Algoritmi de aproximare

Condrea Tudor-Daniel 324CA

Universitatea Politehnica Bucuresti, Automatica si calculatoare

1 Problema comis-voiajorului

1.1 Descrierea problemei rezolvate

Problema comis-voiajorului ne pune următoarea întrebare: "Fiind dată o listă de orașe și distanța dintre fiecare pereche de orașe, care este cel mai scurt drum care trece prin fiecare oraș și se întoarce și la cel inițial". Dorim să găsim un algoritm care întoarce secvența de orașe cu cel mai scurt drum de parcurs între ele, asigurându-ne că primul și ultimul oraș coincid.

1.2 Aplicații reale ale problemei:

- Logistică. Transportul, în special cel industrializat, necesită o eficiență sporită.
- Planificare. Managementul timpului între taskuri poate fi optimizat cu ajutorul acestei probleme.

1.3 Specificarea soluțiilor alese

Algortimi exacți. Prima soluție la care ne gândim apelează la permutarea tuturor drumurilor pentru a genera toate posibilitățile din care extragem pe cea cu drumul optim. Această abordare are o complexitate de $\mathcal{O}(n!)$ și devine imposibil de utilizat pentru chiar și 20 de orașe.
Putem utiliza programare dinamică, mai exact algoritmul lui Held-Karp, acesta rulând în complexitate de timp $\mathcal{O}(2^n n^2)$ care este mult mai bună decât cea factorială, dar folosește mai mult spațiu de memorie $\mathcal{O}(n2^n)$ pentru rezultatele funcțiilor, comparativ cu backtracking care utilizează doar $\mathcal{O}(n^2)$ pentru graf.

Algoritmi aproximativi. Putem utiliza metoda greedy căutând cel mai apropiat vecin pentru fiecare oraș. Această metodă dă în medie un drum cu 25% mai lung decât cel optim, dar este foarte rapid.
Putem apela la teoria grafurilor, mai exact la algoritmul dezvoltat de Christofides-Serdyukov care oferă un drum care poate fi cu maxim 1.5 ori mai mare ca cel optim. Acesta folosește proprietățile grafurile Euleriene și arbori minim de acoperire.

1.4 Criterii de evaluare

Pentru a modela o rețea cu N orașe vom folosi un graf complet G cu N noduri, iar pentru reprezentarea acestuia, o matrice de adiacență $A \in M_n(\mathbb{N})$ unde $a_{ij} = d$, d fiind distanța dintre orașele cu indicii i și j.
Folosim numere naturale pentru ușurința calculelor și pentru a nu permite drumuri de lungimi negative. Niște seturi de valori for fi generate aleator și testate cu fiecare soluție. Rezultatul va afișa distanța totală și secvența de orașe gasită. Pentru testele unde $N > 20$ nu se va mai testa backtracking din cauza lipsei de eficiență.

2 Prezentarea soluțiilor

2.1 Brute-force

Algoritmul funcționează prin generarea tuturor permutărilor orașelor diferite de cel de început și adunarea distanțelor dintre ele. Pentru generarea permutărilor este folosit backtracking, adăugând într-o stivă, pe rând, orașele nevizitate. Orice oraș care nu se află în stivă este nevizitat. Atunci când este adăugat un oraș, se reține ce distanță adaugă la drumul final. Când avem $N - 1$ elemente în stivă, înseamnă că am obținut o posibilă soluție. Verificăm dacă soluția are distanța finală mai mică decât ultimele soluții propuse, dacă da, aceasta devine noua soluție optimă.
Clasa de complexitate în care se încadrează această metodă este $\mathcal{O}(n!)$ pentru timp și $\Theta(n^2)$ pentru memorie deoarece calculează toate permutările pentru cele $N - 1$ orașe și reține doar matricea de adiacență alături de câțiva vectori ajutători. Verificarea de apartenență la stivă se efectuează în $\mathcal{O}(n)$ care, deși se repetă de destule ori în interiorul algoritmului, devine neglijabilă prin comparație cu generarea permutărilor.
Avantaje:

- Implementare simplă.
- Complexitate de memorie adițională mică.

Dezavantaje:

- Pentru N suficient de mare, apelurile recursive pot supraîncărca memoria și chiar dacă nu, timpul real de execuție trece peste speranța de viață a unui om obișnuit.

2.2 Held-Karp

Algoritmul funcționează prin calcularea unei matrici de distanțe parțiale $P \in M_{n,2^n}(\mathbb{N})$ cu elementele p_{ij}. Indicele j al coloanelor este rescris în baza 2. Pentru fiecare bit setat de pe poziția $k, k = \overline{0, N-1}$, orașul cu indicele k aparține submulțimii de orașe din p_{ij}. Indicele i al liniilor arată că drumul parțial ce conține orașele j se termină în orașul i.

