16. Zgłoszenie, że program liniowy jest niewykonalny, z $H^* \cup h_i$ jako certyfikaty i zakończyć.
17. **return** v_n

Poniższe twierdzenie określa wydajność RANDOMIZEDLP. Chociaż uważamy d za stałą, co oznacza, że możemy określić granicę $O(n)$ na czas działania, warto przyjrzeć się bliżej zależności czasu działania od d - *patrz* koniec dowodu poniższego twierdzenia.

Twierdzenie 4.12 *Dla każdego ustalonego wymiaru d, d-wymiarowy problem programowania liniowego z n ograniczeniami może być rozwiązany w czasie oczekiwanym* $O(n)$.

Dowód. Musimy udowodnić, że istnieje stała C_d taka, że algorytm zajmuje co najwyżej C_d n oczekiwanego czasu. Postępujemy przez indukcję na wymiarze d. Dla dwóch wymiarów wynik wynika z Twierdzenia 4.10, więc załóżmy, że $d > 2$. Krok indukcji jest w zasadzie identyczny z dowodem dla przypadków dwuwymiarowych.

Zaczynamy od rozwiązania co najwyżej d programów liniowych o wymiarze $d - 1$. Zgodnie z założeniem indukcji, zajmuje to czas $O(dn)+ dC_{d-1}$ n.

Algorytm poświęca $O(d)$ czasu na obliczenie v_d . Sprawdzenie, czy $v_d \in h_i$ zajmuje $O(d)$ czasu. Czas działania wynosi zatem $O(dn)$, o ile nie liczymy czasu spędzonego w linii 13.

W linii 13 musimy rzutować $\vec{c}$ na g_i , *w czasie* $O(d)$, i przecinać i półprzestrzenie z g_i , w czasie $O(di)$. Ponadto wykonujemy wywołanie rekurencyjne o wymiarze d - 1 i i - 1 półprzestrzeni.

Zdefiniuj zmienną losową X_i , która wynosi 1, jeśli $v_{i-1} \notin h_i$, i 0 w przeciwnym wypadku. Całkowity oczekiwany czas spędzony przez algorytm jest ograniczony przez

$$O(dn)+ dC_{d-1}\, n + \sum_{i=d+1}^{n} (O(di)+C_{d-1}\, (i - 1)) \cdot \mathrm{E}[X_i].$$

Aby ograniczyć $\mathrm{E}[X_i]$, stosujemy analizę wsteczną. Rozważmy sytuację po dodaniu $h_1, \ldots, h_i$. Optymalnym punktem jest wierzchołek v_i z C_i , więc jest on zdefiniowany przez d półprzestrzeni. Teraz wykonujemy jeden krok wstecz w czasie. Punkt optymalny zmieni się tylko wtedy, gdy usuniemy jedną z półprzestrzeni definiujących v_i . Ponieważ $h_{d+1}, \ldots, h_i$ jest losową permutacją, prawdopodobieństwo, że tak się stanie, wynosi co najwyżej $d/(i - d)$.

W rezultacie otrzymujemy następujące ograniczenie dla oczekiwanego czasu działania algorytmu:

$$O(dn)+ dC_{d-1}\, n + \sum_{i=d+1}^{n} (O(di)+C_{d-1}\, (i - 1)) \frac{d}{i - d}$$

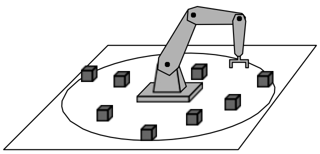
To może być ograniczone przez C_d n, z $C_d = O(C_{d-1}\, d)$, więc $C_d = O(c^d\, d!)$ $\square$

dla stałej *c* niezależnej od wymiaru.

Gdy *d* jest stałą, poprawne jest stwierdzenie, że algorytm działa w czasie liniowym. Byłoby to jednak dość mylące. Stały współczynnik C_d rośnie tak szybko jako funkcja *d*, że algorytm ten jest użyteczny tylko dla raczej małych wymiarów.

