

# Online plan recognition

גברת מור ורד

אלעד חמילבסקי | עידו בן אל

יולי 2017



## תוכן עניינים

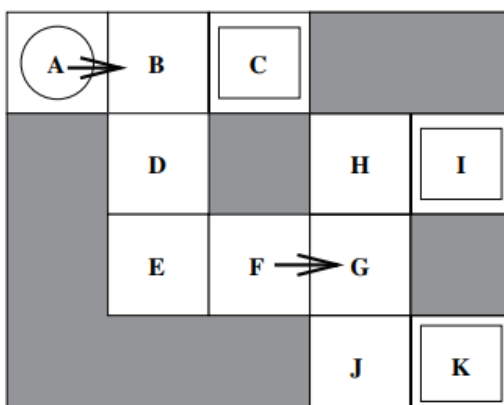
2	מבוא
3	רקע תיאורטי
5	רקע פרקטי – התקנה והרצה
5	הקוד המקורי ממאמר 3
5	חלק א - demo
5	חלק ב - prob-plan-recognition
5	שלב 1- הורדת הקוד וקימפול
5	שלב 2- הורדת planners
5	LAMA
6	HSPS
6	חלק ג - הורדת קבצי benchmarks
6	חלק ד - הרצת הקוד
7	הסבר על מהלך התוכנית
7	הקוד של הפרויקט
8	מבנה הקוד ותחזוקה
8	תיאור קבצי הקוד
9	UML
10	קבצים חשובים
10	קבצי PDDL
11	קובץ template
12	קובץ soln.
13	מהלך העבודה
14	הפיכה לonline
19	קבצי תוצאות
20	הרצת tests
20	עיבוד התוצאות
22	ניסויים והרצות
23	ייצוג התוצאות
25	הצעות לשיפור בעתיד

## מבוא

בפרויקט זה עסקנו במשימת plan recognition. plan recognition בפרט וplanning 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, B ו-F, 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 Ramirez Miquel ו- Hector Geffner. לצורך כך, נדרשנו ללמוד את תחום planning בכלל ואת המודל של Ramirez ו- Geffner בפרט, על ידי קריאת המאמרים הבאים:

1. Ramirez, Miquel, and Hector Geffner. **"Plan recognition as planning."** *Proceedings of the 21st international joint conference on Probabilistic Artificial 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. Ramirez, 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 planning- חישוב ישיר וחישוב מקורב ע"י שינויים קלים של אלגוריתמים אופטימליים והיוריסטיקות נוספות מעולם planning. 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 ספרייה.

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 Ramirez 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 וקימפול

הקוד ניתן להורדה מהאתר <https://sites.google.com/site/prasplanning/home>. יש להכנס ללשונית downloads ולהוריד מתקניית software את קבצי הtar של תיקיית ההרצה הראשית prob-plan-recognition, ושל obs-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 הקימפול 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 ב readme 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 executable search ולשנות את שמו ל- release-search. לבסוף יש להעביר את התיקייה (הראשית של planner) 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 flex ו bison (גרסאות חדשות יותר לא תואמות ל planner וגוררות שגיאת קומפילציה), אך יש אפשרות להשתמש באופציות מובנות של הפלאנר שנקראות fake-bison ו- fake-flex.  
 תחילה יש לחלץ את קובץ ה tar דרך הטרמינל (חילוץ אחר לא אפשרי) ולאחר מכן יש להעתיק מתת התיקייה locals את הקבצים config.h ו- makedefs ולישים אותם בתיקייה הראשית של ה planner. בנוסף, יש להעתיק את הקבצים scanner.linux.h ו- scanner.linux.cc ולהשמיט משמם את המילה linux ולישים 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 #define CPLEX\_INCREMENTAL לא בהערה ו- #define USE\_CACHE כן בהערה.

בקובץ ilb.cc יש לוודא כי #define ILA\_USE\_HS\_EXTERN ו- #define USE\_ CPLEX לא בהערה, ו- #define USE\_SCIP כן בהערה.

כעת יש לעשות התאמות כדי לאפשר שימוש בfake-bison ו fake-flex בקובץ makedefs (לפי המצוין בקובץ עצמו).

לצורך הרצת הפלאנר יש להשתמש בתוכנת CPLEX של IBM הניתנת להורדה בחינם עבור שימוש אקדמי.<sup>1</sup> על מנת להתקין את CPLEX במיקום הדיפולטיבי 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 הפלאנר יש צורך ללנקג' את התקיות הסטטיות ilcplex.a, libcplex.a, libconcert.a, libilcplex.a. אופן קימפול הפלאנר- 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.