Pentru a completa matricea, adăugăm mai întâi drumurile cu un singur oraș ($p_{k,2^k}$), care sunt egale cu distanța dintre ele și orașul inițial ($d_{start,k}$). Continuăm să completăm cu drumurile formate din $c = \overline{2,N}$ orașe utilizând un generator de combinări de N luate câte c. Pentru fiecare drum care se termină cu orașul i eliminăm bitul i și căutăm un drum parțial minim $p'_{i'j'}$ anterior calculat. Îi atașăm orașul i și adunăm distanța de la i' la i și obținem distanța drumului parțial dorit.
Soluția este găsită prin adăugarea orașului de plecare la elementele $p_{i,N}$, calcularea distanțelor și alegerea celui mai bun rezultat.
Pentru a calcula complexitatea, numărăm buclele și numărul de iterații ale acestora. Algoritmul populează o matrice de mărime $N \times 2^N$ pe care o iterează de N ori, ducând la o complexitate de timp $\Theta(n^2 2^n)$ și complexitate de memorie $\Theta(n2^n)$.
Avantaje:

- Complexitate mai bună ca a brute-force-ului, merge pentru $N < 30$
- Folosește foarte multe operații pe biți, reducând costul operațiilor datorită eficienței

Dezavantaje:

- Nu merge pe seturi mari de date.
- Complexitate exponențială de memorie înseamnă că necesită resurse hardware foarte mari.

2.3 Christofides-Serdyukov

Euristica aproximativă aleasă pentru implementare. Algoritmul întoarce rezultate cu maxim 50% mai lungi pentru orice graf care respectă un spațiu metric, mai exact, oricare 3 noduri ale grafului, u, v, x, se poate forma un triunghi cu acestea ($d_{uv} + d_{vx} \geqslant d_{ux}$). Pentru grafuri pur abstracte care nu reprezintă un set de orașe posibil algoritmul poate avea eroare de peste 200%.
Euristica aceasta propune următorul algoritm de rezolvare:

1. Găsește un arbore de cost minim T.
2. Face o listă a tuturor nodurilor cu grad impar într-o mulțime O.
3. Găsește în subgraful indus de nodurile din O cu $\frac{Card(O)}{2}$ muchii astfel încât fiecare muchie atinge 2 noduri unice și suma costurilor e minimă. Aceste muchii formează un graf M.
4. Combină T și M într-un multigraf.
5. Face un tur eulerian din nodul de start.
6. Transformă circuitul eulerian într-unul hamiltonian prin eliminarea nodurilor duplicate după prima apariție.

Pentru pașii următori sunt folosiți acești algoritmi:

- Arborele de cost minim ⇒ algoritmul lui Prim.
- Minimum matching ⇒ metodă greedy.
- Turul eulerian ⇒ algoritmul lui Hierholzer.

Complexitatea depinde de algoritmii utilizați în pașii intermediari. Arborele de cost minim utilizează o soluție care rulează în $\Theta(n^2)$, minimum matching este făcut în $\Theta(n)$, iar turul eulerian și transformarea în tur hamiltonian rulează în $\Theta(n)$. Toate rezultatele necesită $\Theta(n)$ spațiu, memorând rezultatele în vectori unidimensionali, dar matricea de adicență și de multiplicitatea a muchiilor necesită $\Theta(n^2)$, ajungând la complexitatea finală de $\Theta(n^2)$.

Avantaje:

- Complexitate polinomială de grad 2 permite unor seturi de date de ordinul miilor de noduri.
- Fiind compus din mai mulți algoritmi fundamentali individuali, permite introducerea unor soluții optimizate pentru rezultate generale mai apropiate de soluția exactă sau număr de calcule redus.

Dezavantaje:

- Întoarce drumuri mai lungi decât cele optime.

3 Evaluare

Pentru a construi seturile de date am folosit 3 metode.

1. Generarea unor grafuri aleatorii fără a ține cont de regula triunghiului.
2. Transformarea unei liste de coordonate ale unor orașe într-o matrice de adiacență.
3. Preluarea unor date de test de pe acest site.

Testele au fost rulate pe o mașină virtuală cu următoarele specificații tehnice:

- Software de virtualizare: VMware Virtual Platform
- Distro: Ubuntu 20.04.3 LTS
- Kernel: 5.11.0-40-generic x86_64
- CPU: 2 nuclee, AMD Ryzen 5 3600, frecvență 3.6 GHz, cache 1 MB
- Memorie: 4 GB RAM

Timpii de execuție sunt următorii:

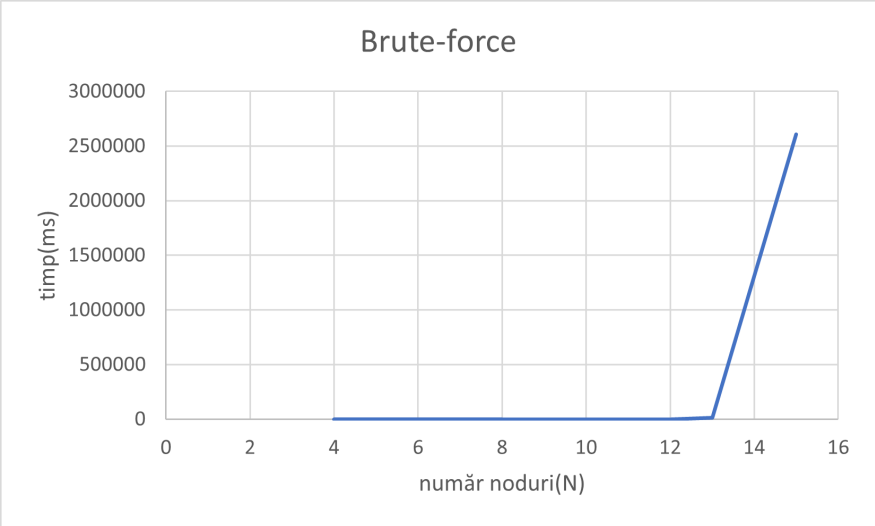

Fig. 1. Brute-force la scară originală

Timpul de execuție crește de la 12.283 secunde la 43 minute și 27.87 secunde dacă adăugăm doar 2 noduri.

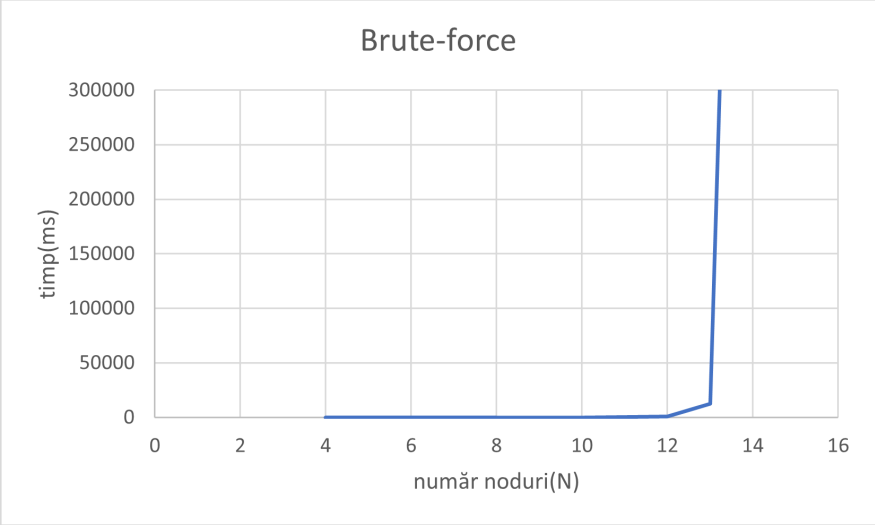

Fig. 2. Brute-force la scară comparabilă

Se observă cât de bruscă este într-adevăr creșterea.

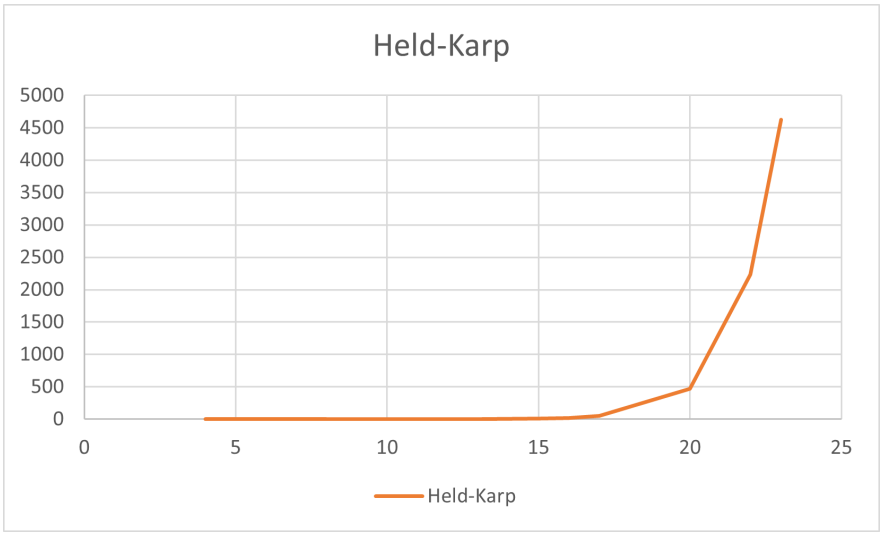

Fig. 3. Held-Karp

Se observă o creștere exponențială mult mai atenuată. Testele cu mai mult de 25 de noduri necesitau mai multă memorie decât era posibil de alocat.

