המחלקה למדעי המחשב
אוניברסיטת בר-אילן

בס"ד

Online plan recognition

גברת מור ורד

אלעד חמילבסקי | עידו בן אל

יולי 2017

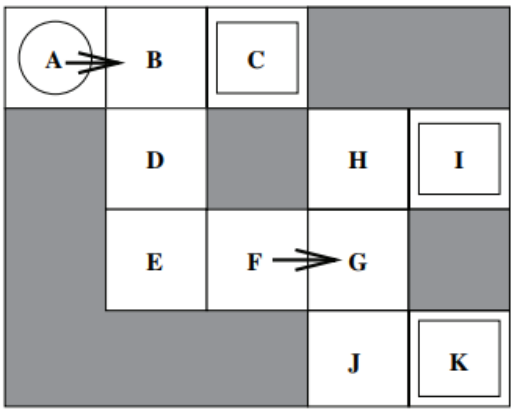
תוכן עניינים

מבוא

בפרויקט זה עסקנו במשימת plan recognition. plan recognition בפרט ו planning בכלל, הוא ענף של בינה מלאכותית הנוגע למימוש אסטרטגיות או רצפי פעולה, בדרך כלל לביצוע על ידי סוכנים נבונים, רובוטים אוטונומיים וכלי רכב בלתי מאוישים. להבדיל מבעיות בקרה וסיווג קלאסיות, הפתרונות מורכבים ויש לגלותם ולבצע אופטימיזציה בחלל רב-ממדי.

במשימה זו מבצעים planning בצורה הפוכה. בעוד שב-planning, בהינתן עולם וקבוצת מטרות, אנו מחפשים את הפעולות שבאמצעותן נגיע למטרה (כלומר תכנון קדימה), ב- plan recognition, בהינתן עולם, קבוצת מטרות ותצפיות, אנו מחפשים את המטרות שיסבירו בצורה הטובה ביותר את התצפיות שראינו (כלומר שחזור אחורה).

האיור הבא מציג בעיית plan recognition פשוטה. חדר A (מסומן בעיגול) הוא המיקום הראשוני של הסוכן, ואילו חדרים C, I ו- K (מסומנים בריבוע) הם היעדים האפשריים. חצים בין חדרים A ו- B, ו- F ו- G, הם תנועות הסוכן הנצפות בסדר זה. המטרות האפשריות היחידות שיש להן תכניות אופטימליות התואמות את רצף התצפית הן I ו- K. בטרמינולוגיה הפורמלית, קבוצת המטרות האפשריות G ניתנת על ידי האטומים ב- (C), ב- (I) ו ב (K), בעוד המטרה האופטימלית G^*_T המורכבת ב (I) ו ב (K), מוותרת על המטרה האפשרית ב (C).



בפרויקט זה נעשה שימוש בplanners המבצעים חיפוש במרחב המצבים בעזרת היוריסטיקה שניתן לחלץ באופן אוטומטי מקידוד הבעיה, ולאחר מכן בשילוב עם אלגוריתמים לחיפוש רגיל. ההצלחה מושפעת מגורמים כגון בחירת ההיוריסטיקה, אלגוריתם החיפוש וכיוון החיפוש. מידע נוסף על אופן השימוש יובא בהמשך.

בעבודתנו ניסינו להיצמד למאמרם של גברת ורד מור ופרופסור גל קמינקא- online recognition of navigation goals through goal mirroring.

בהמשך נפרט אודות מהלך העבודה שלנו החל משלבי ההתקנה, תיאור המימוש, הצגת הבעיות שנתקלנו בהן וכלה בהצעות לשיפור בעתיד.

רקע תיאורטי

בפרויקט זה נתבקשנו לבנות מודל online plan recognition המבוסס על המודל שהציעו Ramırez Miquel ו - Hector Geffner. לצורך כך, נדרשנו ללמוד את תחום הplanning בכלל ואת המודל של Ramırez ו- Geffner בפרט, על ידי קריאת המאמרים הבאים:

1. Ramırez, Miquel, and Hector Geffner. "**Plan recognition as planning**." *Proceedings of the 21st international joint conference on* Probabilistic *Artifical intelligence. Morgan Kaufmann Publishers Inc*. 2009.

2. Plan Recognition Using Off-The-Shelf planners Sohrabi, Shirin, Anton Riabov, and Octavian Udrea. "**Plan recognition as planning revisited**." *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*. 2016.

3. Ramırez, Miquel, and Hector Geffner. "**Probabilistic plan recognition using off-the-shelf classical planners**." *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI 2010)*. 2010.

4. Vered, Mor, Gal A. Kaminka, and Sivan Biham. "**Online Goal Recognition through Mirroring: Humans and Agents**."

5.**online recognition of navigation goals through goal mirroring** (To be published, Mor Vered, Gal A. Kaminka)

מאמר 1 סיפק את הידע התיאורטי על planning והציע שתי היוריסטיקות חישוב G* על טרנספורמציה של בעיית הplanning- חישוב ישיר וחישוב מקורב ע"י שינויים קלים של אלגוריתמים אופטימליים והיוריסטיקות נוספות מעולם הplanning. המאמר מראה ששימוש בהיוריסטיקות אלו מקרב בצורה טובה את קבוצת המצבים האופטימלית .G*

מאמר 2 מציע הרחבה למודל ממאמר 1 על ידי הרפיה של נוסחת ה-plan-recognition ובכך גורם להעדפה של תצפיות בודדות על פני זרימה (fluent), מגביר את אמינות התצפיות (פחות נוטה לרעשים), ובנוסף מזהה גם מסלולים בנוסף לקבוצת המטרות G*. הרפיה זו מתבצעת על ידי הוספת מטרות ביניים לחישוב העלות- חישוב עבור תצפיות רועשות וחישוב עבור תצפיות מוחמצות, וביצוע אופטימיזציה על הצירוף הלינארי של תתי-מטרות אלו. אופטימיזציה זו היא חישוב ההסתברויות על המסלולים האחרונים שנוצרו תוך כדי לקיחה בחשבון של העלויות המשולבות והענשה של תצפיות חסרות או רועשות.

לאחר מסקנות המאמר הקודם, במאמר 3 מרחיבים את הגישה של שימוש באלגוריתמי תכנון עם שינויים קלים לבעיה הכללית יותר של תכנון מסלול בצורה הסתברותית. ההתפלגות ההסתברותית נובעת מתוך הנחה שלפעולות יש השפעה דטרמיניסטית, ובנוסף, גם לסוכן וגם לצופה (observer) יש מידע מלא על המצב ההתחלתי. במאמר זה הראו שניתן לפתור בעיות תכנון בצורה יעילה באמצעות אלגוריתמי תכנון (planner) קלאסיים ובלבד שההסתברות על ביצוע של תצפיות חלקיות בהינתן מטרה מוגדרת כהפרש העלות השגת המטרה כאשר התצפית נמצאת במסלול וכאשר אינה נמצאת במסלול. העלות, ולכן גם ההסתברויות הקודמות להשגת המטרה, מחושבות על ידי שתי קריאות לאלגוריתמי תכנון קלאסיים ללא שום שינוי.

מאמר 4 מציע גישה חדשה, Online goal mirroring, המושפעת מהדרך בה עובד המוח האנושי. בניגוד לגישות הקודמות, גישה זו לא מסתמכת על ספריית תכניות קבועה מראש (ועקב כך מוגבלת), אלא שימוש באלגוריתם שעובד בצורת Online על ידי שימוש ב -Black box planner ליצירת היפותזות זיהוי המושוות באופן מתמשך מול תצפיות חדשות. המאמר גם מציג השוואה בין ביצועי goal mirroring לביצועים אנושיים, ומראה שיש התאמה בין השניים, וכן עמידות של השיטה אל מול תצפיות של תכניות לא אופטימליות. בנוסף המאמר מדגים שעד גבול מסוים, mirroring מזהה תכניות באותה יעילות של שיטות מבוססות ספריה.

במאמר 5, מובאת גישה התומכת בשיקוף יעדים (goal mirroring), גישה של probabilistic plan recognition אשר מתמודדת ישירות עם היעילות של זיהוי מטרה מקוון, דבר המקנה מספר הרצות נמוך יותר מאשר בגרסאות הקודמות. השיפור מתבטא בכך שהחוקרים זיהו שתי נקודות החלטה עצמאיות בתוך אלגוריתם שיקוף המטרה (GM), שבהן ניתן להשתמש בהיוריסטיקה כדי לשפר עוד יותר את זמן הריצה. היוריסטיקות צוינו עבור תחום ההכרה של מטרות ניווט, אשר נוסו על מאות סביבות תלת ממדיות.

רקע פרקטי – התקנה והרצה

הפרויקט נכתב והורץ במערכת ההפעלה LINUX, ובסביבת העבודה PyCharm.

מכיוון שהתבססנו בפרויקט זה על הקוד של Ramırez ו- Geffner ממאמר 3 כשלב מקדים להרצת הפרויקט יש לוודא שהקוד המקורי עובד. לשם כך מובאים מדריכי התקנה לקוד המקורי ממאמר 3 וכן הסברים על אופן ההתקנה וההרצה של הפרויקט.