## 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 planner על כל ההיפותזות. במהלך הריצה מעדכנים את עלות המסלולים וההסתברויות לייטכנותם  $P(G|O)$  (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 תתיקיות: 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: -O, -n, -online) 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 planners. כמו כן, קיימות בקובץ זה מחלקות נוספות הממשות מתודות לשימוש והרצה של planners השונים.

**prob\_PR.py** – הלולאה הראשית של התכנית- קביעת סוג planner שירוצ, 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 planner במצב offline ובמצב online.

**Main.py** – הרצה מלאה של התכנית כמקשה אחת (online, offline), 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 benchmark שמריצים. קובץ תוצאות זה מכיל כל המידע הרלוונטי שמתקבל מההרצה.

**translation.py** – יצירת קבצי pddl (pr\_domain, pr\_problem) ספציפיים להיפותזה (המתאימים לplanner) על פי קבצי המקור של benchmark.

**csv\_to\_output.py** – יצירת קובץ מסכם של תוצאות סטטיסטיות בפורמט csv לפי קבצי התוצאות בתיקיה.

**problems\_info.py** – מחלקת domain\_info. על ידי פירסור קבצי pddl מייצר מילונים עבור פעולות ופרדיקטים. כמו כן, המחלקה מכילה מתודות המממשות planner נאיבי שעובד לפי pddl.

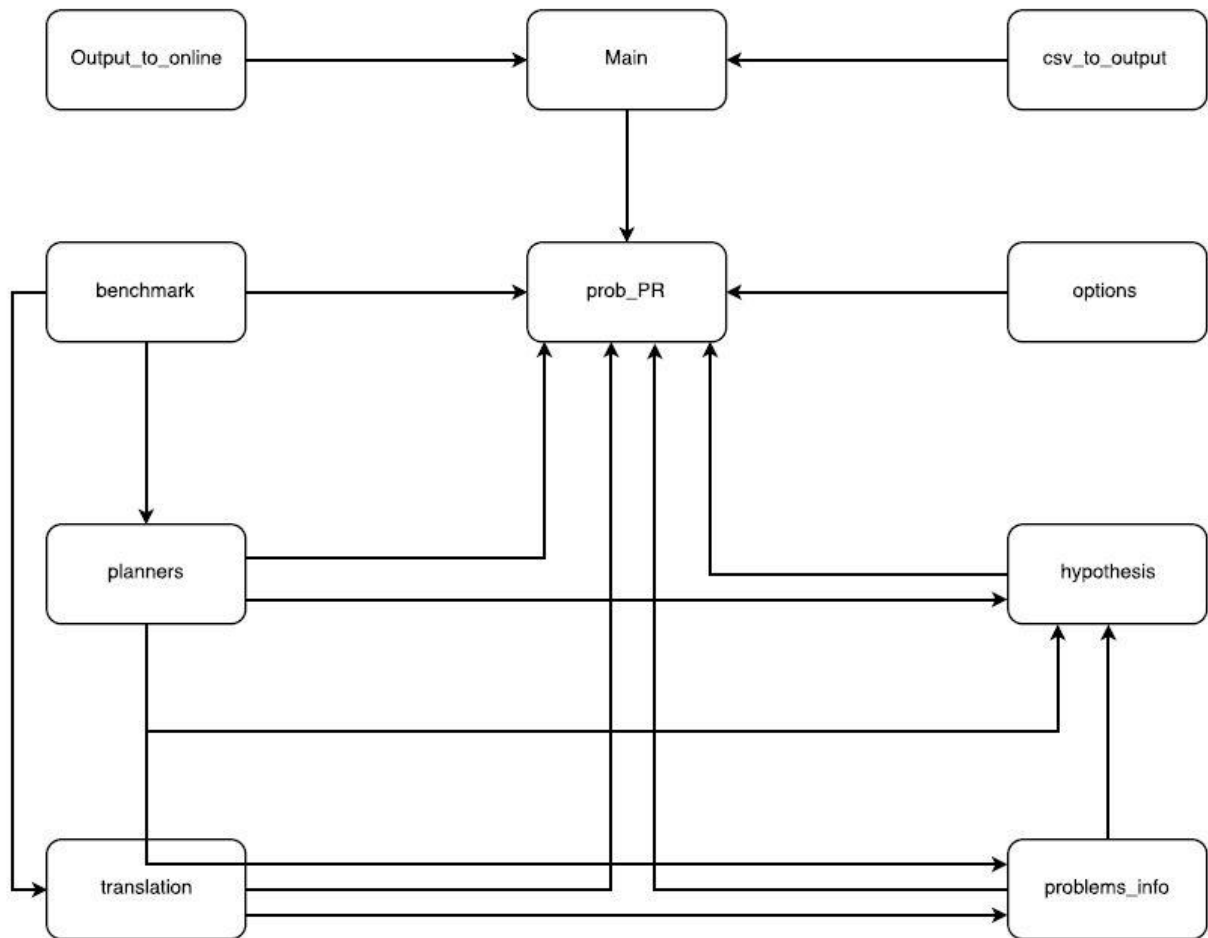
**action.py** – מכיל מחלקות של פעולה (action) ופרדיקט (predicate).

