Problem komiwojażera w praktyce

*Wydział Matematyki, Informatyki i Mechaniki, Uniwersytet Warszawski

*Łukasz KOWALIK**

Korzystając z popularnych serwisów internetowych, błyskawicznie znajdziemy najkrótszą trasę między dwoma miastami. A co, gdybyśmy chcieli znaleźć najkrótszą trasę, która pozwoli po wyruszeniu z domu odwiedzić wszystkie interesujące nas miasta i wrócić do punktu wyjścia? Możemy to pytanie sformalizować w następujący sposób. Wybrane miasta (wraz z tym, w którym mieszkamy) numerujemy od 1 do n. Zakładamy, że mamy tabelę $n \times n$ o wartościach d_{ij} dla $i, j = 1, \ldots, n$, gdzie d_{ij} oznacza odległość (np. podaną przez nasz ulubiony serwis) z miasta i do miasta j. Tabela jest symetryczna, tzn. $d_{ij} = d_{ji}$. Naszym celem jest znalezienie kolejności odwiedzania miast, czyli permutacji $v_1, \ldots, v_n$, która minimalizuje długość trasy, tzn. $\sum_{i=0}^{n} d_{v_i v_{i+1}}$, przyjmując $v_0 = v_n$. To zadanie znane jest jako *problem komiwojażera* i od kilkudziesięciu lat spędza sen z oczu informatykom.

Z jednej strony jest to problem NP-trudny, a najlepszy w sensie złożoności pesymistycznej algorytm działa w czasie $O(2^n n^2)$. Z drugiej strony, istnieją implementacje, które dla instancji pochodzących ze świata rzeczywistego uzyskują spektakularne wyniki (tabela poniżej), ale ich pesymistyczna złożoność czasowa nie jest znana.

rok	autorzy	liczba miast
1954	Dantzig, Fulkerson, Johnson	49
1977	Grötschel	120
1987	Padberg, Rinaldi	2 392
2004	Applegate, Bixby, Chvátal, Cook, Helsgaun	24 978
2016	Cook, Espinoza, Goycoolea, Helsgaun	49 603

Wszystkie powyższe rekordy dotyczą optymalnych rozwiązań dla rzeczywistych map połączeń drogowych (w USA, Niemczech, Szwecji), z wyjątkiem rekordu Manfreda Padberga i Giovanniego Rinaldiego, którzy znaleźli najkrótszą trasę dla wiertarki wiercącej otwory w płytce obwodu drukowanego. Aktualnie rozwiązanie problemu na zwykłym laptopie dla 1000 miast to kwestia kilkudziesięciu sekund, ale już wynik z ostatniego wiersza tabelki (2016) uzyskano za pomocą 310 procesorów pracujących przez 8 miesięcy. Choć pierwszy wiersz tabelki wygląda skromnie, w rzeczywistości stanowi wielkie osiągnięcie, a kolejne rekordy były wynikiem rozwinięcia technik wypracowanych przez George'a Dantziga, Delberta Fulkersona i Selmera Johnsona. W dalszej części artykułu zobaczymy, jak działa ich metoda na nieco bliższym nam przykładzie 51 dużych miast w Polsce.

Kluczowy pomysł polega na zapisaniu naszego problemu jako zbioru równań i nierówności liniowych. Dla każdej pary różnych miast $i, j = 1, \ldots, 51$ wprowadźmy zmienną x_{ij}, przy czym x_{ij} oraz x_{ji} traktujemy jako dwie nazwy tej samej zmiennej. Na początek przyjmijmy, że zmienne x_{ij} mogą przyjmować tylko dwie wartości: 0 i 1. Myślimy o nich w ten sposób, że połączenie między i oraz j jest używane w naszym rozwiązaniu wtedy i tylko wtedy, gdy $x_{ij} = 1$. Zauważmy, że długość takiej trasy wynosi $\sum_{1 \leqslant i < j \leqslant n} d_{ij} x_{ij}$.

Czy możemy za pomocą prostych równań i nierówności wymusić, aby spełniające je wartości zmiennych odpowiadały dokładnie możliwym trasom komiwojażera (czyli cyklom długości n)? Pierwsza obserwacja jest prosta: dla każdego miasta i dokładnie *dwie* zmienne x_{ij} powinny mieć wartość 1, co można wyrazić jako $\sum_{j \neq i} x_{ij} = 2$. To jednak nie wystarczy, gdyż zamiast jednej trasy moglibyśmy dostać kilka krótszych, rozłącznych cykli (w skrajnym przypadku $n/3$ rozłącznych trójkątów).

