A-tokovo kritické grafy – Letný report

Študent: Gergő Varga

1. Úvod a Optimalizácia SAT riešenia

Počas letného semestra bolo hlavným cieľom práce navrhnúť efektívnejší algoritmus na overovanie kritických grafov. Pôvodná implementácia využívala backtrackingový algoritmus, ktorý poskytoval dobré výsledky pri menších grafoch, avšak jeho výpočtová náročnosť rástla exponenciálne so zvyšujúcou sa veľkosťou vstupu.

Na účely testovania boli vytvorené nové testovacie množiny obsahujúce grafy s viac ako 100 vrcholmi. Pri týchto inštanciách dosahoval pôvodný algoritmus časy približne 14 minút, čo predstavovalo limit jeho praktického využitia. Z tohto dôvodu bolo potrebné navrhnúť efektívnejšie riešenie založené na SAT solveri.

Prvé implementácie SAT riešenia nepriniesli očakávané zrýchlenie. Hlavným dôvodom bolo využívanie externého programu MiniSAT, ktorý bol pri každom teste spúšťaný ako samostatný proces. Každé spustenie vyžadovalo vytvorenie DIMACS súboru, jeho zápis na disk, spustenie externého programu a následné načítanie výsledku. Režijné náklady spojené s týmito vstupno-výstupnými operáciami výrazne znižovali celkový výkon algoritmu.

Prechod na knižnicu Sat4j

Tento problém bol odstránený použitím knižnice Sat4j, ktorá obsahuje implementáciu SAT solvera priamo v prostredí Java. Formulácia je vytvorená priamo v pamäti programu a jednotlivé klauzuly sú odovzdávané solveru prostredníctvom jeho API. Tento prechod priniesol dve zásadné výhody:

1. • Eliminácia I/O operácií: Kompletná komunikácia s riešiteľom prebieha priamo v operačnej pamäti JVM, vďaka čomu algoritmus nie je brzdený diskovými operáciami.

2. • Absolútna nezávislosť od operačného systému: Kým pôvodné riešenie vyžadovalo skompilovanú binárnu formu MiniSATu špecifickú pre Linux, nová implementácia funguje natívne na akejkoľvek platforme (Windows, Linux, macOS) bez nutnosti konfigurácie externého prostredia.

Samotná výmena solvera priniesla výrazné zrýchlenie. Čas riešenia grafov s viac ako 100 vrcholmi sa skrátil približne zo 14 minút na približne jednu minútu.

Následne bola prepracovaná aj samotná SAT formulácia. Namiesto jednoduchého modelovania problému bola navrhnutá nová formulácia založená na postupnom počítaní súčtov toku modulo (k) vo vrcholoch grafu. Každá hrana je reprezentovaná množinou booleovských premenných určujúcich hodnotu toku na danej hrane. Pomocou pomocných premenných sa postupne počíta výsledný tok vo vrcholoch, pričom na konci formulácie je vynútené splnenie podmienky nulového súčtu modulo (k).

Významnou optimalizáciou bolo aj využitie mechanizmu assumptions, ktorý poskytuje Sat4j. Základná CNF formulácia sa vytvorí iba raz a pri jednotlivých testoch sa mení iba množina predpokladov (assumptions). Nie je preto potrebné opakovane generovať celú SAT formuláciu ani vytvárať nový problém pre každé overenie.

Zdrojový odkaz na použitú knižnicu:
https://www.sat4j.org/index.php

https://content.iospress.com/articles/journal-on-satisfiability-boolean-modeling-and-computation/sat190075

Viacvláknové spracovanie

Ďalšie zrýchlenie bolo dosiahnuté využitím viacvláknového spracovania. Jednotlivé kontroly dvojíc vrcholov sú navzájom nezávislé, a preto ich možno vykonávať paralelne. Každé vlákno pracuje s vlastnou inštanciou SAT solvera a overuje inú časť priestoru riešení. Moderné procesory obsahujú viacero výpočtových jadier, takže paralelné spracovanie umožňuje efektívne využiť dostupný hardvér. Výpočet sa navyše ukončí okamžite po nájdení prvého protipríkladu, čím sa eliminuje vykonávanie zbytočných testov.

