

המחלקה להנדסת תוכנה

פרויקט גמר – ה'תשפ"ד

אלגוריתמי תזמון עבור חישוב מקבילי בסביבה הטרוגנית

Task Scheduling For Parallel Computation In Heterogeneous Systems

מאת

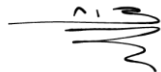
אליה אטלן

318757200

אביב זבולוני

211313333

תאריך: 14\04\2024



אישור:

מנחה אקדמי/ת: ד"ר צור לוריא

מערכות ניהול הפרויקט :

#	מערכת	מיקום
1	מאגר קוד	Repositories (github.com)
2	קישור ליומן	יומן פגישות
3	קישור לסרטון דוח אלפא	דוח אלפא mp4

מידע נוסף (מחקו את המיותר)

סוג הפרויקט	1. מחקרי ממצא במכללה 2. תעשייתי חברת hi-tech
פרויקט ממשיך	פרויקט המשך של שיר גולה שכותרתו "אופטימיזציית המתזמן עבור מחשוב מקבילי בזמן אמת". פרויקט זה בא להציע גישה שונה לפתרון אותה הבעיה ששיר ניסתה לפתור

תוכן עניינים

4.....	נאום המעלית או תקציר (רשות)
5.....	מבוא
6.....	תיאור הבעיה
6.....	דרישות ואפיון הבעיה
6.....	הבעיה מבחינת הנדסת תוכנה
7.....	סקירת עבודות דומות בספרות והשוואה (לפרויקט מחקרי בלבד)
8.....	תיאור הפתרון
9.....	המשך הפרק תיאור הפתרון (פרויקט מחקרי יכול להוריד את הסעיף הזה באישור המנחה)
10.....	מה עשינו עד כה?
11.....	תכנית בדיקות (פרויקט מחקרי יכול להוריד את הסעיף הזה באישור המנחה)
12.....	נספחים
12.....	טבלת סיכונים
12.....	רשימת/טבלת דרישות
13.....	תכנון הפרויקט – ברזולוציה של שבועיים
13.....	רשימת ספרות (ביבליוגרפיה)

נאום המעלית

בעולם הטכנולוגי המודרני, מופיעים קשיים חדשים שדורשים פתרונות חדשניים. אחד מהם הוא הצורך לתזמן\לשבץ מספר רב של משימות התלויות זו בזו על מספר מעבדים שונים (גם נקרא סביבה הטרוגנית).

בימינו, התעשייה משתמשת באלגוריתמים פשוטים שמשבצים משימות למעבדים באופן שרירותי.

בפרויקט שלנו, נציע אלגוריתמים חדשניים לתזמון מעבדים בסביבה זו. באמצעות אלגוריתמים אלו, אנו מתכננים למזער את זמן הריצה הכולל של המערכות.

מבוא

בעיית תזמון מעבדים היא בעיה ידועה ונפוצה בתעשייה שמתעסקת בבניהול ובשיפור שימוש המעבדים במערכות תוכנה. הבעיה עוסקת בניסיון לתזמן משימות למערכות שמכילות מספר מעבדים, על מנת לייעל את המערכת על פי קריטריונים שונים לפי דרישות המערכת.

עולם בעיות התזמון הוא רחב, וכל בעיה בעולם זה אפשר לתאר על ידי שלושת הקריטריונים הבאים:

המעבדים:

1: * רק מעבד אחד, עליו רצים כל המשימות.

P: * מספר מעבדים מאותו סוג, כאשר מספר המעבדים הוא P

Q: * מספר מעבדים מסוגים שונים (Q סוגי מעבדים)

R: * מספר מעבדים מסוגים שונים, כאשר זמן הריצה של כל משימה תלוי במעבד.

וכולי.

אילוצים על תזמון המשימות:

prec: * תחילת ריצה של משימה יכולה להיות תלויה בהשלמת משימה\משימות אחרות.

due date: * על משימות מסוימות להסתיים עד זמן נתון.

release time: * משימות מסוימות יכולות לרוץ רק לאחר שמאורע מסוים קרה.

processing time: * זמני הריצה של המשימות, יכולים להיות כולם שווים, או חלק מקבוצה מסוימת.