Aby zagwarantować, że miasta z pewnego zbioru S nie tworzą krótszej trasy, możemy dodać warunek *eliminacji podtras*, który mówi, że do tego zbioru musimy co najmniej jeden raz wejść i co najmniej jeden raz z niego wyjść. Zwykle ten warunek zapisujemy w równoważnej formie $\sum_{i,j \in S,\, i<j} x_{ij} \leqslant |S| - 1$ (czyli $\sum_{i \in S,\, j \notin S} x_{ij} \geqslant 2$). Równoważność łatwo dostaniemy, mnożąc oryginalną nierówność przez -1, dodając do niej równości $\sum_{j \neq i} x_{ij} = 2$ dla wszystkich $i \in S$ i dzieląc otrzymaną nierówność przez 2. Niestety, taki warunek musimy dodać dla każdego zbioru miast S o mocy między 3 a $\lfloor n/2 \rfloor$. Otrzymujemy następujące zadanie, które jest przykładem tzw. *programu liniowego całkowitoliczbowego*.

Zminimalizuj $\displaystyle\sum_{1 \leqslant i < j \leqslant n} d_{ij} x_{ij}$ przy ograniczeniach

- $\sum_{j \neq i} x_{ij} = 2$, dla $i = 1, \ldots, n$,
- $\sum_{i,j \in S,\, i<j} x_{ij} \leqslant |S| - 1$, dla $S \subseteq \{1, \ldots, n\}$ i $3 \leqslant |S| \leqslant \lfloor \frac{n}{2} \rfloor$,
- $x_{ij} \in \{0, 1\}$, dla $1 \leqslant i < j \leqslant n$.

Skąd pomysł na zapisanie problemu w języku równań i nierówności liniowych? Otóż teraz rozluźniamy założenia naszego zadania: warunki $x_{ij} \in \{0, 1\}$ zamieniamy na $0 \leqslant x_{ij} \leqslant 1$ i szukamy rozwiązania w liczbach rzeczywistych. Wówczas otrzymujemy tzw. *program liniowy*. Tak się składa, że nasi bohaterowie umieli szybko rozwiązywać programy liniowe, gdyż jeden z nich, George Dantzig, w 1947 roku opublikował do dziś najskuteczniejszy w praktyce algorytm realizujący ten cel, nazywany algorytmem *simplex*. Jest on obecnie dostępny w bibliotekach dla wielu popularnych języków programowania, np. przygotowując ten artykuł, użyliśmy modułu PuLP dla języka Python.

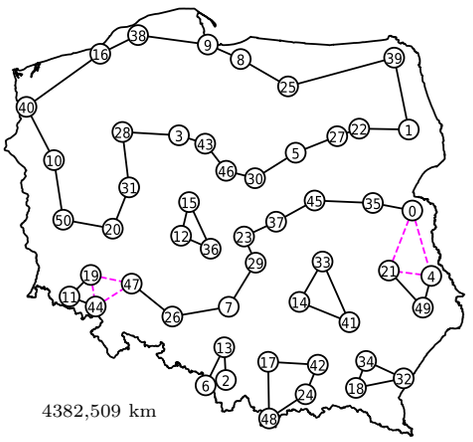
Plan jest następujący. Jeśli szczęśliwie jako rozwiązanie optymalne naszego programu liniowego dostaniemy same zera i jedynki, to będą one definiowały pewną trasę, która musi być optymalna, ponieważ *każda* trasa komiwojażera spełnia wszystkie warunki programu. Niestety może się okazać, że dostaniemy rozwiązanie *ułamkowe*, w którym niektóre zmienne będą miały wartość w przedziale $(0, 1)$. Wtedy będziemy musieli jakoś zareagować.

Dodatkowy kłopot polega na tym, że nasz program liniowy ma gigantyczną liczbę warunków eliminacji podtras: tylko dla $|S| = 25$ jest ich $\binom{51}{25} = 247\,959\,266\,474\,052$. To za dużo nawet dla współczesnych komputerów. Pomysł Dantziga i jego kolegów polegał na tym, aby zacząć od prostego programu liniowego postaci:

Zminimalizuj $\sum_{1 \leqslant i < j \leqslant n} d_{ij} x_{ij}$ przy ograniczeniach

