Geometria obliczeniowa

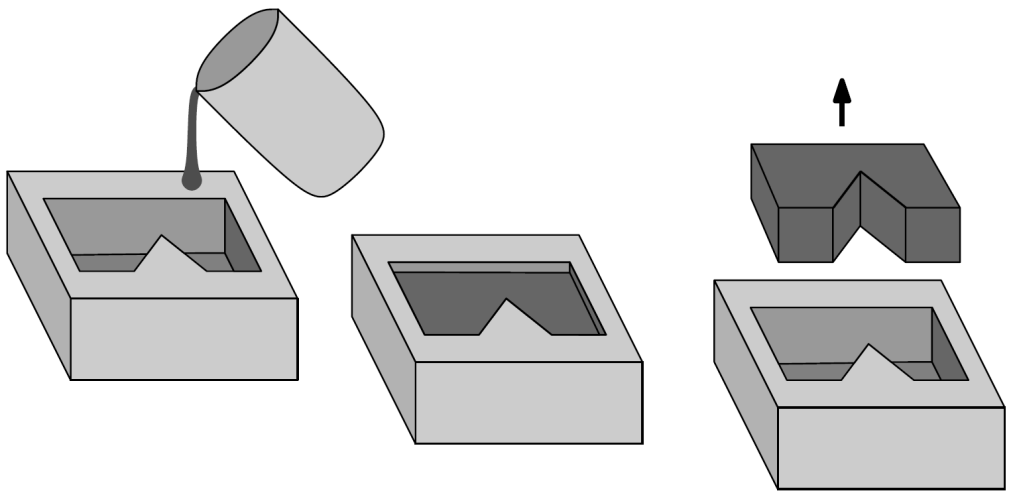
Krzysztof Gdawiec

UNIWERSYTET ŚLĄSKI
INSTYTUT INFORMATYKI

Programowanie liniowe

Geometria odlewania

Zajmiemy się teraz badaniem geometrycznych aspektów produkcji z użyciem form odlewniczych.



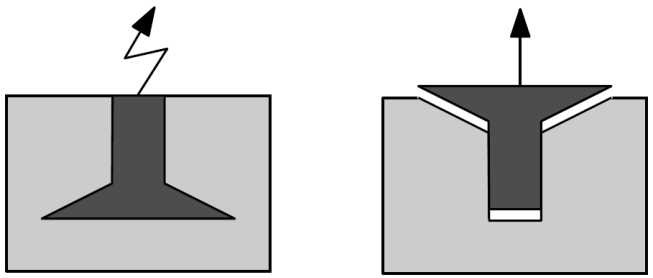
Ostatni krok, który widzimy na rysunku nie zawsze jest tak łatwy, jak to się wydaje. Przedmiot może przylepić się do formy. Czasami możemy obejść ten problem stosując różne formy.

Problem jaki będziemy rozważać to: czy dla danego obiektu istnieje forma, z której można go wyjąć?

Wprowadzimy pewne założenia:

- obiekt jest wielościanem,
- rozważamy tylko formy jednoczęściowe,
- obiekt może być wyjmowany z formy tylko z użyciem pojedynczego przesunięcia.

Kształt wgłębienia w formie jest określony przez kształt obiektu, ale różne orientacje obiektu wymagają różnych form.



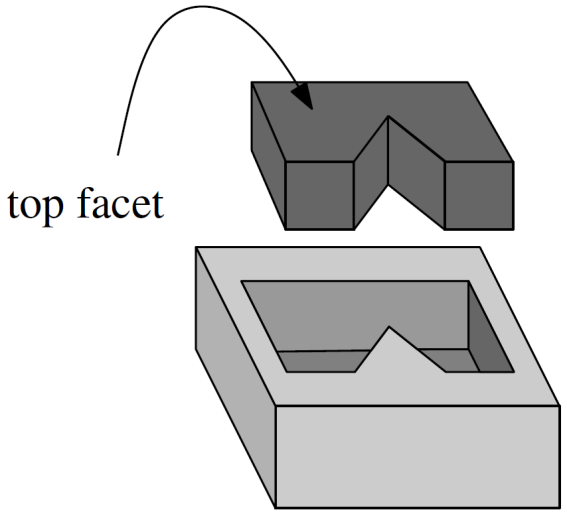
Ograniczeniem nałożonym na orientację jest że obiekt musi mieć poziomą górną ścianę (ang. top facet). Będzie to jedyna ściana pozbawiona kontaktu z formą.

