Bounded Diameter Minimum Spanning Tree

Teoria Obliczeń i Złożoność Obliczeniowa

Adrian Mucha

23 stycznia 2021

1 Wprowadzenie

Mając ważony, nieskierowany graf G i dodatnią liczbę D w problemie *Bounded-Diameter Minimum Spanning Tree (BDMST)* szukamy najniższego kosztem drzewa rozpinającego spośród wszystkich drzew rozpinających G, których ścieżki składają się z co najwyżej D krawędzi. Formalnie *BDST* jest drzewem $T \subset E$ na $G = (V, E)$, którego średnica jest nie większa niż D. *BDMST* ma na celu znalezienia drzewa rozpinającego o minimalnym koszcie $w(T) = \sum_{e \in T} w(e)$. Zawężając do grafów Euklidesowych, czyli takich w których wierzchołki są punktami na przestrzeni Euklidesowej a wagi krawędzi reprezentują dystans między parami wierzchołków nazywamy *Euclidean BDMST*.

Problem BDMST jest NP-trudny dla $4 \leqslant D < |V| - 1$, oraz trudny w aproksymacji co motywuje w poszukiwaniach efektywnej strategii opartej na heurystykach, które potrafią szybko znaleźć BDST o niskim koszcie.

Oczywistym zastosowaniem *EBDMST* jest znalezienie najtańszej sieci kabli lub rur by połączyć zbiór miejsc zakładając, że koszt połączenia zależny jest od jego długości.

1.1 Definicja

W książce [1, p. 206] Garey and Johnson pokazali, że Bounded Diameter Spanning Tree (BDST) problem jest NP-trudny.

Mając ważony graf nieskierowany $G = (V, E)$ oraz dwa parametry D i C, decyzja *czy drzewo rozpinające o koszcie C oraz średnicy ograniczonej z góry przez D istnieje* jest NP-zupełny. Jako pierwsi opisali ten problem i dowiedli NP-zupełności autorzy pracy [2].

Formalnie, mając graf G, funkcję kosztu $W(e) \in \mathbb{Z}^+, \forall_{e \in E}$, oraz nieujemne liczby całkowite C i D, stwierdzić czy istnieje drzewo rozpinające T, takie że $\sum_{e \in T} W(e) \leqslant C$ oraz $\sum_{e \in p} W(e) \leqslant D$ gdzie p jest zbiorem wszystkich ścieżek w T. Ten problem nazywać będziemy Bounded Diameter Bounded Cost Spanning Tree (BDBCST).

1.2 Przykłady

Najprostszym grafem o zadanej maksymalnej średnicy D drzewa rozpinającego jest *graf liniowy* zawierający nie więcej niż $D+1$ ($|V| \leqslant D+1$, $|E| \leqslant D$) wierzchołków o funkcji kosztu $W(u,v) = \lfloor \frac{D}{C} \rfloor$.

Przykładem negatywnym będzie więc graf w którym istnieje ścieżka między dwoma liśćmi dłuższa niż D, lub graf którego drzewo rozpinające zawiera krawędź e_0 której koszt przekracza ograniczenie $W(e_0) > C$.

2 NP-zupełność

Twierdzenie 1. *BDBCST jest NP-zupełny*

Dowód. BDBCST jest NP. Najcięższą ścieżką musi być ścieżka łącząca dwa liście, więc możemy zgadnąć $n-1$ krawędzi i sprawdzić czy drzewo spełnia ograniczenia w czasie wielomianowym.

Weźmy klauzule $C = \{C_1, C_2, \ldots, C_q\}$ z problemu `3SAT` na zbiorze zmiennych $\{X_1, X_2, \ldots, X_n\}$. Skonstruujemy graf $G = (V, E)$ i funkcję W w taki sposób, że C jest spełnialne wtedy i tylko wtedy gdy istnieje drzewo rozpinające T dla grafu G, takie że $\sum_{e \in T} W(e) \leqslant 3n + 3q + 5$ oraz $\sum_{e \in p} W(e) \leqslant 10$ gdzie p jest zbiorem ścieżek w drzewie T.

