Programy liniowe, gry i algorytmy

Łukasz KOWALIK Instytut Informatyki, Wydział Matematyki, Informatyki i Mechaniki, Uniwersytet Warszawski

1. Bliźniacze programy liniowe. *Program liniowy* jest to minimalizacja (maksymalizacja) liniowej *funkcji celu* o n argumentach $x_1, x_2, \ldots, x_n$ przy zachowaniu pewnej liczby równości lub nierówności liniowych zawierających zmienne x_i. Oto przykład:

(1) zmaksymalizuj $x_1 + 2x_2$
z zachowaniem warunków $x_1 + 3x_2 \leqslant 9$,
$2x_1 + x_2 \leqslant 8$,
$x_1, x_2 \geqslant 0$.

Ponieważ w powyższym przykładzie mamy tylko dwie zmienne, możemy łatwo wyobrazić sobie *zbiór rozwiązań dopuszczalnych*, tzn. zbiór punktów w przestrzeni dwuwymiarowej spełniających podane nierówności. Jest to czworokąt o wierzchołkach $(0,0)$, $(4,0)$, $(0,3)$ i $(3,2)$. Interesują nas *rozwiązania optymalne*, czyli rozwiązania dopuszczalne, które minimalizują (maksymalizują) funkcję celu. W szczególności, jeśli zbiór rozwiązań dopuszczalnych jest nieograniczony, rozwiązanie optymalne może nie istnieć. Dla programów o wszystkich zmiennych nieujemnych można dość łatwo wykazać (zachęcam Czytelnika, aby spróbował swoich sił dla dwóch wymiarów), że jeśli jakieś rozwiązanie optymalne istnieje, to istnieje też takie, które jest wierzchołkiem zbioru rozwiązań dopuszczalnych. Po sprawdzeniu wszystkich wierzchołków okazuje się, że rozwiązanie optymalne dla powyższego przykładu to $(3,2)$.

Czy programy liniowe mają coś wspólnego z informatyką? Okazuje się, że bardzo wiele. Po pierwsze, istnieją (bardzo ciekawe!) efektywne algorytmy rozwiązujące programy liniowe. Ponadto, programy liniowe są kluczowym narzędziem w optymalizacji kombinatorycznej, algorytmach aproksymacyjnych czy algorytmach online. Dla przykładu, problem maksymalnego przepływu jest szczególnym przypadkiem programu liniowego (dla sieci o n wierzchołkach i m krawędziach taki program ma m zmiennych i $O(n+m)$ ograniczeń).

Przejdźmy teraz do głównego tematu tego artykułu: każdy program liniowy ma swojego **brata bliźniaka**. Aby się o tym przekonać, spróbujmy znaleźć jakiś prosty (istotnie prostszy niż dowodzenie twierdzenia o wierzchołkach) sposób, żeby przekonać kolegę, że rozwiązanie $(3,2)$, o wartości funkcji celu 7, jest faktycznie rozwiązaniem optymalnym, albo chociaż że jest bliskie optymalnemu. Innymi słowy, szukamy *oszacowania górnego* na wartość funkcji celu rozwiązań dopuszczalnych. Zauważmy, że tak się szczęśliwie złożyło, iż rozwiązania dopuszczalne naszego programu są nieujemne. Dlatego proste oszacowanie możemy dostać już na podstawie pierwszej nierówności: $x_1 + 2x_2 \leqslant x_1 + 3x_2 \leqslant 9$. Dodając pierwsze dwie nierówności pomnożone przez $\frac{1}{2}$, otrzymujemy jeszcze lepsze oszacowanie: $x_1 + 2x_2 \leqslant \frac{3}{2}x_1 + 2x_2 \leqslant 8\frac{1}{2}$. Spróbujmy pójść tą drogą jeszcze dalej. Mianowicie, chcemy znaleźć takie liczby $y_1, y_2 \geqslant 0$, że mnożąc i-tą nierówność przez y_i i dodając otrzymane nierówności, dostaniemy jak najlepsze oszacowanie na wartość funkcji celu. Żeby wartość funkcji celu faktycznie szacowała się z góry przez taką sumę, musi zachodzić $1 \leqslant 1 \cdot y_1 + 2 \cdot y_2$ oraz $2 \leqslant 3 \cdot y_1 + 1 \cdot y_2$. A więc otrzymaliśmy pewne zadanie do rozwiązania. Spostrzegawczy Czytelnik zapewne zauważył już, że jest to zadanie programowania liniowego! Możemy je zapisać następująco:

