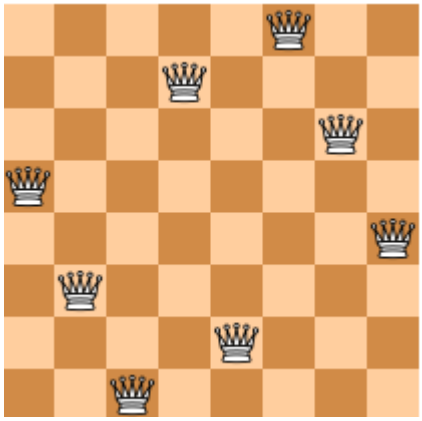
איתמר לרדו 311547087
עמית טויטו 208445866

Solving n-Queens with Tabu Search

הבעיה

בעבודה זו בחרנו לפתור את בעיית ה-n-queens בעזרת אלגוריתם האופטימיזציה Tabu Search.

N-Queens



בעיית שמונה המלכות היא הבעיה של הצבת שמונה מלכות שחמט על לוח שחמט בגודל 8×8 כך שלא קיים זוג מלכות כך שהן מאיימות אחת על השניה; כלומר, **פתרון לבעיה מחייב שאף שתי מלכות לא חולקות את אותה שורה, עמודה או אלכסון**.הבעיה הוצגה לראשונה באמצע המאה ה-19. בעידן המודרני, הוא משמש לעתים קרובות כבעיה לדוגמה עבור טכניקות תכנות שונות כמו תכנות עם אילוצים, אלגוריתמים גנטים ואופטימיזציות.

בעבודתנו נתייחס למקרה הכללי של בעיה זו שהוא בעיית n המלכות של הצבת n מלכות לא מאיימות על לוח שחמט n × n. לבעיה זו קיימים פתרונות לכל המספרים הטבעיים חוץ מ- n = 2 ו-n = 3. נכון להיום, לא קיימת נוסחה מדויקת למציאת מספר הפתרונות עבור כל n בהצבת n מלכות על לוח בגודל n × n , אך כן ידוע המספר המדויק של פתרונות לכל n≤27.

Tabu Search

אלגוריתם חיפוש טאבו נוצר על ידי פרד וו גלובר בשנת 1986. האלגוריתם הוא שיטת חיפוש מטא-אוריסטית תוך שימוש בחיפוש מקומי. האלגוריתם בשימוש רחב בתחום האופטימיזציה, בבעיות מתמטיות ובתחומים רבים אחרים במדעי המחשב.

אילו בעיות ניתן לפתור באמצעות אלגוריתם זה?

האלגוריתם שפיתח פרד וו גלובר אינו מקובע תבניתית כלפי בעייה אחת או אחרת ולכן ניתן לשימוש בפתרון בעיות רבות. הדרישה היחידה של האלגוריתם הוא שניתן שיהיה ליצור ייצוג של הבעיה על גרף באופן כזה שפתרון הבעיה הוא מעבר כלשהו על קודקודים שהם מצבים שפותחו הניתנים לבדיקה על ידי האלגוריתם.

בחיפוש סיסטמתי אנחנו עורכים חיפוש עמוק ויסודי ובוחרים את הפתרון הטוב ביותר שהגענו אליו. אנו מבטיחים שלמות ואופטימליות לפתרון. אולם, החיפוש אורך זמן רב, צורך הרבה מקום ולפעמים בלתי אפשרי בבעיות גדולות או בעיות הדורשות פיתוחים רבים. לכן, בשונה מחיפוש סיסטמתי אנו נוכל לבצע חיפוש מקומי שהוא חיפוש חלקי אשר משתמש בפונקציות היוריסטיות לעמידת טיב הפתרון. ישנם אלגוריתמים רבים שהם מסוג של חיפוש מקומי כמו

איתמר לרדו 311547087
עמית טויטו 208445866

Hill Climbing, simulated annealing, gradient descent וכן, גם חיפוש טאבו הוא מסוג של חיפוש מקומי.