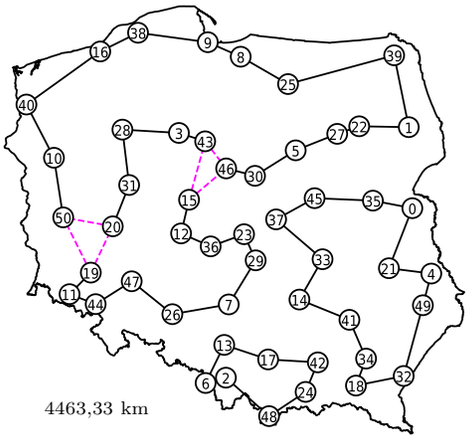
- $\sum_{j \neq i} x_{ij} = 2$, dla $i = 1, \ldots, n$,
- $0 \leqslant x_{ij} \leqslant 1$, dla $1 \leqslant i < j \leqslant n$;

a następnie dodawać warunki w miarę potrzeby. Taki program ma $\binom{51}{2} = 1275$ zmiennych i 2601 warunków liniowych. Uruchamiamy na komputerze algorytm simplex i w ułamku sekundy dostajemy poniższe rozwiązanie.



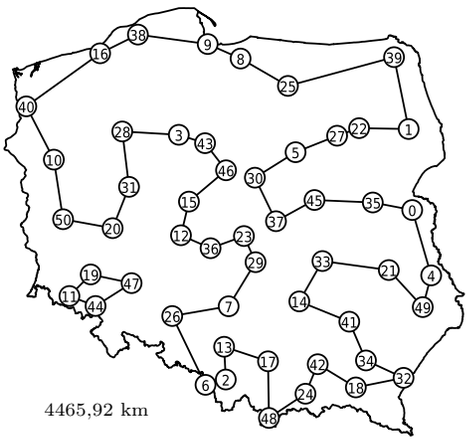
4382,509 km

Liniami ciągłymi oznaczyliśmy zmienne o wartości 1, natomiast przerywanymi o wartości $\frac{1}{2}$ (innych ułamkowych wartości w uzyskanym rozwiązaniu nie było). Widzimy, że możemy dodać warunki eliminacji podtras dla zbiorów $\{14, 33, 41\}, \{2, 6, 13\}, \{32, 34, 18\}, \{17, 24, 42, 48\}$ oraz $\{12, 15, 36\}$. Chociaż miasta 4, 21 i 49 nie utworzyły krótkiego cyklu, zauważmy, że dla $S = \{4, 21, 49\}$ oraz dla $S = \{11, 44, 19\}$ warunek eliminacji podtras też nie jest spełniony; dodajemy więc jeszcze te dwa warunki i ponownie uruchamiamy algorytm.



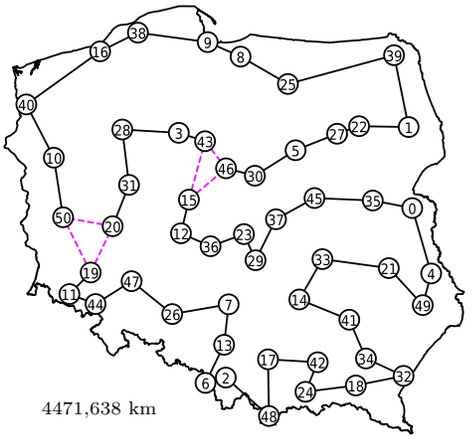
4463,33 km

W nowym rozwiązaniu mamy nowe krótkie cykle, dodajemy warunki eliminacji podtras dla kolejnych dwóch zbiorów $\{2, 6, 13, 17, 42, 24, 48\}$ oraz $\{4, 21, 0, 35, 45, 37, 33, 14, 41, 34, 18, 32, 49\}$ i generujemy kolejne rozwiązanie.



4465,92 km

Otrzymaliśmy dwa cykle, a więc dodajemy warunek eliminacji podtras dla krótszego z nich o zbiorze wierzchołków $\{11, 19, 44, 47\}$. Efekt kolejnego uruchomienia algorytmu simplex widzimy poniżej.

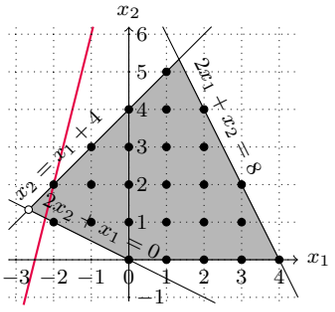


4471,638 km

Niestety *wszystkie* warunki eliminacji podtras są spełnione, a jednak nie uzyskaliśmy pojedynczego cyklu. Jest to spowodowane dopuszczeniem ułamkowych wartości zmiennych. Ratunkiem w takiej sytuacji jest znalezienie nowej nierówności, którą spełniają trasy komiwojażera, ale nie spełnia jej nasze rozwiązanie. Takie nierówności nazywamy *płaszczyznami odcinającymi*. Dlaczego? Wyobraźmy sobie, że nasz program liniowy ma tylko dwie zmienne, a wszystkie warunki to nierówności. Wówczas zbiór punktów spełniających te warunki jest wielokątem na płaszczyźnie, jak poniżej.