הקוד המקורי ממאמר 3

חלק א - demo

תחילה ניסינו להריץ את חבילת הדמו של ההרצה הנמצאת בקישור הבא:
https://github.com/miquelramirez/pr-as-planning-demo
הרצת הדמו דרשה הורדה של ספריית עזר בשם LAPKT-LIB הניתנת להורדה מכאן:
https://github.com/LAPKT-dev/LAPKT-public
ספריית LAPKT דרשה התקנה של SCONS (התקנה סטנדרטית משורת הפקודה) וכן ספרייה נוספת בשם JUDY שהורדנו מהקישור: http://judy.sourceforge.net/index.html
הרצת הדמו נתקלה בבעיות של PATH שגוי, אולם ההתקנות המובאות נדרשות לצורך ריצת הקוד המלא.

חלק ב - prob-plan-recognition

שלב 1- הורדת הקוד וקומפילציה

הקוד ניתן להורדה מהאתר https://sites.google.com/site/prasplanning/home. יש להיכנס ללשונית downloads ולהוריד מתקיית software את קבצי הtar של תיקיית ההרצה הראשית prob-plan-recognition, ושל הobs-compiler.
ה obs-compiler מכיל בתוכו תיקייה בשם metric-ff בה יש להריץ את הפקודה make, ולאחר מכן להריץ make נוסף בתקיה הראשית של הקומפיילר. ייווצר קובץ executable של pr2plan, שאותו יש לשים בתיקיית התכנית prob-plan-recognition. לצורך קימפול הקומפיילר יש לוודא כי multilib ++g מותקן במערכת.

שלב 2- הורדת planners

LAMA

הורדת LAMA Planner מאתר https://github.com/rock-planning/planning-lama או מתיקיית קבצי הפרויקט (גרסה חדשה יותר).
בנייה אוטומטית כמצוין בreadme לא עובדת ולכן יש צורך בבניה ידנית, כלומר לקמפל כל תת תיקיה בנפרד. יש לשים לב כי צריך להוסיף תת תיקיות בשם obj לתיקיות search ו preprocess. בנוסף, יש להעתיק את הexecutable search ולשנות את שמו ל- release-search. לבסוף יש להעביר את התיקייה (הראשית של הplanner) לתוך תיקיית הפרויקט prob-plan-recognition.

HSPS

הורדת HSP planner מהאתר http://users.cecs.anu.edu.au/~patrik/un-hsps.html.
ה HSP planner דורש שימוש בגרסאות ישנות של bison ו flex שכבר לא מופצות (גרסאות חדשות יותר לא תואמות ל planner וגוררות שגיאת קומפילציה), אך יש אפשרות להשתמש באופציות מובנות של הplanner שנקראות fake-bison ו- fake-flex.
תחילה יש לחלץ את קובץ הtar דרך הטרמינל (חילוץ אחר לא אפשרי) ולאחר מכן יש להעתיק מתת התיקייה locals את הקבצים makedefs ו-config.h ולשים אותם בתיקייה הראשית של ה planner. בנוסף, יש להעתיק את הקבצים scanner.linux.h ו-scanner.linux.cc ולהשמיט משמם את המילה linux ולשים גם אותם בתיקיית הפלאנר.

בקובץ makedefs יש לשנות את הערך של buildwithcplex מ-0 ל-1, ולוודא ש-0=buildwithscip.
בקובץ ilb.h יש לוודא כי השורה #define CPLEX_INCREMENTAL אינה מופיעה כהערה ומנגד, השורה-#define USE_CACHE כן מסומנת כהערה.
כמו כן, בקובץ ilb.cc יש לוודא כי השורות #define ILA_USE_HS_EXTERN ו- #define USE_ CPLEX אינן מופיעות כהערה, והשורה- #define USE_ SCIP כן מסומנת כהערה.
כעת יש לעשות התאמות כדי לאפשר שימוש בfake-bison ו fake-flex בקובץ makedefs (לפי המצוין בקובץ עצמו).

לצורך הרצת הplanner יש להשתמש בתוכנת CPLEX של IBM הניתנת להורדה בחינם עבור שימוש אקדמי.[1] על מנת להתקין את CPLEX במיקום ברירת המחדל יש לשנות את ההרשאות עבור קובץ ההתקנה על ידי הפקודה chmod u+x <file name> ולאחר מכן להריץ את קובץ ההתקנה על ידי הפקודה sudo ./<file name> .

לשם קימפול הplanner יש צורך ללנקג' את התיקיות הסטטיות ilcplex.a, libcplex.a, libconcert.a, libilocplex.a. אופן קימפול הplanner- הרצת הפקודה make libs בתיקיית הפלאנר. לאחר מכן הרצת הפקודה make all ליצירת קבצי הריצה. לבסוף, יש להעתיק את קובץ ה- hsp_f executable לתיקיית ה prob-plan-recognition.

חלק ג - הורדת קבצי benchmarks

ה-benchmarks הרלוונטיים אלינו ניתנים להורדה מהאתר https://sites.google.com/site/prasplanning/home. יש להיכנס ללשונית downloads ולהוריד מתיקיית benchmarks את הקובץ aaai-10-benchmarks.

חלק ד - הרצת הקוד

לצורך הרצת הקוד יש להריץ דרך ה-terminal את הפקודה:
python prob_PR.py <option><benchmark.tar>

כאשר prob_PR.py הוא קובץ הmain של התוכנית, option היא אחת או יותר מהאפשרויות המופיעות בקובץ option.py (לדוגמה -e עבור ניסוי, -o עבור אופטימלי וכו') ו-benchmark.tar הוא קובץ הבעיה של הדוגמה.

[1] https://www.ibm.com/developerworks/community/blogs/jfp/entry/CPLEX_Is_Free_For_Students?lang=en

הסבר על מהלך התוכנית

מערך הארגומנטים נקלט בmain של prob_PR.py ויוצר אובייקט options המחזיק שדות על אופן ריצת התכנית ומעדכן את הפרמטרים השונים לפי הקלט. לאחר מכן, מייצר רשימת היפותזות על פי קובץ הבעיה, מעדכן את זמני המערכת ומריץ בלולאה את הplanner על כל ההיפותזות. במהלך הריצה מעדכנים את עלות המסלולים וההסתברויות לייתכנותם P(G|O) (הגעה להיפותזה עם מעבר בכל התצפיות) ו- P(G|negO) (הגעה להיפותזה ללא התייחסות לתצפיות). האלגוריתם פועל בצורת offline. עבור כל היפותזה מתבצעת הרצה עם כל התצפיות או בלעדיהן.

מצורף לתיקיית הפרויקט קובץ ZIP המכיל את הקוד המקורי ובו הסברים על הפונקציות השונות ומיפוי של הגדרות ומשוואות מהמאמר אל הקוד עצמו (תחת #EXPLAIN:).

הקוד של הפרויקט

את תיקיית הפרויקט ניתן להוריד דרך קישור QR המופיע בעמוד השער של הספר או דרך הקישור https://github.com/EladCh/Online-Plan-Recognition. בתיקייה ישנן שלוש תת תיקיות: תיקיית הקוד המקורי של מאמר 3, תיקיית הקוד של הפרויקט ותיקייה של הLAMA planner שעבדנו אתה (כמצוין בחלק של התקנת LAMA). כל קבצי הקוד מכילים הערות והסברים.

במידה ולא בוצעה עדין התקנה של הקוד המקורי ממאמר 3, הוראות ההתקנה של הפרויקט זהים להוראות ההתקנה של הקוד המקורי.

מבנה הקוד ותחזוקה

תיאור קבצי הקוד

להלן תיאור קבצי הקוד. לכל קובץ מובא תיאור קצר של אופן השימוש בו וכן מידע על הפונקציות העיקריות שבו. תיאור מפורט לחלקים העיקריים בקוד יובאו בהמשך.

options.py – מחלקה המקבלת את הפרמטרים השונים להרצת התכנית על פי הקוד המקורי של גפנר ורמירז (לדוגמה: online -n, optimal -O) ובאמצעותה ניתן להריץ את האופציות השונות של התכנית.

planners.py – בקובץ זה קיימת מחלקת factory בשם Planner המכילה תכונות (members) הנדרשות להרצה והפעלה של כל סוגי הplanners. כמו כן, קיימות בקובץ זה מחלקות נוספות הממשות מתודות לשימוש והרצה של הplanners השונים.

prob_PR.py – הלולאה הראשית של התכנית- קביעת סוג הplanner שירוץ, קבלת הנתונים מoption, הרצה לפי mode מסוים (online/offline), יצירת דוחות זמניים וקבצי tar זמניים.

hypothesis.py – מכיל את המחלקה Probabilistic שהיא למעשה מייצגת אובייקט של היפותזה. מכילה מתודות להרצת test על ההיפותזה באמצעות הplanner במצב offline ובמצב online.

Main.py – הרצה מלאה של התכנית כמקשה אחת (offline, online, יצירת קובץ תוצאות סופי וקובץ תוצאות סטטיסטי).

benchmark.py – כתיבת קבצי ה-log של התכנית.