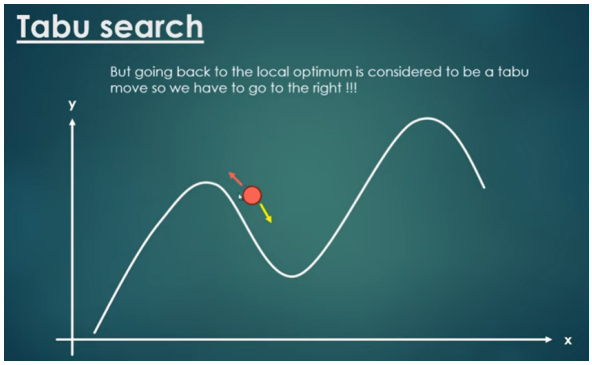
כפי שהוזכר בקורס וכן בקורסים נוספים, בעיה נפוצה באלגוריתמים של חיפושים מקומיים היא הישארות בנקודת קיצון מקומית. זאת אומרת, האלגוריתם מוצא נקודת קיצון, כל הצעדים סביבו פחות טובים ממיקומו הנוכחי ולכן האלגוריתם בוחר לא להמשיך, לעצור ולהחזיר את הפתרון.

בכך, אנו מקבלים פתרון שהוא לא אופטימלי, כי לא הגענו לנקודות קיצון מוחלטת.

אלגוריתם חיפוש טאבו הוא אלגוריתם חיפוש מקומי משופר שיודע להתחמק מבעיה זו. חיפוש טאבו משפר את ביצועי החיפוש המקומי על ידי כך שהאלגוריתם מאפשר במקרים מסוימים לבחור פיתוח של צעד באלגוריתם גם אם הצעד פחות טוב מהמיקום הנוכחי בו האלגוריתם נמצא. בנוסף, כדי להיות יעיל ואופטימלי, האלגוריתם יוצר רשימת איסורים שהם מצבים שהאלגוריתם ביקר בהם בעבר ולכן מונע מלחזור לפתרונות אלו.

שימוש ברשימת טאבו

הזכרנו לעיל שבהינתן והאלגוריתם הגיע לנקודת קיצון חשודה, נניח A, אז הוא יסרוק את כל שכניו (הצעדים הבאים) וילך לשכן הכי טוב מבין כל שכניו. אם אין כזה (כי בדוגמה A היא נקודת קיצון מקומית), האלגוריתם יבחר את השכן הכי טוב מבין כל שכניו שפחות טובים ממנו, נניח B. כאשר נסרוק את כל שכניו של B יתכן ו-A יהיה השכן הכי טוב של B וכך האלגוריתם בעצם נכנס ללולאה אינסופית. על מנת להימנע ממצב זה, האלגוריתם מתחזק מבנה נתונים בגודל k, שמחזיק את המידע על הבחירות שהאלגוריתם ביצע ב-k איטרציות האחרונות. לכל איבר במבנה הנתונים מוצמד מספר שלם וחיובי שמסמן את כמות האיטרציות שנשאר לאותו איבר עד שהוא ימחק ממבנה הנתונים (כמו TTL). על מנת להימנע מלחזור על פתרונות, האלגוריתם לא בוחר בחירות שהוא ביצע במהלך ה-k איטרציות האחרונות, ובכך בעצם ימנע את הלולאה האינסופית שתוארה קודם. מצד שני, לפעמים כן נרצה לחזור לאחד מהצעדים שהאלגוריתם כבר ביצע בעבר ולכן נגביל את גודל רשימת הטאבו. כאשר הרשימה תגדל לגודל המקסימלי שלה נתחיל לשחרר מצבים שביקרנו בהם כבר כך שנוכל לחזור אליהם בהדרגה. לבסוף, האלגוריתם יתחיל להתכנס לנקודת קיצון מוחלטת ועל כן האלגוריתם הוא שלם ואופטימלי.



איתמר לרדו 311547087
עמית טויטו 208445866

חיפוש טאבו - פסאודו-קוד:
(תנאי העצירה שלנו הוא כל עוד לא הגענו לפתרון, כלומר מצב הסופי עם היוריסטיקה 0 ובשורה 7 בחרנו שכן רנדומי):

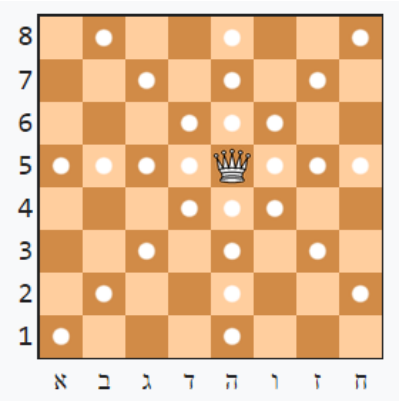
```
sBest ← s0
bestCandidate ← s0
tabuList ← []
tabuList.push(s0)
while (not stoppingCondition())
    sNeighborhood ← getNeighbors(bestCandidate)
    bestCandidate ← sNeighborhood[0]
    for (sCandidate in sNeighborhood)
        if ( (not tabuList.contains(sCandidate)) and (fitness(sCandidate) > fitness(bestCandidate)) )
            bestCandidate ← sCandidate
        end
    end
    if (fitness(bestCandidate) > fitness(sBest))
        sBest ← bestCandidate
    end
    tabuList.push(bestCandidate)
    if (tabuList.size > maxTabuSize)
        tabuList.removeFirst()
    end
end
return sBest
```