4.7* Najmniejsze dyski otaczające

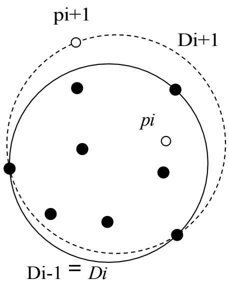
Prosta technika randomizacji, której użyliśmy powyżej, okazuje się być zaskakująco skuteczna. Można ją zastosować nie tylko do programowania liniowego, ale także do wielu innych problemów optymalizacyjnych. W tej sekcji przyjrzymy się jednemu z takich problemów.



Rozważmy ramię robota, którego podstawa jest przymocowana do podłogi roboczej. Ramię musi podnosić przedmioty w różnych punktach i umieszczać je w innych punktach. Jaka byłaby dobra pozycja dla podstawy ramienia? Byłoby to gdzieś "pośrodku" punktów, do których ramię musi być w stanie dotrzeć. Mówiąc dokładniej, dobrą pozycją jest środek najmniejszego dysku, który obejmuje wszystkie punkty. Punkt ten minimalizuje maksymalną odległość między podstawą ramienia a dowolnym punktem, do którego musi ono dotrzeć. Otrzymujemy następujący problem: biorąc pod uwagę zbiór P n punktów na płaszczyźnie (punkty na podłodze roboczej, do których ramię musi być w stanie dotrzeć), znajdź najmniejszy otaczający dysk dla P, to znaczy najmniejszy dysk, który zawiera wszystkie punkty P. Ten najmniejszy otaczający dysk jest unikalny - patrz Lemat 4.14(i) poniżej, który jest uogólnieniem tego stwierdzenia.

Podobnie jak w poprzednich sekcjach, podamy losowy algorytm przyrostowy dla tego problemu: Najpierw generujemy losową permutację $p_1, \dots, p_n$ punktów w
P. Niech $P_i := \{p_1, \dots, p_i\}$. Dodajemy punkty jeden po drugim, zachowując D_i
, najmniejszy otaczający dysk P_i.

W przypadku programowania liniowego istniał miły fakt, który pomagał nam zachować optymalny wierzchołek: jeśli bieżący optymalny wierzchołek jest zawarty w następnej półpłaszczyźnie, to się nie zmienia, aw przeciwnym razie nowy optymalny wierzchołek leży na granicy półpłaszczyzny. Czy podobne stwierdzenie jest prawdziwe dla najmniejszych otaczających dysków? Odpowiedź brzmi: tak:



Lemat 4.13 *Niech $2 < i < n$ oraz niech P_i i D_i będą zdefiniowane jak powyżej. Wtedy mamy*
(i) Jeśli $p_i \in D_{i-1}$, to $D_i = D_{i-1}$.
(ii) Jeśli $p_i \notin D_{i-1}$, to p_i leży na granicy D_i.

Udowodnimy ten lemat później, po tym jak zobaczymy, jak możemy go wykorzystać do zaprojektowania randomizowanego algorytmu przyrostowego, który jest dość podobny do algorytmu programowania liniowego.

Algorytm MINIDISC(P)
Wejście. Zbiór P n punktów na płaszczyźnie.
Wynik. Najmniejszy otaczający dysk dla P.
1. Oblicz losową permutację $p_1, \dots, p_n$ dla P.
2. Niech D_2 będzie najmniejszym otaczającym dyskiem dla $\{p_1, p_2\}$.
3. **dla** $i \leftarrow 3$ **do** n
4.   **do if** $p_i \in D_{i-1}$
5.   **wtedy** $D_i \leftarrow D_{i-1}$
6.   **else** $D_i \leftarrow$ MINIDISCWITHPOINT($\{p_1, \dots, p_{i-1}\}, p_i$)
7. **powrót** D_n

Krytyczny krok ma miejsce, gdy $p_i \notin D_{i-1}$. Potrzebujemy podprogramu, który znajdzie najmniejszy dysk otaczający P_i , wykorzystując wiedzę, że p_i musi leżeć na granicy tego dysku. Jak zaimplementować tę procedurę? Niech $q := p_i$. Używamy tej samej struktury jeszcze raz: dodajemy punkty P_{i-1} w losowej kolejności i utrzymujemy najmniejszy otaczający dysk $P_{i-1} \cup \{q\}$ z dodatkowym ograniczeniem, że powinien mieć q na swojej granicy. Dodanie punktu p_j będzie ułatwione dzięki następującemu faktowi: jeśli p_j jest zawarty w aktualnie najmniejszym otaczającym dysku, to dysk ten pozostaje taki sam, a w przeciwnym razie musi mieć p_j na swojej granicy. W tym drugim przypadku dysk ma zarówno q, jak i p_j oraz swoją granicę. Otrzymujemy następujący podprogram.