Output_to_online.py – יצירת קובץ תוצאות וקובץ אטומים בפורמט csv עבור הbenchmark שמריצים. קובץ תוצאות זה מכיל כל המידע הרלוונטי שמתקבל מההרצה.

translation.py – יצירת קבצי pddl (pr_problem, pr_domain) ספציפיים להיפותזה (המתאימים לplanner) על פי קבצי המקור של הbenchmark.

csv_to_output.py – יצירת קובץ מסכם של תוצאות סטטיסטיות בפורמט csv לפי קבצי התוצאות בתיקייה.

problems_info.py – מחלקת domain_info. על ידי ביצוע parsing קבצי הpddl מייצר מילונים עבור פעולות ופרדיקטים. כמו כן, המחלקה מכילה מתודות הממשות planner נאיבי שעובד לפי הpddl.

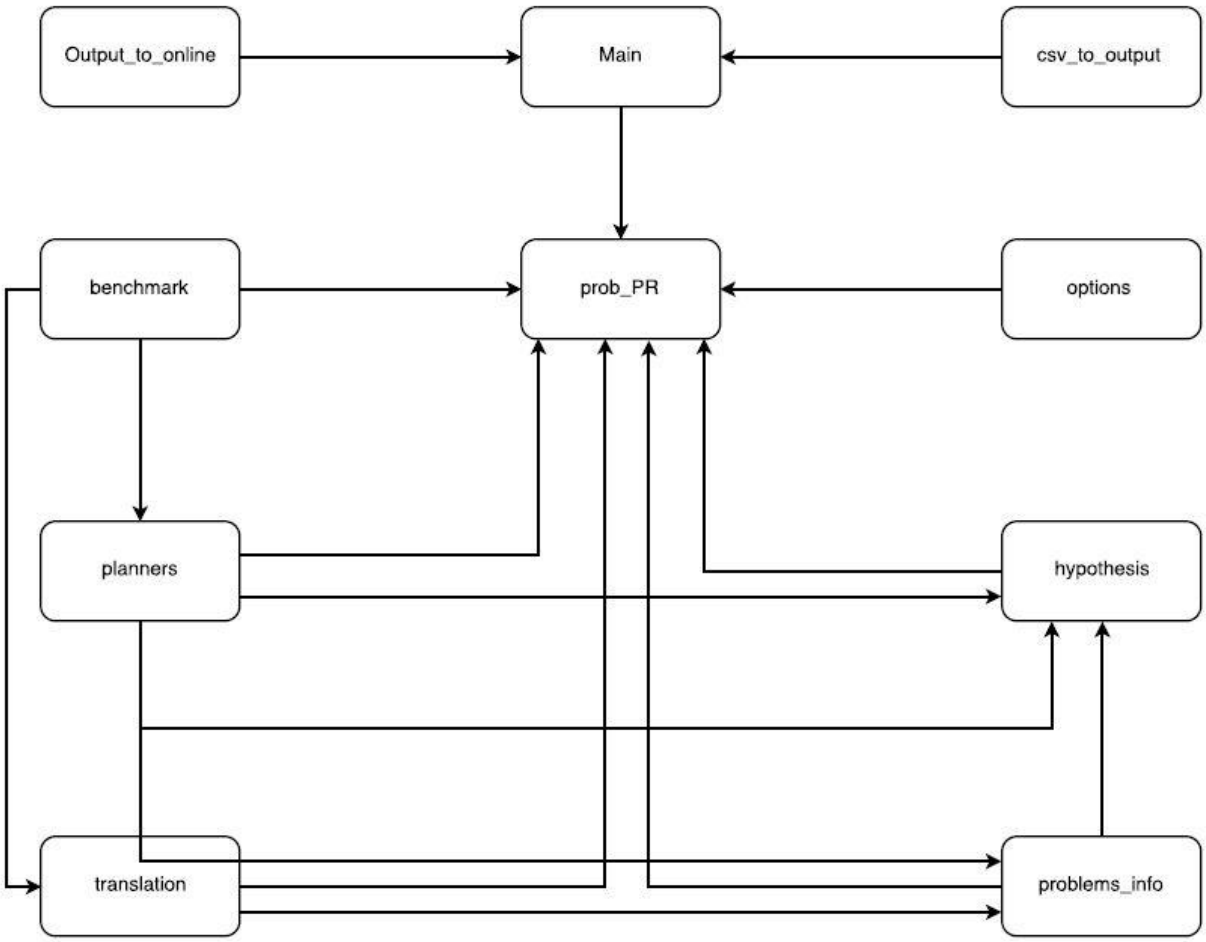
action.py – מכיל מחלקות של פעולה (action) ופרדיקט (predicate).

benchmark_testing.py – משמש להרצה נוחה של אוסף גדול של benchmarks.

PR_sim.py – משמש להרצת סימולציה (לא בשימוש בקוד שלנו).

simulation.py – משמש להרצת סימולציה (לא בשימוש בקוד שלנו).

UML


Output_to_online
Main
csv_to_output
benchmark
prob_PR
options
planners
hypothesis
translation
problems_info

קבצים חשובים

קבצי PDDL

קבצי PDDL הינם קבצים שמטרתם ליצור סטנדרט אחיד עבור שפות של בעיות תכנון.
בתוכנית שלנו אנו משתמשים בשני קבצי PDDL עיקריים.

pr-domain.pddl:

קובץ המכיל את רשימת הפרדיקטים של domain, וכן רשימה של כל הפעולות האפשריות בdomain כאשר לכל פעולה ישנם תנאים מקדימים ואפקטים.
במהלך ההרצה, התוכנית משתמשת בקובץ על מנת למצוא בdomain מסלולים רצויים בהתאם לדרישות ההרצה, כאשר מציאת המסלול נעשית בהתאם למצב העולם הנוכחי (אילו פרדיקטים אנו יודעים שקרו, אילו אפקטים התממשו בעקבות פעולות וכו'), והמטרה אליה אנו שואפים להגיע.

דוגמה (חלקית) לרשימת פרדיקטים:

```
(:predicates
    ( RECON-PERFORMED_PERSEUS )
    ( RECON-PERFORMED_CASSIOPEA )
    ( RECON-PERFORMED_ANDROMEDA )
    ( RECON-PERFORMED_SAGITTARIUS )
    ( RECON-PERFORMED_SCORPIO )
    ( RECON-PERFORMED_VIRGO )
    ( RECON-PERFORMED_ARIES )
    ( RECON-PERFORMED_LEO )
    ( RECON-PERFORMED_LIBRA )
    ( RECON-PERFORMED_TAURUS )
    ( INFORMATION-GATHERED_PERSEUS )
    ( INFORMATION-GATHERED_CASSIOPEA )
    ( INFORMATION-GATHERED_ANDROMEDA )
    ( INFORMATION-GATHERED_SAGITTARIUS )
    ( INFORMATION-GATHERED_SCORPIO )
    ( INFORMATION-GATHERED_VIRGO )
    ( INFORMATION-GATHERED_ARIES )
    ( INFORMATION-GATHERED_LEO )
    ( INFORMATION-GATHERED_LIBRA )
    ( INFORMATION-GATHERED_TAURUS )
    ( ACCESS-OBTAINED_PERSEUS )
    ( ACCESS-OBTAINED_CASSIOPEA )
```

דוגמה לפעולה מתוך רשימת הפעולות:

```
(:action STEAL-DATA_ANDROMEDA
    :parameters ()
    :precondition
    (and
        ( DELETED-LOGS_ANDROMEDA )
        ( FILES-DOWNLOADED_ANDROMEDA )
    )
    :effect
    (and
        (increase (total-cost) 1)
        ( DATA-STOLEN-FROM_ANDROMEDA )
    )
)
```

:pr-problem.pddl

מכיל את המצב הנוכחי של domain ויעד אליו נרצה להגיע (היעד משתנה בהתאם להיפותזה הנוכחית). בהרצת ה Online ערכנו את הקובץ בכל איטרציה כדי שנתחיל את ההרצה החדשה ממצב העולם המתקבל לאחר שראינו את התצפית החדשה.

דוגמה לקובץ:

```
(define
    (problem grounded-INTRUSION-DETECTION-10-HOSTS)
    (:domain grounded-INTRUSION-DETECTION)
    (:init
        (= (total-cost) 0)
        ( NOT_EXPLAINED_BREAK-INTO_LEO_1 )
        ( NOT_EXPLAINED_BREAK-INTO_TAURUS_1 )
        ( NOT_EXPLAINED_FULL_OBS_SEQUENCE )
    )
    (:goal
        (and
        ( INFORMATION-GATHERED_TAURUS )
        ( INFORMATION-GATHERED_LIBRA )
        ( INFORMATION-GATHERED_LEO )
        ( INFORMATION-GATHERED_ARIES )
        ( INFORMATION-GATHERED_VIRGO )
        ( INFORMATION-GATHERED_SCORPIO )
        ( INFORMATION-GATHERED_SAGITTARIUS )
        ( INFORMATION-GATHERED_ANDROMEDA )
        ( INFORMATION-GATHERED_CASSIOPEA )
        ( INFORMATION-GATHERED_PERSEUS )
        ( EXPLAINED_FULL_OBS_SEQUENCE )
        )
    )
    (:metric minimize (total-cost))

)
```

קובץ tamplate

קובץ עזר המכיל מעין תבנית עבור קבצי pr-problem.