Mamy tyle potencjalnych orientacji (form) ile obiekt ma ścian.

Mówimy, że obiekt można odlać (ang. castable), gdy można go wyjąć z formy dla co najmniej jednej z tych orientacji.

Niech $\mathcal{P}$ będzie trójwymiarowym wielościanem z wyznaczoną górną ścianą. Załóżmy, że forma jest prostopadłościennym blokiem z wgłębieniem, które odpowiada $\mathcal{P}$.

Gdy wielościan jest włożony do formy jego górna ściana powinna być współpłaszczyznowa z najwyższą ścianą formy, o której zakładamy, że jest równoległa do płaszczyzny xy.



Ścianę $\mathcal{P}$, która nie jest ścianą górną nazywamy ścianą zwykłą (ang. ordinary). Każda ściana zwykła f ma odpowiadającą jej ścianę w formie, którą oznaczymy przez $\hat{f}$.

W naszym problemie chcemy rozstrzygnąć czy istnieje kierunek $\overrightarrow{d}$ taki, że $\mathcal{P}$ można przesunąć do nieskończoności w kierunku $\overrightarrow{d}$, nie przecinając wnętrza formy w trakcie przesuwania.

Ponieważ ściana $\mathcal{P}$ niedotykająca formy jest jego górną ścianą, więc kierunek wyjmowania musi być ku górze, tzn. mieć dodatnią współrzędną z.

W naszym problemie chcemy rozstrzygnąć czy istnieje kierunek $\overrightarrow{d}$ taki, że $\mathcal{P}$ można przesunąć do nieskończoności w kierunku $\overrightarrow{d}$, nie przecinając wnętrza formy w trakcie przesuwania.

Ponieważ ściana $\mathcal{P}$ niedotykająca formy jest jego górną ścianą, więc kierunek wyjmowania musi być ku górze, tzn. mieć dodatnią współrzędną z.

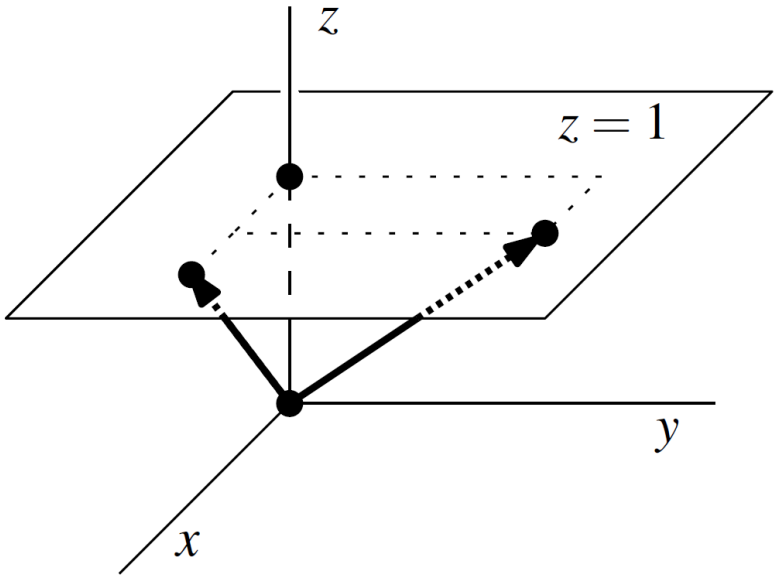
Dla naszych celów musimy zdefiniować kąt między dwoma wektorami w przestrzeni trójwymiarowej.
Bierzemy płaszczyznę rozpiętą na tych wektorach. Kąt między tymi wektorami jest mniejszym z dwóch kątów mierzonych na płaszczyźnie.

Ściana $\hat{f}$ blokuje każde przesunięcie w kierunku tworzącym kąt mniejszy niż 90° z wektorem normalnym ściany f, oznaczanym $\overrightarrow{\eta}(f)$.

Lemat. Wielościan $\mathcal{P}$ można wyjąć z formy za pomocą przesunięcia w kierunku $\overrightarrow{d} \iff \overrightarrow{d}$ tworzy kąt co najmniej 90° z wektorami normalnymi wszystkich ścian zwykłych $\mathcal{P}$.