הפתרון

המשתמש בוחר n גדול מ-3 (כיוון שאין פתרון חוקי עבור n שלם אי שלילי קטן מ-4, למעט 1). לפי הפסאודו קוד, נמקם מלכות באופן רנדומלי, כל שורה מלכה, ונתחיל לבדוק באופן איטרטיבי את הפתרון. נשתמש בהיוריסטיקה אשר מודדת את איכות הפתרון או המרחק מהפתרון על ידי ספירת כמות הקונפליקטים - כמות המלכות אשר מאיימות אחת על השניה. במימוש עצמו החזרנו את הערך כ-שלילי.

תזכורת מחוקי שחמט: מלכה היא הכלי החזק ביותר במשחק השחמט. המלכה יכולה לנוע באלכסון, אופקית או אנכית אך אינה יכולה לדלג מעל כלים אחרים.

בפונקציה ההיוריסטית נבדוק כלפי כל מלכה האם היא מאיימת על אחרות (או מאויימת מאחרות). תחילה ניצור משתנים עבור הפיתוח הנוכחי, הפיתוח האופטימלי ורשימת טאבו. בנוסף נקבע את גודל הרשימה להיות 300.



נעבור בלולאה עד שנגיע לפתרון האופטימלי. לכל קודקוד המייצג הצבה בלוח הנוכחי נפתח את שכניו המייצגים את ההצבות האפשריים הבאות.

לכל הצבה שפותחה, נבדוק האם ההצבה כלומר האם מצב הלוח אינו נמצא ברשימת טאבו. אם כן, לא נרצה לחזור אליו ולכן נמשיך להצבה הבאה.

אחרת, נבדוק על ידי הפונקציה ההיוריסטית האם ההצבה הבאה תיצור יותר קונפליקטים עם כל המלכות מאשר ההצבה הנוכחית שלנו. אנו מחפשים הצבה שתיצור כמה שפחות קונפליקטים - 0 קונפליקטים הוא הצבה חוקית של המלכות ופתרון לבעיית המלכות.

הצעד הבא באלגוריתם הוא עדכון הפתרון האופטימלי (עד לשלב זה). לבסוף, נבצע בדיקת גודל הרשימת טאבו. אם עברנו את הגודל שקבענו בהתחלה, נוציא מהרשימה מצבים שכבר ביקרנו בהם ולכן נוכל לבקר בהם שוב.

לבסוף נחזיר את הפתרון האופטימלי.

הניסוי

בכל בדיקה שלנו אנחנו מריצים פונקציה שמקבלת מצב התחלתי של הלוח, סוג אלגוריתם ומספר איטרציות (דיפולטיבית הוא 10). היא מריצה את האלגוריתם במספר איטרציות וכל פעם בודקת האם הפתרון שהאלגוריתם החזיר הוא נכון - בודקת מול פונקציית ההיוריסטיקה האם הערך שווה ל-0 (כלומר, אין קונפליקטים). אם כן, סופרת אותו ועוברת לאיטרציה הבאה. אנחנו בודקים זמנים מתחילת האיטרציה הראשונה ועד האחרונה, וגם hit rate/דיוק - כמה פתרונות נכונים קיבלנו מתוך מספר האיטרציות.

לצורך בדיקת איכות ומהירות האלגוריתם השתמשנו בכמה אלגוריתמים שונים שהוזכרו בקורס (שאת הקוד שלהם מצאנו באינטרנט) וביצענו השוואה על אותה סביבת ריצה.
נזכיר בקצרה את האלגוריתמים שהבאנו לבדיקה:

אלגוריתם Hill Climbing:
האלגוריתם מפתח מצבים שיותר טובים מהמצב הנוכחי. בבעיה שלנו, תחילה הצבנו את המלכות על הלוח באופן רנדומלי. נבדוק את כל האפשרויות לשינוי המצבים של המלכות. נבחר בשינוי כך שכמות הקונפליקטים קטנה מהמצב הנוכחי והיא קטנה מכל השינויים האחרים. הבעיה באלגוריתם זה היא שאנו עלולים להתכנס לקיצון מקומי ולא מוחלט. בהמשך מתוארת טבלת תוצאות וניתן לראות שרק חלק מהרצות האלגוריתם זה אכן מגיעות לפתרון הרצוי.

