Válaszok **Prof. Dr. Hajdu András tszv. egyetemi tanár** kérdéseire az MTA doktori értekezésemhez

Tisztelt Tanár Úr!

Köszönöm a bírálatot az MTA doktori értekezésemhez. Az általános kritikai megjegyzéseket a jövőbeli kutatásaim során igyekszem figyelembe venni. A bírálatban foglalt kérdésekre az alábbiakban válaszolok. A könnyebb követhetőség érdekében a kérdéseket sorszámmal azonosítom.

HA1: Milyen méretű és típusú rendszereken végezték a gráfmentes programszeletelési algoritmusok összehasonlító teljesítmény méréseit, és ezek mennyire reprezentálják a jelenlegi ipari méretű kódokat?

A legelső kísérleteket kisméretű C programrészleteken végeztük (a 1999-es publikációnk néhány száz soros kódra hivatkozik [14]). Ezután már valódi, 1-21 ezer soros C programokkal kísérleteztünk a [1,2] cikkekben. Ezek jellemzően olyan programok, amelyeket abban az időszakban (2000-es évek eleje) gyakran használtak a hasonló empirikus kutatásokban. A módszerünk implementációját Java programokra hasonló méretű, de ugyancsak valódi programokon teszteltük (a [28]-as publikációban 1-17 ezer soros kódok szerepelnek). A korabeli empirikus kísérletek e rendszereket "közepes" méretűnek tekintik, de ma már inkább kisméretűnek számítanak. Ugyanakkor, a mai számítási kapacitás mellett ugyanazon nem optimalizált implementáció már sok százezer soros programokat is képes lenne feldolgozni. Valamint, a megvalósítás további optimalizálásával valódi, ipari méretű projektek feldolgozása is lehetővé válna az algoritmusok kedvező elméleti bonyolultsági tulajdonságainak köszönhetően.

Véleményem szerint a valódi kihívást nem is az elemzett szoftver mérete, hanem a modern szoftverek heterogén technológiai jellege, valamint az új programozási paradigmák jelentik.

HA2: Vizsgálták-e a gráfmentes programszeletelési módszerek skálázhatóságát többmagos vagy elosztott feldolgozási környezetben?

Ilyen kísérleteket nem végeztünk, de elvi szinten az algoritmusok jól adaptálhatók többszálú és elosztott környezetre. Mivel a programszeletek számítása a végrehajtási lépések sorozatos feldolgozásán alapul, ha egy lépésnél több függőségi halmazt kell kiszámolni, azok párhuzamosíthatók (ezek függetlenek egymástól az adott lépésnél). Kapcsolódó kérdés, hogy ha maga az elemzendő program többszálú, akkor milyen adaptációkra lenne szükség. Az egyes szálak végrehajtásával egyidejűleg a függőségi halmazok is párhuzamosan számíthatók, viszont ami megoldandó, a szálak által közösen használt memóriaterületekhez kapcsolódó függőségek, de itt sem látok elvi akadályt az adaptációra (amennyiben maga az elemzett program is helyesen kezeli a konkurenciát).

HA3: Hogyan teljesít a DFC-alapú megközelítés olyan rendszerekben, ahol a végrehajtás erősen párhuzamos és nem determinisztikus?

Ez a módszer az eljáráshívások sorrendiségén és távolságán alapul, amit konkrét programvégrehajtások feldolgozásával számítunk. Az eredeti módszer egyszálú végrehajtást feltételez, és a csatolási kapcsolatokat egy-egy szálra lehet meghatározni. Az így kapott eredményeket ugyanakkor lehetne aggregálni az egész rendszerre valamilyen súlyozott átlag megoldással. A konkurens módon végrehajtott függvények minden bizonnyal zajt eredményeznének a számításban, így azokat valamilyen módon kezelni kellene.

Az alap heurisztika általánosítható más paradigmákra és környezetekre, akár párhuzamos végrehajtásra is. A módszernek az a kiindulási gondolata, hogy amikor két számítás (függvényhívás) időben, azaz a végrehajtási szekvencia során, közel van egymáshoz, akkor nagyobb a valószínűsége annak, hogy kapcsolódnak egymáshoz. Párhuzamos számítás esetén ennek a feltételezésnek nincs alapja, így más megközelítést kellene alkalmazni. Egyrészt, egy többszálú program esetében a szálak létrehozásának körülményeit lehetne vizsgálni (pl. melyik

eljárás hozza létre melyik szálat), és ez alapján meghatározni az egyes szálak közötti csatolási kapcsolatokat. Másrészt, mivel a különböző párhuzamos számítási egységek jellemzően közös adatterületeken keresztül kommunikálnak egymással, a módszert ki lehetne terjeszteni úgy, hogy a közös adatelérések között határozunk meg egy távolsági mértéket, és azzal számoljuk tovább a csatolást. Például, az adat írás és olvasás műveleteket végző eljárások közötti csatolást az adott műveletek sorrendisége és időben egymástól való távolsága alapján határozzuk meg.