(2) zminimalizuj $9y_1 + 8y_2$
z zachowaniem warunków $y_1 + 2y_2 \geqslant 1$,
$3y_1 + y_2 \geqslant 2$,
$y_1, y_2 \geqslant 0$.

Ponownie stosując trik z wyznaczeniem wierzchołków wielokąta, otrzymujemy, że $(y_1, y_2) = (\frac{3}{5}, \frac{1}{5})$ jest rozwiązaniem optymalnym naszego nowego programu. A jaką otrzymaliśmy wartość funkcji celu? Mamy $9y_1 + 8y_2 = 7$, czyli idealnie! Możemy więc łatwo przekonać kolegę, że rozwiązanie $(3,2)$ oryginalnego problemu jest nie tylko bliskie optymalnemu, ale nawet że jest optymalne.

Program (2) nazywamy programem *dualnym* do programu (1), który z kolei nazywamy programem *prymalnym*. Rozumowanie, które zastosowaliśmy powyżej, można łatwo przenieść na dowolny program maksymalizacyjny, w którym wszystkie zmienne są nieujemne. Jeśli dodatkowo zapiszemy wszystkie m nierówności (oprócz „nieujemnościowych") w postaci $\sum_j a_{i,j}x_j \leqslant b_i$, mamy do czynienia z programem *w postaci standardowej*. Dla takich programów „przepis" na program dualny jest wyjątkowo prosty: transponujemy macierz współczynników (tzn. zamieniamy wiersze z kolumnami), zamieniamy maksymalizację na minimalizację i odwracamy kierunek nierówności.

$$\max c_1x_1 + \ldots + c_nx_n, \quad \left\{\begin{array}{ccccc} a_{11}x_1 & + \ldots + & a_{1n}x_n & \leqslant & b_1, \\ \vdots & \ddots & \vdots & \vdots & \\ a_{m1}x_1 & + \ldots + & a_{mn}x_n & \leqslant & b_m, \end{array}\right\} \quad x_1, \ldots, x_n \geqslant 0$$

$$\longrightarrow \quad \min b_1y_1 + \ldots + b_my_m, \quad \left\{\begin{array}{ccccc} a_{11}y_1 & + \ldots + & a_{m1}y_m & \geqslant & c_1, \\ \vdots & \ddots & \vdots & \vdots & \\ a_{1n}y_1 & + \ldots + & a_{mn}y_m & \geqslant & c_n, \end{array}\right\} \quad y_1, \ldots, y_m \geqslant 0.$$

Widzimy, że dla programów w postaci standardowej zmienne nieujemne w programie prymalnym odpowiadają nierównościom w programie dualnym i nierówności w programie prymalnym odpowiadają zmiennym nieujemnym w programie dualnym. Za pomocą nieco bardziej skomplikowanych reguł możemy dla *dowolnego* programu napisać program dualny. Mianowicie, zmienne bez warunku nieujemności generują równość, a równości generują zmienne bez warunku nieujemności (w dalszej części artykułu pojawi się przykład).

Zapraszam w tym miejscu Czytelnika, aby stosując podobne rozumowanie jak powyżej, poszukał ograniczenia *dolnego* na rozwiązania programu (2). Jak można się domyślić, to ograniczenie otrzymujemy jako kombinację liniową nierówności o współczynnikach będących rozwiązaniem optymalnym programu (1). Tak więc dla programu minimalizacyjnego również możemy określić program dualny. Ponadto, program dualny do programu dualnego to program prymalny.