דוגמה עבור intrusion-detection:

```
(define (problem intrusion-detection-10-hosts)
    (:domain intrusion-detection)
    (:objects
        perseus cassiopea andromeda sagittarius scorpio
        virgo aries leo libra taurus - host )
    (:init
        (dummy)
    )
    (:goal
        (and
<HYPOTHESIS>
        )
    )
)
```

soln. קובץ

קובץ הנוצר על ידי הHSP planner ומכיל את המסלול שבחר הplanner, עלותו, וכן נתונים סטטיסטיים (לא השתמשנו בהם).

דוגמה לקובץ:

```
; Time 0.032
; ParsingTime 0
; NrActions
; MakeSpan
; MetricValue 15
0 : (recon_taurus) [1]
1 : (break-into_taurus) [1]
2 : (clean_taurus) [1]
3 : (modify-files_taurus) [1]
4 : (vandalize_taurus) [1]
5 : (recon_perseus) [1]
6 : (break-into_perseus) [1]
7 : (clean_perseus) [1]
8 : (modify-files_perseus) [1]
9 : (vandalize_perseus) [1]
10 : (recon_leo) [1]
11 : (break-into_leo) [1]
12 : (clean_leo) [1]
13 : (modify-files_leo) [1]
14 : (vandalize_leo) [1]
;; stats: grounded-intrusion-detection::grounded-intrusion-detection-10-hosts 1 1 9 15 221 0.032 0 1
0 218 0.032
;; (:heuristic ((not_explained_full_obs_sequence)(vandalized_taurus)(vandalized_leo)
(vandalized_perseus)) 15)
```

מהלך העבודה

על מנת שנוכל להריץ את הplanner בצורת online על הbenchmarks נדרשה התאמה של הקוד בכמה מישורים.

מישור ראשון הוא התאמת הפלט של הקוד מהמאמר של Geffner & Ramirez לפלט הרצוי לנו מריצת online. לשם כך, עבור כל הרצה של benchmark יצרנו קובץ חדש שמכיל את המידע הדרוש: שם הבעיה, מספר המטרות האפשריות, כמות התצפיות, זמן התכנסות, דירוגים של המטרות ועוד.

הפלט של התכנית מהמאמר של Geffner & Ramirez הוא קובץ tar.bz2 שמכיל קבצים רבים כגון- קובץ report שבו מידע על זמנים ועלויות, קבצי transcription המכילים מידע נוסף על המסלולים השונים עבור המטרות, קבצי log המכילים תיאורי שגיאות ותיקיה עבור כל מטרה בה מוכל תיאור הבעיה והdomain של הbenchmark עבור (G|O) ו- (G|negO). לשם התאמת הקוד, ומאחר ולא היה קיים קובץ תוצאות המכיל את כל המידע הדרוש, בחרנו את הקבצים שמכילים מידע רלוונטי עבור התכנית שלנו- קובץ report וקבצי הtranscription ובאמצעותם יצרנו קובץ חדש המכיל את המידע הרלוונטי עבור כל היפותזה (מטרה).

מכיוון שהמידע הנדרש קיים בקבצים שונים הנמצאים בתוך קובץ tar.bz2 יצרנו סקריפט המחלץ את הקבצים לתיקיה זמנית, בוחר את הקבצים הרלוונטיים ויוצר קובץ חדש בתיקיית הפרויקט (בה נמצא הtar וקבצי התכנית) המכיל את הפרמטרים השונים שנדרשים בסדר אחיד. שם הקובץ נכתב בפורמט PLANNER_DATE_TIME (סוג ה-planner בו השתמשנו, תאריך ושעה). לאחר יצירת הקובץ התיקייה הזמנית נמחקת. כך מקובץ tar המכיל המון מידע לא רלוונטי "מזקקים" קובץ יחיד המכיל את כל המידע הרלוונטי להרצת התכנית שלנו. כמו כן, הוספנו לקוד הקיים פקודת ליצירת התיקייה results בה יהיו קבצי הפלט הסופיים, וכן מחקנו את הקבצים המיותרים מהתיקייה הראשית על מנת שסביבת העבודה תישאר נקייה ונוחה לעבודה.

אחד הפרמטרים שנדרשים לקובץ הקלט לאלגוריתם החדש דורש חישוב של המסלול האידיאלי, כלומר ללא השפעת תצפיות- המסלול הנאיבי. מכיוון שהיה הגיוני שמסלול כזה יופיע גם בקוד מהמאמר (3) חיפשנו בקבצי התוצאות את תיאור המסלול הזה. ואכן, לאחר הרצה של הקוד הופיע בתיקיית הקוד קובץ .soln שחשדנו שבו ימצא המסלול, אך כשפתחנו אותו מצאנו שהקובץ ריק (קובץ ריק פירושו כישלון של ריצת הplanner). לאחר מספר הרצות שבהן לא קיבלנו נתונים בקובץ הsoln חזרנו לבדוק את הplanner בתקווה שנמצא מה גורם לכישלון. באותו שלב, עבדנו עם HSPS planner מבוסס SCIP שלמעשה לא עבד, ולכן התאמנו את הקוד לשימוש בLAMA planner באופן בלעדי. לאחר חיפוש באינטרנט מצאנו כי ניתן להריץ את הHSPS planner עם CPLEX מבית IBM (ניתנת להורדה בחינם עבור מטרות לימודים ומחקר). לאחר התקנת התוכנה הplanner אכן עבד ובקובץ .soln הופיע המסלול שאותו חיפשנו.

לאחר שווידאנו שקובץ הפתרון עבור HSPS תקין ונותן את הדרוש, רצינו להוסיף לקובץ הפלט החדש שלנו את היחס שבין העלות של המסלול האידיאלי לעלות של המסלול המתקבל הנותר. לצורך חישוב זה יש להשוות בין המסלולים המתקבלים מהאלגוריתם האופטימלי בהרצת offline (ללא תצפיות) לבין האלגוריתם כאשר מריצים אותו בצורת online (עם הוספת התצפיות). יש לציין כי הרצה של planners שונים מניבה קבצי תוצאות שונים, קרי- הרצת HSPS מייצרת קובץ soln המכיל את המסלול ומחירו, בעוד הרצת

הLAMA planner אינה מייצרת קובץ שכזה. לאחר הרצה ידנית של LAMA דרך הטרמינל מצאנו שישנו שינוי בשם של אחד הקבצים (עקב שינוי גרסאות), התיקון מופיע במדריך ההתקנה לעיל. קבצי התוצאה של LAMA הנוצרים בתיקיית הראשית אינם רלוונטיים, אולם המידע שאנו זקוקים לו מופיע בקבצי הlog שנמצאים בtar הפלט של התכנית.

לשם נוחות, יצרנו קובץ Main.py האחראי על הרצת התכנית בצורה מלאה ללא צורך בהגדרות נוספות. הקובץ מכיל פונקציית run המקבלת כקלט את שם הbenchmark שעליו נרצה את ההרצה. יש לציין שקובץ הbenchmark צריך להיות באותה תיקייה שבה נמצא הקובץ Main.py.

הפיכה לonline

המישור השני הוא התאמת הקוד לצורת פעולה online. כאמור, יש צורך בתוצאות האלגוריתם כשהוא פועל באופן offline וכן בתוצאות האלגוריתם כשהוא פועל בצורת online. לכן, הוספנו אופציה חדשה לקוד המקורי של גפנר ורמירז, online, המסומנת בflag –n. כאשר flag זה מתקבל בקלט, התכנית תשתמש בלולאת הרצה המתאימה לצורת פעולה של online ואילו כערך ברירת המחדל תישאר הרצה בoffline. בחרנו לממש כך על מנת לא לחסום אפשרויות הרצה קיימות וכן הרצה בonline עבור כל מצב פעולה אפשרי.

כאשר מתקבל הflag -n הקוד יכנס ללולאה הבאה (בקובץ prop_PR.py):

```
if options.online:
  obs = load_observations()
  remainder = hyp_time_bounds[0]
  # iterating the observations
  for j in range(0, len(obs)):
    global obs_ind
    # for each observation, iterate all hyps
    for i in range(0, len(hyps)):
      hyps[i].test_online(i, j, hyp_time_bounds[i], options.max_memory,
options.optimal)
      if hyps[i].cost_O == 1e7 and hyps[i].cost_Not_O == 1e7:
        hyps[i].test_failed = True
      remainder = remainder - hyps[i].total_time
      if remainder > 0:
```

בלולאה זו מתבצעת קריאה של קובץ התצפיות אל תוך רשימה כך שנוכל לרוץ בלולאה על התצפיות השונות. בתוך כל לולאה אנו רצים על כל ההיפותזות בצורה דומה לאלגוריתם המקורי, אלא שבמקום לקרוא לפונקציית test_offline (מהקוד המקורי) נקרא לפונקציה test_online. שאר החישובים נשארו כמו שהיו בקוד המקורי. על מנת שנוכל לדעת על איזו תצפית אנו מסתכלים הגדרנו משתנה גלובלי אותו נוכל לעדכן גם במודולים הנוספים.