ובולי, אנחנו נתמקד באילוצים מסוג prec, כאשר ה processing times שייכים ל R^+

המטרה:

C_{max} * מזעור הזמן מתחילת המשימה הראשונה, ועד סיום המשימה האחרונה

C_j : $\sum_{j=0}^P$ * מזעור סכום זמני הריצה אילו היה ברשותנו רק מעבד אחד

אנחנו נתמקד ב C_{max} .

כל הווריאציות ומידע עליהן, נמצא ב[1]

נוכל לחשוב על המשימות כאילו הן מסודרות בתוך גרף מכון ללא מעגלים (DAG) כאשר כל צומת היא משימה, וכל קשת מסמלת את התלות בין המשימה שהקשת יוצאת ממנה,

למשימה אליה היא נכנסת. תלות זו מתבטאת בכך שמשימה שתלויה במשימות אחרות צריכה לחכות לסוף הריצה של כלל המשימות שהיא תלויה בהן.

אלגוריתמי online ואלגוריתמי offline:

בעת חישוב סדר המשימות כפי שהאלגוריתם מתאר, ניתן להוסיף אילוץ על זמן ריצת המתזמן.

אלגוריתם: online דורש זמן חישוב מאוד מהיר כך שיהיה זניח לעומת זמן הריצה הכולל של המשימות (על מנת שלא יפגע בביצועי המערכת), מפני שהתזמון מחושב בזמן ריצת המשימות, בכל פעם שהן צריכות להשתבץ למעבד.

אלגוריתם: offline מבצע חישוב מראש של התזמון שיכול לערוך זמן רב (ביחס למשימות להרצה), ולבסוף מחזיר את רשימת סדר הריצות של המשימות על כל המעבדים.

מאז שנות ה-90, עולם המחשוב המבוזר צמח. עקב כך התפשטו רשתות מחשבים גדולות ומורכבות יותר, בנוסף לשימוש בארכיטקטורות client-server.

תהליך זה הביא לצורך גדול יותר בתזמון מעבדים, שכן המערכות המורכבות האלה דרשו תיאום וניהול יעיל של משאבי המערכת. על מנת למקסם את רווחיהם, חברות הענן שאפו (ושואפות) "לשבץ" כמה שיותר משימות בזמן הקצר ביותר (מזעור זמן ה-idle של המעבדים).

בנוסף להתפתחות הענן, התפתחות הלמידה העמוקה באמצעות רשתות נוירונים עמוקות

דרש פיתוח מעבדים מסוג חדש, המעבדים החדשים הותאמו עבור עיבוד מרובה משימות (כגון עיבוד מקבילי של מידע בכמויות עצומות). התפתחות הגרפיקה הביאה להפצת המעבדים החדשים הללו שתמכו בעיבוד גרפי. עם הלמידה העמוקה, עלה הצורך בתזמון מעבדים מסוגים שונים המותאמים למשימות ספציפיות על מנת לתמוך במטלות שדורשות חישובים מורכבים, מקביליים ומהירים.

הפרויקט שלנו הוא גם פרויקט תעשייתי, בשיתוף עם חברת Mobileye והמנחה התעשייתי יוסי קרייניו. בנוסף למערכות ענן, למערכות הפועלות בזמן אמת כמו אלה המפותחות על ידי Mobileye (העוסקת בפיתוח מערכות נהיגה אוטונומית) עולה הדרישה לאלגוריתמים יעילים שיתזמנו משימות בזמן מינימלי, וישתמשו בחומרה ייעודית המתאימה למשימות ספציפיות. במערכות כמו של Mobileye, שעוסקות בעיבוד תמונה וקבלת החלטות בזמן אמת, חוץ מדרישה למהירות ביצוע מרבית, ישנו גם צורך באמינות ובדיוק גבוה. אלגוריתמי תזמון ישפרו את יכולת המערכת לתפקד בזמן אמת.

כיום בתעשייה משתמשים באלגוריתם תזמון חמדן. אלגוריתם זה משבץ משימה מוכנה למעבד פנוי באופן שרירותי. האלגוריתם החמדן הוא $1 + P$ מקרב בוריאציה הבעיה שלנו, כאשר P הוא מספר סוגי המעבדים. בנוסף לכך החמדן חסכן במשאבים מכיוון שהוא לא מתחזק מבנה נתונים מלבד תור רשימה של משימות מוכנות.

