pierwszy, zgodnie z literaturą, to znalezienie wyniku, który równy jest górnemu ograniczeniu. Drugi warunek wynika bezpośrednio ze sposobu implementacji problemu i zależy od liczby elementów trzymanych na stosie. Jeżeli wszystkie węzły drzewa zostają zamknięte wynikami sprzecznymi lub nieograniczonymi, algorytm podziału i ograniczeń zapętliłby się w nieskończoność.

Liczba iteracji L zwiększa się przy wywołaniu funkcji liczącej algorytm simpleks, jednak warunek stopu znajduje się w argumencie pętli `while()`, dlatego niekiedy algorytm podziału i ograniczeń może wykonać ponad L iteracji. Dzieję się tak, gdyż do końca jednej iteracji pętli głównej mogą pozostać węzły do sprawdzenia dla badanego właśnie przypadku. Pozostawiano tę możliwość, gdyż może to pomóc znaleźć lepsze rozwiązanie, choć jest to mało prawdopodobne, aby przy stu iteracjach algorytm miał znaleźć optimum równe górnemu ogarniczeniu w iteracji 101.

Wybór zmiennej wokół której budowane są nowe ograniczenia według literatury dobierany powinien być heurystycznie. Po konsultacjach zaimplementowano przeszukiwanie całego drzewa poprzez sprawdzenie wszystkich

możliwych zmiennych. Dla kilku sprawdzanych instancji problemu wielkości dwóch-trzech zmiennych funkcji celu i od trzech do pięciu ograniczeń wynik całkowitoliczbowy był znajdowany w pierwszych trzech iteracjach. W wyjątkowych przypadkach znaleziony wynik nie był równy górnemu ograniczeniu i algorytm musiał bezskutecznie przejść przez 100 iteracji.

6.2 Program

Program uruchamia się poprzez plik `index.html` znajdujący się w głównym katalogu aplikacji. W pliku znajduje się szkielet strony napisany w języku HTML. W tym samym katalogu znajdują się jeszcze tylko pliki wejściowe ze współczynnikami funkcji celu oraz zestawem ograniczeń: `target.json` oraz `boundaries.json`.

W katalogach `\css` i `\fonts` znajdują się pliki bibliotek służących do budowania interfejsu aplikacji od strony graficznej. Dodatkowo w pliku `\css\style.css` zawarte są wszystkie opisy dodatkowych klas graficznych.

Najważniejszy katalog aplikacji to `\js`. W katalogu `\js\vendor` znajdują się wszystkie javascriptowe biblioteki, z których korzysta program. Główny plik interfejsu według koncepcji MVC to `\js\mvc.js` - tam znajdują się wszystkie podpięcia eventów, możliwość dodawania ograniczeń, zmiany funkcji celu, konwersja danych wejściowych na tablice oraz renderowanie widoków. Plik `\js\simplex.js` zawiera w sobie funkcje bezpośrednio powiązane z algorytmem simpleksowym: przygotowanie tablic simplekowych, wybór kolumn, eliminację Gaussa, sprawdzanie kombinacji liniowej oraz przede wszystkim sam algorytm. W pliku `\js\bb.js` zaimplementowano sprawdzanie całkowitoliczbowości, funkcję dodającą ograniczenie do poprzedniej instancji problemu oraz sam algorytm podziału i ograniczeń.

Program - zgodnie z sugestiami na konsultacjach - został napisany w konwencji open-source. Wykorzystano język Javascript, w którym pisane programy mogę zostać uruchomione praktycznie na wszystkich popularnych systemach operacyjnych, włączając w to systemy mobline. Z racji skryptowego charakteru języka, program nie wymaga instalowania żadnych kombilatorów i może być uruchomiony bezpośrednio z poziomu większości przeglądarek internetowych.

Do budowy interfejsu wykorzystano popularny wśród programistów i web-developerów frameworkiem Backbone.js. Dzięki wykorzystaniu wzorca architektonicznego jakim jest *Model-View-Controller*, interfejs od strony kodu jest czytelny i łatwy do zrozumienia nawet dla osoby, nie mającej wcześniej styczności z językiem JS. Wykorzystano również inne popularne i darmowe biblioteki, aby interfejs wyglądał nowocześnie i był łatwy w implementacji: Bootstrap, Underscore, Font Awesome i przede wszystkim jQuery.

Implementacja algorytmów obyła się bez poważniejszych problemów. Wektory i macierze zostały zbudowane na tablicach i tablicach tablic. Utrudniło to niektóre operacje, takie jak edycję lub usuwanie kolumny, jednak przyspieszyło przepisywanie danych, dostęp do poszczególnych komórek czy podstawowe operacje algebraiczne.

Javascript jest językiem typizowanym dynamicznie, więc początkowo tablice simpleksowe zostały zaimplementowane jako właściwa macierz liczbowa z tekstowymi etykietami w zerowej kolumnie i wierszu. Ostatecznie zrezygnowano z tego pomysłu na rzecz przyspieszenia wykonywania operacji, a etykiety przepisano do dodatkowej tablicy pomocniczej.

Program został przetestowany na trzech najpopularniejszych przeglądarkach: Mozilli Firefox, Google Chrome oraz Internet Explorerze. Okazało się, że ten ostatni ma najbardziej ścisły interpreter języka Javascript, który nie pozwala na przykład na zwracanie kilku wartości i odczytywanie ich jako tablica (podobnie jak w MATLAB-ie: `[a, b] = func(x);`), przez co trzeba było wprowadzić kilkanaście drobnych zmian w kodzie.

Pełny kod źródłowy programu znajduje się na prywatnym repozytorium autora projektu. Każdy zainteresowany może przejrzeć wszystkie pliki, ściągnąć je na własny dysk albo zrobić *fork* projektu i wprowadzić własne zmiany. Repozytorium jest dostępne pod adresem `https://bitbucket.org/krewak/pwr-opt/`

Literatura

[1] Maciej M. Sysło, Narsingh Deo, Janusz S. Kowalik, *Algorytmy optymalizacji dyskretnej*, wydanie trzecie, Wydawnictwo Naukowe PWN, Warszawa 1999

[2] Andrzej Stachurski, Andrzej P. Wierzbicki, *Podstawy optymalizacji*, Oficyna Wydawnicza PW, Warszawa 1999

[3] Jacek Stadnicki, *Teoria i praktyka rozwiązywania zadań optymalizacji z przykładami zastosowań technicznych*, Wydawnictwo WNT, Warszawa 2006

[4] Robert Garfinkel, George L. Nemhauser, *Programowanie całkowitoliczbowe*, Wydawnictwo Naukowe PWN, Warszawa 1978

[5] Zbigniew Jędrzejczyk, Jerzy Skrzypek, Karol Kukuła, Anna Walkosz, pod redakcją Karola Kukuły, *Badania operacyjne w przykładach i zadaniach*, Wydawnictwo Naukowe PWN, Warszawa 2003

Spis treści