Facultatea de Matematică şi Informatică, Universitatea Babes-Bolyai,
Cluj-Napoca

Studiul grafurilor dinamice

Rezumatul tezei de doctorat

Doctorand: Pătcaş Csaba
Conducător ştiinţific: Prof. Dr. Kása Zoltán

2011

Rezumat

Grafurile dinamice sunt grafuri care se pot schimba în timp printr-o serie de actualizări locale. Într-o problemă de grafuri dinamice, o proprietate a grafului trebuie menţinută de-a lungul acestor actualizări şi răspunsul la unele interogări specifice trebuie găsit cât mai eficient posibil, fără a rezolva problema de la început folosind un algoritm static clasic. Problemele pe grafuri dinamice au o gamă largă de aplicaţii practice şi pot fi folosite şi pentru a accelera algoritmi statici existenţi.

Teza reprezintă un studiu cuprinzător al structurilor de date şi al algoritmilor folosiţi în rezolvarea problemelor pe grafuri dinamice. În teză se pune accent deosebit pe aspectul practic al acestor soluţii, prin prezentarea studiilor experimentale efectuate în cazul fiecăreia dintre problemele importante de grafuri dinamice. În cazul fiecărei probleme, starea actuală a ştiinţei este descrisă. În funcţie de presupusa structură a grafului şi a secvenţei de actualizări, diferiţi algoritmi sunt recomandabili în implementarea unor aplicaţii practice. Aceste recomandări se găsesc în capitole separate în teză, alături de tabele şi figuri explicative.

Contribuţia cea mai originală a autorului o reprezintă studiul problemei datoriilor, o problemă cu aplicaţii în primul rând în economie, care se poate modela într-un mod natural folosind teoria grafurilor. În teză se demonstrează că problema este NP-hard, alături de alte rezultate în privinţa relaţiei cu diferite clase de complexitate. Se prezintă un algoritm exact, care este capabil să obţină rezultatul optim pentru intrări de dimensiuni rezonabile. Problema enunţată pe grafuri dinamice poate avea un set aparte de aplicaţii. Autorul dezvoltă o structură de date pentru varianta dinamică a problemei, care la rândul ei se poate folosi într-un nou algoritm static. Soluţiile obţinute sunt comparate într-un studiu experimental amplu. În încheierea tezei, se prezintă un algoritm genetic capabil să rezolve problema pentru exemple de dimensiuni mari.

Cuvinte cheie: graf dinamic, datorii

Mulţumiri

Doresc să mulţumesc conducătorului meu ştiinţific Kása Zoltán, pentru ajutorul ştiinţific acordat de-a lungul anilor. Trebuie să-i mulţumesc de asemenea pentru răbdarea avută, fiindcă sunt conştient că acesta a fost necesar în colaborarea noastră. El a fost întotdeauna amabil şi zâmbitor cu mine, chiar şi când nu mă aşteptam. Se pare că ştia, că a fi strict nu este metoda cea mai bună pentru a mă motiva, cel puţin nu pe termen lung.

Sunt recunoscător fostei mele profesoare şi actualei mele colege, Robu Judit, pentru motivaţia oferită şi pentru aranjarea mobilităţii mele la Linz, asigurând cele mai bune condiţii pentru a-mi finaliza teza. Fără ajutorul ei, nu aş fi reuşit să termin în timp.

Doresc să mulţumesc familiei mele pentru suportul financiar, emoţional şi de alt fel acordat, când treceam prin perioade dificile - ceea ce s-a întâmplat de câteva ori în ultimii cinci ani. Mama mea m-a ajutat deseori când îmi făceam temele de casă în clasele elementare; câteodată, când nu eram satisfăcut cu rezultatul obţinut nici şi după ore în şir de lucru, obişnuia să-mi zică: "Lasă fiule, că îi bine deja, nu acuma îţi scrii teza de doctorat!".[1]. Iată, că acuma îl scriu.

Nu în ultimul rând sunt recunoscător prietenilor, colegilor şi tuturor persoanelor din jurul meu. Unii au făcut parte din viaţa mea la un moment dat şi au avut un impact mai semnificativ ca alţii. Prietenii mei cei mai apropiaţi mi-au organizat o petrecere de adio excelentă înainte să plec în Linz şi m-au ajutat în probleme de tot felul înaintea şi în timpul şederii mele. Ionescu Klára, fostă profesoară şi actual prieten, mi-a împărtăşit o mare parte din experienţa ei, cu ocazia pregătirii problemelor pentru concursurile de programare şi a conversaţiilor private. Csató Lehel a făcut nişte comentarii folositoare în legătură cu versiunea preliminară a tezei. Mulţumesc lui Bartha Attila pentru implementarea algoritmului genetic pentru rezolvarea problemei datoriilor.

Toate aceste persoane au contribuit la dezvoltarea mea în persoana care sunt astăzi.

[1] Propoziţia exactă în limba maghiară era: „Ó, jó lesz az már fiam, nem a doktori disszertációdat írod!"

Cuprins

Tabelul notaţiilor

Bold font	Termen nou
Capital font	Nume de metodă abstractă, nume de problemă
`Typewriter font`	Abrevierea unui algoritm, implementarea concretă a unei metode
Italic font	Nume de variabilă, notaţie matematică
n	Numărul nodurilor într-un graf
m	Numărul muchiilor sau arcelor într-un graf
nr_q	Numărul interogărilor într-un algoritm de grafuri dinamice
nr_u	Numărul actualizărilor într-un algoritm de grafuri dinamice
nr	Numărul total de operaţii într-un algoritm de grafuri dinamice
t_u	Complexitatea de timp a actualizărilor în cel mai rău caz
t_q	Complexitatea de timp a interogărilor în cel mai rău caz
$\overline{t_u}$	Complexitatea de timp amortizată a actualizărilor
$\overline{t_q}$	Complexitatea de timp amortizată a interogărilor
t_a	Complexitatea de timp amortizată pe operaţie
t_p	Complexitatea de timp pentru faza de preprocesare
t_t	Complexitatea de timp totală aşteptată în cel mai rău caz
$u \cdots v$	Drum de la nodul u la nodul v

Lista acronimelor

`Abd97`	Algoritmul lui Abdeddaïm, descris în [Abd97]
`Abd00`	Algoritmul lui Abdeddaïm, descris în [Abd00]
`DSF`	Păduri de mulţimi disjuncte, descris în Secţiunea 2.1
`Debt`	Algoritmul autorului pentru problema datoriilor, descris în Secţiunea 4.2
`ES`	Algoritmul de conexitate decrementală de Even şi Shiloach
`FredI-85`	Partiţie topologică de Frederickson
`FredI-91`	Partiţie restricţionată de Frederickson
`FredI-Mod`	Partiţie uşoară de Amato et. al
`FredII-85`	Arbore topologic bazat pe partiţii topologice de Frederickson
`FredII-91`	Arbore topologic bazat pe partiţii restricţionate de Frederickson
`FredIII-85`	Arbore topologic 2-dimensional bazat pe `FredII-85`
`FredIII-91`	Arbore topologic 2-dimensional bazat pe `FredII-91`
`HK`	Algoritmul de conexitate complet dinamică de Henzinger şi King
`HT`	Rafinament la `HK` de Henzinger şi Thorup
`HDT`	Algoritmul de conexitate complet dinamică de Holm, De Lichtenberg şi Thorup
`HDTMST`	Algoritmul de arbore parţial minim complet dinamică de Holm, De Lichtenberg and Thorup
`Ital`	Algoritmii de închidere tranzitivă parţial dinamice de Italiano
`Ital-Gen`	Generalizare la `Ital` de Frigioni et. al
`KUF`	Arbore k-UF de Blum, descris în Secţiunea 2.2
`Spars(X)`	Sparsificare descris în Secţiunea 2.6 peste algoritmul `X`
`ThoDec`	Algoritmul de conexitate decrementală de Thorup

1 Introducere

1.1 Motivaţie

Teoria grafurilor este un domeniu consacrat al matematicii combinatoriale. Este de asemenea unul dintre cele mai active domenii al matematicii, care şi-a găsit numeroase aplicaţii în diverse domenii, incluzând nu numai informatica, dar şi chimia, fizica, biologia, antropologia, psihologia, geografia, istoria, economia şi multe ramuri ale ingineriei. Teoria grafurilor este în special folositoare în informatică, fiindcă orice structură de date poate fi reprezentată ca un graf. Mai mult, are aplicaţii în reţele, proiectarea arhitecturii calculatoarelor şi în general în orice ramură a informaticii ([HarGup97]).

Algoritmii tradiţionali de grafuri lucrează pe grafuri statice, adică se ocupă de dezvoltarea unui algoritm, care dându-se un graf fixat ca intrare, rezolvă o anumită problemă, de exemplu: ”este graful conex?”.

Grafurile dinamice nu sunt fixate în timp, ele pot evolua prin unele actualizări locale. Problema trebuie rezolvată eficient după fiecare modificare. Provocarea unui algoritm dinamic este să menţină proprietate cerută de-a lungul actualizărilor, fără a recalcula totul de la început de fiecare dată. Grafurile dinamice modelează mult mai precis multe grafuri, care apar în aplicaţii din viaţa reală, fiindcă niciun sistem mare nu este static cu adevărat ([AlbEtAl98, Zar02]).