בתוך הפונקציה test_online (בקובץ hypothesis.py) נוצר קובץ הבעיה. מכיוון שאלגוריתם online מקבל כל פעם תצפית חדשה (על פי הלולאה בקובץ prop_PR.py) וידאנו כי מספר התצפיות שיתווספו לקובץ הבעיה בשלב הנוכחי יהיה נכון. הגדרנו משתנה

גלובלי count המאותחל ל(1-) והשתמשנו במשתנה הגלובלי obs_ind הנ"ל. בכל פעם שהאינדקס המתקבל גדול מהcounter נגדיל את הcounter באחד ונקרא לפונקציה modify_obs_file. הפונקציה קוראת את קובץ התצפיות ויוצרת קובץ תצפיות חדש בעל מספר תצפיות כגודל הcounter (במקום קובץ עם כל התצפיות כגוש אחד כמו בoffline) ודורסת את הקובץ המקורי. לאחר מכן מתבצע תרגום של הבעיה כמו בקוד המקורי. בכל איטרציה של הפונקציה נוצרת תיקייה חדשה עם עותק מקורי של התצפיות (obs.dat) וכן קובץ בעיה מקורי (hyp_problem) (כתוצאה מהפעלת הסקריפט translation, שבכל פעם משתמש בקבצים הנמצאים בתיקיית הbenchmark) ועליהם מתבצעת המניפולציה אין איבוד של מידע בין האיטרציות השונות והקבצים אכן תקינים.

```
# generate the problem with G=H
hyp_problem = 'hyp_%d_problem.pddl'%index
self.generate_pddl_for_hyp_plan( hyp_problem )
global count
if obs_index > count:
  count += 1
  self.curr_obs_num = count+1
  modify_obs_file(count)
# creating the derived problem with G_Obs
trans_cmd = translation.Probabilistic_PR('domain.pddl', hyp_problem,
'obs.dat')
trans_cmd.execute()
```

בנוסף לשינוי זה, נדרשנו לשנות את קבצי pr_problem.pddl של כל ההיפותזות לאור התצפיות שראינו. על פי אלגוריתם ה-online בכל איטרציה הסוכן נחשף לתצפית חדשה, כך שבמקרה וההיפותזה נכונה, ככל שהסוכן ראה יותר תצפיות המסלול שלו אל ההיפותזה אמור להתקצר. כפי שנכתב בהקדמה על קבצי pr_problem.pddl, לכל תצפית (פעולה בקובץ pr_domain.pddl) קיימים פרדיקטים המבטאים את האפקט שלה, וכיוון שהסוכן ראה את התצפית, האפקט הזה חל על הdomain, כלומר יש להוסיף את הפרדיקטים של האפקט למצב ההתחלתי (init) בקובץ pr_problem.pddl.

על מנת לדעת מהם האפקטים של כל פעולה בנינו מחלקה הנקראת Domain_info (בקובץ problems_info.py) שלוקחת את הקובץ pr_domain.pddl ויוצרת ממנו שני מילונים- מילון פרדיקטים ומילון פעולות. רשומה במילון הפרדיקטים מכילה את שם הפרדיקט, האם הוא מתקיים (valid) ורשימה של פעולות שאחד הפרדיקטים באפקט שלהן הוא הפרדיקט הזה (שימוש ברשימה זו מקצר את זמן הריצה של planner הבדיקה הנאיבי, יפורט בהמשך). רשומה במילון הפעולות מכילה את שם הפעולה, האם הפעולה התקיימה (valid) ומשתנה האומר האם הפעולה היא חלק מקבוצת פעולות בעלת אותו שם (מכיוון שיתכנו מספר פעולות בעלות שם זהה, flag זה חיוני לפעילות planner הבדיקה הנאיבי).

```
▼ predicates_dict = {dict} {' DATA-STOLEN-FROM_ANDROMEDA ': <action.Predicates instance at 0x7f8472fb0200>, ' INFORMATION-GA
    __len__ = {int} 96
  ▼ ' ACCESS-OBTAINED_ANDROMEDA ' (140206841328368) = {instance} Predicates: <action.Predicates instance at 0x7f8472fa8f80>
      index = {int} 22
      name = {str} ' ACCESS-OBTAINED_ANDROMEDA '
    ▶ possible_actions = {list} <type 'list'>: ['BREAK-INTO_ANDROMEDA']
      times_visited = {int} 0
      valid = {bool} False
```

```
actions_dict = {dict} {'GAIN-ROOT_VIRGO': [<action.Action instance at 0x7f8472b0e4d0>], 'BREAK-INTO_LIBRA': [<actio
    __len__ = {int} 91
    'BREAK-INTO_ANDROMEDA' (140206841436464) = {list} <type 'list'>: [<action.Action instance at 0x7f8472fbaab8>]
        __len__ = {int} 1
        0 = {instance} Action: <action.Action instance at 0x7f8472fbaab8>
            effect = {list} <type 'list'>: ['increase (total-cost) 1', ' ACCESS-OBTAINED_ANDROMEDA ']
            index = {int} 69
            is_compressed = {bool} False
            name = {str} 'BREAK-INTO_ANDROMEDA'
            parameters = {list} <type 'list'>: []
            precondition = {list} <type 'list'>: [' RECON-PERFORMED_ANDROMEDA ']
            times_visited = {int} 0
            valid = {bool} False
```

יצרנו אובייקט של domain הנ"ל פעם אחת בלבד בתחילת התכנית ושמרנו אותו בתוך האובייקט של ההיפותזה על מנת לאפשר גישה נוחה אל domain לכל אורך התכנית.

בכל איטרציה התכנית מייצרת תיקיה מקוננת להיפותזה הנוכחית בה קיימות שתי תיקיות- O ו-neg-O. בכל תיקייה ישנם שני קבצים: קובץ pr_problem.pddl וקובץ pr_domain.pddl. קבצים אלו נוצרים כתוצאה מהפעלת הסקריפט translatoin.py המתרגם את קבצי הdomain.pddl הגנרי והתבנית template לכדי בעיה ספציפית. לאחר ניסוי וטעייה מצאנו שלמרות שהדרך המתבקשת היא להכניס את הפרדיקטים של התצפית לקובץ הtemplate כדי שמההתחלה יוצרו קבצים מעודכנים, הדבר גורר שגיאה (ככל הנראה פגיעה בפורמט שהplanner מאשר) ולכן הכנסת הפרדיקטים נעשית לאחר יצירת התיקיות והקבצים שבהן.

```
# modify the pr_problem file according to observation
for id, domain, instance in self.walk('prob-%s-PR' % index):
    if id == '0':
        self.modify_init_pr_problem_file(self.domain, instance)
```

עדכון הקבצים הוא די טריוויאלי- עבור על התיקייה שנוצרה, אם שם תת התיקייה הוא O (כיוון שאנו רוצים לעדכן את הקובץ עבור ריצה עם התצפיות ולא בלעדיהן) קרא את הקובץ dlpr_problem.pd, הכנס את הפרדיקטים של המצב ההתחלתי (init) לרשימה, מצא את האובייקט של התצפית (פעולה) והכנס את האפקטים שלה לרשימה נפרדת. מצא את הפרדיקטים השוללים את התצפית (על ידי הוספת NOT_EXPLAINED לשם התצפית). כעת, עבור על הinit ועל רשימת התצפיות, אם הפרדיקט הוא הפרדיקט השולל את התצפית הוצא אותו. לאחר מכן הכנס את כל האפקטים (לאחר הורדת כפילויות). אפקטים המכילים שלילה (not (predicate)) לא הוכנסו לinit. בנוסף, כיוון שבכל שלב הסוכן ראה את כל התצפיות עד כה אנו משנים את הפרדיקט NOT_EXPLAINE_FULL_OBS ל-EXPLAINED_FULL_OBS.

המשך ריצת התכנית זהה להרצת הקוד המקורי.

על מנת לוודא שהplanner שהרצנו אכן מצא את המסלול הנכון בנינו planner שפועל בצורה נאיבית על domain. לשם כך יצרנו את המחלקה Domain_info (המוזכרת לעיל). בעת יצירת אובייקט של המחלקה מתבצע מיפוי domain מקובץ pr_domain.pddl לשני מילונים- מילון פרדיקטים ומילון פעולות. המתודה run_hyp עוברת על כל האטומים בהיפותזה ועבור כל אטום מוצאת בצורה רקורסיבית את כל הפעולות עבורן הפרדיקט (ובפרט האטום) מופיע כאפקט. לאחר מעבר רקורסיבי על אטום בודד מתבצעת בדיקה אילו פעולות אכן התקיימו על ידי בדיקת התנאים המקדימים של הפעולה (preconditions), כך

שבסוף איטרציה על אטום מסוים ידוע לנו המסלול (מספר הפעולות והפעולות עצמן) שהובילו אליו. כמו כן קיימות מתודות ייעודיות המבצעות את אותה הלוגיקה עבור תצפיות (שהן למעשה פעולות).