Pozostaje nam znalezienie kierunku $\overrightarrow{d}$, który tworzy kąt co najmniej 90° z wektorem normalnym każdej ściany zwykłej $\mathcal{P}$.

Wiemy, że wektor kierunku musi mieć dodatnią współrzędną z. Wszystkie kierunki możemy reprezentować jako punkty na płaszczyźnie $z = 1$, gdzie punkt $(x, y, 1)$ odpowiada kierunkowi wektora $(x, y, 1)$.



Niech $\overrightarrow{\eta} = (\eta_x, \eta_y, \eta_z)$ będzie wektorem normalnym zwykłej ściany.

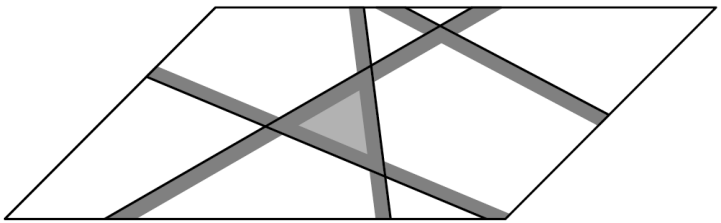
Kierunek $\overrightarrow{d} = (d_x, d_y, 1)$ tworzy kąt co najmniej 90° z $\overrightarrow{\eta} \iff$ iloczyn skalarny $\overrightarrow{d}$ i $\overrightarrow{\eta}$ jest niedodatni, tzn.

$$\eta_x d_x + \eta_y d_y + \eta_z \leqslant 0.$$

Nierówność opisuje półpłaszczyznę na płaszczyźnie $z = 1$, tzn. obszar na lewo lub prawo od prostej na tej płaszczyźnie. W przypadku ścian poziomych ($\eta_x = \eta_y = 0$) warunek jest albo niemożliwy albo zawsze spełniony (można to łatwo sprawdzić).

Każda niepozioma ściana $\mathcal{P}$ definiuje domkniętą półpłaszczyznę na płaszczyźnie $z = 1$ i dowolny punkt z części wspólnej tych półpłaszczyzn odpowiada kierunkowi, w którym można wyjąć $\mathcal{P}$. Część wspólna może być oczywiście pusta i w takim przypadku $\mathcal{P}$ nie można wyjąć z danej formy.

Przekształciliśmy problem produkcji do problemu geometrycznego na płaszczyźnie: dany jest zbiór półpłaszczyzn, znajdź punkt w ich części wspólnej lub stwierdź, że część wspólna jest pusta.



Przecięcie półpłaszczyzn

Niech $H = \{h_1, h_2, \ldots, h_n\}$ będzie zbiorem liniowych warunków z dwoma zmiennymi, tzn.