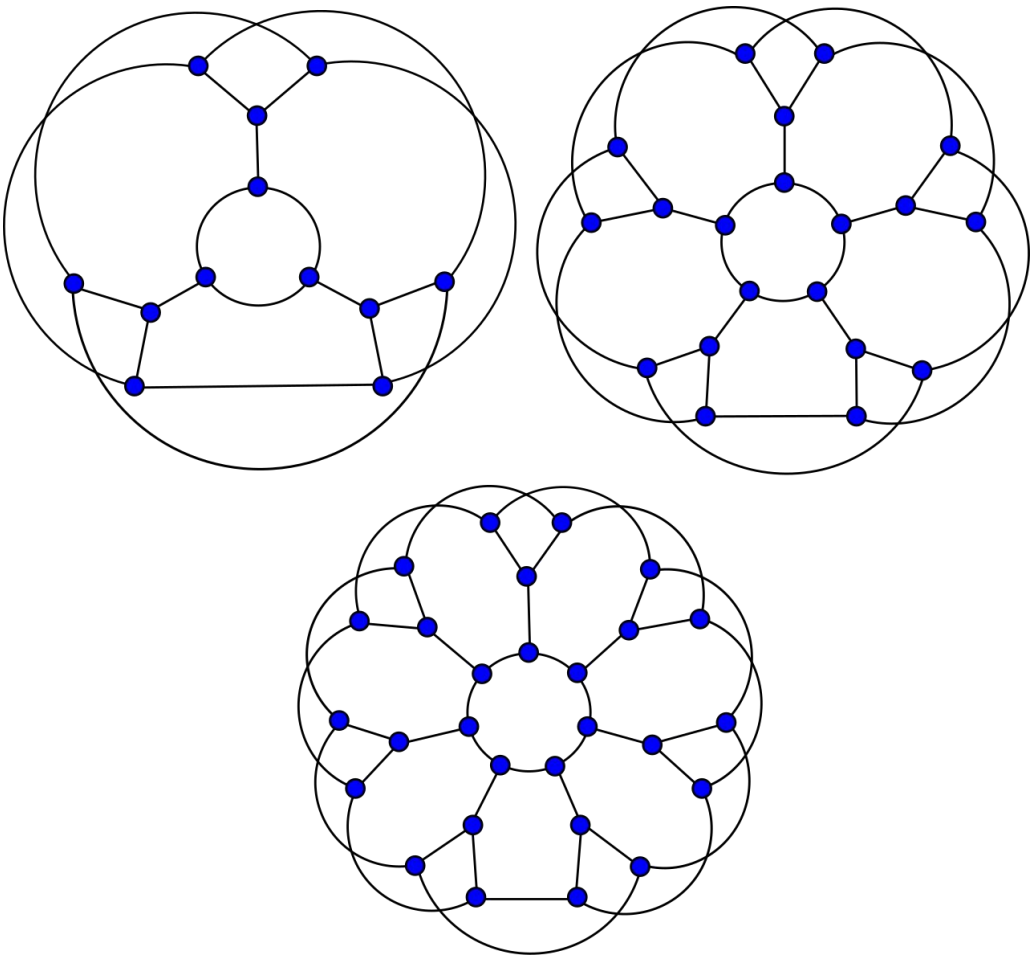
2.Analýza ILP riešenia a porovnanie

Okrem SAT solverov bolo experimentálne testované aj riešenie založené na celočíselnom lineárnom programovaní (Integer Linear Programming – ILP). Predpokladalo sa, že moderné MILP solvery budú schopné efektívne riešiť aj túto formuláciu problému.
Experimentálne výsledky však tieto očakávania nepotvrdili. Pri testovacích grafoch s viac ako 100 vrcholmi dosahovali ILP solvery časy presahujúce 10 minút, čo bolo výrazne horšie než optimalizované SAT riešenie.
Zaujímavým pozorovaním bolo aj správanie pri jednoduchých testovacích prípadoch typu flower graph. Aj pri týchto grafoch dosahoval ILP solver pomerne vysoké výpočtové časy. Ukázalo sa teda, že napriek všeobecnosti ILP prístupu nie je pre riešený problém rovnako vhodný ako SAT formulácia.