בהרצת ה- planner הנאיבי מצאנו שהplanner המקורי אכן הגיע לתוצאה הרצויה. מאחר והplanner הנאיבי לא משתמש בplanner חיצוני אלא רק עובר על הpddl, לא השתמשנו בו להמשך התכנית ולכן גם לא הרחבנו עליו בספר הפרויקט, אולם הסבר כללי מופיע כנספח לספר הפרויקט וקוד מפורט עם הסברים מצורף לתיקיית הפרויקט.

לפי מאמר (5) המדרג (rank) מחושב כחלוקה של עלות המסלול האידאלי בסכום עלות המסלול הנותר לאחר צפייה בתצפיות ועלות התצפיות עצמן. בשלב הראשוני חישבנו את עלות התצפיות כמספר התצפיות שנצפו עד כה. בהרצת הקוד אכן מצאנו שהמסלול הנותר מתקצר ככל שרואים יותר תצפיות אולם חיבור מספר התצפיות עם עלות המסלול הנותר לא הייתה שווה לעלות המסלול האידאלי עבור ההיפותזה הנכונה, כך שהמדרג לא יוצא נכון (בהיפותזה הנכונה הדירוג אמור להתכנס ל-1 או יותר ככל שנצפות יותר תצפיות).

כתוצאה מכך, החלטנו לבדוק האם העלות של הרצת הplanner על התצפיות (בonline) תשלים את הפער בין המסלול האידאלי למסלול הנותר. לצורך ההרצה על התצפיות נדרשנו לייצר קבצי pddl חדשים בהם הgoal יהיה הpreconditions של התצפית ולא ההיפותזה.

```
def generate_pddl_for_obs_plan(self, obs):
    obs_preconditions=[]
    # get obs preconditions
    for o in obs:
        o = "EXPLAIN_OBS_"+o.replace(" ","_").replace("(","").replace(")","")+"_1"
        o_obj = self.actions_dict.get(o)
        if o_obj is None:
            continue
        temp_list=[]
        for p_str in o_obj[0].precondition:
            p_str="("+p_str.replace("_"," ").lower().strip()+")"
            if not "explained" in p_str:
                temp_list.append(p_str)
        obs_preconditions.append(temp_list)

    # create the new files
    for i in range(obs.__len__()):
        out_name="obs_"+str(i)+"_problem.pddl"
        instream = open('template.pddl')
        outstream = open(out_name, 'w')

        for line in instream:
            line = line.strip()
            if '<HYPOTHESIS>' not in line:
                print >> outstream, line
            else:
                # insert obs preconditions instead of hypothesis
                if i>0:
                    for j in range(1,i+1):
                        for atom in obs_preconditions[j]:
                            print >> outstream, atom
                else:
                    for atom in obs_preconditions[i]:
                        print >> outstream, atom
        # able to insert number of goals here (atoms)
        outstream.close()
        instream.close()
```

בנוסף, על מנת ליצר הרצת online יש צורך "להזרים" תצפיות כך שבכל הרצה תהיה רק תצפית אחת חדשה והמסלול שנעשה כבר יעבור דרך כל התצפיות הקודמות (על מנת שתהיה התאמה בין הרצת התצפיות לבד ובין ההרצה המקורית).

```
def run_planner_on_obs(self, obs_index):
    obs_plans=[]
    obs_actions=[]
    for i in range(obs_index):
        hyp_problem = 'obs_%d_problem.pddl' % i

        self.modify_obs_file(i)
        # creating the problem pddl
        trans_cmd = translation.Probabilistic_PR('domain.pddl', hyp_problem, 'obs.dat')
        trans_cmd.execute()

        os.system('mv prob-PR obs_prob-%s-PR' % i)
        # execute the planner on the problem and gets the plan
        for id, domain, instance in self.walk('obs_prob-%s-PR' % i):
            if id == "0":
                plan_for_G_Obs_cmd = planners.HSP(domain, instance, i)
                plan_for_G_Obs_cmd.execute()
                obs_actions= plan_for_G_Obs_cmd.get_plan()
                obs_plans.append(obs_actions)
    return obs_plans
```

```
# incrementally editing obs.dat according to current iteration
def modify_obs_file(self, count):
    lines = []
    for i in range(count + 1):
        line = linecache.getline('obs.dat', i + 1)
        obs_line = line.replace('(', '( ')
        obs_line = obs_line.replace(')', ' )')
        lines.append(obs_line)

    with open(os.getcwd() + "/obs.dat", "w") as outstream:
        for line in lines:
            outstream.write(line)
```

לאחר הרצת הplanner על התצפיות מצאנו שעלות תצפית איננה רק מספרה הסידורי אלא העלות שנדרשה על מנת להגיע לתצפית, כלומר מספר הפעולות שבוצעו מנקודת ההתחלה עד להגעה אל התצפית.

קבצי תוצאות

באלגוריתם online ישנו מעבר על כל ההיפותזות הנתונות לכל תצפית. בסוף הרצה של תצפית מסוימת נכתב קובץ report המכיל את המידע הרלוונטי. השדות של הקובץ נערכו על גבי הקוד המקורי. לאחר סיום ריצת האלגוריתם מופעל קוד (output_to_online.py) ליצירת קובץ פלט מסוג csv, ששמו מוצג בפורמט <שעה_ תאריך_ שם הplanner>, המאגד את כל המידע שהתקבל מריצת האלגוריתם- עבור כל תצפית ולכל היפותזה. כמו כן מופיע דירוג עבור כל תצפית. בנוסף, לכל קובץ תוצאה נוצר קובץ בעל שם זהה עם התוספת atoms בה מופיעות ההיפותזות המתאימות לקובץ התוצאות.

פורמט של שורה בקובץ התוצאה:

obs_num, hyp_num, hyp_index, hyp_test_failed, hyp_is_true, timeout, heap_restriction, obs_cost, current_cost, ideal_cost, hyp_GR_rank, hyp_VK_rank, hyp_cost_not_O, hyp_prob_O, hyp_prob_not_O, hype_plan_time_O, hyp_plan_time_not_O, hyp_trans_time, hyp_plan_time, hyp_test_time

obs_num - מספר התצפית.

hyp_num - מספר ההיפותזות בקובץ.

hyp_index - מספר ההיפותזה הנוכחית.

hyp_test_failed - שדה בוליאני המראה אם ההרצה נכשלה מסיבה כלשהי.

hyp_is_true – שדה בוליאני המראה האם ההיפותזה היא ההיפותזה הנכונה.

timeout - מגבלת הזמן על ריצת הplanner.

heap_restriction - מגבלת המקום על ריצת הplanner.

obs_cost - העלות להגיע מנקודת ההתחלה לתצפית הנוכחית (להרחבה ראה פרק הפיכה לאונליין).

current_cost - העלות של ההיפותזה הנוכחית עם התצפיות עד כה (כל שורה בקובץ היא תצפית חדשה).

ideal_cost - העלות של המסלול אל ההיפותזה ללא תצפיות בoffline.

hyp_GR_rank - הדירוג לפי מאמר (3) של גפנר ורמירז:

$$cost(O) - cost(neg\_O)$$

hyp_VK_rank - הדירוג לפי מאמר (5) של ורד וקמינקא:

$$\frac{ideal\ cost}{obs\ cost + current\ cost}$$

hyp_cost_not_O - העלות של ההיפותזה ללא התצפיות.

hyp_prob_O - ההסתברות שההיפותזה הזו היא הנכונה.

hyp_prob_not_O - ההסתברות שההיפותזה הזו אינה הנכונה.

hyp_plan_time_O - זמן ריצת הplanner על ההיפותזה עם התצפיות.

hyp_plan_time_not_O - זמן ריצת הplanner על ההיפותזה ללא התצפיות.

hyp_trans_time - זמן תרגום הבעיה לקבצי pddl ספציפיים.

hyp_plan_time - זמן ריצת הplanner על ההיפותזה בסה"כ, עם תצפיות ובלעדיהן.

hyp_test_time - סכום זמן התרגום וזמן הריצה הכולל.

הרצת tests

לצורך נוחות ההרצה, יצרנו סקריפט נוסף בשם benchmark_testing.py המריץ את Main.py באופן מסודר על כמה בעיות. על מנת להשתמש בסקריפט יש ליצור תיקייה בשם test_benchmarks בתיקייה הראשית של הפרויקט ולתוכה יש להכניס קבצי tar.bz2 של הbenchmarks הרצויים. הרצת הסקריפט דרך הטרמינל תבוצע על ידי הפקודה:

python benchmark_testing.py

עיבוד התוצאות

לאחר קבלת התוצאות נעשה עיבוד לנתונים לכדי קובץ csv יחיד, statistical_result, המכיל את המידע הסטטיסטי הרלוונטי עבור כל בעיה. לשם כך יצרנו הסקריפט csv_to_output, העובר על תיקיית התוצאות (על קבצי התוצאות בלבד, ללא ה-atoms) ומייצר קובץ חדש בעל השדות הבאים עבור כל היפותזה:

Experiment_name, hyps_num, hyp_test_failed, hyp_is_true, Timeout, Heap_restriction, hype_plan_time_O, hyp_plan_time_not_O, hyp_trans_time, hyp_plan_time, hyp_test_time, hyp_test_failed

(הסברים לפרמטרים השונים ניתן למצוא בפרק יצירת קבצי תוצאות) וכמו כן בסוף כל שורה מופיעים convergence, ranked_first עבור הבעיה.