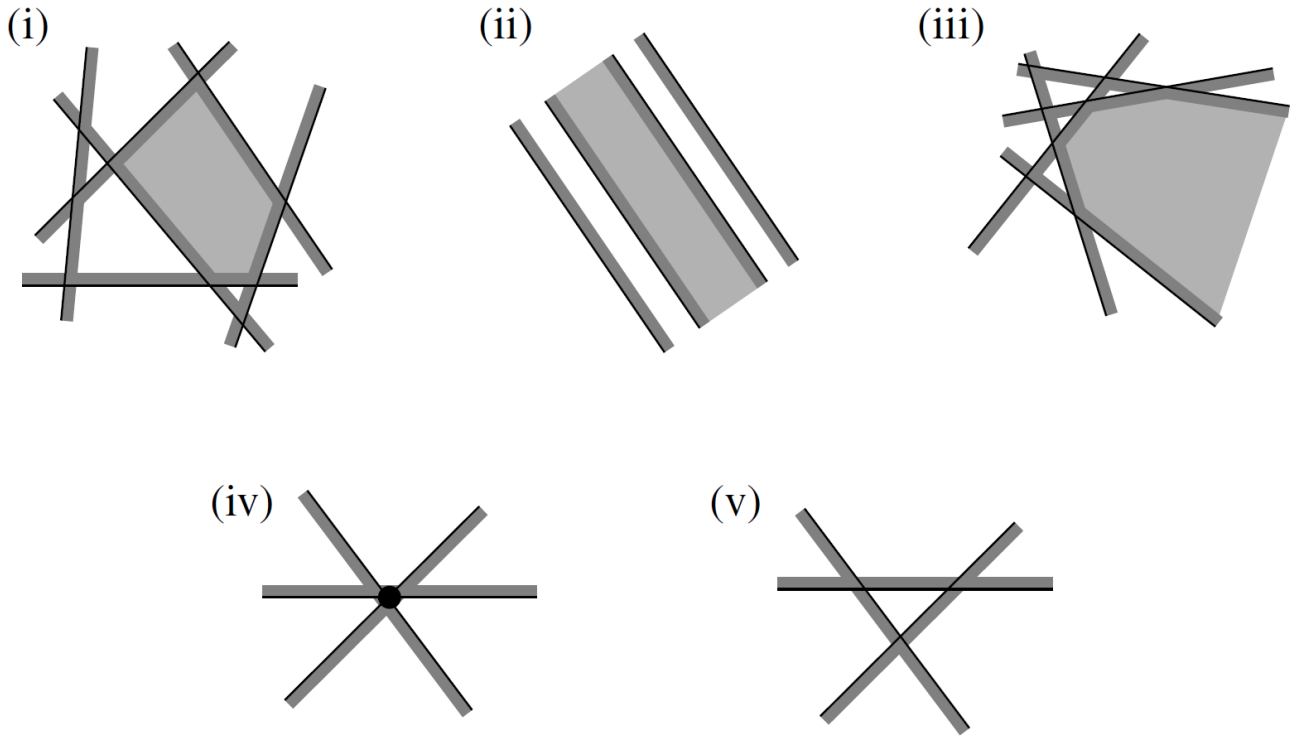
$$a_i x + b_i y \leqslant c_i,$$

gdzie a_i, b_i, c_i są stałymi takimi, że co najmniej jedna z a_i lub b_i nie jest zerem.

Będziemy chcieli znaleźć zbiór punktów C leżących w części wspólnej półpłaszczyzn H, tzn.

$$C = \bigcap_{h \in H} h.$$

Półpłaszczyzna jest zbiorem wypukłym. Ponieważ przecięcie zbiorów wypukłych jest wypukłe, więc zbiór C będzie zbiorem wypukłym. Ponadto zbiór C będzie obszarem wielokątnym składającym się z co najwyżej n krawędzi.



Algorithm IntersectHalfplanes(H)
Input. A set H of n half-planes in the plane.
Output. The convex polygonal region $C := \bigcap_{h \in H} h$.

```
1.  if card(H) = 1
2.      then C ← the unique half-plane h ∈ H
3.      else Split H into sets H1 and H2 of size ⌈n/2⌉ and ⌊n/2⌋.
4.           C1 ←IntersectHalfplanes(H1)
5.           C2 ←IntersectHalfplanes(H2)
6.           C ←IntersectConvexRegions(C1, C2)
```

IntersectConvexRegions oblicza przecięcie dwóch wypukłych, wielokątnych obszarów. Algorytm ten możemy otrzymać poprzez modyfikację algorytmu nakładania map.

Czas działania IntersectHalfplanes wynosi $\mathcal{O}(n \log^2 n)$.

Przyrostowe programowanie liniowe

W poprzednim algorytmie znajdowaliśmy zbiór wszystkich rozwiązań układu n liniowych warunków. W problemie odlewania nie musimy znać wszystkich rozwiązań. Wystarczy tylko jedno.

Znalezienie jednego rozwiązania dla układu warunków liniowych jest blisko związane z dobrze znanym problemem w badaniach operacyjnych nazywanym programowaniem liniowym (ang. linear programming). Jedyną różnicą jest to, że w programowaniu liniowym szukamy rozwiązania, które maksymalizuje daną funkcję liniową kilku zmiennych.