אלגוריתם Hill Climbing: Random Restart:
אלגוריתם זה הוא שיפור של אלגוריתם Hill Climbing. האלגוריתם בוחר כמה נקודות התחלה באופן רנדומלי ומשם מריץ את אלגוריתם Hill Climbing שתואר לעיל. האלגוריתם שומר את כל התוצאות ומחזיר את התוצאה הטובה ביותר. בטבלה שנראה בהמשך, ניתן להבחין שאיכות האלגוריתם טובה יותר מ-Hill Climbing כיוון שהוא מגיע לפתרון אופטימלי יותר פעמים. אולם, זמן הריצה שלו גבוה מ-Hill Climbing ולכן יש כאן Trade-off.

אלגוריתם Simulated Annealing:
אלגוריתם זה מגיע לפתרון אופטימלי באופן הבא: בכל צעד באלגוריתם הוא יבדוק את כל ההצבות הניתנות לפתח מהמצב הנוכחי. אם קיימות הצבות שיותר טובות מהמצב הנוכחי - נזכיר הצבה שיותר טובה היא הצבה שיוצרת פחות קונפליקטים כמו המצב הנוכחי - נבחר בהצבה הכי טובה מבין ההצבות שיותר טובות מהמצב הנוכחי. אם לא קיימת הצבה כזו, נבחר בהסתברות כלשהי (תלות בפונקציית טמפרטורה) הצבה שהיא פחות טובה מהמצב הנוכחי.
פונקציית הטמפרטורה מאפשרת יד יותר חופשית לבחירת מצבים פחות טובים בתחילת הריצה, וככל שמתקדם האלגוריתם, הפונקציה מקשה יותר את התנאים, האלגוריתם יותר זהיר, ובוחר פחות ופחות מצבים פחות טובים. בנוסף, הטמפרטורה גם תלוייה במרחק של המצב שפחות טוב

איתמר לרדו 311547087
עמית טויטו 208445866

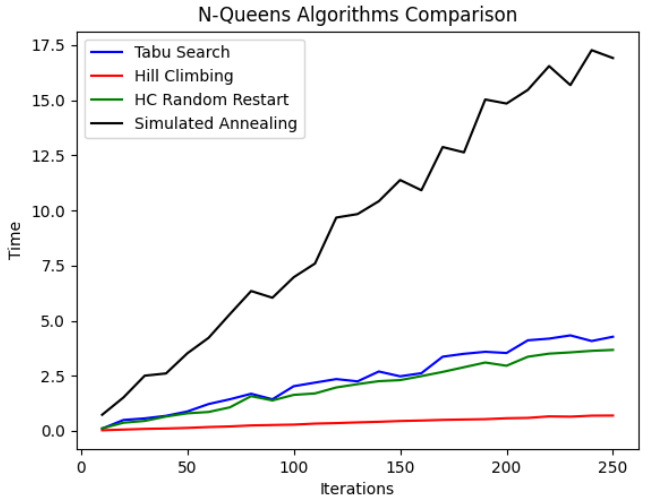
מהמצב הנוכחי. זאת אומרת, ככל שמצב יוצר יותר קונפליקטים עם המלכות, ככה יורדת ההסתברות שנבחר לפתח אותו ולהתקדם אליו.

תוצאות

בכל השוואה לפי פרמטרים שונים, התחלנו בכל פעם ממצב ראשוני של לוח זהה עבור כל האלגוריתמים, ובכל פעם הגרלנו מצב ראשוני אחר מהפעם הקודמת. כלומר, בכל נקודת זמן האלגורתמים מתחילים מאותו מצב ראשוני של הלוח, עד לנקודת הזמן הבאה.