תיאור הבעיה

אנחנו ננסה לתת מענה ל 3 שאלות מרכזיות במהלך הפרויקט:

שאלה 1:

האם קיים אלגוריתם קירוב יותר טוב מהחמדת לתזמון מעבדים בסביבה הטרוגנית (מעבדים מסוגים שונים)? ננסה להתקרב לאלגוריתם האופטימלי (התיאורטי) כמה שיותר.

אפילו בעיית התזמון על שני מעבדים זהים עם משימות בעלי זמני ריצה שונים היא NP קשה (זו בעיית Partition). מכאן שניתן לבנות רדוקציה מווריאציה זו לכל ווריאציה אחרת של הבעיה, ולכן גם הבעיה שלנו היא NP קשה.

לכן לא ננסה למצוא אלגוריתם אופטימלי לבעיה. במקום, ננסה להתקרב כמה שיותר לפתרון האופטימלי.

בנוסף לכך, הוכח [2] שעבור בעיית תזמון מעבדים בסביבה הומוגנית (כאשר כל המעבדים מאותו סוג), לא ניתן לשפר את הקירוב של החמדת [3] (2-מקרב) בזמן פולינומי. ננסה להשליך על הבעיה שלנו בסביבה הטרוגנית. האם האלגוריתם החמדת הוא 2-מקרב גם במקרה שלנו? או אולי יותר גרוע? ואם כן, האם קיים אלגוריתם קירוב יעיל יותר?

שאלה 2:

האם קיים\קיימים אלגוריתמים טובים יותר מהחמדת על קלטים\בעיות מהעולם האמיתי? על מנת לענות על שאלה זו, נצטרך לפתח אלגוריתמי תזמון על ידי שימוש בהיוריסטיקות שונות על מנת לנסות לייעל את החמדת כאשר זמני

הריצה והתלויות בין המשימות ידועים מראש.

את נושא זה נוכל לחקור יותר לעומק באמצעות העזרה של Mobileye, אשר יספקו לנו דוגמאות מהעולם האמיתי. בנוסף לכך, נפתח סימולטור שיסמלץ סביבה הטרוגנית כללית על מנת לבחון אלגוריתמים שונים ולהשוות ביניהם.

שאלה 3:

מהו האלגוריתם הכי טוב לחומרה הספציפית של Mobileye?

בנוסף לסימולציה שנפתח בשלב השני (שאלה 2), נוסיף אילוצים על האלגוריתם כך שיתאים למתזמן של Mobileye (אשר הוטמע בחומרה) שבשימוש כיום. כיום Mobileye משתמשים במתזמן חמדן שעובד עם רשימות של משימות בעלי 2 רמות דחיפות, ומשבץ מהתור הדחוף למעבד זמין (אם התור הדחוף ריק, אז הוא מתזמן מהתור הפחות דחוף).

כך נוכל להשוות את האלגוריתם החמדן ש Mobileye מריצים כיום עם האלגוריתמים שנפתח.

לאחר שנענה על שאלות 2,3 נוכל להציע ל Mobileye את השיפור בתוכנה (אם קיים).

דרישות ואפיון הבעיה

הדרישות של הלקוח (Mobileye) הן:

- (1) אלגוריתם שנותן תזמונים יותר טובים מאשר האלגוריתם החמדי הקלאסי שבשימוש כיום. אפילו אם זה רק משפר את המקרה הממוצע, ולא את המקרה הגרוע.
- (2) אלגוריתם שאפשר להטמיע בחומרה הקיימת.

התוצר ש Mobileye יקבלו יהיה מספר אלגוריתמים שונים, שנפתח בשפת Python בנוסף ל Pseudo-Code שממנו פותחו האלגוריתמים.

הבעיה מבחינת הנדסת תוכנה

הפרויקט דורש:

- (1) מציאת חסמים על זמני ריצה של אלגוריתמי תזמון.
- (2) פיתוח אלגוריתמי תזמון חדשים.
- (3) מציאת היוריסטיקות על מנת לייעל את האלגוריתמים הקיימים.
- (4) פיתוח מערכת סימולציה על מנת להריץ ולבחון את האלגוריתמים השונים.