שימוש בספריית numpy הועיל לשליפה נוחה של המידע מהקבצים ומעבר מהיר ופשוט מרשימת מידע למטריצה פשוטה. לשם נוחות, יצרנו תת מטריצה המכילה רק את הדירוגים, בה כל שורה היא תצפית וכל עמודה היא היפותזה, עבור חישוב ההתכנסות ומספר הפעמים שההיפותזה דורגה במקום הראשון.

ranked_first סוכם את מספר הפעמים שההיפותזה אכן דורגה במקום הראשון ביחס לכלל ההיפותזות, מתוך כלל התצפיות (התוצאות עוגלו לחמש ספרות אחרי הנקודה). מחושב על ההיפותזה הנכונה (real_hyp).

```
# ranked first, only for real hyp
r_first_counter = 0
for i in range(rows-4):
  m = max(ranking_matrix[i])
  real_hyp_rank = ranking_matrix[i][true_hyp_col-1]
  if float(real_hyp_rank) >= float(m):
      r_first_counter += 1
first = float(r_first_counter)
data += str(round(first/r, 5))
```

convergence מחושב על כל היפותזה עבור כל אחת מהתצפיות ובודק מתי ההיפותזה מגיעה להתכנסות, כלומר מונוטונית לא יורדת (התוצאות עוגלו לחמש ספרות אחרי הנקודה).

```
# Convergence
conv_array = []
cou = 0
num_of_obs = ranking_matrix.__len__()
for j in range(x.__len__() - cols):
    cou = 0  # counter of the converging elements
    con_value = float(1)
    con_index = 0
    # if the rank (value of the cell) is less than 1
    if float(ranking_matrix[num_of_obs-1][j]) < con_value:
        conv_array.append(0)
        continue
    else:
        inserted_flag = False
        # run on all the column
        for i in range(num_of_obs-1, -1, -1):
            current_value = ranking_matrix[i][j]
            # if the value is equal or greater than the con_value
            if float(current_value) >= con_value:
                cou +=1
            else:
                # calculate the convergence
                conv_array.append(cou/float(num_of_obs))
                inserted_flag = True
                break
        # calculate the convergence
        if not inserted_flag:
            conv_array.append(cou / float(num_of_obs))
```

עבור על השורה האחרונה במטריצת הדירוגים. אם הערך בעמודה הנוכחית קטן מ-1 המשך לעמודה הבאה. אם הערך גדול או שווה ל-1 עבור על העמודה מהסוף להתחלה. אם הערך הקודם בעמודה גדול או שווה מהערך הנוכחי המשך להתקדם והעלה את הcounter. אם הערך הנוכחי קטן מהערך הקודם בעמודה בצע את חישוב ההתכנסות וסיים את הפונקציה. אם הלולאה הסתיימה בהצלחה ללא עצירה נבצע את חישוב ההתכנסות.

ניסויים והרצות

תוך כדי תהליך ההפיכה לonline, הרצנו את הקוד שכתבנו על כל קבצי הbenchmark של רמירז וגפנר. הקבצים מכילים 450 בעיות פלאנינג משישה תחומים (domains) שונים- blok words, easy ipc grid, intrusion detection, logistics, campus, kitchen. פירוט על הבעיות השונות נמצא במאמר (3). ההרצה ערכה כ11 ימים.

לאחר צפייה בתוצאות מצאנו שלעיתים הריצה נכשלת ולכן ערכנו שינויים ותיקונים לקוד התכנית על מנת למזער, ובשאיפה למנוע, את הכישלונות האלו. מכיוון שהרצה על כל ששת התחומים אורכת זמן רב, בהמלצת המנחה הרצנו את הבדיקות הבאות על intrusion-detection Domain בלבד.

ייצוג התוצאות

בקבצי הפלט התוצאות מוצגות בצורה מטריציונית. כל שורה מייצגת תצפית, וכל עמודה מייצגת היפותזה. להלן מספר דוגמאות, כאשר לכל דוגמה מובאת ההצגה כמו שהיא בקובץ, וכן הצגה גרפית.

דוגמה I:

עבור הבעיה intrusion-detection_p10_hyp-5_50_2.

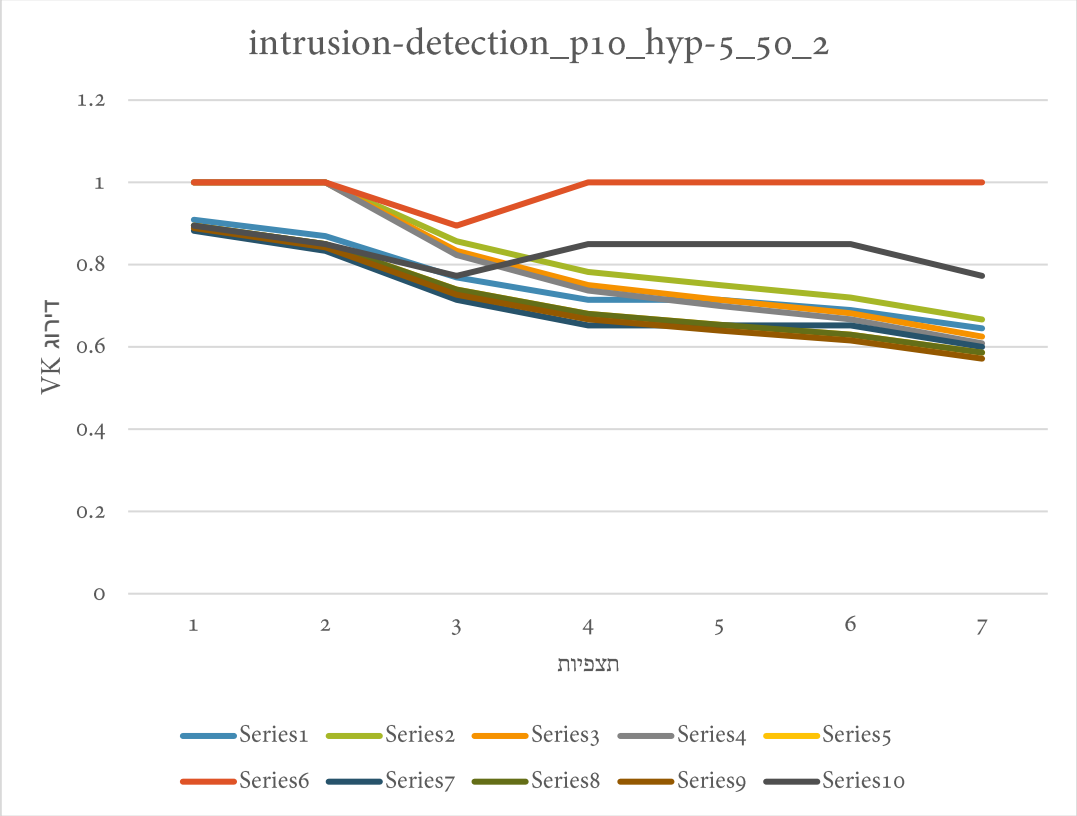
ייצוג טבלאי (כמו שמופיע בקובץ הפלט):

0.894737	0.888889	0.894737	0.882353	1	0.894737	1	1	1	0.909091
0.85	0.842105	0.85	0.833333	1	0.85	1	1	1	0.869565
0.772727	0.727273	0.73913	0.714286	0.894737	0.73913	0.823529	0.833333	0.857143	0.769231
0.85	0.666667	0.68	0.652174	1	0.68	0.736842	0.75	0.782609	0.714286
0.85	0.64	0.653846	0.652174	1	0.653846	0.7	0.714286	0.75	0.714286
0.85	0.615385	0.62963	0.652174	1	0.62963	0.666667	0.681818	0.72	0.689655
0.772727	0.571429	0.586207	0.6	1	0.586207	0.608696	0.625	0.666667	0.645161

ה real-hyp בבעיה זו היא ההיפותזה השישית, המיוצגת כאמור בעמודה השישית. ניתן לראות בטבלה שהיא מדורגת ראשונה לאורך כל התצפיות, ושההתכנסות שלה מתחילה בתצפית הרביעית.

ייצוג גרפי:

ציר ה Y בגרף מייצג את ערך דירוג VK, ציר ה X מייצג את כמות התצפיות שנראו.



דוגמה II:

עבור הבעיה intrusion-detection_p10_hyp-1_70_1.