1.2 Structura tezei

Secţiunea 1.1 conţine descrierea motivaţiei pentru studiul grafurilor dinamice. Lista publicaţiilor proprii şi a rezultatelor originale apare în Secţiunea 1.3. În Secţiunea 1.4 bazele teoretice şi conceptele principale folosite în această teză sunt prezentate. În încheierea capitolului (Secţiunea 1.5) se face o prezentare a modelelor computaţionale teoretice, care se folosesc în primul rând la determinarea limitelor inferioare posibile privind timpii de rulare ai algoritmilor pentru rezolvarea unor probleme.

În Capitolul 2 sunt prezentate structurile de date uzuale în algoritmii de grafuri dinamice. Capitolele 3 şi 4 conţin cele mai importante probleme pe grafuri dinamice neorientate respectiv orientate.

1.3 Contribuţii originale

Contribuţiile originale ale autorului se găsesc în lista de mai jos. Mare parte din ele a fost publicat în [Pat09], [Pat11], [Pat11b] şi [PatBar11]. Publicaţii având o oarecare legătură sunt [PatIon08] şi [IonPat08].

- Un studiu cuprinzător al celor mai importante probleme pe grafuri dinamice neorientate şi orientate, cu o comparaţie a algoritmilor recenţi nu numai din punct de vedere teoretic, dar şi practic.
- Primul pseudocod detaliat pentru descrierea algoritmului de conexitate inventat de Even şi Shiloach.
- Demonstraţii riguroase din punct de vedere matematic în privinţa relaţiei problemei datoriilor cu clasele de complexitate NP-hard, strongly NP-hard şi NP-easy pe cazul general şi pe cazul restricţionat la un singur drum.
- Algoritmi şi structuri de date dezvoltate pentru a rezolva problema datoriilor pe cazul static şi cel dinamic, testat într-un şir de experimente.
- Operatori noi de recombinare şi mutaţie folosiţi în algoritmul genetic.

1.4 Definiţii şi notaţii

În această secţiune dăm unele definiţii bine cunoscute din teoria grafurilor, urmat de definiţia grafului dinamic şi o scurtă clasificare a problemelor pe grafuri dinamice.

Definiţie 1.1 Spunem că $G = (V, E)$ este un **graf**, unde V este mulţimea **nodurilor** sau a **vârfurilor**, şi $E \subseteq V \times V$ este mulţimea **muchiilor** sau a **arcelor**. □

Definiţie 1.2 Dacă $(i, j) \in E \Leftrightarrow (j, i) \in E$ atunci graful este **neorientat**, şi E se numeşte mulţimea muchiilor, altfel graful este **orientat** (uneori numit **digraf**) şi E este numit mulţimea arcelor (câteodată notat cu A). □

Definiţie 1.3 Dacă G este orientat şi nu conţine cicluri, atunci este numit **graf orientat aciclic**, prescurtat cu **DAG**. □

Definiţie 1.4 Într-un **graf ponderat** avem o pondere asociată cu fiecare muchie sau arc, $w : E \to \mathbb{R}$. □

Definiţie 1.5 Un **graf dinamic** care se schimbă în timp printr-un şir de actualizări. O actualizare este o operaţie care adaugă sau şterge un nod sau o muchie din graf sau schimbă atributele asociate unui nod sau unei muchii. □

Definiţie 1.6 O problemă de grafuri dinamice se numeşte **incrementală (incremental)** dacă numai adăugări sunt permise. □

Definiţie 1.7 O problemă de grafuri dinamice se numeşte **decrementală (decremental)** dacă numai ştergeri sunt permise □

Definiţie 1.8 O problemă de grafuri dinamice se numeşte **parţial dinamică (partially dynamic)** dacă este incrementală sau decrementală. □

Definiţie 1.9 O problemă de grafuri dinamice se numeşte **complet dinamică (fully dynamic)** dacă nu sunt restricţii în legătură cu tipul actualizărilor. □

1.5 Modele computaţionale

Au fost elaborate numeroase modele computaţionale pentru a facilita analiza teoretică a grafurilor dinamice. O descriere scurtă a celor mai importante dintre acestea se găseşte în părţile corespunzătoare din teză.

2 Structuri de date

2.1 Pădurile de mulţimi disjuncte

Introducere Această structură de date se foloseşte pentru reprezentarea unor mulţimi disjuncte de elemente şi stă la baza algoritmilor de conexitate incrementală.

Operaţii suportate Fiecare mulţime are un **nume**, care în majoritatea cazurilor este un element al mulţimii (numit şi reprezentant). Operaţiile suportate sunt:

- MAKE(x): Creează o mulţime nouă, al cărui singur membru este x. Fiindcă mulţimile sunt disjuncte, x nu trebuie să facă parte din altă mulţime.
- UNION(x, y): Uneşte mulţimile având x şi y ca reprezentanţi. Se presupune $x \neq y$.
- FIND(x): Returnează numele mulţimii, care îl conţine pe x.

Performanţă Fie nr_u numărul operaţiilor UNION, nr_q numărul operaţiilor FIND, n numărul operaţiilor MAKE şi $nr = nr_u + nr_q + n$. Dat fiind că mulţimile sunt disjuncte, este uşor de dedus, că după $n-1$ operaţii UNION rămâne o singură mulţime, deci $nr_u \leq n-1 \Rightarrow nr_u = O(n)$. Presupunem de asemenea, că operaţiile MAKE sunt primele n operaţii efectuate.

MAKE şi UNION sunt suportate în timp constant, iar FIND este $t_q = O(\log n)$ în cel mai rău caz. Întreaga secvenţă de operaţii are $t_t = O(n + nr_q \cdot \alpha(nr_q + n, n))$ timp total şi $\Theta(n)$ memorie este necesară.

α este o funcţie care creşte foarte lent şi valoarea ei nu depăşeşte 4 pentru aplicaţiile practice. Este definită ca inversa funcţiei lui Ackermann, definită în felul următor:

$A(1, j) = 2^j$, dacă $j \geq 1$

$A(i, 1) = A(i-1, 2)$, dacă $i \geq 2$

$A(i,j) = A(i-1, A(i,j-1))$, dacă $i,j \geq 2$
$\alpha(m,n) = \min\{i \geq 1 | A(i, \lfloor m/n \rfloor) > \log n\}$

2.2 Arborele k-UF

Introducere Arborele k-UF a fost introdus în [Blu85] pentru a rezolva problema mulţimilor disjuncte, asigurând şi cea mai bună complexitate pe operaţie în cel mai rău caz pentru problema conexităţii incrementale.

Operaţii suportate Aborele k-UF suportă aceleaşi operaţii ca pădurile de mulţimi disjuncte: MAKE, UNION şi FIND.

Performanţă FIND şi UNION sunt $O(\log n / \log \log n)$ în cel mai rău caz, însă timpul lor de rulare nu se amortizează. Complexitatea de memorie este $\Theta(n)$.

2.3 Clusterul de vârfuri (Vertex cluster)

Introducere Clusterii de vârfuri, arborele topologic şi arborele topologic 2-dimensional a fost introdus în [Fre83] pentru a suporta operaţiile de schimbarea costului unei muchii în arborele parţial minim al unui graf. Aceste structuri de date însă au numeroase aplicaţii cum ar fi menţinerea arborelui parţial minim în grafuri planare, conexitate, generarea celor mai mici k arbori parţiali sau biconexitate ([Fre85, Fre97])).

Clusterii de vârfuri funcţionează peste un arbore parţial al grafului şi se bazează pe gruparea nodurilor din graf în mulţimi, care induc subgrafuri conexe. Toate cele patru strategii de grupare sunt descrişi în teză, fiecare având propriul ei set de avantaje şi dezavantaje.

Definiţie 2.1 Pentru a evita confuzia, în restul tezei ne vom referi la arborele original ca **arbore de bază (underlying tree)**, pentru a-l diferenţia de arborele construit peste el. Arborele de bază este uneori arborele parţial al grafului de la intrare, care se va numi **graf de bază (underlying graph)**.□

Definiţie 2.2 Muchiile din graful de bază, care nu aparţin arborelui de bază se numesc **muchii din afara arborelui (nontree edges)**. □

Operaţii suportate Următoarele operaţii sunt suportate:

- SWITCH(u, v, x, y): înlocuieşte muchia (x, y) cu (u, v). Se presupune, că (x, y) este pe drumul $u \cdots v$ din arbore.
- REMOVE(u, v): şterge muchia (u, v) din arborele parţial şi returnează o muchie de înlocuire, dacă aceasta există. Se presupune, că (u, v) face parte din arborele parţial.

Performanţă SWITCH şi REMOVE pot fi suportate în $O(m^{2/3})$, folosind $O(m)$ spaţiu şi timp de preprocesare $t_p = O(m)$.

2.4 Arborele topologic (Topology tree)

Introducere Arborele topologic este o reprezentare ierarhică a clusterelor de vârfuri, construit recursiv prin aplicarea unei partiţii a nodurilor, până când rămâne un singur nod.

Operaţii suportate Arborele topologic suportă aceleaşi operaţii ca clusterii de vârfuri. În plus pentru a implementa aceste operaţii, un arbore topologic poate fi spart, sau doi arbori topologici pot fi uniţi.

- SPLIT(T, u, v): sparge arborele topologic T după ştergerea muchiei (u, v).
- MERGE(T_1, T_2): uneşte arborii topologici T_1 şi T_2..
- SWITCH(u, v, x, y): înlocuieşte muchia (x, y) cu (u, v). Se presupune, că (x, y) este pe drumul $u \cdots v$ din arbore.
- REMOVE(u, v): şterge muchia (u, v) din arborele parţial şi returnează o muchie de înlocuire, dacă acesta există. Se presupune, că (u, v) face parte din arborele parţial.