G konstruujemy w następujący sposób. G zawiera następujące typy wierzchołków:

- ustanawiający prawdę wierzchołek t
- wierzchołki zmiennych $\{x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n\}$
- wierzchołki klauzul $\{c_1, c_2, \ldots, c_q\}$
- specjalny wierzchołek s

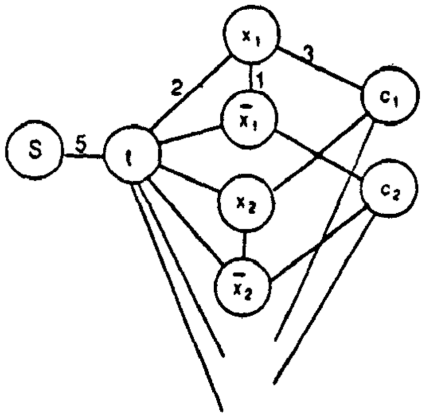
G zawiera następujące krawędzie:

- krawędzie przypisania (waga 2) $\{(t,x) | \forall x\}$
- krawędzie regulujące (waga 1) $\{(x_i, \bar{x}_i) | i \in [1,n]\}$
- krawędzie powstrzymujące (waga 3) $\{(u_i, c_j) | u_i = \begin{cases} x_i, & \text{jeśli } X_i \in C_j \\ \bar{x}_i, & \text{jeśli } \bar{X}_i \in C_j \end{cases}\}$
- krawędź (t,s) - waga 5

Ta konstrukcja jest możliwa w czasie wielomianowym.

Następnie załóżmy, że C jest spełnialne. Istnieje więc przypisanie, takie że każda klauzula w C zawiera conajmniej jeden prawdziwy literał. Możemy wybrać takie drzewo rozpinające T składające się z następujących krawędzi:

- (t,s)



Rysunek 1: Konstrukcja grafu G w dowodzie twierdzenia 1

- (t, x_i) dla każdego i gdzie X_i jest przypisany jako `TRUE` lub $(t, \bar{x}_i)$ w p.p.
- wszystkie krawędzie regulujące
- krawędzie powstrzymujące (x_{i_j}, c_i), gdzie X_{i_j} jest pierwszym literałem `TRUE` w klauzuli C_i.

Łatwo zauważyć, że $\sum_{e \in T} W(e) = 3n + 3q + 5$. Waga ścieżki od liścia do wierzchołka t wynosi co najwyżej 5, więc nie istnieje ścieżka w T z wagą większą niż 10.

Załóżmy że mamy drzewo rozpinające T, takie że $\sum_{e \in T} W(e) \leqslant 3n + 3q + 5$ oraz żadna ścieżka p w T nie jest dłuższa niż 10. W T jest $2n+q+2$ wierzchołków i $2n + q + 1$ krawędzi. T musi zawierać krawędź (t, s), ponieważ (t, s) jest jedyną krawędzią łączącą s. Wierzchołki klauzul są połączone jedynie za pomocą krawędzi powstrzymujących, a jest ich co najmniej q w T. Załóżmy, że drzewo rozpinające T składa się z i krawędzi przypisania, j krawędzi regulujących, $k+q$ krawędzi powstrzymujących oraz krawędzi (t, s). Wtedy mamy

$$i, j, k \geqslant 0 \tag{1}$$

$$j \leqslant n \tag{2}$$

$$i + j + k = 2n \tag{3}$$

$$2i + j + 3k \leqslant 3n \tag{4}$$

Załóżmy, że $j < n$. Odejmując od (4) - 2 × (3), mamy

$$k \leqslant j - n < 0$$

, czyli sprzeczność. Ponadto, T musi zawierać wszystkie n krawędzi regulujących. Podstawiając $j = n$ otrzymujemy

$$i + k = n \tag{5}$$

$$2i + 3k \leqslant 2n \tag{6}$$

Dalej odejmując od (6) - $2 \times (5)$ otrzymujemy $k \leqslant 0$. Stąd T zawiera dokładnie q krawędzi powstrzymujących, n krawędzi regulujących i n krawędzi przypisania.

Kolejno twierdzimy, że wszystkie wierzchołki klauzul sąsiadują tylko z tymi wierzchołkami zmiennych, które sąsiadują z wierzchołkiem t w drzewie rozpinającym T. Teraz załóżmy, że to twierdzenie jest fałszywe. Ponieważ wierzchołki klauzul sąsiadują tylko z wierzchołkami zmiennych, musi więc istnieć wierzchołek ze zmienną x oraz taki wierzchołek z klauzulą c, że x sąsiaduje z c ale nie sąsiaduje z t w T (patrz rysunek 1). Ścieżka z c do t, która musi przebiegać przez x, ma więc wagę 6. Ścieżka z s do c w T musi wtedy ważyć 11, co jest sprzeczne.