HA4: A függőségi klaszterek linchpin-azonosítási eredményeit validálta-e emberi szakértők bevonásával, és ha igen, milyen egyezési arányt ért el?

Történt ehhez hasonló vizsgálat, de a másik irányból megközelítve: szakértői eredményből kiindulva kerestük meg a legjobban közelítő heurisztikus megoldást ([11] és [10] publikációk). Első lépésként a közepes méretű kísérleti programjainkon (29 darab valódi program, melyek sorainak száma néhány ezertől 200 ezerig terjed) megkerestük a legvalószínűbb linchpin elemeket nyers erővel, minden szóba jöhető elem (függvény) kézi és metrika alapú vizsgálatával. A kézi vizsgálatot a kapcsolódó publikációk szerzőivel közösen végeztük, egymás eredményeit validálva. Ehhez a programok klaszterezettségének vizualizációját használtuk a linchpin jelölt elemek eltávolítása előtt és után.

Emellett kiszámoltuk minden programra a klaszterezettségi metrikák változása szerinti legvalószínűbb elemeket is, amit a kézi vizsgálattal is összevetettünk, és magas hasonlóságot kaptunk (0,9-1 Mean Average Precision értékek). Így, a továbbiakban olyan heurisztikus előrejelző metrikákat kerestünk, amelyek jól korrelálnak a klaszterezettségi metrika különbségekkel. Ez azért lényeges eredmény, mert a nyers erő alapú kézi keresés és a klaszterezettségi metrika alapú megközelítések nem használhatók a gyakorlatban. Azt kaptuk, hogy a függvényekből kimenő hívások száma (NOI metrika) nagyon jó heurisztikus közelítés a kézi eredményekhez és metrikákhoz képest: magas klaszterezettség esetén 92% az egyezés.

HA5: Vizsgálta-e a SEA és a szelet-alapú függőségi klaszterek közötti eltérések hatását konkrét hibajavítási forgatókönyvekben?

Ez a kérdés nem volt a kutatási céljaink között, de két észrevétel is kapcsolódhat a kérdéshez. Egyrészt, a tézisben hivatkozott kutatások és más eredmények alapján az a mintázat rajzolódik ki, hogy a programszeletek és a SEA alapú függőségi halmazok közötti különbség annál kisebb, minél nagyobb az elemzett program. Másrészt, a magasabb klaszterezettségű programok esetében kisebbek a SEA és a szelet-alapú függőségi klaszterek közötti eltérések. Így, várhatóan a nemtriviális, magasan klaszterezett programok esetében lenne a legkisebb hatása az eltéréseknek konkrét hibajavítási forgatókönyvekben.

HA6: Mely vizsgált kódlefedettség mérő eszközök alkalmazhatók nagy, többnyelvű kódalapokra, és a módszer kiterjeszthető-e ezekre?

Kutatásunkban 24 ingyenes és kereskedelmi terméket azonosítottunk be, amelyek alkalmasak lehetnek Java nyelvű rendszerek kódlefedettségének mérésére. A részletes kiértékelésünket végül öt eszközön végeztük el.

Egyrészt, azon eszközök, amelyek forráskód instrumentálást alkalmaznak kizárhatók abból, hogy más nyelvű rendszereket elemezhessenek, de a többi, bytecode alapú eszköz elvileg minden olyan rendszerre és komponensre alkalmazható, amely bytecode-ra fordítható és JVM-en végrehajtható (pl. Kotlin, Scala, Groovy, JRuby). A JVM technológiától független, illetve vegyes kódalapok csak olyan eszközökkel elemezhetők, amelyek önmagukban többnyelvűek, vagy különböző nyelvspecifikus eszközök eredményeit képesek egységesen kezelni és prezentálni. A részletesen vizsgált eszközök közül a Semantic Designs cég Test Coverage Tools eszközcsaládja képes többnyelvű rendszerek elemzésére (Java mellett pl. C/C++, C#, PHP). További hasonló eszközök a piacon egyebek mellett a TestWorks és a SonarQube.

A kutatásunkban alkalmazott eszköz kiértékelési módszertan alkalmas lehet más nyelvű és akár többnyelvű eszközök szisztematikus vizsgálatára, a megfelelő adaptációkkal. A kapcsolódó publikációinkban részletesen ismertettük a több fázisból álló kiértékelési módszertanunkat.

HA7: Mennyire érzékenyek az eljárás hívási láncokat és az eljárás hívási gyakoriságokat használó spektrum-alapú hibalokalizációs módszerek a tesztkészlet minőségére és lefedettségére, és hogyan változik a teljesítmény gyengébb lefedettség mellett?

A tesztkészlet minősége talán a legfontosabb tényező, amely alapvetően meghatározza a spektrum-alapú hibalokalizációs módszerek teljesítményét, és ez érvényes a klasszikus módszerekre, de az általunk javasolt továbbfejlesztett algoritmusokra is. Viszont, a tesztkészlet minőségét sokféleképpen lehet vizsgálni, ugyanakkor arra vonatkozóan viszonylag kevés kutatási eredmény létezik, hogy melyik minőségi szempont milyen mértékben és milyen körülmények között befolyásolja a módszerek hatékonyságát.