Performanţă Arborele topologic poate fi construit în timp liniar în numărul nodurilor din arborele parţial. SPLIT şi MERGE pot fi suportate în $O(\log n)$, unde n este numărul nodurilor din arbore. SWITCH şi REMOVE sunt suportate în $O(\sqrt{m \log m})$, timpul de preprocesare fiind $t_p = O(m)$ şi complexitatea de memorie de asemenea $O(m)$.

2.5 Arborele topologic 2-dimensional (2-dimensional topology tree)

Operaţii suportate Arborele topologic 2-dimensional suportă aceleaşi operaţii ca şi arborele topologic.

Performanţă Toate operaţiile durează $O(\sqrt{m})$ timp, cu timp de preprocesare $t_p = O(m)$ şi $O(m)$ memorie.

2.6 Arborele de sparsificare (Sparsification tree)

Introducere Sparsificarea (sparsification) este o tehnică generală ce se poate aplica la o varietate largă de probleme pe grafuri dinamice. Se aplică peste algoritmi de grafuri pentru a le accelera şi se poate folosi ca o cutie neagră (black box), adică nu necesită cunoştinţe despre detaliile interne ale algoritmului peste care se aplică. A fost introdus în [EppEtAl92] şi a îmbunătăţit complexitatea de timp a multor algoritmi de grafuri dinamice, cum ar fi arborele parţial minim, biconexitatea, generarea celor mai mici k arbori parţiali, sau triconexitatea. De asemenea prin sparsificare s-au obţinut primii algoritmi dinamici pentru problema 4-conexităţii, k-conexităţii şi a bipartităţii. Mai târziu tehnica a fost uşor îmbunătăţită în [EppGalIta93], apoi rezultatele rezumate în [EppEtAl97]. În [AmaCatIta97] prima versiune a fost denumită **sparsificare simplă (simple sparsification)**, iar a doua **sparsificare îmbunătăţită (improved sparsification)**. Ambele variante sunt descrise în detaliu în teză.

Operaţii suportate Există trei strategii de sparsificare:

- Sparsificarea elementară (basic sparsification) se poate folosi pentru dinamizarea algoritmilor statici.
- Sparsificarea stabilă (stable sparsification) se poate folosi pentru accelerarea algoritmilor complet dinamici existenţi.
- Sparsificarea asimetrică (asymmetric sparsification) este util în aplicaţii în care numărul adăugărilor este mai mare ca cea a ştergerilor şi pentru care există algoritmi dinamici parţiali care suportă adăugări.

Performanţă Pentru a aplica sparsificarea elementară trebuie calculate eficient certificatele sparse (sparse certificates). Dacă notăm timpul necesar calculării certificatului cu $f(n, m)$, timpul necesar construcţiei unei structuri de date capabilă să testeze proprietatea cu $g(n, m)$, care poate furniza răspunsuri la interogări în $q(n, m)$, atunci o actualizare poate fi suportată cu sparsificare elementară în $O(f(n, O(n)) \cdot \log(m/n) + g(n, O(n)))$ cu sparsificare simplă şi în $O(f(n, O(n)) + g(n, O(n)))$ cu sparsificare îmbunătăţită, iar o interogare poate fi suportată în $q(n, O(n))$.

Sparsificarea stabilă este utilă când putem menţine eficient certificatele sparse stabile (stable sparse certificates). Această variantă transformă limite de timp de forma $O(m^p)$ în cele de forma $O(n^p)$. Mai general, dacă notăm cu $f(n, m)$ timpul necesar menţinerii unui certificat sparse stabil, pentru care există o structură de date capabilă să testeze proprietatea cu timp de actualizare $g(n, m)$ şi timp de interogare $q(n, m)$, atunci aceleaşi limite de timp se aplică ca în cazul sparsificării elementare.

Dacă notăm cu $f(n, m)$ timpul necesar găsirii unui certificat sparse, cu $g(n, m)$ timpul necesar construcţiei unei structuri de date parţial dinamice pentru testarea proprietăţii capabilă să suporte adăugări de muchii în $p(n, m)$ şi să răspundă la interogări în $q(n, m)$, atunci folosind sparsificarea asimetrică putem obţine un algoritm complet dinamic, care suportă adău-

gări de muchii în $O(\frac{f(n,O(n))+g(n,O(n))}{n} + p(n, O(n)))$, ştergeri de muchii în $f(n, O(n)) \cdot O(\log(m/n)) + g(n, O(n))$ şi interogări în $q(n, O(n))$.

Memoria necesară stocării arborelui de sparsificare este $O(m)$ în cazul sparsificării simple. Pentru sparsificarea avansată o limită de $O(m \log(n^2/m))$ se obţine uşor, care poate fi îmbunătăţit la $O(m)$ pentru SPARSIFICAREA ELEMENTARĂ şi $O(\frac{m}{n} \cdot h(n))$ în cazul SPARSIFICĂRII STABILE, unde $h(n)$ este spaţiul necesar unui singur nod în arborele de sparsificare.

Timpul de preprocesare este $O(m)$ pentru sparsificarea simplă. Pentru sparsificarea îmbunătăţită $t_p = O(m \log(n^2/m))$ se obţine trivial şi se poate optimiza la $O(\frac{m}{n} \cdot h(n))$ în cazul SPARSIFICĂRII ELEMENTARE şi al SPARSIFICĂRII STABILE, unde $h(n)$ este timpul necesar procesării unui singur nod în arborele de sparsificare.

2.7 Arborele Euler Tour

Introducere Arborele Euler Tour a fost introdus în [HenKin99] ca un ingredient al algoritmului de conexitate complet dinamic descris, care a fost primul algoritm cu limite polilogaritmice pentru problema conexităţii complet dinamice.

Operaţii suportate Numeroase operaţii pot fi efectuate eficient:

- TREE(u): returnează rădăcina arborelui, care conţine nodul u.
- NONTREEEDGES(T): returnează lista muchiilor din afara arborelui vecine cu T. Muchiile cu ambele capete în T sunt returnate de două ori.
- INSERTTREE(u, v): adaugă (u, v) în arbore, conectând arborele, care conţine nodul u cu arborele, care conţine nodul v.
- INSERTNONTREE(u, v): adaugă muchia din afara arborelui (u, v).
- DELETETREE(u, v): sparge arborele, care conţine nodurile u şi v prin ştergerea muchiei (u, v).

- DeleteNontree(u, v): şterge muchia din afara arborelui (u, v).
- SampleAndTest(T): selectează aleator o muchie din afara arborelui vecin cu T şi îl returnează dacă are exact un capăt în T. Muchiile cu ambele capete în T au probabilitate de două ori mai mare să fie selectate.

Performanţă Algoritmul de conexitate din [HenKin99] foloseşte două metode pentru implementarea arborelui Euler Tour, prima cu arbori binari şi a doua cu arbori $\log n$-ari. În prima implementare NontreeEdges rulează în $O(m' \log n)$, unde m' este mărimea intrării, în timp ce restul operaţiilor au nevoie de $O(\log n)$ timp. În a doua implementare DeleteTree şi InsertTree devin $O(\log^2 n / \log \log n)$, în timp ce Tree devine $O(\log n / \log \log n)$, iar restul operaţiilor rămâne neschimbat.

Un arbore Euler Tour poate fi stocat în spaţiu $O(n)$, fiind nevoie de memorie adiţională $O(m)$ dacă şi muchiile din afara arborelui trebuie menţinute. Timpul de preprocesare este $t_p = O(m \log n + n)$ ([AlbCatIta97]).

3 Probleme pe grafuri dinamice neorientate

3.1 Conexitate incrementală

Problema conexităţii incrementale poate fi definită după cum urmează. Dat fiind un graf neorientat, iniţial conţinând n noduri izolate, să se suporte următoarele operaţii:

- Insert(u, v): adaugă o muchie între nodurile u şi v.
- Connected(u, v): returnează *true* dacă nodurile u şi v se află in aceeaşi componentă conexă şi *false* altfel.

Un tabel comparativ cu algoritmii prezentaţi în teză se găseşte în Figura 1. Pentru semnificaţia prescurtărilor vezi Tabelul notaţiilor şi Lista acronimelor de la începutul lucrării.

Acronimul algoritmului	t_p	t_u	t_q	t_a	Complexitate de memorie
`DSF`	$\Theta(n)$	$O(\log n)$	$O(\log n)$	$O(\alpha(nr_q + n, n)))$	$\Theta(n)$
`KUF`	$\Theta(n)$	$O(\frac{\log n}{\log \log n})$	$O(\frac{\log n}{\log \log n})$	$O(\frac{\log n}{\log \log n})$	$\Theta(n)$

Figura 1: Comparaţia algoritmilor de conexitate incrementală

3.2 Conexitate decrementală

Problema conexităţii decrementale poate fi definită după cum urmează. Dat fiind un graf neorientat $G(V, E)$, să se suporte următoarele operaţii:

- DELETE(u, v): şterge muchia dintre nodurile u şi v. Se presupune, că $(u, v) \in E$.
- CONNECTED(u, v): returnează *true* dacă nodurile u şi v se află in aceeaşi componentă conexă şi *false* altfel.