MINIDISCWITHPOINT(P, q)
Wejście. Zbiór P składający się z n punktów na płaszczyźnie oraz punkt q taki, że istnieje okrąg zamykający zbiór P z punktem q na jego brzegu.
Wynik. Najmniejszy otaczający dysk dla P z q na jego granicy.
1. Oblicz losową permutację $p_1, \dots, p_n$ dla P.
2. Niech D_1 będzie najmniejszym dyskiem z q i p_1 na jego brzegu.
3. **dla** $j \leftarrow 2$ **do** n
4. **do if** $p_j \in D_{j-1}$
5. **wtedy** $D_j \leftarrow D_{j-1}$
6. **else** $D_j \leftarrow$ MINIDISCWITH2POINTS($\{p_1, \dots, p_{j-1}\}, p_j, q$)
7. **powrót** D_n

Jak znaleźć najmniejszy otaczający dysk dla zbioru przy ograniczeniu, że dwa dane punkty q_1 i q_2 znajdują się na jego granicy? Po prostu stosujemy to samo podejście jeszcze raz. W ten sposób dodajemy punkty w losowej kolejności i zachowujemy optymalny dysk; gdy dodany punkt p_k znajduje się wewnątrz bieżącego dysku, nie musimy nic robić, a gdy punkt p_k nie znajduje się wewnątrz bieżącego dysku, musi znajdować się na granicy nowego dysku. W tym drugim przypadku mamy trzy punkty na granicy dysku: q_1 , q_2 i p_k . Oznacza to, że pozostaje tylko jeden dysk: unikalny dysk z q_1 , q_2 i p_k na jego granicy. Poniższa procedura opisuje to bardziej szczegółowo.

MINIDISCWITH2POINTS(P, q_1, q_2)
Dane wejściowe. Zbiór *P składający się* z n punktów na płaszczyźnie oraz dwóch punktów q_1 i q_2 takich, że istnieje okrąg ograniczający zbiór P z punktami q_1 i q_2 na jego brzegu.
Wynik. Najmniejszy otaczający dysk dla P z q_1 i q_2 na jego granicy.
1. Niech D_0 będzie najmniejszym dyskiem z q_1 i q_2 na jego brzegu.
2. **dla** $k \leftarrow 1$ **do** n
3. **do if** $p_k \in D_{k-1}$
4. **wtedy** $D_k \leftarrow D_{k-1}$
5. **else** $D_k \leftarrow$ dysk z q_1 , q_2 i p_k na jego brzegu
6. **powrót** D_n

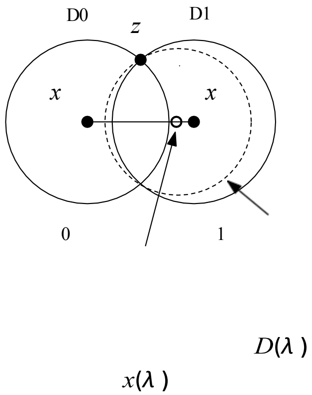
To ostatecznie kończy algorytm obliczania najmniejszego otaczającego dysku zbioru punktów. Zanim go przeanalizujemy, musimy zweryfikować jego poprawność poprzez udowodnienie pewnych faktów, które wykorzystaliśmy w algorytmach. Na przykład, użyliśmy fakt, że gdy dodaliśmy nowy punkt i punkt ten znajdował się poza bieżącym optymalnym dyskiem, to nowy optymalny dysk musi mieć ten punkt na swojej granicy.