Ponieważ T zawiera wszystkie krawędzie regulujące, T zawiera dokładnie jedną z krawędzi (t, x_i) i $(t, \bar{x}_i)$ dla każdego i. Przypisując `TRUE` każdej zmiennej sąsiadującej z t oraz `FALSE` ich zmiennym uzupełniającym, wszystkie klauzule w C będą spełnione, ponieważ każda zawiera przynajmniej jeden prawdziwy literał. Stąd `3SAT` jest wielomianowo redukowalny do problemu $BDBCST$. □

3 Podstawowe heurystyki

Poniżej przedstawiono przegląd podstawowych heurystyk

3.1 One-Time Tree Construction (OTTC)

Jest to zachłanny algorytm oparty na heurystyce, która oblicza średnicę drzewa rozpinającego w każdym kroku i upewnia się, że następny wierzchołek nie przekroczy ograniczenia. Następnie do budowanego drzewa rozpinającego w każdym kroku dodaje krawędź o najniższym koszcie, który nie łamie obostrzeń na średnicę. Dodanie wierzchołka wymaga pracy $\mathcal{O}(n^2)$, a całość wykonywana jest $n-1$ razy, więc całkowity czas pracy algorytmu rozpoczynającego w pojedynczym wierzchołku to $\mathcal{O}(n^3)$. Aby znaleźć najniższe kosztem BDST, algorytm OTTC uruchamiany jest dla każdego wierzchołka grafu (n razy), więc całkowita złożoność wynosi $\mathcal{O}(n^4)$ [3].

3.2 Center-Based Tree Construction (CBTC)

W drzewie o średnicy D, żaden wierzchołek nie znajduje się dalej niż $\frac{D}{2}$ skoków (lub krawędzi) od korzenia. Dzięki temu zabiegowi otrzymujemy szybszy algorytm bazujący na algorytmie Prima, który poprawia OTTC dzieki budowaniu BDST od środka drzewa. Zapamiętywanie stopnia wierzchołków i zapewnianie, że żaden nie przekroczy głębokości $\lfloor \frac{D}{2} \rfloor$ pozwala zaoszczędzić ciągłego przeliczania średnicy drzewa przed dołączeniem wierzchołka do BDST. Otrzymujemy nieco lepszą złożoność $\mathcal{O}(n^3)$ w porównaniu do OTTC (tutaj również należy rozważyć algorytm startując z każdego wierzchołka osobno i wybrać drzewo o najniższym koszcie) [4].

3.3 Randomized Tree Construction (RTC)

W losowej konstrukcji drzewa, korzeń (centrum) jest wybierany na początku jako losowy wierzchołek (jeśli D jest parzyste) lub losowane są dwa wierzchołki połączone ze sobą (jeśli D jest nieparzyste). Każdy następny dołączany wierzchołek jest również wybierany losowo w sposób zachłanny, taki że przyłączenie wierzchołka nie przekroczy ograniczenia D na średnicę drzewa. Jest to identyczny w implementacji algorytm jak CBTC z tą różnicą że wprowadzono element losowości przy wybieraniu wierzchołków. Również posiada tę samą złożoność $\mathcal{O}(n^3)$.

3.4 Pozostałe heurystyki

Po skonstruowaniu BDST jednym z algorytmów (CBTC lub RTC) dodatkowo sprawdzane jest dla każdego wierzchołka $v \in V$ którego głębokość jest większa niż 1 czy można go odłączyć i połączyć z innym wierzchołkiem BDST mającym niższy stopień głębokości krawędzią o mniejszym koszcie.

Heurystyka przetrzymuje posortowaną macierz kosztów w celu szukania krawędzi o niskim koszcie by dodać do BDST wierzchołek v. Aby zachować kolejność używa się macierzy pomocniczej pamiętającej indeksy [5].

4 Heurystyki CBLSoC-lite oraz CBLSoC

Center-Based Least Sum of Costs *Lite* wybiera wierzchołek na korzeń (lub dwa wierzchołki w przypadku parzystej średnicy) z najmniejszym kosztem do reszty wierzchołków $v_0 = \arg\min_v \left\{ v \in V | \sum_{u \in N_G(v)} w((v,u)) \right\}$. Podążając tym podejściem, kolejne wierzchołki również są wybierane na podstawie najniższej sumy kosztów do reszty wierzchołków z uwzględnieniem heurystyki CBTC, a zatem upewnianiem się, że żaden wierzchołek nie przekroczy głębokości $\lfloor \frac{D}{2} \rfloor$. Otrzymany algorytm (wersja *Lite*) wykonuje pracę $\mathcal{O}(n^2)$.