În cazul algoritmului lui Even şi Shiloach (`ES`) $t_p = \Theta(n + m), t_u = O(m), t_q = O(1), t_a = O(n)$ şi spaţiul folosit este $\Theta(n + m)$. În cazul algoritmului lui Thorup (`ThoDec`) asemenea limite sunt mult mai dificil de obţinut, în afară de t_a, care a fost demonstrat în [Tho99]. Folosind tehnica din [Tho00] complexitatea de memorie pentru fiecare nivel al recursivităţii se poate reduce la $O(m)$. Chiar şi aşa, constanta ascunsă este mare, fiindcă la fiecare nivel se stochează numeroase grafuri.

3.3 Conexitate complet dinamică

În problema conexităţii complet dinamice următoarele operaţii trebuie suportate:

- INSERT(u, v): adaugă o muchie între nodurile u şi v. Se presupune, că $(u, v) \notin E$.
- DELETE(u, v): şterge muchia dintre nodurile u şi v. Se presupune, că $(u, v) \in E$.

Acronimul algoritmului	t_u	t_q	t_a	Timp de rulare mediu
`FredI-85`, `FredI-91`	$O(m^{2/3})$	$O(1)$	$O(m^{2/3})$	$O(\frac{n}{m^{1/3}} + \log n)$
`FredII-85`, `FredII-91`	$O(\sqrt{m \log m})$	$O(1)$	$O(\sqrt{m \log m})$	$O(\frac{n \cdot \sqrt{\log m}}{\sqrt{m}} + \log n)$
`FredIII-85`, `FredIII-91`	$O(\sqrt{m})$	$O(1)$	$O(\sqrt{m})$	$O(\frac{n}{\sqrt{m}} + \log n)$
`Spars(FredIII)`	$O(\sqrt{n})$	$O(1)$	$O(\sqrt{n})$	$O(\sqrt{n})$
`HK`	$O(m \log n)$	$O(\frac{\log n}{\log \log n})$	$O(\log^3 n)$	
`HT`, `HDT`	$O(m \log n)$	$O(\frac{\log n}{\log \log n})$	$O(\log^2 n)$	

Figura 2: Comparaţia timpilor de actualizare şi interogare ai algoritmilor de conexitate complet dinamică

Acronimul algoritmului	t_p	Complexitate de memorie
`FredI-85`, `FredI-91`	$O(m)$	$O(m)$
`FredII-85`, `FredII-91`	$O(m)$	$O(m)$
`FredIII-85`, `FredIII-91`	$O(m)$	$O(m)$
`Spars(FredIII)`	$O(m)$	$O(m)$
`HK`	$O(m + n \log n)$	$O(m + n \log n)$
`HT`, `HDT`	$O(m + n \log n)$	$O(m + n \log n)$

Figura 3: Comparaţia timpilor de preprocesare şi al complexităţilor de memorie pentru algoritmii de conexitate complet dinamică

- CONNECTED(u, v): returnează *true* dacă nodurile u şi v se află in aceeaşi componentă conexă şi *false* altfel.

În Figura 2 sunt listaţi timpii de rulare cunoscuţi.

În Figura 3 este comparat timpul de preprocesare şi complexitatea de memorie a algoritmilor. Pentru `HDT` complexitatea de memorie se referă la cel descris în articolul original, care se poate îmbunătăţi la $O(m)$ folosind tehnica din [Tho00].

3.4 Arborele parţial minim complet dinamic

Nu tratăm separat versiunile parţial dinamice ale problemei din următoarele motive. Varianta incrementală se poate rezolva cu uşurinţă în $O(\log n)$ pe actualizare folosind arbori link-cut. Pe de altă parte soluţia variantei

Acronimul algoritmului	t_a	Complexitate de memorie
`FredI-85`, `FredI-91`	$O(m^{2/3})$	$O(m)$
`FredII-85`, `FredII-91`	$O(\sqrt{m \log m})$	$O(m)$
`FredIII-85`, `FredIII-91`	$O(\sqrt{m})$	$O(m)$
`Spars(FredIII)`	$O(\sqrt{n})$	$O(m)$
`HDTMST`	$O(\log^4 n)$	$O(m \log n)$

Figura 4: Comparaţia algoritmilor de arbore parţial minim complet dinamic

decrementale este unul dintre ingredientele algoritmului lui Holm et. al, descris în teză.

În problema arborelui parţial minim complet dinamic, dându-se un graf neorientat, ponderat $G(V, E, W)$, vrem să suportăm următoarele operaţii:

- INSERT(u, v, w): adaugă muchia (u, v) în graf cu ponderea w. Se presupune $(u, v) \notin E$ înaintea operaţiei.
- REMOVE(u, v): şterge muchia (u, v) din graf. Se presupune $(u, v) \in E$.
- CHANGE(u, v, w): schimbă ponderea muchiei (u, v) la w. Se presupune $(u, v) \in E$.
- MST(): returnează costul total al arborelui parţial minim şi muchiile pe care acesta le conţine, dacă se cere acest lucru. Folosim termenul "arbore" chiar dacă este vorba despre o pădure, când graful nu este conex.

Notăm că operaţia CHANGE(u, v, w) nu este critică, fiindcă poate fi înlocuită cu o secvenţă de REMOVE(u, v) şi INSERT(u, v, w).

În Figura 4 este listat timpul de rulare amortizat pe operaţie şi complexitatea de memorie pentru fiecare algoritm de arbore parţial minim complet dinamic.

4 Probleme pe grafuri dinamice orientate

4.1 Închidere tranzitivă dinamică

Nu tratăm separat versiunile parţial dinamice şi complet dinamice, fiindcă experimentele au demonstrat ([KroZar08]), că cei mai buni algoritmi complet dinamici cunoscuţi în acest moment sunt clar inferiori unor soluţii triviale şi a unor algoritmi hibrizi bazaţi pe soluţii parţial dinamice. Aşadar prezentăm doar algoritmii incrementali şi decrementali având semnificaţie practică.

Dându-se un graf orientat $G(V, A)$, avem nevoie de următoarele definiţii.

Definiţie 4.1 Un nod v este **accesibil (reachable)** din vârful u dacă şi numai dacă există un drum orientat de la u la v în G. □

Definiţie 4.2 Digraful $G(V, A^*)$, având aceeaşi mulţime de noduri ca G dar având arcul $(u, v) \in A^*$ dacă şi numai dacă v este accesibil din u în G se numeşte **închiderea tranzitivă (transitive closure)** a lui G. Notăm $|A^*|$ cu m^*. □

Definiţie 4.3 Dacă v este accesibil din u (în G), numim v **urmaşul (descendant)** sau **succesorul** lui u şi u este un **strămoş (ancestor)** sau un **predecesor** al lui v. □

Operaţiile care trebuie suportate sunt:

- INSERT(u, v): adaugă arcul (u, v) în graf.
- REMOVE(u, v): şterge arcul (u, v) din graf.
- REACHABLE(u, v): returnează *true* dacă există un drum orientat din nodul u în nodul v şi *false* altfel.
- SEARCHPATH(u, v): returnează un drum de la nodul u la nodul v, sau $\emptyset$ dacă nu există.

Acronimul of algoritmului	t_p	t_t	Complexitate de memorie
`Abd97`	$O(n \cdot m)$	$O(k^2 \cdot (m + nr_u) + (m + nr_u)^*)$	$O(k \cdot n)$
`Abd00`	$O(n \cdot m)$	$O(k \cdot (m + nr_u)^*)$	$O(k \cdot n)$
`Ital`	$O(n^2 + n \cdot m)$	$O(n \cdot (m + nr_u))$	$O(n^2)$
`Ital-Gen`	$O(n^2 + n \cdot m)$	$O(m^2)$	$O(n^2)$
`RZ`	$O(n^2 + n \cdot m)$	$O(n \cdot m)$	$O(n^2)$

Figura 5: Comparaţia algoritmilor de închidere tranzitivă dinamică

În Figura 5 diverse complexităţi ale algoritmilor de închidere tranzitivă dinamică sunt prezentaţi. `Abd97` şi `Abd00` sunt incrementali, şi k este numărul drumurilor disjuncte din puncte de vedere al nodurilor, în care graful este descompus. `Ital` este ori incremental ori decremental, limitele de timp nu se păstrează în cazul unor şiruri mixte de operaţii. Timpul total aşteptat pentru `Ital-Gen` este pentru partea decrementală, cea incrementală având aceeaşi complexitate ca `Ital`. `RZ` este decremental.

4.2 Problema datoriilor

Introducere În această secţiune este discutată o problemă originală propusă în 2008 de autor la concursul de selecţie al lotului naţional de informatică pentru Olimpiada de Informatică a Europei Centrale şi Olimpiada Balcanică de Informatică.

Enunţul problemei este următorul:

Se consideră n entităţi (de ex. persoane, companii), şi o listă de m împrumuturi între aceste entităţi. Un împrumut poate fi descris prin trei parametri: indicele entităţii care cere împrumutul, indicele entităţii care dă împrumutul şi suma de bani împrumutată. Problema cere găsirea unei liste minime de tranzacţii prin care se rezolvă datoriile create ca urmare a celor m împrumuturi.