סקירת עבודות דומות בספרות והשוואה) לפרויקט מחקרי בלבד)

[The scheduling zoo](#) (1)

[Svensson, Ola. "Hardness of precedence constrained scheduling on identical machines." SIAM Journal on Computing 40.5 \(2011\): 1258-1274.](#) (2)

[Graham, Ronald L. "Bounds for certain multiprocessing anomalies." Bell system technical journal 45.9 \(1966\): 1563-1581.](#) (3)

[Pinedo, M. L. \(2012\). Scheduling \(Vol. 29\). New York: Springer.](#) (4)

תיאור הפתרון

הפתרון דורש לענות על שלושת השאלות המרכזיות בפרויקט:

שאלה 1:

ננתח זמני ריצה, נוכיח חסמים על ידי בניית (קונפיגורציות) שנמצא ובעזרת היוריסטיקות ננסה למטב את החמדן ה"טיפש".

שאלה 2:

ניקח קלט מהעולם האמיתי של Mobileye תספק לנו, ונפתח אלגוריתמים שיכולים לייעל את זמן הריצה של החמדן על הקלטים שנלקחו מהעולם האמיתי.

שאלה 3:

ניעזר באלגוריתמים שפיתחנו בחלק השני של הפרויקט (שאלה 2) על מנת לפתח אלגוריתם

שיתאים לחומרה הקיימת במערכות של Mobileye.

נפתח סימולטור על מנת לבחון את האלגוריתם שלנו אל מול האלגוריתם החמדן הקיים באמצעות מידע קיים של Mobileye. במידה ומצאנו אלגוריתם יותר טוב, נציע אותו ל Mobileye.

המשך הפרק תיאור הפתרון

הפרויקט שלנו מסתמך מאוד על הסימולציה שיצרנו לתזמון המעבדים.

קודם כל רשמנו פסאודו-קוד שיסמלץ הרצת מעבדים בסביבה הטרוגנית (ראו נספח [1]).
(בשביל דוגמת הרצה ראו נספח [5])

לאחר מכן היינו צריכים לחשוב איך נסמלץ מעבדים ומשימות, לשם כך יצרנו את המחלקות Task, Processor אשר מייצגות משימות ומעבדים, וכל המידע שדרוש על מנת להריץ את הפסאודו-קוד (ראו נספח [2])

איך השגנו דאטא?

הפרויקט הוא בשיתוף עם Mobileye, ולכן נציג מ Mobileye נתן לנו גישה להמון דאטא. עד שהדאטא הגיע, עבדנו עם קבצים json, ולכן כשהדאטא הגיע רשמנו קוד במחלקה ששמה Parser, שהמטרה שלה היא לנתח את הקבצים הללו ולהפוך אותם לפורמט ה json שהשתמשנו בו. (דוגמא ל input בנספחים למטה ראו נספח [3])

לאחר מכן, מכיוון שהפרויקט עצמו דורש לבנות אלגוריתמים חדשים, יצרנו מחלקה ששמה Algorithm כאשר כל אלגוריתם שנממש ירש ממחלקה זו. למחלקה הזו יש פונקציית decide, ובעצם זה הלב של המתזמן והיא מה שקובעת איך המשימות תשובצנה. (ראו נספח [4])

לבסוף, כתבנו את המחלקה Sim שקודם כל קוראת את הדאטא, ולאחר מכן מריצה את מימוש הפסאודו-קוד.

מה עשינו עד כה?

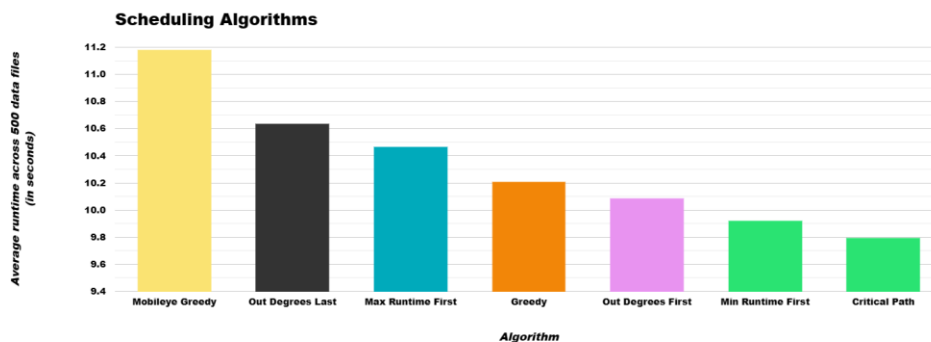
- מציאת חסמים עליונים ותחתונים לאלגוריתם החמדה הבסיסי בסביבה הטרוגנית
- פיתוח פסאודו-קוד להרצת הסימולציות ע"מ לבחון את האלגוריתמים השונים
- פיתוח ב-Python את מערכת הסימולציה בהשראת הפסאודו-קוד
- הוספת אילוסטרציה גרפית כדי שנוכל לנתח את זמני הריצה ולממש את היוריסטיקות
- דיון ומחקר על היוריסטיקות\אלגוריתמים שבדאי לבדוק
- מימוש האלגוריתמים
- הרצת האלגוריתמים על כל ה-Data ובדיקת ממוצע זמני הסיום לצורך השוואה.
- דיון וחשיבה על ההיוריסטיקה המבטיחה critical path
- מימוש critical path והרצתו
- הצעת ודיון על יוריסטיקות\אלגוריתמים נוספים
- הצגת התוצאות עד כה לנציג מובילאי
- הצעת אלגוריתם אופטימלי (מתוך אלו שחקרנו) עבור החומרה הקיימת במערכת מובילאי
- במידה וה critical path (שהוא אלגוריתם offline) משפר בהרבה את זמן הריצה נשקול לפתח מתזמן שיתזמן לפי critical path ו clustering ע"מ לאפשר את הרצת התזמון בזמן אמת (online))