Fig. 4. Christofides-Serdyukov

Se observă o creștere mult mai lentă, deoarece euristica aproximativă rulează în timp polinomial, nu exponențial. Datorită creșterii pătratice, în loc de exponențiale a memoriei, putem analiza cazuri mari de mii de orașe (nu mai mult de 4000).

Mai jos este un tabel care conține rezultatele testelor. Este trasă o linie orizontală de fiecare dată când un algoritm nu mai poate rula.

Nr. test	Expected dist.	Brute-force	Held-Karp	Christofides	Spațiu metric?	Raport aproximare + Obs.
1	95	95	95	95	da	100%
2	19	19	19	23	da	121.05%
3	29	29	29	33	da	113.79%
4	1064	1064	1064	1143	da	107.42%
5	61	61	61	61	da	120.83%
6	144	144	144	174	da	100%
7	1733	1733	1733	2284	da	131.79%
8	13	13	13	13	da	100%, teoretic worst-case
9	291	-	291	329	da	113.05%
10	79	-	79	192	nu	243.03%
11	2085	-	2085	2490	da	119.42%
12	444	-	444	490	da	110.36%
13	67	-	67	111	nu	165.67%
14	106	-	106	237	nu	223.58%
15	937	-	-	1058	da	112.91%
16	699	-	-	882	da	126.18%
17	33523	-	-	41833	da	124.78%
18	N/A	-	-	15388	da	57 de orașe
19	N/A	-	-	46806	da	128 de orașe americane
20	41626	-	-	46512	da	111.73%
21	N/A	-	-	2673	da	535 de aeropoarte din lume
22	86891	-	-	102081	da	117.48%

Pentru testul numărul 8, testăm următorul caz, care în teorie atinge pragul de 50% eroare.

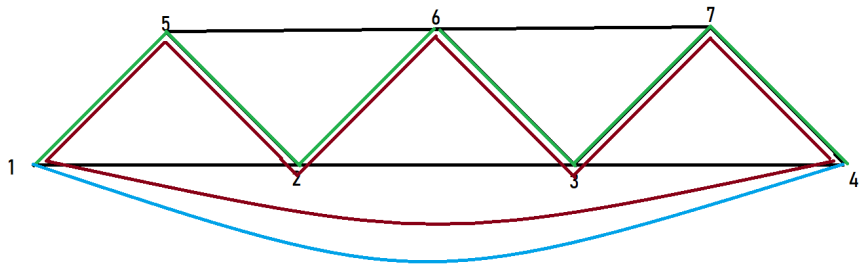

Fig. 5. Cel mai ineficient caz

Cu verde este colorat arborele de cost minim care ar trebui să fie selectat, cu albastru scurtăturile găsite, iar cu roșu drumul total. Fiecare muchie are costul 1. Astfel, drumul total rezultat ar trebui să coste 9 (18 pentru graful nostru cu 13 noduri), față de cel optim care doar înconjoara forma de trapez și costă 7 (13 pentru inputul nr. 8).

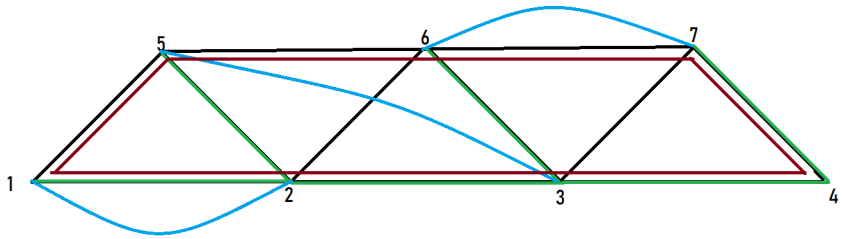

Fig. 6. Ce se întamplă defapt

Deoarece este folosit algoritmul lui Prim pentru determinarea arborelui minim, acesta nu ia forma zig-zag de mai sus, ci una zimțată care permite mai multor scurtături să fie create. Când calculează drumul eulerian și reducerea la drum hamiltonian, rămâne doar cu drumul optim de cost 7.

4 Concluzii

Considerând datele analizate, dacă ar trebui să implementez o soluție de rutare cu punct de plecare și întoarcere identic, aș opta pentru soluțiile optimizate pentru memorie a metodei dinamice, deoarece puterea de calcul este mult mai accesibilă, iar astfel economisesc resurse de transport, menținând un timp de calcul acceptabil.

5 Referințe

http://www.wseas.us/e-library/transactions/economics/2011/54-095.pdf
https://en.wikipedia.org/wiki/Held%E2%80%93Karp_algorithm
https://en.wikipedia.org/wiki/Travelling_salesman_problem
https://slaystudy.com/hierholzers-algorithm/
https://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html
http://www.math.uwaterloo.ca/tsp/data/