**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



## 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 הdomain, 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 הdomain 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 template**

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>
    )
  )
)
```

## קובץ soln.

קובץ הנוצר על ידי הפלאנר HSP ומכיל את המסלול שבחר הפלאנר, 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 benchmarks נדרשת 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 transcription 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 tar 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 אנו מוחקים את התיקייה הזמנית. כך מקובץ tar המכיל המון מידע לא רלוונטי "זיקקנו" 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 מצריך את חישוב 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 מופיע בתיקיית הקוד קובץ soln. שחשדנו שבו ימצא המסלול, אך כשפתחנו אותו מצאנו שקובץ ריק (קובץ ריק פירושו כישלון של ריצת הplanner). לאחר מספר הרצות שבהן לא קיבלנו נתונים בקובץ החל 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 הפלאגר אכן עבד ובקובץ soln הופיע המסלול שאותו חיפשנו.

לאחר שוודאנו שקובץ הפתרון עבור HSPS תקין ונותן את הדרוש, רצינו להוסיף לקובץ הפלט החדש שלנו את היחס שבין costs של מסלול האידיאלי לcosts של המסלול המתקבל הנותר. לצורך כך 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 LAMA 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 log שנמצאים ב-tar 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 benchmark שעליו נרצה את ההרצה. יש לציין שקובץ benchmark צריך להיות באותה תיקייה שבה נמצא הקובץ Main.py.

## הפיכה ל-online

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 בפלאג -n. כאשר פלאג זה מתקבל בקלט התכנית תשתמש בלולאת הרצה המתאימה לצורת פעולה של online ואילו כערך דיפולטיבי תישאר הרצה ב-offline. בחרנו לממש כך על מנת לא לחסום אפשרויות הרצה קיימות וכן הרצה ב-online עבור כל מצב פעולה אפשרי.

כאשר מתקבל הפלאג -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) אנו יוצרים את קובץ הבעיה. מכיוון שאלגוריתם online מקבל כל פעם תצפית חדשה (על פי הלולאה בקובץ prop\_PR.py) אנו מוודאים שמספר 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 counter באחד ונקרא לפונקציה modify\_obs\_file. הפונקציה קוראת את קובץ התצפיות ויוצרת קובץ תצפיות חדש בעל מספר תצפיות כגודל counter (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 benchmark) 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 ההיפותוזות 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': [action.Action instance at 0x7f8472b0e4d0>], 'BREAK-INTO_ANDROMEDA': [action.Action instance at 0x7f8472fbaab8>]}
  _len_ = {int} 91
▼ 'BREAK-INTO_ANDROMEDA' (140206841436464) = {list} <type 'list'>: [action.Action instance at 0x7f8472fbaab8>]
  _len_ = {int} 1
  0 = {instance at 0x7f8472fbaab8> 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

```

אנו מייצרים אובייקט של הדומיין הנ"ל פעם אחת בתחילת התכנית ושומרים אותו בתוך האובייקט של ההיפותזה על מנת לאפשר גישה נוחה אל הdomain 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 translatein.py המתרגם את קבצי domain.pddl והתבנית template לכדי בעיה ספציפית. לאחר ניסוי וטעייה מצאנו שלמרות שהדרך המתבקשת היא להכניס את הפרדיקטים של התצפית לקובץ templaten כדי שמההתחלה יוצרו קבצים מעודכנים, הדבר גורר שגיאה (ככל הנראה פגיעה בפורמט שה-planner 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 התיקיה שנוצרה, אם שם תת התיקיה הוא O (כיוון שאנו רוצים לעדכן את הקובץ עבור ריצה עם התצפיות ולא בלעדיו) קרא את הקובץ pr\_problem.pddl, 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 init 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\_EXPLAINED\_FULL\_OBS ל-EXPLAINED\_FULL\_OBS.