Problem programowania liniowego wygląda następująco:
Znajdź maksimum $c_1x_1 + c_2x_2 + \ldots + c_dx_d$
Z uwzględnieniem

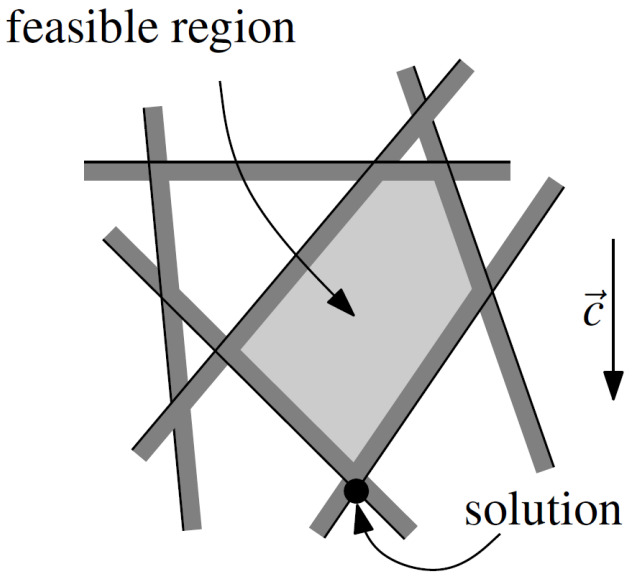
$$\begin{aligned} a_{1,1}x_1 + \ldots + a_{1,d}x_d &\leqslant b_1, \\ a_{2,1}x_1 + \ldots + a_{2,d}x_d &\leqslant b_2, \\ &\vdots \\ a_{n,1}x_1 + \ldots + a_{n,d}x_d &\leqslant b_n, \end{aligned}$$

gdzie $c_i, a_{i,j}$ i b_i są liczbami rzeczywistymi, które tworzą dane wejściowe dla problemu.

Maksymalizowana funkcja nazywa się funkcją celu (ang. objective function), a zbiór warunków wraz z funkcją celu jest programem liniowym (ang. linear program). Liczba zmiennych d jest wymiarem programu liniowego.

Zbiór punktów spełniających wszystkie warunki nazywa się obszarem dopuszczalnym (ang. feasible region). Punkty w tym obszarze nazywamy dopuszczalnymi, a punkty na zewnątrz są niedopuszczalne.

Funkcję celu można rozpatrywać jako kierunek w $\mathbb{R}^d$.
Maksymalizacja $c_1x_1 + c_2x_2 + \ldots + c_dx_d$ oznacza wówczas znalezienie punktu $(x_1, x_2, \ldots, x_d)$, który jest skrajny w kierunku $\overrightarrow{c} = (c_1, c_2, \ldots, c_d)$. Oznaczmy przez $f_{\overrightarrow{c}}$ funkcję celu zdefiniowaną przez wektor kierunku $\overrightarrow{c}$.



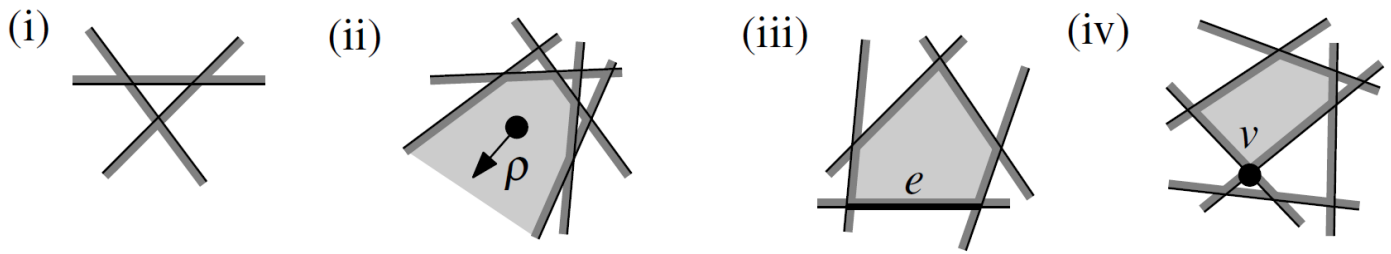
W naszym problemie mamy n liniowych warunków z dwiema zmiennymi i chcemy znaleźć rozwiązanie dla tego układu warunków. Możemy to zrobić biorąc dowolną funkcję celu i użyć dowolnej metody programowania liniowego, np. metody sympleks.

W metodach stosowanych w badaniach operacyjnych zarówno liczba warunków, jak i liczba zmiennych są duże, a w naszym przypadku liczba zmiennych wynosi tylko dwa. Zatem tradycyjne metody nie będą zbyt efektywne dlatego musimy wprowadzić nową metodę.