În [Pat09] problema este modelată folosind teoria grafurilor:

Definiţie 4.4 Fie $G(V, A, W)$ un multigraf orientat ponderat fără bucle, $|V| = n$, $|A| = m$, $W : A \to \mathbb{Z}$, unde V este mulţimea nodurilor, A mulţimea

Lista împrumuturilor:

Entitatea care cere	Entitatea care dă	Suma
1	2	10
2	3	5
3	1	5
1	4	5
4	5	10

Solution:

Entitatea care plăteşte	Entitatea care primeşte	Suma
1	5	10
4	2	5

Figura 6: Exemplu pentru problema datoriilor

arcelor şi W funcţia de pondere. G reprezintă împrumuturile efectuate, aşadar va fi denumit **graf de împrumuturi (borrowing graph)**. □

Graful de împrumuturi corespunzător exemplului din Figura 6 se găseşte în Figura 7.

Definiţie 4.5 Se defineşte pentru fiecare nod $v \in V$ **suma absolută datorată (absolute amount of debt)** pe graful G:

$$D_G(v) = \sum_{\substack{v' \in V \\ (v,v') \in A}} W(v,v') - \sum_{\substack{v'' \in V \\ (v'',v) \in A}} W(v'',v)$$

Uneori pentru simplicitate, suma absolută datorată corespunzătoare unui nod se va numi **valoarea** D. □

Definiţie 4.6 Fie $G'(V, A', W')$ un multigraf orientat ponderat fără bucle, fiecare arc (i, j) reprezentând tranzacţia sumei $W'(i, j)$ de la entitatea i la entitatea j. Acest graf se numeşte **graf de tranzacţii (transaction graph)**. Aceste tranzacţii rezolvă datoriile formate de împrumuturile modelate de graful $G(V, A, W)$ dacă şi numai dacă:

$D_G(v_i) = D_{G'}(v_i), \forall i = \overline{1,n}$, unde $V = \{v_1, v_2, \ldots, v_n\}$

Acest lucru se notează cu: $G \sim G'$. □

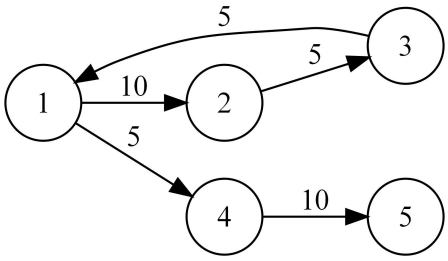

Figura 7: Graful de împrumuturi asociat cu exemplul dat. Un arc de la nodul i spre nodul j cu ponderea w are semnificaţia, că entitatea i trebuie să plătească suma w entităţii j.

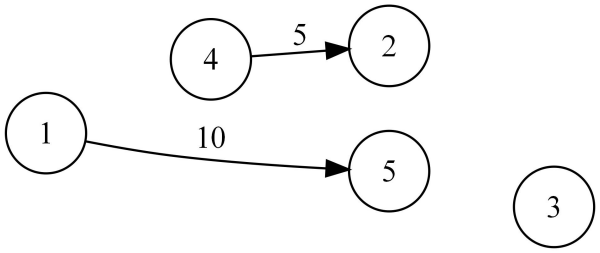

Figura 8: Graful de tranzacţii minim. Un arc de la nodul i spre nodul j cu ponderea w are semnificaţia, că entitatea i plăteşte suma w entităţii j.

Vezi Figura 8 pentru un graf de tranzacţii, care corespunde exemplului din Figura 6, cu număr minim de arce.

Folosind termenii de mai sus, problema datoriilor se poate reformula în felul următor:

Dându-se graful de împrumuturi $G(V, A, W)$ *să se găsească graful de tranzacţii minim* $G_{min}(V, A_{min}, W_{min})$*, în aşa fel încât* $G \sim G_{min}$ *şi* $\forall G'(V, A', W') : G \sim G', |A_{min}| \leq |A'|$.

Relaţia problemei cu clasele de complexitate Să notăm problema din introducere cu DEBT. Problema de decizie corespunzătoare se va numi DEBT-DECISION, definit în felul următor:

Dându-se un graf de împrumuturi $G(V, A, W)$ şi un număr natural $M \leq |A|$, să se determine existenţa unui graf de tranzacţii $G'(V, A', W'), G \sim G'$, în aşa fel încât $|A'| \leq M$.

Lemă 4.7 DEBT-DECISION *este NP.* □

Lemă 4.8 SUBSET SUM *este reductibil la* DEBT-DECISION. □

Teoremă 4.9 DEBT-DECISION *este NP-completă.* □

Corolar 4.10 DEBT *este NP-grea (NP-hard).* □

Lemă 4.11 3-PARTITION *este transformabil pseudopolinomial în* DEBT-DECISION. □

Teoremă 4.12 DEBT-DECISION *este tare NP-completă (NP-complete in the strong sense).* □

Corolar 4.13 DEBT *este tare NP-grea (NP-hard in the strong sense).* □

Se defineşte problema DEBT-DECISION-PARTIAL după cum urmează:

Dându-se un graf de împrumuturi $G(V, A, W)$, un "graf parţial" $G^p(V, A^p, W^p)$ şi un număr natural $M \leq |A|$, să se determine, dacă G^p poate fi "completat" într-un graf de tranzacţii cu cel mult M arce. Adică, să se determine existenţa unui graf de tranzacţii $G'(V, A', W'), G \sim G'$, în aşa fel încât $|A'| \leq M$ şi $A^p \subset A, W^p(a) = W'(a), \forall a \in A^p$.

Lemă 4.14 DEBT-DECISION-PARTIAL *este NP.* □

Lemă 4.15 ([Pat11b]) DEBT *este Turing reductibil la* DEBT-DECISION-PARTIAL. □

Teoremă 4.16 ([Pat11b]) DEBT *este NP-uşor (NP-easy).* □

Corolar 4.17 DEBT *is NP-echivalent (NP-equivalent).* □

O variantă restricţionată Problema DEBT-PATH este definită în felul următor:

Dându-se un graf de împrumuturi $G(V, A, W)$, al cărui arce formează un drum, să se găsească graful de tranzacţii minim $G'(V, A', W'), G \sim G'$. Reformulând matematic $A = \bigcup_{i=1}^{n-1} \{(v_{p_i}, v_{p_{i+1}})\}$, $v_{p_i} = v_{p_j} \Rightarrow i = j, \forall i, j = \overline{1, n}$.

Teoremă 4.18 ([Pat11b]) DEBT-PATH *este NP-grea.* □

Teoremă 4.19 ([Pat11b]) DEBT-PATH *este tare NP-grea.* □

Teoremă 4.20 DEBT-PATH *este NP-uşor.* □

Corolar 4.21 DEBT-PATH *este NP-echivalent.* □

Menţionăm că Lema 4.8, Corolarul 4.10, Lema 4.11 şi Corolarul 4.13 au fost enunţate şi în [Ver04].

O soluţie bazată pe metoda programării dinamice Metoda propusă foloseşte tehnici similare celor descoperite independent de Bellman ([Bel62]), respectiv Held şi Karp ([HelKar62]) pentru rezolvarea problemei comis voiajorului.

Următoarea observaţie este crucială în această soluţie.

Teoremă 4.22 ([PatBar11]) *Orice caz al problemei datoriilor se poate rezolva trivial cu cel mult $n - 1$ tranzacţii.* □

Se notează cu V_{left} mulţimea nodurilor având valoare D pozitivă şi cu V_{right} mulţimea nodurilor având valoare D negativă, adică $V_{left} = \{u | D(u) > 0\}$, $V_{right} = \{u | D(u) < 0\}$. Fie $n_1 = |V_{left}|$, $n_2 = |V_{right}|$ şi $V_{left} = \{left_1, \ldots, left_{n_1}\}$, $V_{right} = \{right_1, \ldots, right_{n_2}\}$. Subproblemele din programarea dinamică se definesc cu doi parametri i şi j, unde i este o reprezentare binară de n_1 biţi, şi j o reprezentare binară de n_2 biţi ($i = \overline{0, 2^{n_1} - 1}, j = \overline{0, 2^{n_2} - 1}$). O subproblemă va avea următoarea semnificaţie:

$dp_{i,j}$ = numărul arcelor în graful de tranzacţii minim, care conţine doar nodurile din V_{left} determinate de biţii lui i şi nodurile din V_{right} determinate de biţii lui j.

Formula recursivă pentru determinarea valorilor asociate subproblemelor este următoarea[2]:

$dp_{i,j} = \min(dp_{i \texttt{ XOR } i',j \texttt{ XOR } j'} + bitcount(i') + bitcount(j') - 1)$, unde

1. $i \texttt{ AND } i' = i'$
2. $j \texttt{ AND } j' = j'$
3. $\sum_{i' \texttt{ AND } 2^k \neq 0} D(left_k) = - \sum_{j' \texttt{ AND } 2^k \neq 0} D(right_k)$
4. $bitcount(x)$ returnează numărul de biţi ai lui x egali cu 1.

Urmează analiza performanţei algoritmului propus. Numărul subproblemelor este $2^{n_1} \cdot 2^{n_2} = 2^{n_1+n_2}$, în cel mai rău caz ajungând la 2^n. Aşadar complexitatea de memorie este $\Theta(2^n)$. Pentru a rezolva o subproblemă (i, j) este nevoie de toate perechile (i', j'), în aşa fel încât i' este o submulţime a lui i şi j' o submulţime a lui j. O pereche (i, i') se poate codifica cu un şir de n_1 cifre ternare. O cifră are valoarea 0, dacă nodul respectiv nu face parte din i, 1 dacă face parte din i dar nu face parte din i' şi 2 dacă face parte din i' (aşadar şi din i). Aceeaşi codificare se poate utiliza de asemenea pentru fiecare pereche (j, j'). Deci numărul paşilor efectuaţi de algoritm este proporţional cu $3^{n_1} \cdot 3^{n_2} = 3^n$

Problema datoriilor în grafuri dinamice În problema datoriilor dinamice ([Pat11]) următoarele operaţii trebuie suportate:

- InsertNode(u) - adaugă nodul u în graful de împrumuturi.
- RemoveNode(u) - şterge nodul u din graful de împrumuturi. Pentru ca nodul să fie şters, întâi toate datoriile legate de acesta trebuie rezolvate.

