**פרוייקט סופי בקורס ישומי בינה מלאכותית לפתירת תקלות – דו"ח תוצאות**

1) מטרת הפרוייקט:

מטרת הפרוייקט היא לבדוק כיצד תלות בין קומפונטות משפיעה על אלגוריתם בארינל, והאם יש חשיבות לסדר הקומפוננטות.

2) האלגוריתם:

1. נריץ JAVA\_TRACER על מנת לגלות באיזה קומפוננטות הטסטים עברו.

2. ניקח את הפלט משלב (1) ונשנה אותו בצורה הבאה:

2.1. נשמור את הקומפוננטות כבודדים

2.2. נחבר ביחד כל שתי קומפוננטות צמודות בטסט לזוג

2.3. נחבר ביחד כל שלוש קומפוננטות צמודות בטסט לשלשה

2.4. נאחד את (2.1) עם (2.2) ו (2.3) לקבוצה אחת ונסמנה ב – A.

3. ניצור קובץ לפי הקלט של ברינל:

3.1. שמות כל הקומפוננטות (כאן מתבצעת גם בדיקת כפילות – אם יש שתי קומפוננטות שמופיעות כזוג/שלשה במקומות שונים, לפי הסימון כאן נדע שמדובר באותם הקומפוננטות).

3.2. הסתברות – ניתן עדיפות שווה לכל הקומפוננטות (0.1)

3.3. ניצור את רשימת כל הטסטים

3.4. ניצור רשימה של טסטים שעברו או לא לפי השמות שלהם

3.5. ניצור את רשימת כל הטסטים ועבור כל אחד מהם נרשום באיזה קומפוננטות הוא עבר (הבודדים, הזוגות, והשלשות) ונוסיף גם עבור כל טסט אם הוא עבר או לא.

4. נריץ על הקובץ שנוצר בסעיף את (3) את אלגוריתם ברינל המקורי.

3) הסבר על הניסויים:

יצרנו קובץ קלט לבארינל עבור 4 פרוייקטי open source. לכל אחד מהפרוייקטים הנ"ל יצרנו קובץ "רגיל" עבור האלגוריתם המקורי, וכמו כן קובץ נוסף עבור האלגוריתם שלנו (כלומר, קובץ שמשקף תלות בין קומפוננטות)

4) התוצאות שקיבלנו:

השוונו לפי הערכים:

['entropy', 'num\_bugs', 'num\_comps', 'num\_distinct\_traces', 'num\_failed\_comps', 'num\_failed\_tests', 'num\_tests', 'only\_failed\_comps', 'only\_passed\_comps', 'precision', 'recall', 'top\_k', 'wasted']

\*שימו לב כי בתוצאות הסדר הוא משמאל לימין (הפוך מהטבלה לעיל).

**Tika**:

Regular algorithm:

[6.4877645214631929, 1, 4391, 71, 3764, 23, 72, 945, 627, 0.0, 0.0, inf, inf]

Our algorithm:

[7.4993140315402878, 1, 34325, 71, 25487, 23, 72, 12467, 8838, 0.0, 0.0, inf, inf]

**Reef:**

all tests in this project passed.

**Accumulo:**

Regular algorithm:

[0.0, 1, 2862, 464, 2169, 242, 466, 838, 693, 0, 0, inf, inf]

Our algorithm:

[0.0, 1, 21379, 464, 13879, 242, 466, 7486, 7500, 0, 0, inf, inf]

**Ant:**

Regular algorithm:

[0.0, 1, 2148, 639, 1853, 1064, 1263, 1233, 295, 0, 0, inf, inf]

Our algorithm:

[0.0, 1, 13905, 646, 11632, 1064, 1263, 9238, 2273, 0, 0, inf, inf]

5) סיכום:

קל לראות, כי אכן יש הבדל בתוצאות המתקבלות מהחישוב הנוסף, על הקלט לאלגוריתם בארינל, לפני הרצתו.

**נספחים:**

1) איך מריצים את הקוד ומשחזרים את הניסויים:

1. הרץ את פרוייקט JAVA\_TRACER

2. התאם את מיקום קבצי הפלט של 1 (תיקיית DebuggerTests) לתיקיות בקוד הפרוייקט (בקובץ – makeTableFromTestsTraces).

\*אם רוצים ליצור קבצים עבור האלגוריתם המקורי יש להוסיף את המיקום בקובץ makeTableFromTestsTracesOnlyOne ואז לשנות ב Main לקובץ הנכון (זה נמצא בהערה – צריך להוריד אותה ולשים את השורה שאחריה בהערה)

3. שנה את מיקום תיקיית surfer למיקום הנכון במחשבך בקובץ readTestsResults.

4. הרץ את הפרוייקט על הקובצים המתקבלים ב - 1 ועל קבצי הפלט של הטסטים (סעיפים 2 ו-3) – בשלב זה מריצים את הקוד שלנו, ונוצר קובץ קלט חדש לאלגוריתם בארינל.

5. הרץ את SFL-diagnoser על הפלט של סעיף 4.

2) קבצי תוצאות:

כל קבצי הפלט של האלגוריתם שלנו (קלט לבארינל עבור התוספת שלנו ועבור המקורי) – נמצאים ב- git בתיקייה out-files.

כל קבצי התוצאות של הרצת בארינל – נמצאים ב- git בתיקייה results.

3) הקוד עצמו:

קישור ל git: <https://github.com/elior19/double_tracer>