Oznaczmy zbiór n liniowych warunków przez H. Wektorem definiującym funkcję celu jest $\overrightarrow{c} = (c_x, c_y)$ czyli funkcja ma postać $f_{\overrightarrow{c}}(p) = c_x p_x + c_y p_y$.

Naszym celem jest znalezienie punktu $p \in \mathbb{R}^2$ takiego, że $p \in \cap H$ i $f_{\overrightarrow{c}}(p)$ jest maksymalizowana. Oznaczmy program liniowy przez $(H, \overrightarrow{c})$ i jego obszar dopuszczalny przez C.

Mogą zajść cztery przypadki rozwiązań naszego programu liniowego:



Algorytm, który podamy będzie przyrostowy. Będzie dodawać ograniczenia jedno po drugim utrzymując optymalne rozwiązanie dla pośrednich programów liniowych. Ponadto, będzie zakładać, że każdy pośredni obszar dopuszczalny ma jednoznaczny wierzchołek optymalny (ang. optimal vertex).

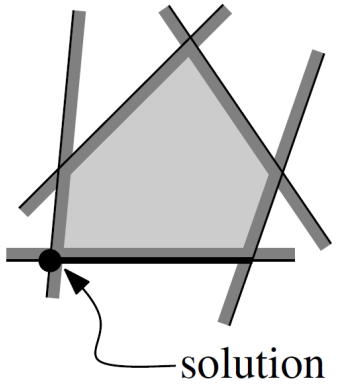
Aby spełnić wymagania musimy dodać dodatkowe ograniczenia. Będą one zapewniały, że program liniowy jest ograniczony.

$$m_1 = \begin{cases} p_x \leqslant M, & \text{gdy } c_x > 0, \\ -p_x \leqslant M, & \text{gdy } c_x \leqslant 0, \end{cases}$$

$$m_2 = \begin{cases} p_y \leqslant M, & \text{gdy } c_y > 0, \\ -p_y \leqslant M, & \text{gdy } c_y \leqslant 0, \end{cases}$$

gdzie $M \in \mathbb{R}$ jest tak duże, aby dodatkowe warunki nie wpływały na rozwiązanie optymalne jeśli oryginalny program liniowy był ograniczony.

Musimy również przyjąć konwencję, że jeśli mamy kilka punktów optymalnych, to wybieramy leksykograficznie najmniejszy.



Przy takich dwóch konwencjach każdy program liniowy, który jest dopuszczalny, ma jednoznaczne rozwiązanie będące wierzchołkiem obszaru dopuszczalnego. Nazywamy ten wierzchołek wierzchołkiem optymalnym.

Niech $(H, \overrightarrow{c})$ będzie programem liniowym. Oznaczmy $C_0 = m_1 \cap m_2$. Numerujemy półpłaszczyzny $h_1, h_2, \ldots, h_n$ oraz niech

$$H_i = \{m_1, m_2, h_1, h_2, \ldots, h_i\},$$
$$C_i = m_1 \cap m_2 \cap h_1 \cap h_2 \cap \ldots \cap h_i.$$

Każdy obszar C_i ma jednoznaczny wierzchołek optymalny v_i. Z definicji od razu widać, że

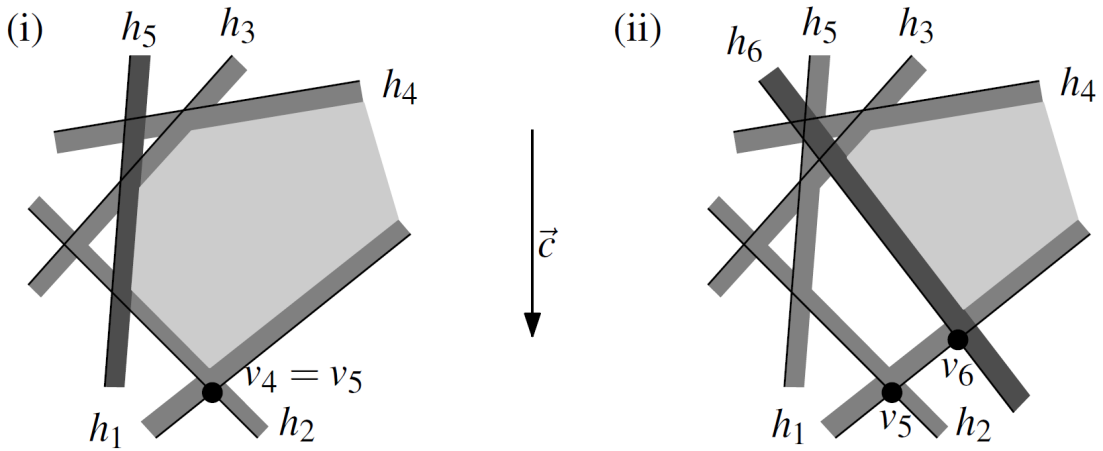
$$C_0 \supseteq C_1 \supseteq C_2 \supseteq \ldots \supseteq C_n = C.$$