[2]Se notează cu `AND` operaţia "şi" binară şi cu `XOR` operaţia "sau exclusiv" binară

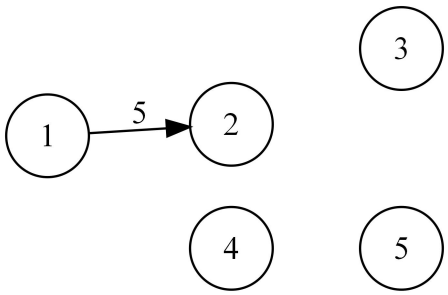

Figura 9: Rezultatul operaţiei QUERY apelat după adăugarea celui de-al treilea arc

Pentru a afecta cât mai puţin restul nodurilor, datoriile sunt rezolvate în aşa fel, încât să afecteze un număr minim de noduri fără a compromite soluţia optimă pentru întreg graful.

- INSERTARC(u, v, x) - adaugă un arc în graful de împrumuturi. Adică, u trebuie să plătească suma x lui v.
- REMOVEARC(u, v) - şterge datoria dintre u şi v.
- QUERY() - returnează un graf de tranzacţii minim.

De exemplu un apel al operaţiei QUERY după adăugarea celui de-al treilea arc în graful de împrumuturi corespunzător Figurii 6 rezultă în graful de tranzacţii minim din Figura 9.

O structură de date pentru rezolvarea datoriilor dinamice Dat fiind că varianta statică a problemei este NP-grea, nu este posibil suportarea tuturor operaţiilor în timp polinomial (dacă $P \neq NP$). Altfel s-ar putea construi întreg graful arc cu arc, apelând de m ori INSERTARC, după care s-ar obţine graful de tranzacţii minim printr-un apel al operaţiei QUERY, ceea ce ar duce la un algoritmi polinomial pentru problema statică.

Structura de date folosită este bazată pe menţinerea submulţimii nodurilor care au suma datorată absolută diferită de zero $V^* = \{u | D(u) \neq 0\}$. Suma

tuturor valorilor D a celor $2^{|V^*|}$ submulţimi a lui V^* sunt de asemenea stocate într-o tabelă de dispersie $sums$.

InsertNode Dat fiind că pentru structura de date numai nodurile având valoare D diferite de zero sunt importante, şi că un nod nou întotdeauna începe fără datorii, înseamnă că nimic nu rămâne de făcut la un apel al lui InsertNode.

InsertArc Când InsertArc este apelat, valorile D a două noduri se schimbă, aşadar şi V^* se poate schimba. Când un nod părăseşte V^*, folosim o metodă de "lazy update" în cazul submulţimilor care îl conţin, deoarece când un nod nou intră în V^*, oricum trebuie calculate toate sumele corespunzătoare submulţimilor din care acesta face parte.

Dacă u şi v erau în V^* şi au rămas acolo după schimbarea valorilor D, se adaugă x la suma tuturor submulţimilor care îl conţin pe u, dar nu pe v, şi se scade x din suma acelora, care îl conţin pe v dar nu pe u. Suma submulţimilor, care conţin ambele noduri nu se schimbă.

Dacă unul dintre noduri tocmai a intrat în V^* ($D[u] = x$, sau $D[v] = -x$), toate sumele submulţimilor care îl conţin trebuie recalculate, ceea ce se poate efectua în $O(1)$ pe submulţime, folosind sumele deja calculate pentru submulţimi mai mici.

Query Pentru a efectua Query se observă, că găsirea unui graf de tranzacţii minim este echivalentă cu partiţionarea lui V^* într-un număr maximal de submulţimi disjuncte având suma zero, adică $V^* = P_1 \cup \ldots \cup P_{max}, sums[P_i] = 0, \forall i = \overline{1, max}$ şi $P_i \cap P_j = \emptyset, \forall i, j = \overline{1, max}, i \neq j$. Motivul este, că datoriile dintr-o mulţime de sumă zero P_i se pot rezolva prin $|P_i| - 1$ tranzacţii (după Teorema 4.22, vezi şi [Pat09, Pat11b, Ver04]), deci pentru a rezolva toate datoriile este nevoie de $|V^*| - max$ tranzacţii.

Fie S^0 mulţimea submulţimilor lui V^*, având suma zero: $S^0 = \{S | S \subset V^*, sums[S] = 0\}$. Pentru a găsi partiţia maximă, folosim metoda programării dinamice.

Fie $dp[S]$ numărul maximal de submulţimi cu sumă zero în care $S \subset V^*$ poate fi partiţionat.

$$dp[S] = \begin{cases} \text{nedefinit}, & \text{dacă } sums[S] \neq 0 \\ 0, & \text{dacă } S = \emptyset \\ \max\{dp[S \setminus S'] + 1 | S' \subset S, S' \in S^0\}, & \text{altfel} \end{cases}$$

Construcţia lui dp durează cel mult $2^{|V^*|} \cdot |S^0|$ paşi.

Fiindcă viteza operaţiei QUERY depinde în mare parte de cardinalitatea mulţimii S^0, în teză se propun două euristici pentru reducerea ei, fără a compromite soluţia optimă.

RemoveNode Ştergerea nodului u cu condiţiile de mai sus este echivalentă cu găsirea mulţimii P de cardinalitate minimă, care îl conţine pe u şi poate face parte dintr-o soluţie optimă, adică $dp[V^*] = dp[V^* \setminus P] + 1$.

RemoveArc Dat fiind că ştergerea unui arc dintre două noduri este echivalentă cu adăugarea unui arc în direcţia opusă, această operaţie poate fi implementat cu uşurinţă folosind INSERTARC. Dacă valorile D a celor două noduri au acelaşi semn, înseamnă că într-un graf de tranzacţii minim nu poate apărea un arc între cele două noduri, deci nimic nu este de făcut.

Un algoritm nou pentru problema statică Se observă, că operaţia QUERY are nevoie doar de mulţimea S^0, iar pentru a construi S^0, suma tuturor submulţimilor lui V^* trebuie calculată. Deci, după procesarea arcelor din graful de împrumuturi în $\Theta(m)$ şi găsirea valorilor D, tabela de dispersie $sums$ poate fi construită în $\Theta(2^{|V^*|})$ folosind metoda programării dinamice:

$$sums[S = \{s_1, \ldots s_k\}] = \begin{cases} 0, & \text{dacă } S = \emptyset \\ D[s_1], & \text{dacă } |S| = 1 \\ sums[\{s_2, \ldots s_k\}] + D[s_1], & \text{altfel} \end{cases}$$

După construirea lui $sums$, se poate construi S^0 printr-o nouă iteraţie asupra tuturor submulţimilor lui V^* şi selectarea acelora care au suma zero în S^0. După acesta se efectuează euristicile şi se apelează QUERY. Timpul total de rulare este $\Theta(m + 2^{|V^*|} + |S^0|^2 + 2^{|V^*|} \cdot |S^0|)$.

Rezultate experimentale A fost efectuat un şir de experimente pentru a compara algoritmii noi cu cel propus în [Pat09]. S-au folosit aceleaşi 15 teste care au fost folosite când problema a fost propusă la barajul de calificare al lotului naţional de informatică. Structura acestora este descrisă în teză.

În primul experiment au fost comparaţi trei algoritmi: algoritmul static vechi, algoritmul static nou şi algoritmul dinamic. Pentru ultimul algoritm InsertArc a fost apelat pentru fiecare arc şi Query efectuat o singură dată, la sfârşit.

În al doilea experiment au fost comparate algoritmul static vechi şi algoritmul dinamic. În cazul primului algoritm întreaga soluţie a fost recalculată după adăugarea fiecărui arc, iar în cazul celui de-al doilea algoritm după fiecare InsertArc a fost apelat şi un Query. Rezultate detaliate se găsesc în teză.

Pentru o mai bună înţelegere a rezultatelor, experimente adiţionale au fost efectuate. Prima dată s-a comparat timpul de rulare ai algoritmilor statici pe grafuri aleatoare având $n = 16$ noduri şi $m = 20$ arce cu costuri din intervalul $[1, MAXVALUE)$. Pentru fiecare valoare pară $MAXVALUE \in [2, 80]$ s-au generat 1000 grafuri diferite şi ambii algoritmi au fost executaţi pe ele. După cum se vede în Figura 10 pentru valori mici ale lui $MAXVALUE$ algoritmul static vechi este mai rapid, dar începând cu $MAXVALUE = 16$ acesta devinde din ce în ce mai lent. Se observă de asemenea că algoritmul nou este mult mai robust, fiindcă timpul lui de rulare nu are fluctuaţii atât de mari ca cel al algoritmului vechi.

Pentru a înţelege mai bine detaliile interioare ale algoritmilor, s-a măsurat separat timpul petrecut în fiecare fază a acestora.