3. Problémová topológia: Flower grafy (Kvetové grafy)

Počas experimentov sa ukázalo, že medzi najnáročnejšie testovacie prípady patria rôzne varianty flower grafov. (Kvetový graf, resp. Flower snark v teórii grafov). Veľké kvetové grafy spôsobovali hlavné výkonnostné problémy pri optimalizácii.

Flower graf je rodina symetrických kubických grafov (každý vrchol má stupeň 3), ktoré obsahujú centrálne jadro, z ktorého do strán vychádzajú cyklické štruktúry pripomínajúce okvetné lístky. Tieto grafy majú vysoký index obvodovej dĺžky a vykazujú silné kombinatorické vlastnosti (často ide o tzv. snarky – grafy, ktoré nie sú hranovo 3-zafarbiteľné a nemajú triviálne toky).



4. Princíp fungovania SAT riešenia

Navrhnutý algoritmus transformuje kombinatorický problém existencie nikde-nulového A-toku na problém splniteľnosti booleovských formúl (SAT). Celý proces kódovania a vyhodnocovania prebieha v troch základných fázach:

1. Binarizácia hodnôt hrán (Premenné “vars”): Keďže SAT solver nepracuje s celými číslami, pre každú hranu *e* sa vytvára sada *k-1* logických premenných *vars[e][i]* prislúchajúcich povoleným hodnotám toku {1, 2, … , k-1}. Absencia hodnoty 0 priamo v definícii premenných garantuje splnenie podmienky nikde-nulovosti. Pomocou podmienky Exactly-One (metóda “exactlyOne”) sa následne vynucuje, aby bol pre každú hranu aktívny práve jeden veľkostný variant.

2. Simulácia zachovania toku (Akumulátory “sumVars”): Kirchhoffov zákon (nulový súčet tokov vo vrchole modulo *k*) je implementovaný prostredníctvom priebežných sčítacích premenných “sumVars[i][sum]”, ktoré fungujú ako diskrétny akumulátor. Algoritmus sekvenčne prechádza incidentné hrany daného vrcholu a generuje implikačné klauzuly prechodu (napr. ak bol predchádzajúci súčet s_1 a na aktuálnej hrane je hodnota *v*, nový súčet musí byť s_2). Na záver sa pomocou kontrolnej premennej “vertexForceZeroVars[v]” uzamkne podmienka, že finálny stav akumulátora musí byť rovný nule.

3. Paralelné overovanie kriticizmu (Assumptions): V riadiacej triede “IsCritical” sa kompletná báza klauzúl načíta do solvera iba raz, čo výrazne šetrí režijné náklady na pamäť. Samotná simulácia topologických zmien grafu (zlúčenie dvoch vrcholov *u* a *v*) sa riadi dynamicky prostredníctvom prepínania logických predpokladov (assumptions). Negáciou príslušných riadiacich literálov (“-bVars[u]”, “-bVars[v]”) sa pre danú dvojicu uvoľní podmienka nulového balansu, čím sa exaktne nasimuluje ich prepojenie.

Vďaka využitiu paralelného prúdu “IntStream.parallel()” testovanie dvojíc prebieha na viacerých jadrách procesora súčasne, pričom prvá nájdená nesplniteľnosť okamžite skratuje celý výpočet a potvrdzuje, že graf nie je kritický.