Az általános kódlefedettség egy tesztkészlet vonatkozásában egy nagyon egyszerű aggregált mérőszám, amiben a tesztesetek lefedettségei összesítve szerepelnek, így önmagában nehezen látható az, hogy milyen mértékben befolyásolja a spektrum-alapú módszerek hatékonyságát. Triviális, hogy azok a kódelemek, amelyeket egy teszteset sem fed le, kiesnek az algoritmusok látóköréből (az általunk javasolt módszerek esetében is), gyakorlatilag a spektrum mátrix ezen elemekre vonatkozó oszlopai törölhetők. Így, az ezen elemekben lévő esetleges hibák nem találhatók meg, de a többi elemre relatíve több “teszteset jut”, és a lokalizáció sikeresebb lehet, amennyiben ezekben találhatók a hibák.

Ha csak a bináris spektrum mátrix struktúráját, a lefedettségi információ “mintázatait” tekintjük, egy másik fontos jellemző kerül előtérbe, ami nagyobb mértékben befolyásolja a hibalokalizáció sikerességét. Nevezetesen, az egyes kódelemekre vonatkozó lefedettségi vektorok diverzitása. Ha sok, azonos lefedettségi mintázatú elem található a spektrumban, akkor (mivel ezek nem különböztethetők meg egymástól) romlik a hatékonyság. Az ideális spektrumban kevés azonos lefedettségi vektor van. Erre vonatkozólag léteznek vizsgálatok, amik igazolják, hogy a diverzitás valamilyen metrikával kifejezve jól előre jelezheti a hibalokalizáció sikerességét átlagos esetben (*Horváth et al.: Test Suite Evaluation using Code Coverage Based Metrics. In Proceedings of the 14th Symposium on Programming Languages and Software Tools (SPLST 2015), pages 46-60, 2015*; *Abreu et al.: On the accuracy of spectrum-based fault localization. In Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007, pp. 89–98*).

Ez a tényező az általunk javasolt módszerekre is hatással van, de itt további szempontok is előtérbe kerülnek. A hívási láncok esetében azok a kedvező tesztkészletek, amelyeknél a diverzitás nem csak a lefedettségben mutatkozik meg, hanem abban is, hogy egy-egy teszteset mennyire különböző eljárásokat és eljárás sorozatokat hív meg, másszóval hívási struktúra szerint redundánsak-e a tesztek, vagy kellően különbözőek. A hívási gyakoriságok esetében nagyon hasonló a kérdés: mennyire különböző hívási kontextusokból juthatunk el egy-egy kódelemhez: ha több helyről, akkor az javítja a lokalizációs képességeket.

Végül, legújabb kutatásunkban vizsgáltuk a tesztesetek “egységteszt jellegét”, vagyis, hogy a tesztesetek mennyire izoláltan tesztelnek csak egy-egy eljárást a többi elszigetelésével vagy sem (*Szatmári et al.: Influence of Pure and Unit-Like Tests on SBFL Effectiveness: An Empirical Study. In Proceedings of the 2025 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pages 392-399, 2025*). Ideális esetben, amikor minden teszteset ilyen jellegű, akkor a hibalokalizáció szinte triviális. Viszont, érdekes kérdés, hogy egy tényleges tesztkészlet esetében milyen arányban kell, hogy legyenek az ilyen ideális egységtesztek a sikeres lokalizáláshoz.

HA8: Mennyiben alkalmazható a dinamikus programszeletelés alapú hibalokalizáció elméleti kerete nem procedurális vagy funkcionális nyelvekre, ahol a függőségi struktúra lényegesen eltérő?

A dinamikus programszeletelés alapú hibalokalizáció a két alapvető technika ötvözése. Ezért, ahhoz, hogy a módszer használható legyen nem procedurális nyelvekre, mindkét alap technikának meg kell találni az adaptálási módját, továbbá mindezt úgy, hogy kompatibilis maradjon az elméleti kerettel. Konkrétan, szükséges egyrészt a programelemeknek és teszteseteknek megfelelő nyelvi konstrukciók beazonosítása, valamint a programszeletet meghatározó “számítást befolyásoló” reláció definiálása az adott paradigmában. Ez jelentheti például a függvényeket és a kiértékelések közötti kapcsolatokat funkcionális nyelvekben. Ezen kívül, a gyanússági értékeket kiszámoló formulákat is adaptálni kell.

A programszeletelési algoritmusokat már régebb óta alkalmazzák a hibakeresésben, jelentős mennyiségű szakirodalom létezik a különböző algoritmusok adaptálásáról az egyes programozási paradigmákhoz. Ugyanez a spektrum-alapú módszerekről már kevésbé állapítható meg, itt zömében a procedurális nyelvekhez készültek algoritmusok, néhány kivételtől eltekintve. Így, kombinált módszer kidolgozására más paradigmákhoz vagy konkrét programozási nyelvekhez jelentős további kutatást igényel.

Szeged, 2026. január 19.

Dr. Beszédes Árpád, habilitált egyetemi docens