Wynika stąd, że jeśli $C_i = \emptyset$ dla pewnego i, to $C_j = \emptyset$ dla wszystkich $j \geqslant i$ i program jest niedopuszczalny, więc możemy zatrzymać algorytm.

Następujący lemat mówi nam jak zmienia się wierzchołek optymalny kiedy dodajemy półpłaszczyznę h_i.

Lemat. Niech $1 \leqslant i \leqslant n$ oraz C_i i v_i będą zdefiniowane jak wcześniej. Wówczas:

- jeśli $v_{i-1} \in h_i$, to $v_i = v_{i-1}$,
- jeśli $v_{i-1} \notin h_i$, to albo $C_i = \emptyset$ albo $v_i \in \ell_i$, gdzie ℓ_i jest prostą ograniczającą h_i.



W jaki sposób znaleźć nowy wierzchołek optymalny w drugim przypadku?

Nasz problem ma postać:
Znajdź punkt p na ℓ_i, który maksymalizuje $f_{\overrightarrow{c}}$, z uwzględnieniem warunków $p \in h$ dla $h \in H_{i-1}$.

Załóżmy, że ℓ_i nie jest pionowa, więc możemy ją sparametryzować względem współrzędnej x. Wtedy możemy zdefiniować $\overline{f_{\overrightarrow{c}}} : \mathbb{R} \to \mathbb{R}$ taką, że

$$f_{\overrightarrow{c}}(p) = \overline{f_{\overrightarrow{c}}}(p_x)$$

dla $p \in \ell_i$.

Niech $\sigma(h, \ell_i)$ oznacza współrzędną x punktu przecięcia ℓ_i z prostą ograniczającą h.

Zależnie od tego czy $\ell_i \cap h$ jest ograniczone z lewej lub prawej strony dostajemy warunek na współrzędną x rozwiązania w postaci $x \geqslant \sigma(h, \ell_i)$ lub $x \leqslant \sigma(h, \ell_i)$.

Nasz problem możemy teraz zapisać następująco:
Maksymalizuj $\overline{f_{\overrightarrow{c}}}$
Z uwzględnieniem

$$x \geqslant \sigma(h, \ell_i), h \in H_{i-1} \text{ i } \ell_i \cap h \text{ jest ograniczone z lewej,}$$
$$x \leqslant \sigma(h, \ell_i), h \in H_{i-1} \text{ i } \ell_i \cap h \text{ jest ograniczone z prawej.}$$

Jest to jednowymiarowy program liniowy.

Jego rozwiązanie jest proste. Niech

$$x_l = \max_{h \in H_{i-1}} \{\sigma(h, \ell_i) : \ell_i \cap h \text{ jest ograniczone z lewej}\},$$

$$x_p = \max_{h \in H_{i-1}} \{\sigma(h, \ell_i) : \ell_i \cap h \text{ jest ograniczone z prawej}\}.$$

Przedział $[x_l, x_p]$ jest obszarem dopuszczalnym. Program liniowy jest niedopuszczalny, gdy $x_l > x_p$. Jeśli program liniowy jest dopuszczalny, to punkt optymalny jest punktem na ℓ_i albo w x_l albo x_p zależnie od funkcji celu.

Lemat. Jednowymiarowy program liniowy możemy rozwiązać w czasie liniowym.

Algorithm 2DBOUNDEDLP($H, \vec{c}, m_1, m_2$)
Input. A linear program $(H \cup \{m_1, m_2\}, \vec{c})$, where H is a set of n half-planes, $\vec{c} \in \mathbb{R}^2$, and m_1, m_2 bound the solution.
Output. If $(H \cup \{m_1, m_2\}, \vec{c})$ is infeasible, then this fact is reported. Otherwise, the lexicographically smallest point p that maximizes $f_{\vec{c}}(p)$ is reported.