Lemat 4.14 *Niech P będzie zbiorem punktów na płaszczyźnie, niech R będzie możliwie pustym zbiorem punktów $P \cap R = \emptyset$* i niech $p \in P$. Wtedy zachodzi następująca własność:

(i) *Jeśli istnieje dysk, który otacza P i ma wszystkie punkty R na swojej granicy, to najmniejszy taki dysk jest unikalny. Oznaczamy go przez $md(P, R)$.*

(ii) *Jeśli $p \in md(P \setminus \{p\}, R)$, to $md(P, R) = md(P \setminus \{p\}, R)$.*

(iii) *Jeśli $p \notin md(P \setminus \{p\}, R)$, to $md(P, R) = md(P \setminus \{p\}, R \cup \{p\})$.*



Dowód. (i) Załóżmy, że istnieją dwa różne dyski ograniczające D_0 i D_1 o środkach odpowiednio x_0 i x_1 oraz o tym samym promieniu. Oczywiste jest, że wszystkie punkty P muszą leżeć na przecięciu $D_0 \cap D_1$. Definiujemy ciągłą rodzinę dysków $\{D(\lambda) \mid 0 \leq \lambda \leq 1\}$ w następujący sposób. Niech z *będzie* punktem przecięcia ∂D_0 i ∂D_1, granic D_0 i D_1. Środkiem $D(\lambda)$ jest punkt $x(\lambda) := (1 - \lambda)x_0 + \lambda x_1$, a promieniem $D(\lambda)$ jest $r(\lambda) := d(x(\lambda), z)$. Mamy $D_0 \cap D_1 \subset D(\lambda)$ dla wszystkich λ z $0 \leq \lambda \leq 1$ oraz, w szczególności, dla $\lambda = 1/2$. Stąd, skoro zarówno D_0 jak i D_1 zawierają wszystkie punkty P, to $D(1/2)$ też musi. Co więcej, $\partial D(1/2)$ przechodzi przez punkty przecięcia ∂D_0 i ∂D_1. Ponieważ $R \subset \partial D_0 \cap \partial D_1$, wynika z tego, że $R \subset \partial D(1/2)$. Innymi słowy, $D(1/2)$ jest dyskiem otaczającym dla P z R na jego brzegu. Ale promień $D(1/2)$ jest ściśle mniejszy od promieni D_0 i D_1. Zatem gdy istnieją dwa różne dyski zamykające o tym samym promieniu z R *na* ich brzegu, to istnieje mniejszy dysk zamykający z R na jego brzegu. Zatem najmniejszy okrąg otaczający $md(P, R)$ jest wyjątkowy.

(ii) Niech $D := md(P \setminus \{p\}, R)$. Jeśli $p \in D$, to D zawiera P i ma R na swoim *brzegu.* Nie może istnieć mniejszy dysk zawierający P z R *na jego brzegu,* ponieważ taki dysk byłby również dyskiem zawierającym dla $P \setminus \{p\}$ z R na jego brzegu, co jest sprzeczne z definicją D. Wynika stąd, że $D = md(P, R)$.

(iii) Niech $D_0 := md(P \setminus \{p\}, R)$ i niech $D_1 := md(P, R)$. Rozważmy zdefiniowaną powyżej rodzinę dysków $D(\lambda)$. Zauważmy, że $D(0) = D_0$ i $D(1) = D_1$, więc rodzina ta definiuje ciągłą deformację D_0 do D_1. Z założenia mamy $p \notin D_0$. Mamy też $p \in D_1$, więc przez ciągłość musi istnieć pewne $0 < \lambda^* \leq 1$ takie, że p leży na granicy $D(\lambda^*)$. Podobnie jak w dowodzie (i), mamy $P \subset D(\lambda^*)$ i $R \subset \partial D(\lambda^*)$. Ponieważ promień dowolnego $D(\lambda)$ z $0 < \lambda < 1$ jest ściśle mniejszy niż promień D_1, a D_1 jest z definicji najmniejszym otaczającym dyskiem dla P, musimy mieć $\lambda^* = 1$. Innymi słowy, D_1 ma p na swojej granicy. $\square$

Lemat 4.14 implikuje, że MINIDISC poprawnie oblicza najmniejszy okrąg zamykający zbiór punktów. Analiza czasu działania jest podana w dowodzie poniższego twierdzenia.