Niech v_P będzie wartością funkcji celu dla rozwiązania optymalnego programu maksymalizacyjnego P oraz niech v_D będzie wartością funkcji celu dla rozwiązania optymalnego programu dualnego do P. Z konstrukcji wiemy, że $v_P \leqslant v_D$. Tę własność nazywamy *słabą dualnością*. W naszym przykładzie okazało się jednak, że $v_P = v_D$. Czy to był przypadek? Okazuje się, że nie, równość zachodzi *zawsze*! Tę mocniejszą własność nazywamy twierdzeniem o (silnej) dualności. Jego dowód wykracza poza ramy tego artykułu. Silna dualność ma wiele pasjonujących konsekwencji. W szczególności, liczne tzw. twierdzenia minimaksowe w kombinatoryce (np. twierdzenie o maksymalnym przepływie i minimalnym przekroju czy twierdzenie Königa–Egerváry'ego) są szczególnymi przypadkami twierdzenia o dualności.

2. Bliźniacze strategie. Za odkrywcę dualności programów liniowych uważa się Johna von Neumanna. Odkrycia tego dokonał niejako „przy okazji", badając zagadnienia teorii gier. Przyjrzymy się teraz związkom tych dwóch teorii.

Wojtek i Kasia grają w następującą grę. Wojtek ma dwie monety: 1 zł i 2 zł. Kasia również ma dwie monety: 5 zł i 10 zł. Każde z nich niezależnie wybiera jedną ze swoich monet i kładzie ją na stół. Jeśli suma wartości monet na stole jest parzysta, to zabiera je Wojtek, w przeciwnym przypadku zabiera je Kasia.

Możemy uogólnić tę grę w następujący sposób. Dana jest dowolna (niekoniecznie kwadratowa) macierz $A = [a_{ij}]$ liczb rzeczywistych oraz dwóch graczy: W (gracz wierszowy) i K (gracz kolumnowy). Gracz W wybiera wiersz w macierzy A, natomiast gracz K niezależnie wybiera kolumnę k macierzy A. Następnie równocześnie ujawniają sobie swoje wybory i gracz W dostaje a_{wk} złotych od gracza K. Jest to tzw. *gra o sumie zerowej* (tzn. suma zysków graczy wynosi 0). Dla przykładu, następująca macierz odpowiada opisanej wcześniej grze:

$$A = \begin{bmatrix} 5 & -1 \\ -2 & 10 \end{bmatrix}.$$

Jak mogą wyglądać strategie w grze tego typu? *Strategią mieszaną* gracza W nazywamy dowolny rozkład prawdopodobieństwa na wierszach macierzy A. Np. gracz W mógłby z prawdopodobieństwem $p_1 = \frac{1}{4}$ wybierać wiersz 1 i z prawdopodobieństwem $p_2 = \frac{3}{4}$ wybierać wiersz 2. (Analogicznie definiujemy strategię mieszaną dla gracza K). Jak wygląda *optymalna* strategia mieszana dla W? Gracz W maksymalizuje wartość oczekiwaną swojej wygranej. Gracz K ma do wyboru 2 ruchy. Gdy wybierze kolumnę 1, wartość oczekiwana kwoty, którą zapłaci graczowi W, wynosi $5 \cdot p_1 - 2 \cdot p_2$. Gdy wybierze kolumnę 2, ta wartość wynosi $(-1) \cdot p_1 + 10 \cdot p_2$. Gracz K chce zapłacić jak najmniej, a więc wybierze bardziej korzystną opcję i zapłaci $\min\{5p_1 - 2p_2, -p_1 + 10p_2\}$. Teoretycznie K mógłby grać przeciwko W, również używając strategii mieszanej. Zauważmy jednak, że jeśli, tak jak w powyższej analizie, K zna p_1 i p_2, to opłaca mu się grać jedną z dwóch strategii czystych, tzn. zapłacić

$$\min_{\substack{q_1+q_2=1 \\ q_1,q_2 \geqslant 0}} (q_1(5p_1 - 2p_2) + q_2(-p_1 + 10p_2)) = \\ = \min\{5p_1 - 2p_2, -p_1 + 10p_2\}.$$