1. Let v_0 be the corner of C_0.
2. Let $h_1, \ldots, h_n$ be the half-planes of H.
3. **for** $i \leftarrow 1$ **to** n
4. **do if** $v_{i-1} \in h_i$
5. **then** $v_i \leftarrow v_{i-1}$
6. **else** $v_i \leftarrow$the point p on ℓ_i that maximizes $f_{\vec{c}}(p)$, subject to the constraints in H_{i-1}.
7. **if** p does not exist
8. **then** Report that the linear program is infeasible and quit.
9. **return** v_n

Algorithm 2DBOUNDEDLP($H, \vec{c}, m_1, m_2$)
Input. A linear program $(H \cup \{m_1, m_2\}, \vec{c})$, where H is a set of n half-planes, $\vec{c} \in \mathbb{R}^2$, and m_1, m_2 bound the solution.
Output. If $(H \cup \{m_1, m_2\}, \vec{c})$ is infeasible, then this fact is reported. Otherwise, the lexicographically smallest point p that maximizes $f_{\vec{c}}(p)$ is reported.

```
1.  Let v_0 be the corner of C_0.
2.  Let h_1, ..., h_n be the half-planes of H.
3.  for i ← 1 to n
4.      do if v_{i-1} ∈ h_i
5.            then v_i ← v_{i-1}
6.            else v_i ←the point p on ℓ_i that maximizes f_c(p), subject to the
                       constraints in H_{i-1}.
7.                 if p does not exist
8.                    then Report that the linear program is infeasible and quit.
9.  return v_n
```

Lemat. Algorytm 2DBOUNDEDLP oblicza rozwiązanie ograniczonego programu liniowego z n warunkami i dwoma zmiennymi w czasie $\mathcal{O}(n^2)$ i liniowej pamięci.

Okazuje się, że dla różnych uporządkowań półpłaszczyzn z H możemy dostać różne czasy, które mogą być nawet rzędu $\mathcal{O}(n)$.

W jaki sposób znaleźć takie uporządkowanie?

Niestety, ale nie ma na to sposobu, ale możemy zastosować losowy porządek w H. Oczywiście możemy nie mieć szczęścia i trafić na uporządkowanie z czasem kwadratowym.

Algorithm 2DRANDOMIZEDBOUNDEDLP$(H, \vec{c}, m_1, m_2)$

Input. A linear program $(H \cup \{m_1, m_2\}, \vec{c})$, where H is a set of n half-planes, $\vec{c} \in \mathbb{R}^2$, and m_1, m_2 bound the solution.

Output. If $(H \cup \{m_1, m_2\}, \vec{c})$ is infeasible, then this fact is reported. Otherwise, the lexicographically smallest point p that maximizes $f_{\vec{c}}(p)$ is reported.

1. Let v_0 be the corner of C_0.
2. Compute a *random* permutation $h_1, \ldots, h_n$ of the half-planes by calling RANDOMPERMUTATION$(H[1 \cdots n])$.
3. **for** $i \leftarrow 1$ **to** n
4. **do if** $v_{i-1} \in h_i$
5. **then** $v_i \leftarrow v_{i-1}$
6. **else** $v_i \leftarrow$the point p on ℓ_i that maximizes $f_{\vec{c}}(p)$, subject to the constraints in H_{i-1}.
7. **if** p does not exist
8. **then** Report that the linear program is infeasible and quit.
9. **return** v_n

Algorithm RANDOMPERMUTATION(A)
Input. An array $A[1 \cdots n]$.
Output. The array $A[1 \cdots n]$ with the same elements, but rearranged into a random permutation.

```
1.  for k ← n downto 2
2.      do rndindex ←RANDOM(k)
3.          Exchange A[k] and A[rndindex].
```

Algorytm 2DRandomizedBoundedLP nazywa się algorytmem randomizowanym (ang. randomized algorithm). Jego czas działania zależy od pewnych losowych wyborów dokonywanych przez algorytm.

Lemat. Problem dwuwymiarowego programowania liniowego z n warunkami można rozwiązać randomizacyjnie w oczekiwanym czasie $\mathcal{O}(n)$, używając w pesymistycznym przypadku linowej pamięci.