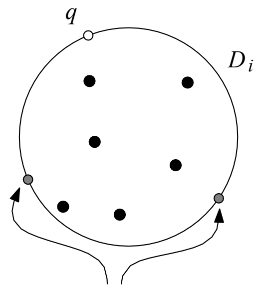
Twierdzenie 4.15 *Najmniejszy otaczający dysk dla zbioru n punktów na płaszczyźnie może być obliczony w $O(n)$ oczekiwanym czasie przy użyciu najgorszego przypadku przechowywania liniowego.*

Dowód. MINIDISCWITH2POINTS działa w czasie $O(n)$, ponieważ każda iteracja

pętli
zajmuje
stały
czas i
wykorz
ystuje
pamięć
liniową
.
MINIDI
SCWIT
HPOINT

i MINIDISC również wymagają liniowej pamięci masowej, więc pozostaje tylko przeanalizować ich oczekiwany czas działania.

Czas działania funkcji MINIDISCWITHPOINT wynosi $O(n)$, o ile nie liczymy czasu spędzonego na wywoływaniu funkcji MINIDISCWITH2POINTS. Jakie jest prawdopodobieństwo wykonania takiego wywołania? Ponownie używamy analizy wstecznej, aby ograniczyć to prawdopodobieństwo: Ustalmy podzbiór $\{p_1, \dots, p_i\}$, i niech D_i będzie najmniejszym dyskiem otaczającym $\{p_1, \dots, p_i\}$ i mającym q na swojej granicy. Wyobraźmy sobie, że usuwamy jeden z punktów $p_1, \dots, p_i$. Kiedy zmienia się najmniejsze otaczające koło? Dzieje się tak tylko wtedy, gdy usuniemy jeden z trzech punktów na granicy. Jednym z punktów na granicy jest q, więc istnieją co najwyżej dwa punkty, które powodują zmniejszenie najmniejszego otaczającego okręgu. Prawdopodobieństwo, że p_i jest jednym z tych punktów wynosi $2/i$. (Gdy na granicy znajdują się więcej niż trzy punkty, prawdopodobieństwo, że najmniejszy otaczający okrąg się zmieni, może być tylko mniejsze). Możemy więc ograniczyć całkowity oczekiwany czas działania MINIDISCWITHPOINT przez



wskazuje, że wraz z

q zdefiniować D_i

$$O(n) + \sum_{i=2}^{n} O(i)\frac{2}{i} = O(n).$$

Stosując ten sam argument ponownie, stwierdzamy, że oczekiwany czas działania MINIDISC również wynosi $O(n)$. □

Algorytm MINIDISC można ulepszyć na różne sposoby. Po pierwsze, nie jest konieczne używanie świeżej losowej permutacji w każdym wystąpieniu podprogramu MINIDISCWITHPOINT. Zamiast tego można obliczyć permutację raz, na początku MINIDISC i przekazać ją do MINIDISCWITHPOINT. Co więcej, zamiast pisać trzy różne procedury, można napisać pojedynczy algorytm MINIDISCWITHPOINTS(P, R), który oblicza $md(P, R)$ zgodnie z definicją w Lemacie 4.14.

4.8 Uwagi i komentarze

W tym rozdziale przeanalizowaliśmy problem algorytmiczny, który pojawia się, gdy chcemy wyprodukować obiekt za pomocą odlewania. Inne procesy produkcyjne również prowadzą do trudnych problemów algorytmicznych, a wiele takich problemów było badanych w geometrii obliczeniowej w ciągu ostatnich lat - patrz na przykład książka Dutta et al. [152] lub badania Janardan i Woo [220] oraz Bose i Toussaint [72].

Obliczanie wspólnego przecięcia półpłaszczyzn jest starym i dobrze zbadanym problemem. Jak wyjaśnimy w Rozdziale 11, problem ten jest dualny do obliczania wypukłego kadłuba punktów na płaszczyźnie. Oba problemy mają długą historię w tej dziedzinie, a Preparata i Shamos [323] podali już szereg rozwiązań. Więcej informacji na temat obliczania dwuwymiarowych wypukłych kadłubów można znaleźć w uwagach i komentarzach do Rozdziału 1.