5. Výsledky projektu a záver

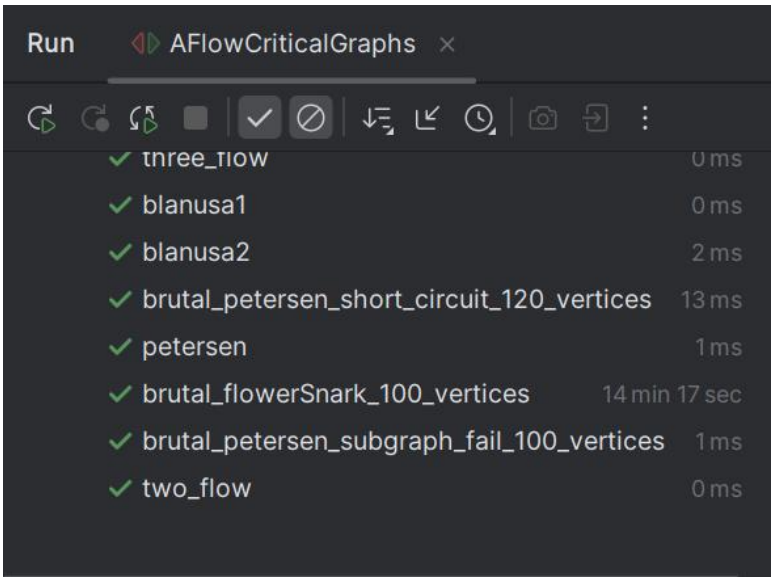
Z nameraných dát jednoznačne vyplýva, že pre malé inštancie grafov je klasický backtracking pomerne efektívny. Vďaka použitým heuristikám bol v niektorých prípadoch dokonca rýchlejší než SAT riešenie, keďže nebolo potrebné vytvárať SAT formuláciu ani inicializovať solver. Pre rozsiahle grafy je však jeho použitie neakceptovateľné. Pri testovacích grafoch s viac ako 100 vrcholmi dosahoval čas približne 14 minút.

Optimalizované SAT riešenie postavené na knižnici Sat4j s podporou paralelizácie sa ukázalo ako jediné konzistentné a optimálne riešenie pre zložité grafy. Okrem samotného zrýchlenia poskytovalo stabilné výsledky aj pri väčších testovacích množinách.

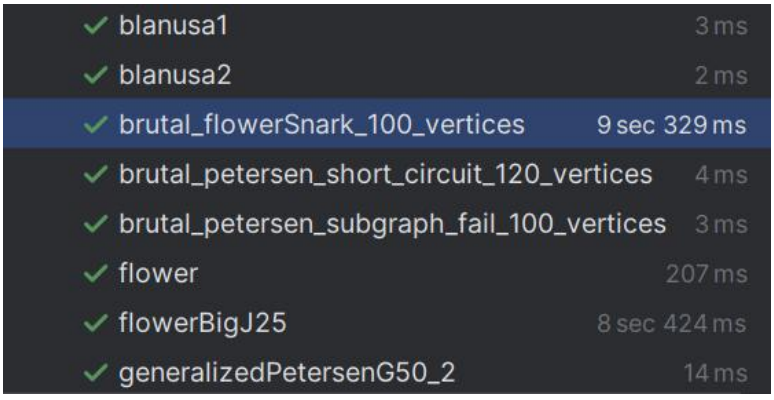
ILP riešenie bolo funkčné, avšak vo všetkých uskutočnených experimentoch bolo pomalšie než optimalizované SAT riešenie a vo viacerých prípadoch dokonca pomalšie než pôvodný backtracking.

Záverom možno konštatovať, že počas letného semestra sa podarilo výrazne zrýchliť riešenie problému kritických grafov. Dosiahnuté výsledky potvrdzujú, že SAT predstavuje vhodný prístup na riešenie daného problému.

Prílohy:



Obr.1 - Backtrack algoritmus



Obr.2 - Sat + multithreading (6-jadrový AMD procesor)