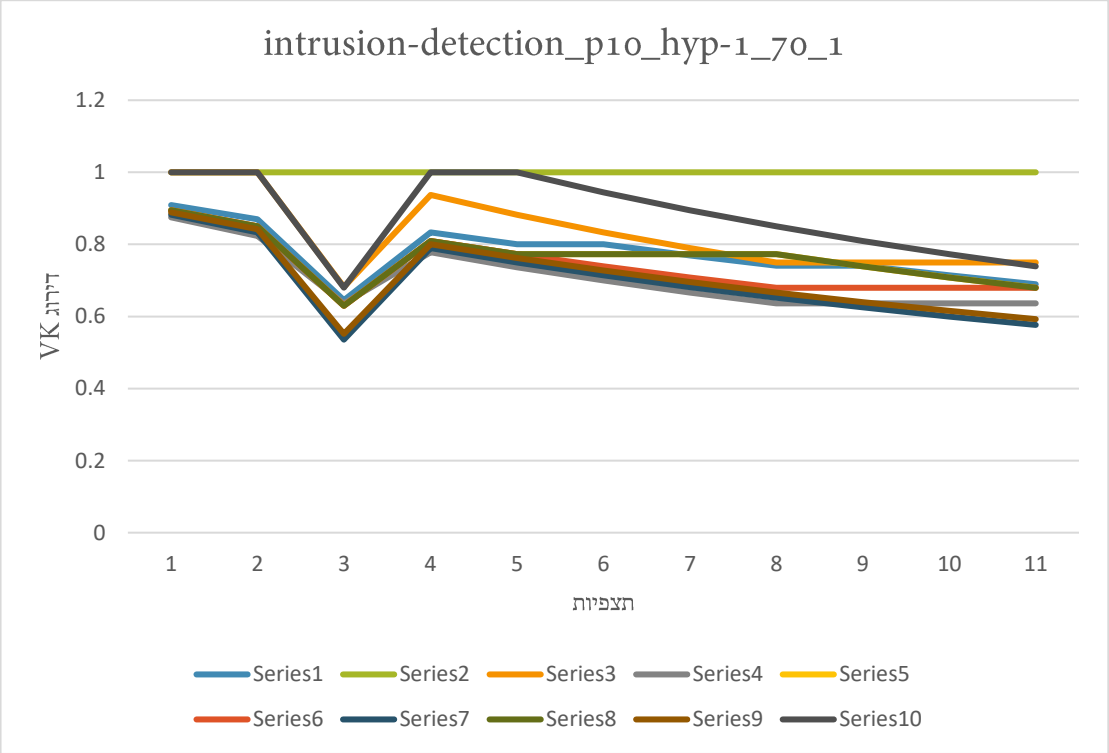
ההיפותזה האמיתית כאן היא ההיפותזה השנייה.

במקרה הזה היא גם מדורגת ראשונה לאורך כל התצפיות, וגם מתכנסת כבר מההתחלה.

ייצוג טבלאי:

1	0.888889	0.894737	0.882353	0.894737	0.894737	0.875	1	1	1	0.909091
1	0.842105	0.85	0.833333	0.85	0.85	0.823529	1	1	1	0.869565
0.68	0.551724	0.62963	0.535714	0.62963	0.62963	0.636364	0.681818	1	1	0.645161
1	0.8	0.809524	0.789474	0.809524	0.809524	0.777778	0.9375	1	1	0.833333
1	0.761905	0.772727	0.75	0.772727	0.772727	0.736842	0.882353	1	1	0.8
0.944444	0.727273	0.772727	0.714286	0.772727	0.73913	0.7	0.833333	1	1	0.8
0.894737	0.695652	0.772727	0.681818	0.772727	0.708333	0.666667	0.789474	1	1	0.769231
0.85	0.666667	0.772727	0.652174	0.772727	0.68	0.636364	0.75	1	1	0.740741
0.809524	0.64	0.73913	0.625	0.73913	0.68	0.636364	0.75	1	1	0.740741
0.772727	0.615385	0.708333	0.6	0.708333	0.68	0.636364	0.75	1	1	0.714286
0.73913	0.592593	0.68	0.576923	0.68	0.68	0.636364	0.75	1	1	0.689655

ייצוג גרפי:



הצעות לשיפור בעתיד

במאמר של ורד וקמינא שעתיד להתפרסם (5), מוצעות שתי היוריסטיקות המצמצמות את מספר הקריאות לplanner תוך שמירה ואף שיפור של ביצועי הplan recognition. בפרויקט זה לא מימשנו את ההיוריסטיקות אולם כן בנינו את התשתית להטמעתן בקוד. הוספנו flag חדש להרצת heuristic online plan recognition, R-. כאשר מתקבל הflag הזה הקוד בקובץ prob_PR יכנס לקטע קוד המתאים להרצת heuristic online. למעשה מימשנו את אלגוריתם 4 במאמר, כאשר נותר רק לממש את ההיוריסטיקות RECOMPUTE ו-PRUNE.

ברמה הטכנית, במהלך העבודה על הפרויקט נתקלנו בקשיים לדבג את הקוד המקורי כיוון שמתבצעות המון קריאות לcmd. השתמשנו בסביבת עבודה pycharm בה אמנם ניתן לדבג קוד python אבל אין אפשרות לדבג פקודות שרצות בcmd. הפתרון שלנו לבעיה היה להמיר את הסקריפטים מהקוד המקורי בפונקציות המבצעות את הפעולה הרצויה, כך גם התאפשר לנו לדבג את הקוד, וגם ראינו שיפור בזמנים של הרצת הplanner עבור מספר בעיות הרצות במקביל. הצעה לשיפור תהיה להמיר את גם את שאר הסקריפטים לפונקציות.

נספחים

בניית הplanner הנאיבי

על מנת לבדוק את התוצאות המתקבלות מהרצת הplanners השונים נדרשנו לבנות planner שעובד בצורה נאיבית, כלומר עובד בהתאם לקובץ domain.pddl ללא שימוש בהיוריסטיקות נוספות.

הplanner הנאיבי מקבל היפותזה המכילה אטומים שונים, פרדיקטים. עבור כל אטום (פרדיקט) הplanner מוצא את הפעולות שעבורן האטום מופיע כאפקט של הפעולה ומכניס את התנאים המקדימים של הפעולה, שהם בעצמם פרדיקטים, לרשימה עליה רצים ברקורסיה לפי המתואר לעיל. במהלך הרקורסיה נרשמות כל הפעולות וכן כל הפרדיקטים בהם עברנו, ולאחר סיום הרקורסיה מתבצעת בדיקה אילו פעולות אכן התממשו על ידי מעבר על רשימת הפעולות שעברנו בהן במהלך הרקורסיה ובדיקה האם הפרדיקטים שמהווים את התנאים המקדמים שלהם מופיעים ברשימת הפרדיקטים שהתקבלה מהרקורסיה.

על פניו מימוש הplanner די טריוויאלי, אולם נתקלנו בשתי בעיות עיקריות- הראשונה, פעולות שונות בעלות שם זהה, והשנייה, קשתות אחורה בעץ הרקורסיה. שתי בעיות אלו גורמות לרקורסיה אין סופית ו/או תוצאות לא נכונות, במידה והplanner בחר בפעולה שונה מהפעולה אך בעלת שם זהה אליה.

על מנת להתמודד עם הבעית השמות הזהים בנינו מחלקה בשם Compressed_Action (בקובץ action.py). מחלקה זו יוצרת אובייקט של פעולה "מכווצת" המכיל רשימה של כל הפעולות בעלות אותו השם בdomain. התנאים המקדימים של אובייקט הפעולה ה"מכווצת" הם אוסף התנאים המקדימים של כל הפעולות הבודדות. האובייקט ה"מכווץ" מחליף את רשימת הפעולות הבודדות ברשימת הפעולות שעליה מריצים את הרקורסיה, כך שבמהלך הרקורסיה כל פעולה מופיעה פעם אחת בלבד. אחרי הרקורסיה, כאשר מבצעים את הבדיקה ממירים את האובייקט ה"מכווץ" ברשימת הפעולות המרכיבות אותו, כך שהבדיקה עצמה מתבצעת על כל הפעולות האפשריות עם אותו השם, ורק הפעולה שהתנאים המקדימים שלה אכן מומשו תוכל להתממש בעצמה.

את בעיית הקשתות אחורה בעץ הרקורסיה פתרנו בעזרת רשימה, אליה הכנסנו את כל הפעולות שנכנסו לרקורסיה ובדיקה אם פעולה המנסה להיכנס לרקורסיה כבר נבדקה או לא. פעולה שכבר עברה ברקורסיה לא תיכנס פעם נוספת, אולם נרשמת כפעולה שרצתה להתממש בשלב זה ויש לבדוק אותה גם במיקום הזה בשלב הבדיקה הסופית.

הplanner עובד בשני שלבים: שלב הרקורסיה, בו יש מעבר על הפעולות כפי שצוין לעיל תוך כדי ניהול רשימות פרדיקטים ופעולות שאליהן הרקורסיה הגיעה, ושלב הבדיקה, בו יש מעבר על רשימת הפעולות המתקבלות מהרקורסיה בסדר הפוך ובדיקה האם התנאים המקדימים שלהם התממשו. אם התנאים המקדימים מומשו, הפעולה נרשמת ברשימת הפעולות הסופית.

הסיבה למעבר הפוך על רשימת הפעולות היא מכיוון שאת הרקורסיה מתחילים במטרה (היפותזה) שהיא בעצם עלה של עץ הרקורסיה ומכיוון שברקורסיה אנו מחפשים את הפעולה שהובילה לפעולה שבה אנו נמצאים סדר הפעולות ברשימה הוא כך שהשורש של עץ הרקורסיה נמצא בסוף הרשימה. כך בעצם מתאפשר מעבר על העץ מהשורש אל העלים תוך עדכון של האפקטים של הפעולות.