המשך ריצת התכנית זהה להרצת הקוד המקורי.

על מנת לוודא שה-planner שהרצנו אכן מצא את המסלול הנכון בנינו planner שפועל בצורה נאיבית על הdomain. לשם כך יצרנו את המחלקה Domain\_info (המוזכרת לעיל). בעת יצירת אובייקט של המחלקה מתבצע מיפוי הdomain 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 שה-planner המקורי אכן הגיע לתוצאה הרצויה. מאחר וה-planner הנאיבי לא משתמש ב-planner חיצוני אלא רק עובר על ה-pddl לא השתמשנו בו להמשך התכנית ולכן גם לא הרחבנו עליו בספר הפרויקט, אולם צירפנו לתיקיית הפרויקט את קוד מבואר.

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 ה-planer על התצפיות (online) תשלים את הפער בין המסלול האידאלי למסלול הנותר. לצורך ההרצה על התצפיות נדרשנו לייצר קבצי pddl חדשים בהם ה-goals יהיה ה-preconditions 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 planer 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 הפלאנר>, 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 planner.

**heap\_restriction** - מגבלת המקום על ריצת planner.

**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:

$$cost(0) - cost(neg\_0)$$

**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 planner על ההיפותזה עם התצפיות.

**hyp\_plan\_time\_not\_O** - זמן ריצת planner על ההיפותזה ללא התצפיות.

**hyp\_trans\_time** - זמן תרגום הבעיה לקבצי pddl ספציפיים.

**hyp\_plan\_time** - זמן ריצת planner 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 `benchmarks` 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` מחושב על ההיפותזה הנכונה (`real_hyp`) כך שסוכם את מספר הפעמים שההיפותזה אכן דורגה במקום הראשון ביחס לכלל ההיפותזות, מתוך כלל התצפיות (התוצאות עוגלו לחמש ספרות אחרי הנקודה).

```