תוצאות ומסקנות:

להפתעתנו, כפי שניתן לראות בגרף, האלגוריתם החמדה של מובילאי הוא בעל הביצועים הנמוכים ביותר.

בנוסף לכך רק האלגוריתם Critical Path הוא אלגוריתם offline, וכל השאר הם אלגוריתמי online, שכן במצופה מאלגוריתם offline שיש לו זמן לחשב מסלול יותר אופטימלי מכל השאר.

השוואת אלגוריתמים לפי זמן ריצה כולל:



להלן השוואת האלגוריתמים ביחס לאלגוריתם החמדן המסורתי:

	Mobiley e Greedy	Out Degrees Last	Max Runtime First	Greedy	Out Degrees First	Min Runtime First	Critical Path
% Runtime From Greedy	~109.5%	~104.2%	~102.5%	100%	~98.8%	~97.2%	~95.9%
% Faster	~8.7%-	~4%-	~2.4%-	0%	~1.2%	~2.9%	~4.2%

נספחים

[1] פסאודו-קוד לסימולציה:

(1) מכניסים את כל המשימות המוכנות לתור: `ready_tasks`
עוברים על כל המעבדים וכל ה `ready_tasks` עד שניתן לשבץ משימה למעבד פנוי
מעדכנים שהמעבד "עסוק"
מכניסים את המשימה ל `current_tasks`
מעדכנים שה `end_time` של המשימה הוא `current_time + duration`
מוחקים את המשימה מ `ready_tasks`
(2) כל עוד יש משימות שרצות:
`task = current_tasks.pop()`
`current_time = task.end_time`
`task.update_in_degrees()`
לבסוף, בצע את (1)

[2] תכונות Task, Processor

```
1 class Processor:
2     def __init__(self, name: str, processor_type: int):
3         self.name = name
4         self.type = processor_type
5         self.idle = True
6         self.current_task = None
7         self.work_order: List["Task"] = []
```

```
1 class Task:
2     def __init__(
3         self,
4         name: str,
5         duration: float,
6         processor_type: int,
7         priority: int,
8         blocking: List["Task"],
9         blocked_by: List["Task"],
10    ):
11         self.name = name
12         self.processor_type = processor_type
13         self.duration = duration
14         self.end_time = 0
15         self.processed_by: Processor = None
16         self.priority = priority
17         # whether a task has stopped running
18         self.done = False
19
20         # In and Out degrees
21         self.blocking = blocking
22         self.blocked_by = blocked_by
```

[3] דוגמא ל input:

```
1  "Tasks": {
2    "A": {
3      "duration": 3,
4      "processor_type": 1,
5      "priority": 1,
6      "blocking": [
7        "B",
8        "C"
9      ]
10   },
11   "B": {
12     "duration": 2,
13     "processor_type": 2,
14     "priority": 0,
15     "blocking": [
16       "D",
17       "E"
18     ]
19   },
20 }
```