Stąd, chcąc mieć optymalną strategię dla W, musimy rozwiązać następujące zadanie:

$$\begin{aligned} &\text{zmaksymalizuj } \min\{5p_1 - 2p_2, -p_1 + 10p_2\} \\ &\text{z zachowaniem warunków } p_1 + p_2 = 1, \\ &\qquad\qquad\qquad\qquad p_1, p_2 \geqslant 0. \end{aligned}$$

Łatwo zauważyć, że rozwiązanie spełnia równanie $5p_1 - 2p_2 = -p_1 + 10p_2$, a więc $p_1 = \frac{2}{3}$, $p_2 = \frac{1}{3}$. Taka strategia mieszana gwarantuje graczowi W średnią wygraną o wartości $\frac{8}{3}$ zł.

A jaka jest optymalna strategia (q_1, q_2) dla gracza K? Z analogicznego rozumowania jak dla W dostajemy

$$\begin{aligned} &\text{zminimalizuj } \max\{5q_1 - q_2, -2q_1 + 10q_2\} \\ &\text{z zachowaniem warunków } q_1 + q_2 = 1, \\ &\qquad\qquad\qquad\qquad q_1, q_2 \geqslant 0. \end{aligned}$$

Otrzymujemy $q_1 = \frac{11}{18}$, $q_2 = \frac{7}{18}$. Wówczas K ma gwarancję, że nie straci średnio więcej niż $\frac{8}{3}$ zł (no cóż, to nie jest sprawiedliwa gra). Zauważmy, że w obu przypadkach dostaliśmy tę samą liczbę. To oznacza, że jeśli W i K obiorą znalezione przez nas strategie, to nawet jeśli je wyjawią sobie nawzajem, i tak żadnemu z nich nie będzie się opłacało zmienić obranej strategii. Jak to się stało? Odpowiedź będzie jasna, jeśli zauważymy, że powyższe dwa zadania optymalizacyjne możemy przedstawić za pomocą programów liniowych:

$$\begin{aligned} &\text{zmaksymalizuj } t \\ &\text{z zachowaniem warunków } t \leqslant 5p_1 - 2p_2, \\ &\qquad\qquad\qquad\qquad t \leqslant -p_1 + 10p_2, \\ &\qquad\qquad\qquad\qquad p_1 + p_2 = 1, \\ &\qquad\qquad\qquad\qquad p_1, p_2 \geqslant 0, \\ &\text{zminimalizuj } r \\ &\text{z zachowaniem warunków } r \geqslant 5q_1 - q_2, \\ &\qquad\qquad\qquad\qquad r \geqslant -2q_1 + 10q_2, \\ &\qquad\qquad\qquad\qquad q_1 + q_2 = 1, \\ &\qquad\qquad\qquad\qquad q_1, q_2 \geqslant 0. \end{aligned}$$

Te programy są do siebie dualne! A więc fakt, że mają takie same optymalne wartości funkcji celu, wynika z twierdzenia o silnej dualności. W ten sposób „udowodniliśmy" (na konkretnym przykładzie) słynne twierdzenie minimaksowe von Neumanna.

Twierdzenie 1. *W każdej dwuosobowej grze o sumie zerowej istnieją strategie dla graczy* W *i* K *oraz taka liczba* $V \in \mathbb{R}$, *że strategia gracza* W *gwarantuje mu średnią wygraną* V *niezależnie od strategii* K *oraz strategia* K *gwarantuje mu średnią wygraną* $-V$ *niezależnie od strategii* W. *Innymi słowy, dla dowolnej macierzy* A *o wymiarach* $n \times m$ *mamy*

$$\max_{\substack{\sum p_j = 1 \\ p_j \geqslant 0}} \min_{\substack{\sum q_i = 1 \\ q_i \geqslant 0}} \sum_{\substack{1 \leqslant i \leqslant m \\ 1 \leqslant j \leqslant n}} p_j q_i a_{ji} = \\ = \min_{\substack{\sum q_i = 1 \\ q_i \geqslant 0}} \max_{\substack{\sum p_j = 1 \\ p_j \geqslant 0}} \sum_{\substack{1 \leqslant i \leqslant m \\ 1 \leqslant j \leqslant n}} p_j q_i a_{ji}.$$