# 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 היפותזה 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 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 counter. 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 benchmark 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 כ-11 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 מטריצינית. כל שורה מייצגת תצפית, וכל עמודה מייצגת היפותזה. להלן מספר דוגמאות, כאשר לכל דוגמה מובאת ההצגה כמו שהיא בקובץ, וכן הצגה גרפית.

### דוגמה 1:

עבור הבעיה intrusion-detection\_p10\_hyp-5\_50\_2.

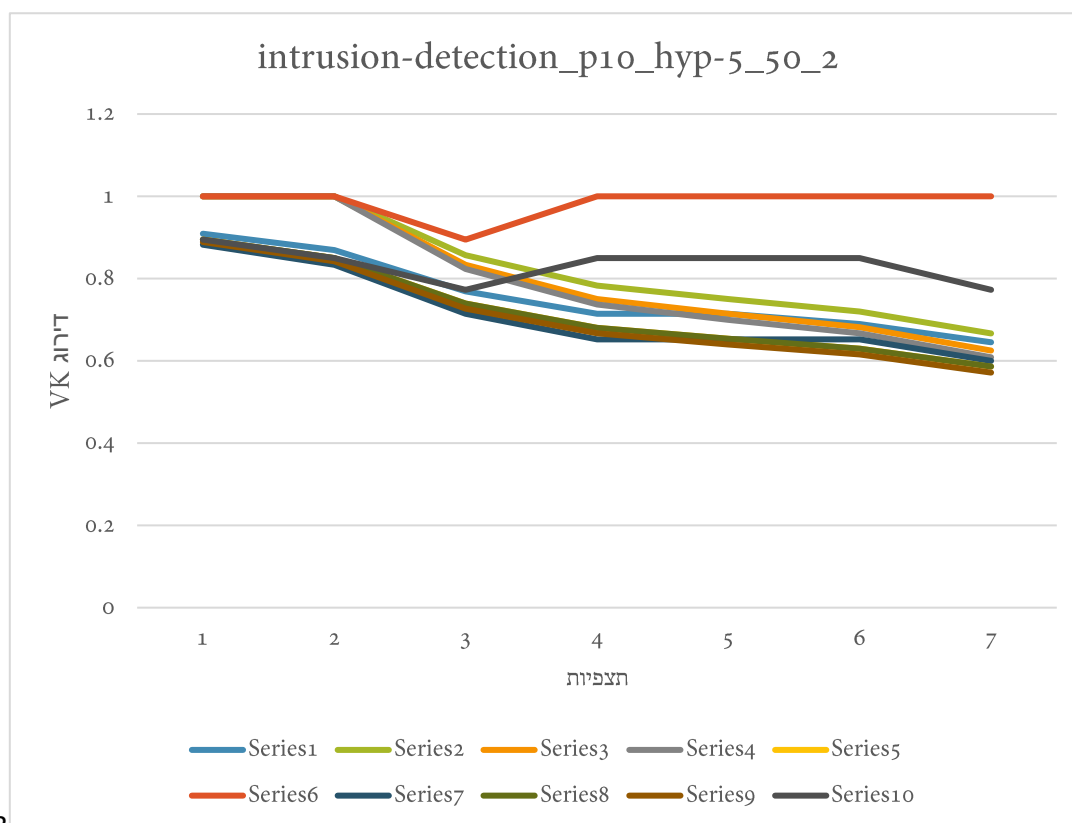
ייצוג טבלאי (כמו שמופיע בקובץ הפלט):

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

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 11:

עבור הבעיה intrusion-detection\_p10\_hyp-1\_70\_1.

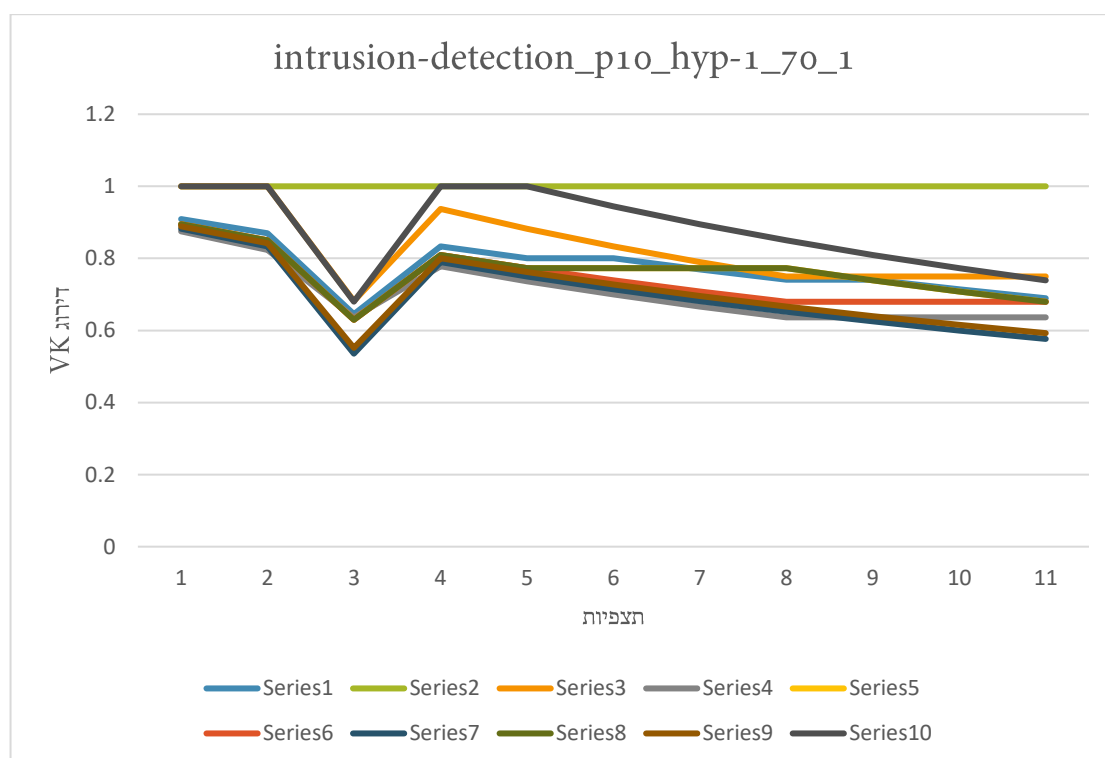
ההיפותזה האמיתית כאן היא ההיפותזה השנייה.

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

ייצוג טבלאי:

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

ייצוג גרפי:



## הצעות לשיפור בעתיד

במאמר של ורד וקמינא שעתידי 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 flag 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 לפונקציות המבצעות את הפעולה הרצויה, כך גם התאפשר לנו לדבג את הקוד, וגם ראינו שיפור בזמנים של הרצת planner עבור מספר בעיות הרצות במקביל. הצעה לשיפור תהיה להמיר את גם את שאר הסקריפטים לפונקציות.