Zminimalizuj x_1
przy ograniczeniach

- $x_2 \leqslant x_1 + 4$,
- $2x_1 + x_2 \leqslant 8$,
- $2x_2 + x_1 \geqslant 0$,
- $x_2 \geqslant 0$.



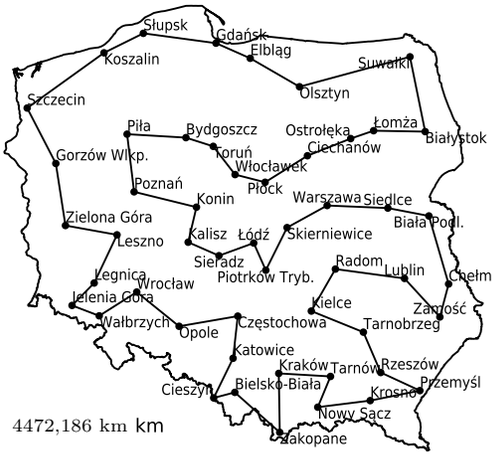
Algorytm simplex ma tę własność, że zwraca nam rozwiązanie optymalne, które jest wierzchołkiem wielokąta. Tymczasem my szukamy całkowitoliczbowego rozwiązania optymalnego. Możemy wówczas do programu liniowego dodać nierówność, która oddzieli nasz wierzchołek od wszystkich punktów o współrzędnych całkowitych wewnątrz wielokąta, odcinając fragment wielokąta

za pomocą linii prostej (na rysunku fioletowa linia jest przykładem takiej prostej). W wyniku tej procedury możemy znów dostać rozwiązanie niecałkowitoliczbowe, wówczas powtarzamy operację. Z analogiczną sytuacją mamy do czynienia w większej liczbie wymiarów, gdzie uogólnieniem linii jest hiperpłaszczyzna.

Wróćmy do ostatniego rozwiązania i spójrzmy na trójkąt $H = \{19, 20, 50\}$, którego krawędzie odpowiadają zmiennym połówkowym $x_{19\,20}$, $x_{20\,50}$ i $x_{50\,19}$, oraz na sąsiadujące z nim połączenia $T = \{19\,11, 20\,31, 50\,10\}$. W naszym rozwiązaniu suma $s = x_{19\,20} + x_{20\,50} + x_{50\,19} + x_{19\,11} + {} + x_{20\,31} + x_{50\,10}$ wynosi $4\frac{1}{2}$. Tymczasem w dowolnej trasie komiwojażera $s \leqslant 4$. Dlaczego? Z warunku $\sum_{j \neq i} x_{ij} = 2$ dla $i \in H$ wiemy, że

$$x_{19\,20} + x_{19\,50} + x_{19\,11} \leqslant 2,$$
$$x_{20\,19} + x_{20\,50} + x_{20\,31} \leqslant 2,$$
$$x_{50\,20} + x_{50\,19} + x_{50\,10} \leqslant 2.$$

Dodając stronami te nierówności i trywialną $\sum_{e \in T} x_e \leqslant |T|$, otrzymujemy $2s \leqslant 9$. Ale jeśli zmienne opisują trasę komiwojażera, to s jest liczbą całkowitą, a wtedy otrzymana nierówność jest równoważna $2s \leqslant 8$, czyli $s \leqslant 4$. Zachęcamy w tym miejscu Pracowitego Czytelnika, aby wykazał, że ogólniej, dla dowolnego zbioru miast H i *nieparzystej* liczby połączeń T wychodzących z H, $\sum_{i,j \in H,\, i<j} x_{ij} + \sum_{ij \in T} x_{ij} \leqslant |H| + \frac{|T|-1}{2}$. Są to tzw. *nierówności 2-skojarzeniowe*, gdyż zostały odkryte w kontekście problemu skojarzeń w grafach. Do naszego programu dodajmy nierówność 2-skojarzeniową $s \leqslant 4$, która pozwoli się pozbyć połówkowego trójkąta $\{19, 20, 50\}$. Okazuje się, że w rozwiązaniu optymalnym takiego programu liniowego wszystkie zmienne mają już wartość 0 lub 1, a odpowiadające im rozwiązanie tworzy pojedynczy cykl, a więc jest optymalną trasą komiwojażera. Wygląda ona następująco.