Obliczanie wspólnego przecięcia półprzestrzeni, które można wykonać w czasie $O(n \log n)$ na płaszczyźnie i w przestrzeni trójwymiarowej, staje się bardziej skomplikowane.
problem wymagający obliczeniowo, gdy zwiększa się wymiar. Powód

jest to, że liczba (niżej wymiarowych) ścian wypukłego wieloboku utworzonego jako wspólne przecięcie może być tak duża jak $\Theta(n^{\lfloor d/2 \rfloor})$ [158]. Jeśli więc jedynym celem jest znalezienie wykonalnego punktu, jawne obliczanie wspólnego przecięcia szybko staje się nieatrakcyjnym podejściem.

Programowanie liniowe jest jednym z podstawowych problemów analizy numerycznej i optymalizacji kombinatorycznej. Przegląd tej literatury wykracza poza zakres tego rozdziału, dlatego ograniczamy się do wspomnienia o algorytmie simpleks i jego wariantach [139] oraz rozwiązaniach wielomianowych Khachiyana [234] i Karmarkara [227]. Więcej informacji na temat programowania liniowego można znaleźć w książkach Chva´tala [129] i Schrijvera [339].

Programowanie liniowe jako problem w geometrii obliczeniowej zostało po raz pierwszy rozważone przez Megiddo [273], który pokazał, że problem sprawdzenia, czy przecięcie półprzestrzeni jest puste, jest ściśle prostszy niż obliczenie przecięcia. Podał on pierwszy deterministyczny algorytm programowania liniowego, którego czas działania jest postaci $O(C_d\, n)$, gdzie C_d jest współczynnikiem zależnym tylko od wymiaru. Jego algorytm jest liniowy w n dla dowolnego ustalonego

wymiar. Współczynnik C_d w jego algorytmie wynosi 2^{2^d} . Został on później ulepszony do 3^{d^2} [130, 153]. Niedawno podano szereg prostszych i bardziej praktycznych algorytmów zrandomizowanych [132, 346, 354]. Istnieje wiele randomizowanych algorytmów, których czas działania jest subwykładniczy, ale nadal nie jest poli

w wymiarze [222, 267]. Znalezienie algorytmu silnie wielomianowego, czyli o złożoności kombinatorycznej wielomianowej, dla programowania liniowego jest jednym z głównych otwartych problemów w tej dziedzinie.

Podany tu prosty randomizowany algorytm przyrostowy dla dwóch i więcej wymiarów zawdzięczamy Seidelowi [346]. W przeciwieństwie do naszej prezentacji, zajmuje się on nieograniczonymi programami liniowymi, traktując parametr M symbolicznie. Jest to prawdopodobnie bardziej eleganckie i wydajne niż prezentowany przez nas algorytm, który został wybrany w celu zademonstrowania związku między nieograniczonymi *d-wymiarowymi* programami liniowymi a wykonalnymi (d 1)-wymiarowymi. W wersji Seidela można wykazać, że współczynnik C_d wynosi $O(d!)$.

Uogólnienie do obliczania najmniejszych otaczających dysków zawdzięczamy Welzl [385], który pokazał również, jak znaleźć najmniejszą kulę otaczającą zbiór punktów w wyższych wymiarach oraz najmniejszą elipsę lub elipsoidę otaczającą. Sharir i Welzl uogólnili tę technikę i wprowadzili pojęcie *problemów typu LP*, które mogą być efektywnie rozwiązywane za pomocą algorytmu podobnego do opisanego tutaj [189, 354]. Ogólnie rzecz biorąc, technika ta ma zastosowanie do problemów optymalizacyjnych, w których rozwiązanie albo nie zmienia się po dodaniu nowego ograniczenia, albo rozwiązanie jest częściowo zdefiniowane przez nowe ograniczenie, tak że wymiar problemu jest zmniejszony. Wykazano również, że specjalne własności problemów typu LP prowadzą do tak zwanych twierdzeń typu Helly'ego [16].

Randomizacja jest techniką, która często tworzy algorytmy, które są proste i wydajne. Zobaczymy więcej przykładów w kolejnych rozdziałach. Ceną, jaką płacimy, jest to, że czas działania jest tylko oczekiwaną granicą i - jak zauważyliśmy - istnieje pewna szansa, że algorytm zajmie znacznie więcej czasu. Niektórzy uważają to za powód, by twierdzić, że algorytmom randomizowanym nie można ufać.