În acest experiment s-au generat 10000 de grafuri, având $n = 16$ noduri şi $m = 20$ arce şi s-a calculat timpul total petrecut în fiecare fază a algoritmilor. S-au investigat două cazuri, unul pentru care algoritmul vechi este mai rapid (Figura 11) şi unul pentru care algoritmul nou este mai eficient (Figura 12).

Timpul de rulare al algoritmului static vechi este dominat de timpul de preprocesare în ambele cazuri. Mai precis faza critică pare a fi partea de

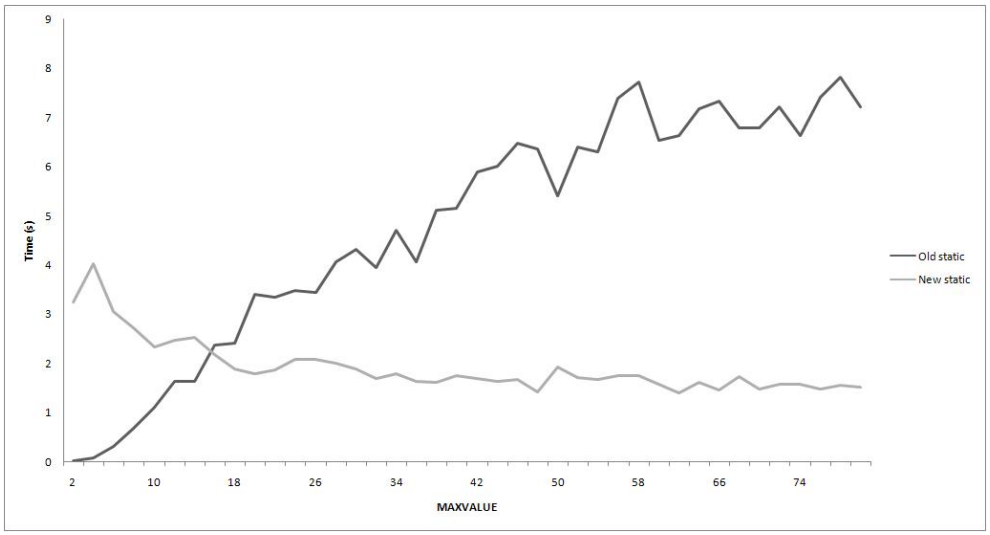

Figura 10: Timpii totali de rulare pe 1000 de grafuri aleatoare având $n = 16$ noduri, $m = 20$ arce cu costuri mai mici ca $MAXVALUE$.

alocarea memoriei. Chiar dacă ambii algoritmi folosesc $\Theta(2^{|V^*|})$ memorie, se pare că alocarea matricelor de această mărime este mult mai costisitor de timp ca şi alocarea unui vector de aceeaşi mărime.

Algoritmul nou se comportă în felul aşteptat, petrecând mult mai mult timp în faza principală şi faza euristică pentru $MAXVALUE = 10$. Motivul este că, cardinalitatea lui S^0 este în mediu de aproximativ patru ori mai mare în comparaţie cu cazul $MAXVALUE = 50$ (615.5 respectiv 153.2).

Tratarea cazurilor mari Se notează $n^* = |V^*| = |\{u|D(u) \neq 0\}|$. Algoritmii prezentaţi mai sus sunt capabili să furnizeze soluţia optimă în timp rezonabil pentru valori de mărimea 20 - 30 ale lui n^*. Poate ar fi de dorit găsirea unor soluţii "suficient de bune" pentru intrări de dimensiuni mai mari.

Autorul propune un algoritm genetic ([Hol75]) pentru rezolvarea problemei datoriilor ([PatBar11]).

Se foloseşte reformularea problemei în care se cere partiţionarea lui V^* în mulţimi disjuncte de sumă zero.

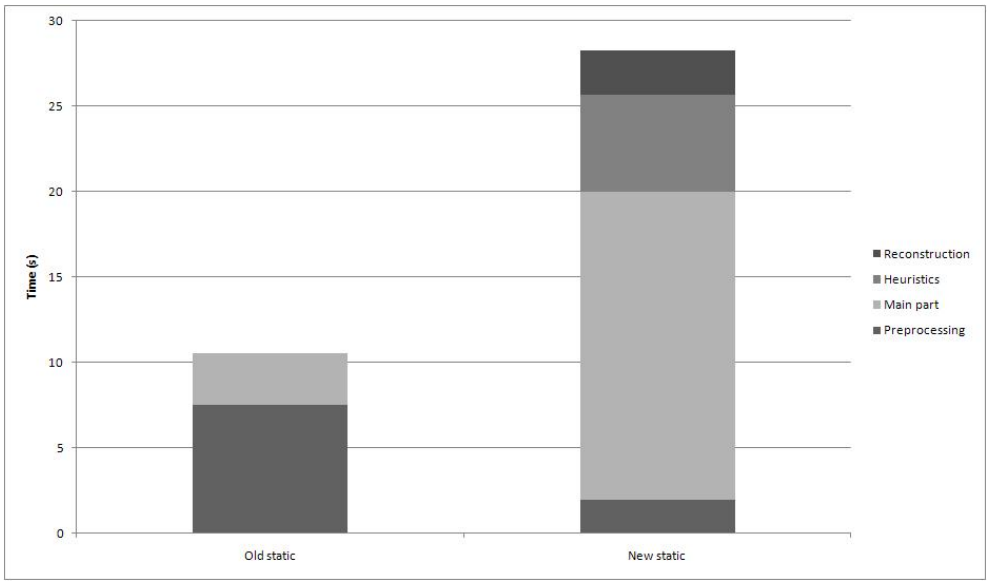

Figura 11: Timpul de rulare total al fazelor algoritmilor pe 10000 de grafuri generate aleator având $n = 16$ noduri, $m = 20$ arce cu costuri mai mici ca 10.

Reprezentare O soluţie a problemei este reprezentată de o permutaţie a valorilor D care corespund nodurilor din V^*. Deci o soluţie candidată este un vector $C = (c_1, c_2, \ldots, c_n)$, în aşa fel încât $c_i = D(u), \forall i \in \overline{1,n}$ pentru un $u \in V^*$ unic.

Atribuirea fitness-ului Pentru a evolua fitness-ul unei cromozome, se iterează peste genele acesteia în ordine crescătoare şi se menţin sumele parţiale obţinute, adică $s_i = \sum_{j=1}^{i} c_j$. Pentru fiecare $s_i = 0$ găsit se adaugă unu la valoarea de fitness.

Recombinarea Se propun noi operatori de recombinare ([PatBar11]).

Operatorul 1 Fie C_1 şi C_2 cele două cromozome şi $k \in [1, n]$ un indice aleator. Primul urmaş C_1' se obţine prin copierea primelor k gene din C_1 şi lipirea elementelor din permutare încă nefolosite, în ordinea în care apar în C_2. Al doilea urmaş C_2' se obţine în mod simetric.

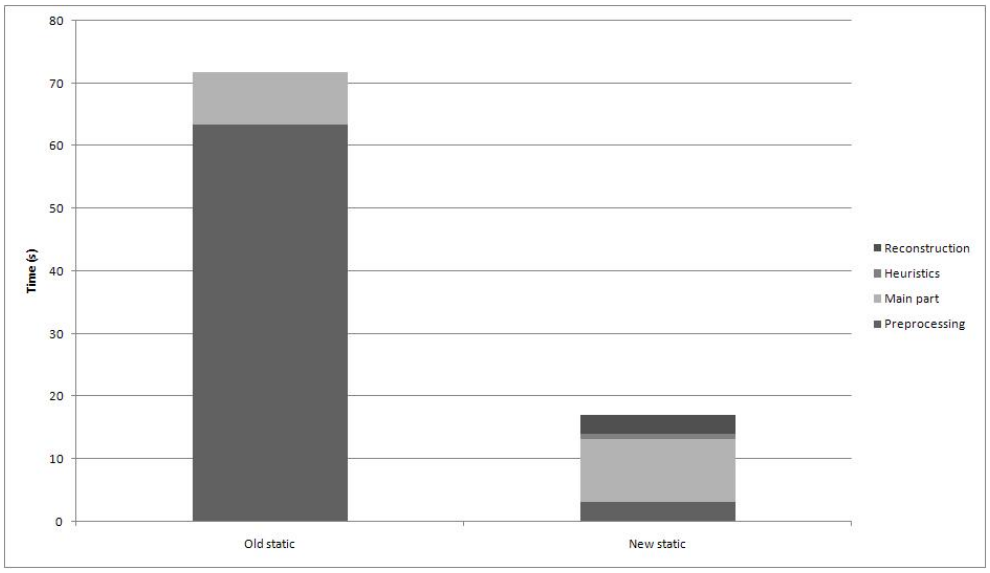

Figura 12: Timpul de rulare total al fazelor algoritmilor pe 10000 de grafuri generate aleator având $n = 16$ noduri, $m = 20$ arce cu costuri mai mici ca 50.

Operatorul 2 Problema cu Operatorul 1 este că primul urmaş moşteneşte majoritatea proprietăţilor de la C_1 şi foarte puţin de la C_2. Acest lucru nu este de dorit, fiindcă C_1 şi C_2 pot conţine submulţimi din partiţia optimă.

Un operator de recombinare mai eficient poate fi următorul. Se determină partiţia codificată de C_1 şi C_2, prin metoda descrisă la atribuirea fitness-ului. Fie acestea $C_1 = P_{1,1} \cup P_{1,2} \cup \ldots$ şi $C_2 = P_{2,1} \cup P_{2,2} \cup \ldots$. Se iniţializează $C_1' := C_1$ şi $C_2' := C_2$.