3. Dolne granice dla algorytmów randomizowanych. Do czego może przydać się nam twierdzenie von Neumanna? Okazuje się, że pojęcie gry pojawia się częściej, niż mogłoby się wydawać. Na przykład złożoność algorytmu możemy interpretować jako wynik gry między algorytmem a przeciwnikiem, który generuje „złośliwe" instancje (tj. dane wejściowe) problemu. Algorytm deterministyczny nie może wykonać w takiej grze żadnego ruchu, czeka

jedynie na posunięcie przeciwnika. Często większe szanse ma algorytm randomizowany, czyli taki, który pewne decyzje podejmuje na podstawie wyniku rzutu monetą. Zajmiemy się tu algorytmami typu Las Vegas, tzn. takimi, które zawsze zwracają poprawną odpowiedź, ale wykorzystują losowość i czas ich działania jest zmienną losową. W 1977 r. Andrew Yao odkrył, jak zastosować twierdzenie von Neumanna, aby wykazywać ograniczenia algorytmów Las Vegas. Zobaczmy, na czym polega odkrycie Yao.

Rozważmy abstrakcyjny problem algorytmiczny. Niech $\mathcal{A}$ oznacza zbiór wszystkich algorytmów deterministycznych dla danych rozmiaru n oraz niech $\mathcal{I}$ oznacza zbiór wszystkich instancji problemu rozmiaru n. Wówczas zbiór $\mathcal{I}$ jest skończony. Zbiór $\mathcal{A}$ nie jest skończony, ale możemy rozważać jedynie algorytmy, których pesymistyczny czas nie przekracza jakiejś określonej (dostatecznie dużej) wartości, i tak ograniczony zbiór jest już skończony. Niech T będzie macierzą o kolumnach indeksowanych elementami zbioru $\mathcal{A}$ i wierszach indeksowanych elementami zbioru $\mathcal{I}$, w której element $t_{A,I} = T(A, I)$ ma wartość równą liczbie kroków, które wykonuje algorytm A na instancji I. Ponadto, dla rozkładu prawdopodobieństwa α na algorytmach ze zbioru $\mathcal{A}$ niech zmienna losowa A_α oznacza algorytm wylosowany zgodnie z rozkładem α. Podobnie, dla rozkładu prawdopodobieństwa β na instancjach ze zbioru $\mathcal{I}$ niech zmienna losowa I_β oznacza instancję wylosowaną zgodnie z rozkładem β.

Zastosujmy do tej macierzy twierdzenie von Neumanna. Ponieważ

$$\min\{a, b\} = \min_{0 \leqslant q \leqslant 1} \{qa + (1 - q)b\},$$

więc równość w tym twierdzeniu możemy zapisać w równoważnej, prostszej postaci:

$$\max_{\substack{\sum p_j = 1 \\ p_j \geqslant 0}} \min_{i=1,\ldots,m} \sum_{j=1}^{n} p_j a_{ji} = \min_{\substack{\sum q_i = 1 \\ q_i \geqslant 0}} \max_{j=1,\ldots,n} \sum_{i=1}^{m} q_i a_{ji}.$$

Stąd

$$\min_\alpha \max_{I \in \mathcal{I}} \sum_{A \in \mathcal{A}} \mathbb{P}[A_\alpha = A] t_{A,I} = \\ = \max_\beta \min_{A \in \mathcal{A}} \sum_{I \in \mathcal{I}} \mathbb{P}[I_\beta = I] t_{A,I}.$$

Równoważnie

$$\min_\alpha \max_{I \in \mathcal{I}} \mathbb{E}[T(A_\alpha, I)] = \max_\beta \min_{A \in \mathcal{A}} \mathbb{E}[T(A, I_\beta)]. \tag{3}$$

Teraz zastanówmy się, co tak naprawdę dostaliśmy. Jak możemy interpretować zmienną losową A_α dla danego rozkładu α? Każdy randomizowany algorytm Las Vegas dla naszego problemu możemy traktować jako pewien rozkład prawdopodobieństwa na algorytmach ze zbioru $\mathcal{A}$. Istotnie, każdy taki algorytm korzysta z pewnej liczby bitów losowych. Możemy wartości tych bitów wylosować na samym początku algorytmu, a wówczas jego dalsze działanie jest już zdeterminowane przez te bity (a więc jest zgodne z pewnym algorytmem ze zbioru $\mathcal{A}$).