CBLSoC jest wariacją pierwszego, gdyż różni się tym, że wybór korzenia nie odbywa się na podstawie najniższego kosztu lecz powtarza się algorytm *Lite* n razy dla każdego wierzchołka grafu. Całkowity czas działania algorytmu wynosi więc $\mathcal{O}(n^3)$.

Podobnie jak w CBTC i RTC otrzymujemy jednakowe złożoności natomiast wynikowe drzewa rozpinające posiadają niższe koszty.

5 Quadrant Centers-based (QCH)

Zachłanność heurystyk OTTC oraz CBTC sprawia że "kręgosłup" (pień) drzewa zazwyczaj składa się z krótkich krawędzi co wymusza na niektórych aby kolejno dołączyły następny wierzchołek za pomocą krawędzi długiej co zwiększa koszt całkowity. LSoC łagodzi ten problem do pewnego stopnia szukając lepszych połączeń po zbudowaniu drzewa.

Inne podejście proponuje rozpocząć od empirycznego wybrania korzenia drzewa oraz dodania kilku wierzchołków do drzewa, które utworzą kręgosłup składający się z małej liczby wierzchołków połączonych relatywnie długimi krawędziami. Pozostałe wierzchołki dołącza się do BDST zachłannie lub za pomocą wcześniej wspomnianej heurystyki CBLSoC.

Tym podejściem kieruje się heurystyka QCH, której głównym zamysłem jest podział wierzchołków na kwadranty. W wariancie Euklidesowym, gdzie modelujemy wierzchołki jako punkty porozrzucane na kwadracie jednostkowym, możemy zastosować heurystykę CBLSoC do wyznaczenia korzenia. Podobnie, jeżeli średnica jest parzysta, to do centrum dodaje się drugi taki wierzchołek, który ma najniższy średni koszt to reszty wierzchołków i dołączony krawędzią o najniższym koszcie. Pozostałe wierzchołki grafu są grupowane w *kwadranty* opisane jako macierz $M \times M$ (elementy macierzy są kwadrantami), takiej że $2 \leqslant M \leqslant \sqrt{N}$. W każdym kwadrancie wierzchołek o najniższym średnim koszcie do reszty wierzchołków spośród tego kwadranta jest dołączany do kręgosłupa drzewa. Następnie reszta wierzchołków jest dołączana zachłannie lub za pomocą heurystyki CBLSoC zachowując warunek na średnicę.

Obie heurystyki próbują wybrać najlepszy kręgosłup zmieniając liczbę kwadrantów ($[2, n]$, gdzie n to ilość wierzchołków). Heurystyka zwraca najniższe kosztem BDST otrzymane w tej procedurze. Tworzenie kręgosłupa wymaga $\mathcal{O}(n^2)$. Przebieg heurystyki po wszystkich ilościach kwadrantów trwa $\mathcal{O}(\sqrt{n})$ co daje całkowity czas $\mathcal{O}(n^2\sqrt{n})$ [6].

Literatura

[1] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., USA, 1990.

[2] Jan-Ming Ho, D. T. Lee, and Chia-Hsiang Chang. Bounded-diameter minimum spanning trees and related problems. In Kurt Mehlhorn, editor, *Proceedings of the Fifth Annual Symposium on Computational Geometry, Saarbrücken, Germany, June 5-7, 1989*, pages 276–282. ACM, 1989.

[3] Narsingh Deo and Ayman M. Abdalla. Computing a diameter-constrained minimum spanning tree in parallel. In Gian Carlo Bongiovanni, Giorgio Gambosi, and Rossella Petreschi, editors, *Algorithms and Complexity, 4th Italian Conference, CIAC 2000, Rome, Italy, March 2000, Proceedings*, volume 1767 of *Lecture Notes in Computer Science*, pages 17–31. Springer, 2000.

[4] Bryant A. Julstrom. Greedy heuristics for the bounded diameter minimum spanning tree problem. *ACM J. Exp. Algorithmics*, 14, 2009.

[5] Alok Singh and Ashok Kumar Gupta. Improved heuristics for the bounded-diameter minimum spanning tree problem. *Soft Comput.*, 11(10):911–921, 2007.

[6] C. Patvardhan, V. Prem Prakash, and Anand Srivastav. Fast heuristics for large instances of the euclidean bounded diameter minimum spanning tree problem. *Informatica (Slovenia)*, 39(3), 2015.