Dantzig z kolegami nie odkryli nierówności 2-skojarzeniowych, podobnie jak w naszym przykładzie nierówności eliminacji podtras nie były wystarczające, więc musieli użyć dwóch dziwacznych nierówności. W kolejnych latach odkryto metody automatycznego znajdowania płaszczyzn odcinających, jednak wciąż najbardziej skuteczne okazują się rodziny płaszczyzn (czyli po prostu nierówności) projektowane z myślą o konkretnym problemie, jak nasze nierówności 2-skojarzeniowe. Dla problemu komiwojażera odkryto wiele takich rodzin. Dla niektórych z nich jesteśmy w stanie szybko (w czasie wielomianowym) znajdować nierówność niespełnioną przez dane rozwiązanie, o ile taka istnieje (jest to możliwe dla nierówności eliminacji podtras i nierówności 2-skojarzeniowych). Dla innych stosujemy heurystyki, które wprawdzie działają szybko, ale mogą pozostawić jakąś niespełnioną nierówność.

Dociekliwy Czytelnik na pewno zadaje sobie teraz pytanie, co się dzieje, gdy program rozwiązujący problem komiwojażera nie miał tyle szczęścia co my z mapą Polski i w pewnym kroku generuje rozwiązanie, które nie jest całkowitoliczbowe, a mimo to wszystkie rodziny nierówności, które jest w stanie sprawdzić, są spełnione. Odpowiedź jest prosta: wówczas używa się brutalnej siły. Wybieramy dowolną zmienną x_{ij} o wartości w przedziale $(0, 1)$ i rekurencyjnie rozwiązujemy dwa podproblemy. W pierwszym dodajemy warunek $x_{ij} = 1$, w drugim $x_{ij} = 0$. Zauważmy, że każda optymalna trasa komiwojażera spełnia warunki dokładnie jednego z tych podproblemów. W każdym z podproblemów możemy znowu szukać płaszczyzn odcinających i rozgałęziać się na kolejne podproblemy. W ten sposób znajdziemy wiele tras komiwojażera, z których najkrótsza będzie optymalna.

Liczba podproblemów może szybko wymknąć się spod kontroli, bo k poziomów rozgałęzień to 2^k podproblemów. Z pomocą przychodzą wówczas heurystyki znajdujące niekoniecznie optymalne trasy komiwojażera (obecnie najskuteczniejsza, heurystyka Lina–Kernighana–Helsgauna, dla mapy świata zawierającej 1 904 711 miast znajduje trasę dłuższą od optymalnej o co najwyżej 0,023 %). Powiedzmy, że heurystyka znalazła nam trasę komiwojażera o długości u, i rozważmy podproblem, w którym wartość rozwiązania optymalnego programu liniowego wynosi l. Każda trasa, którą możemy znaleźć w tym podproblemie, będzie miała długość co najmniej l, a więc jeśli $l \geqslant u$, to taki podproblem możemy zignorować, gdyż nie pozwoli on znaleźć lepszej trasy niż ta, którą już mamy. Dzięki temu możliwe jest znaczne ograniczenie szybkiego wzrostu liczby podproblemów.

Okazuje się, że dla danych pochodzących ze świata rzeczywistego i dostatecznie bogatych rodzin płaszczyzn oddzielających liczba generowanych podproblemów nie jest astronomiczna. Jest to jednak jedynie obserwacja empiryczna. W 2018 roku Stefan Hougardy i Xianghui Zhong wymyślili sposób generowania sztucznych, wyjątkowo trudnych instancji problemu komiwojażera. Na podstawie testów oszacowali, że Concorde, najlepszy obecnie program korzystający z naszkicowanej w tym artykule metody, uruchomiony na współczesnym komputerze potrzebuje $3 \cdot 10^{22}$ lat obliczeń dla wygenerowanej przez nich instancji 1000 miast. To pokazuje, że przynajmniej na razie NP-trudności problemu nie dało się oszukać. Na szczęście duże instancje problemu komiwojażera rozwiązane optymalnie lub niemal optymalnie pokazują, że świat wokół nas rzadko jest aż tak złośliwy jak generator Hougardy'ego i Zhonga.

Optymalna trasa dla 51 polskich miast zaprezentowana w tym artykule została obliczona na podstawie tabeli 1275 długości najkrótszych połączeń drogowych między nimi, znalezionych za pomocą serwisu Google maps we wrześniu 2019 roku. Dane oraz prosty skrypt w języku Python, który znajduje optymalną trasę i tworzy rysunki do tego artykułu, są dostępne na `github.com/lkowalik/tsp`. Zachęcamy do eksperymentów!