Skoro A_α oznacza pewien algorytm randomizowany, to lewa strona naszej równości oznacza *czas najlepszego* (*w sensie oczekiwanej złożoności na najgorszych danych*) *algorytmu randomizowanego* Las Vegas dla naszego problemu.

Natomiast z prawej strony wybieramy taki rozkład na instancjach, który zmaksymalizuje oczekiwany czas najlepszego algorytmu *deterministycznego*.

Wynika stąd, że równanie (3) daje nam sposób na dowodzenie *dolnych granic* na złożoność algorytmów randomizowanych. Wystarczy mianowicie znaleźć jakiś zestaw „trudnych” instancji i podać taki rozkład prawdopodobieństwa na tym zestawie, że żaden algorytm deterministyczny nie poradzi sobie z nim zbyt dobrze. Ten wniosek znany jest jako twierdzenie Yao.

Twierdzenie 2 (Twierdzenie Yao)**.** *Niech A_α będzie pewnym algorytmem Las Vegas. Niech β będzie pewnym rozkładem prawdopodobieństwa na instancjach problemu. Wówczas*

$$\max_{I \in \mathcal{I}} \mathbb{E}[T(A_\alpha, I)] \geqslant \min_{A \in \mathcal{A}} \mathbb{E}[T(A, I_\beta)].$$

Przykład: sortowanie. Chyba najbardziej znanym algorytmem randomizowanym typu Las Vegas jest algorytm sortowania Quicksort, który działa w oczekiwanym czasie $O(n \log n)$. Pokażemy, że żaden inny algorytm Las Vegas nie może działać szybciej, w sensie złożoności asymptotycznej.

Przypomnijmy znany dowód, że każdy *deterministyczny* algorytm sortujący n różnych liczb przez porównania wymaga $\Omega(n \log n)$ porównań. Każdy taki algorytm możemy modelować jako drzewo binarne. Węzły wewnętrzne drzewa odpowiadają wykonywanym porównaniom, każde wykonanie algorytmu odpowiada ścieżce od korzenia do liścia (będąc w liściu, algorytm na podstawie wyników wcześniejszych porównań może jednoznacznie wyznaczyć szukaną permutację liczb). Jeśli h jest długością najdłuższej ścieżki w drzewie, to ma ono nie więcej niż 2^h liści. Ale liści musi być co najmniej $n!$ (bo tyle jest możliwych permutacji), czyli

$$h \geqslant \log_2(n!) \geqslant \log_2((n/2)^{n/2}) = \Omega(n \log n),$$

co kończy dowód.

Zauważmy teraz, że liści na głębokości co najwyżej $\log_2(n!) - 1$ jest nie więcej niż $2^{\lfloor \log_2(n!) \rfloor - 1} \leqslant n!/2$. A więc dla każdego algorytmu deterministycznego jest co najmniej $n!/2$ permutacji, dla których ten algorytm wykona co najmniej $\log_2(n!) - 1$ kroków. Stąd, jeśli wylosujemy permutację na wejściu z rozkładem jednostajnym (tzn. każda permutacja może się pojawić z prawdopodobieństwem $1/n!$), to dla dowolnego algorytmu deterministycznego wartość oczekiwana liczby wykonanych porównań wynosi co najmniej $(\log(n!) - 1)/2$. Na podstawie twierdzenia Yao wnioskujemy więc, że dla dowolnego algorytmu Las Vegas sortującego n liczb, dla pewnej instancji oczekiwana liczba wykonanych porównań wynosi co najmniej

$$(\log_2(n!) - 1)/2 = \Omega(n \log n).$$

Pokazaliśmy więc, że Quicksort jest optymalny w klasie algorytmów Las Vegas sortujących przez porównania. Kto by pomyślał, że do tego wniosku doprowadzi nas pewna teoria dotycząca nierówności liniowych?