Se iterează peste fiecare $P_{1,i}$. Dacă $P_{1,i}$ este conţinut într-un $P_{2,j}$, adică $P_{1,i} \subset P_{2,j}$, se înlocuieşte $P_{2,j}$ în al doilea urmaş cu $P_{1,i} \cup (P_{2,j} \setminus P_{1,i})$. Procedura se repetă pentru C_2 în mod simetric.

Mutaţia Se propun patru operatori noi de mutaţie (Operatorii 3–6), având proprietatea, că fitness-ul cromozomei nu scade după efectuarea operaţiei ([PatBar11]).

Operatorul 1 Operatorul de inversie descris de Holland ([Hol75]) poate fi folosit fără modificări pe secvenţa dintre genele i şi j.

Operatorul 2 O variantă simplificată a Operatorului 1 se poate efectua prin schimbarea genelor i şi j din cromozomă.

Operatorii 3 şi 4 Operatorii 1 şi 2 se pot folosi pe partiţia $C = P_1 \cup P_2 \cup \ldots$ în locul reprezentării pe gene.

Operatorii 5 şi 6 Operatorii 1 şi 2 se pot folosi în interiorul unui P_k fără a scădea valoarea de fitness.

Din cauză că problema este tare NP-grea este o provocare generarea unor teste mari, pentru care se cunoaşte soluţia optimă. În teză sunt descrise patru metode pentru generarea cazurilor mari.

Tranzacţii interzise, dobânzi, reduceri Este firesc să se asume, că tranzacţiile nu sunt posibile între oricare două entităţi din diverse motive.

Intrarea acestei probleme poate fi descrisă prin două grafuri, graful de împrumuturi G şi graful de permisiuni G_P definit mai jos.

Definiţie 4.23 Un **graf de permisiuni (permission graph)** $G_P(V, A_P)$ este un graf orientat neponderat, având un arc $(u, v) \in A_P$, dacă tranzacţiile de la u la v sunt permise. □

Varianta originală a problemei corespunde unui graf de permisiuni egal cu graful complet. Se observă, că prin introducerea grafului de permisiuni, s-a obţinut o generalizare a problemei originale, deci această nouă variantă va fi tare NP-grea de asemenea. Algoritmii descrişi mai sus nu se pot modifica uşor pentru a rezolva problema generală şi se pare că găsirea unor asemenea algoritmi este o sarcină grea.

Pentru ca modelul să devină şi mai realist, graful de permisiuni poate fi unul ponderat şi se poate impune condiţia că orice sumă plătită de u la v se va înmulţi cu ponderea arcului (u, v) din graful de permisiuni. Aşadar o

pondere mai mare ca unu are semnificaţia unei reduceri obţinute de u din partea lui v, şi o pondere mai mică ca unu semnifică o dobândă. În această variantă se poate cere o soluţie care minimizează suma totală plătită de toate entităţile.

Bibliografie selectivă

[Abd97] Saïd Abdeddaïm. *On incremental computation of transitive closure and greedy alignment.* In *Combinatorial Pattern Matching, 8th Annual Symposium*, volume 1264 of *Lecture Notes in Computer Science*, 167–179. 1997.

[Abd00] Saïd Abdeddaïm. *Algorithms and experiments on transitive closure, path cover, and multiple sequence alignment.* In *Proceedings of the 2nd Algorithm Engineering and Experiments*, 157–169. 2000.

[AlbCatIta97] David Alberts, Giuseppe Cattaneo and Giuseppe F. Italiano. *An empirical study of dynamic graph algorithms.* ACM Journal of Experimental Algorithmics, volume 2, 1997.

[AlbEtAl98] David Alberts, Giuseppe Cattaneo, Giuseppe F. Italiano, Umberto Nanni and Christos D. Zaroliagis. *A software library of dynamic graph algorithms.* In *Proceedings of Algorithms and Experiments*, 129–136. 1998.

[AmaCatIta97] Giuseppe Amato, Giuseppe Cattaneo and Giuseppe F. Italiano. *Experimental analysis of dynamic minimum spanning tree algorithms.* In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, 314–323. 1997.

[Bel62] Richard Bellman. *Dynamic programming treatment of the travelling salesman problem.* Journal of the ACM, volume 9(1):61–63, 1962.

[Blu85] Norbert Blum. *On the single-operation worst-case time complexity on the disjoint set union problem.* In *2nd Symposium of Theoretical Aspects of Computer Science*, volume 182 of *lncs*, 32–38. Springer, 1985.

[Dav85] Lawrence Davis. *Applying adaptive algorithms to epistatic domains.* In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, 162–164. Morgan Kaufmann, 1985.

[EppGalIta93] David Eppstein, Zvi Galil and Giuseppe F. Italiano. *Improved sparsification.* Technical Report 93-20, Department of Information and Computer Science, University of California at Irvine, 1993.

[EppEtAl92] David Eppstein, Zvi Galil, Giuseppe F. Italiano and Amnon Nissenzweig. *Sparsification - A technique for speeding up dynamic graph algorithms (extended abstract).* In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, 60–69. IEEE Computer Society Press, 1992.

[EppEtAl97] David Eppstein, Zvi Galil, Giuseppe F. Italiano and Amnon Nissenzweig. *Sparsification - a technique for speeding up dynamic graph algorithms.* Journal of the ACM, volume 44(5):669–696, 1997.

[Fre83] Greg N. Frederickson. *Data structures for on-line updating of minimum spanning trees.* In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, 252–257. ACM Press, 1983.

[Fre85] Greg N. Frederickson. *Data structures for on-line updating of minimum spanning trees, with applications.* SIAM Journal on Computing, volume 14(4):781–798, 1985.

[Fre97] Greg N. Frederickson. *Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees.* SIAM Journal on Computing, volume 26(2):484–538, 1997.

[GolLin85] David E. Goldberg and Robert Lingle, Jr. *Alleles, loci, and the traveling salesman problem.* In *Proceedings of the 1st International Conference on Genetic Algorithms and their Applications*, 154–159. Lawrence Erlbaum Associates, 1985.

[Gor90] Martina Gorges-Schleuter. *Genetic algorithms and population structure - A massively parallel algorithm.* Ph.D. thesis, University of Dortmund, 1990.

[HarGup97] Frank Harary and Gopal Gupta. *Dynamic graph models.* Mathematical and Computer Modelling, volume 25(7):79–87, 1997.

[HelKar62] Michael Held and Richard M. Karp. *A dynamic programming approach to sequencing problems.* Journal of the Society for Industrial and Applied Mathematics, volume 10(1):196–210, 1962.

[HenKin99] Monika R. Henzinger and Valerie King. *Randomized fully dynamic graph algorithms with polylogarithmic time per operation.* Journal of the ACM, volume 46(4):502–516, 1999.

[Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, 1975.

[IonPat08] Klára Ionescu and Csaba G. Pătcaş. *Improving the performance of algorithms using the principles of binary search.* Technical Review, volume 43(3):7–14, 2008.

[KroZar08] Ioannis Krommidas and Christos D. Zaroliagis. *An experimental study of algorithms for fully dynamic transitive closure.* ACM Journal of Experimental Algorithmics, volume 12, 2008.

[OliSmiHol87] I. Oliver, D. Smith and J. Holland. *A study of permutation crossover operators on the traveling salesman problem.* In *Proceedings of the Second International Conference on Genetic Algorithms*, 224–230. Lawrence Erlbaum Associates, 1987.

[Pat09] Csaba G. Pătcaş. *On the debts' clearing problem.* Studia Universitatis Babeş-Bolyai Series Informatica, volume 54(2):109–120, 2009.

[Pat11] Csaba G. Pătcaş. *The debts' clearing problem: a new approach.* Acta Universitatis Sapientiae Informatica (accepted), 2011.

[Pat11b] Csaba G. Pătcaş. *The debts' clearing problem's relation with complexity classes.* Studia Universitatis Babeş-Bolyai Series Informatica (accepted), 2011.

[PatBar11] Csaba G. Pătcaş and Attila Bartha. *Evolutionary solving of the debts' clearing problem.* (submitted), 2011.

[PatIon08] Csaba G. Pătcaş and Klára Ionescu. *Algorithmics of the knapsack type tasks.* Teaching Mathematics and Computer Science, volume 6(INFODIDACT):37–71, 2008.

[Sys91] Gilbert Syswerda. *Schedule optimization using genetic algorithms.* In *Handbook of Genetic Algorithms*, 332–349. Van Nostrand Reingold, 1991.

[Tho99] Mikkel Thorup. *Decremental dynamic connectivity.* Journal of Algorithms, volume 33(2):229–243, 1999.

[Tho00] Mikkel Thorup. *Near-optimal fully-dynamic graph connectivity.* In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, 343–350. ACM Press, 2000.

[Ver04] Tom Verhoeff. *Settling multiple debts efficiently: An invitation to computing science.* Informatics in Education, volume 3(1):105–126, 2004.

[WhiStaFuq89] Darrell Whitley, Timothy Starkwater and D'Ann Fuquay. *Scheduling problems and traveling salesmen: The genetic edge recombination operator.* In *Proceedings of the Third International Conference on Genetic Algorithms*, 133–140. Morgan Kaufmann Publishers, 1989.

[Zar02] Christos Zaroliagis. *Implementations and experimental studies of dynamic graph algorithms.* In *Experimental Algorithmics*, volume 2547, 229–278. 2002.