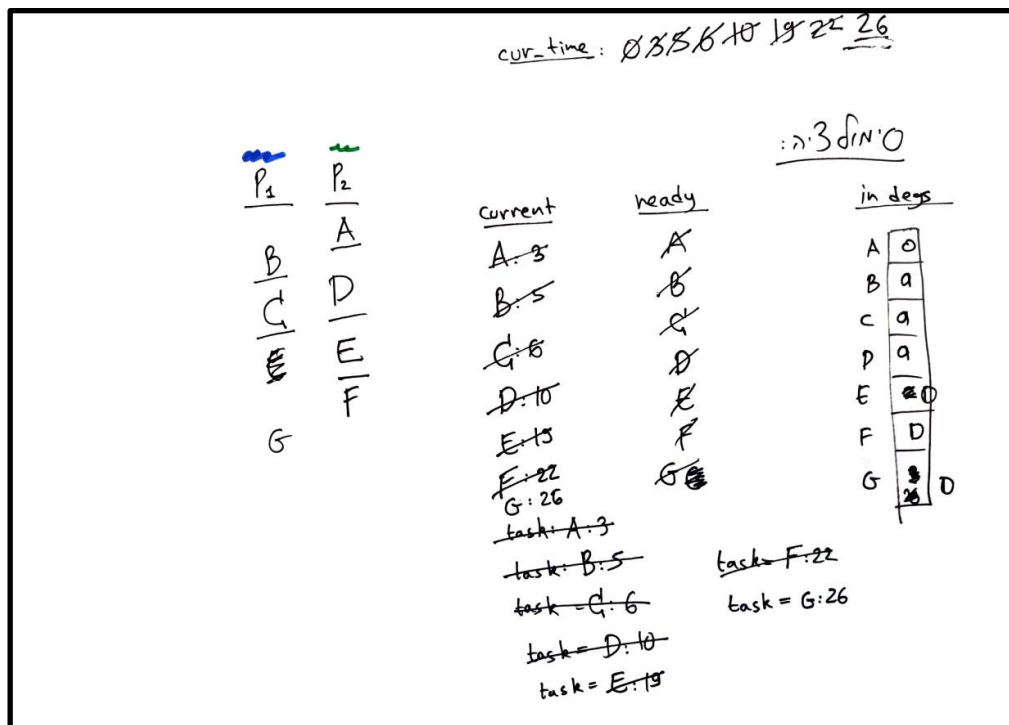
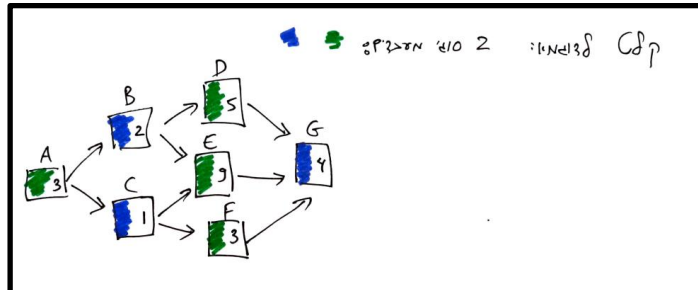
[4] המחלקה Algorithm והפונקצייה decide:

```
1  class Algorithm:
2    def __init__(
3      self,
4      ready_tasks: List["Task"],
5      processors: List["Processor"],
6      all_tasks: List["Task"],
7    ):
8      self.processors = processors
9      self.ready_tasks = ready_tasks
10     self.all_tasks = all_tasks
11
12     def update_lists(self, processors, ready_tasks, all_tasks):
13       self.processors = processors
14       self.ready_tasks = ready_tasks
15       self.all_tasks = all_tasks
16
17     # returns the order of tasks that the algorithm decided we should iterate over
18     def decide(self) -> List["Task"]:
19       pass
```

האלגוריתם שבשימוש כיום Greedy:

```
1  class Greedy(Algorithm):
2    def __init__(
3      self,
4      ready_tasks: List["Task"],
5      processors: List["Processor"],
6      all_tasks: List["Task"],
7    ):
8      super().__init__(ready_tasks, processors, all_tasks)
9
10     # simply returns the order that was given, since the greedy doesn't care
11     def decide(self):
12       result = sorted(self.ready_tasks, key=lambda task: task.priority)
13       return result
```

[5] דוגמת הרצה :



תכנון הפרויקט – ברזולוציה של שבועיים

פגישת היכרות עם הנציג מ Mobileye והצגת הסימולציה והתוצאות עד כה	23.04
פגישת המשך עם צור ודיון על בחירת האלגוריתם האופטימלי עבור Mobileye	30.4