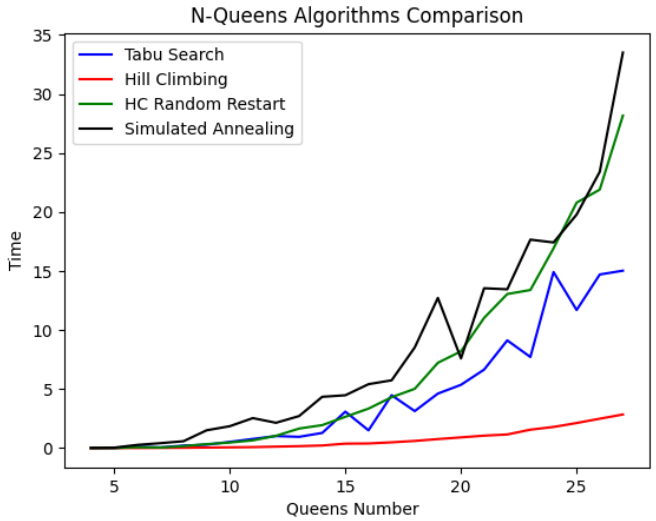
בדיקה 1:

זמן ריצה לפי מספר האיטרציות. מספר המלכות הוא 8. בדקנו מספר איטרציות בקפיצות של 10 מ-10 עד 250.
ניתן לראות שSA לוקח הכי הרבה זמן, ו-Tabu Search קצת מעל HCRR.



בדיקה 2:

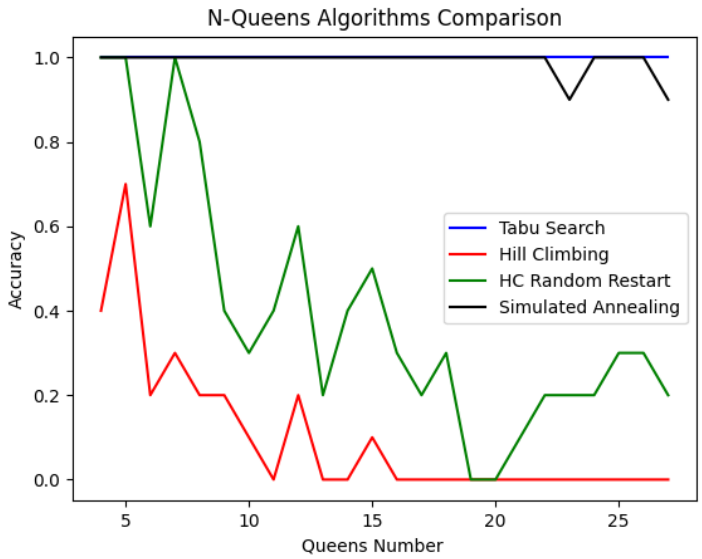
זמן ריצה בשניות לפי גודל לוח. מספר האיטרציות הוא 10.
כאן בדקנו מספר מלכות מ-4 ועד 27.
ניתן לראות שכאן הזמנים של tabu search יותר טובים משל SA ו HCRR.



איתמר לרדו 311547087
עמית טויטו 208445866

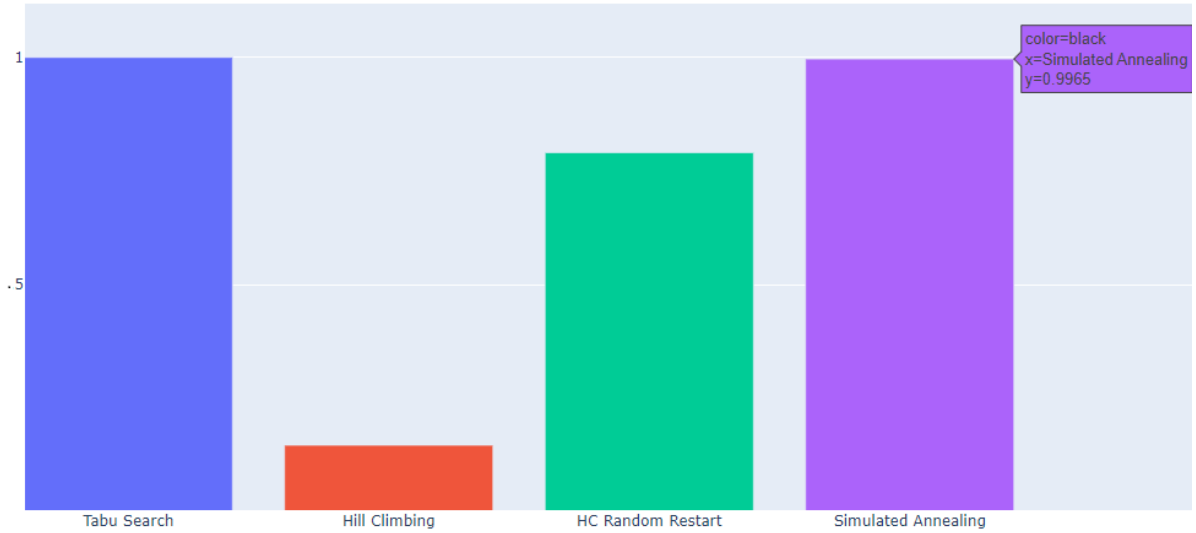
בדיקה 3:

רמת דיוק לפי מספר מלכות (ערכים בין 0 ל 1, מייצגים אחוזים).
כמות האיטרציות היא 10. בדקנו מספר מלכות מ-4 ועד 27. HCRR מתחיל גבוה, יש מגמת ירידה באלגוריתמי הHC ככל שגודל מספר המלכות גדל ומגודל 16 HC כבר מקבל 0 אחוזי דיוק, אך הפתרונות שהוא הביא קרובים מאוד מבחינה היוריסטית. TS ו- SA הכי מדויקים - TS קיבל אחוזי דיוק של 100%, אמנם SA קרוב מאוד לזה.



בדיקה 4:

ממוצע רמת הדיוק לפי אלגוריתם. (ערכים בין 0 ל 1, מייצגים אחוזים). כמות האיטרציות היא 20 ומספר המלכות הוא 8.
הפעלנו 100 פעמים את פונקציית הרצת אלגוריתם באיטרציות, וכאמור, בכל פעם הגרלנו מצב ראשוני אחר של הלוח. לכל אלגוריתם חישבנו את ממוצע הדיוק על גבי 100 הפעמים.



איתמר לרדו 311547087
עמית טויטו 208445866

מסקנות

בעיית N-Queens יכולה להיפתר על ידי אלגוריתמי חיפוש היוריסטים רבים מתחום הבינה מלאכותית. בעבודה זו התעמקנו על פתרון הבעיה באמצעות האלגוריתם Tabu Search והשוואתו לאלגוריתמים השונים. לקחנו בעיה ידועה, התאמנו אלגוריתם חדש, הראנו כיצד הוא פועל ביחס לאחרים ובדקנו את ביצועיו מבחינת זמן ושלמות. המשותף לכל האלגוריתמים הוא הפיתוחים של הצמתים. כל האלגוריתמים מייצרים מעין גרף/עץ חיפוש כך שכל קודקוד הוא מצב של לוח עם הצבה ספציפית של מלכות עליו וכך הם "מנווטים" לפתרון. בשונה מהאלגוריתמים האחרים, חיפוש טאבו מונע לולאות בין מצבים על ידי רשימת הטאבו. הבחירה של הצעד הבא מחושבת ואינה שרירותית. על מנת להגיע לנקודת קיצון מוחלט האלגוריתם יכול לבחור צומת פחות טוב ממיקומו הנוכחי (בשונה מ-Hill Climbing). האלגוריתם לא בוחר צומת שפחות טוב ממנו בהסתברות כלשהי (בשונה מ-Simulated Annealing) אלא מבצע בחירה מושכלת. כפי שניתן לראות מהגרפים לעיל, אלגוריתם החיפוש המקומי 'חיפוש טאבו' הכי מדויק ביחס לחיפושים מקומיים נפוצים עבור הבעיה של N-Queens, אמנם לא הכי מהיר אבל עדיין מהיר הרבה יותר ביחס ל-Simulated Annealing שגם הוא מדויק מאוד (קרוב מאוד ל- 100% דיוק) .

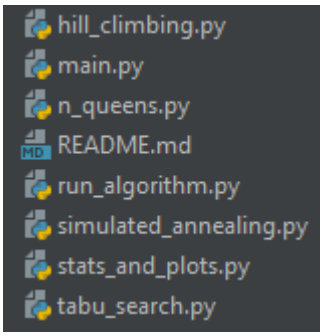
אנו יכולים להסיק שהתוצאות הטובות נובעות מהחזקת רשימת הטאבו. האלגוריתם מונע פיתוח חוזר של מצבים שכבר ביקר בהם ובכך מאפטם את זמן הריצה שלו ומגיע לפתרון חוקי מהר יותר. האלגוריתם שלם, אופטימלי ויחסית לאחרים שנמדדו מהיר ולכן שימוש בו היא בחירה מצוינת לבעיות אופטימיזציה מורכבות כמו זו. כמובן, שהאלגוריתם אינו תלוי בבעיה זו (את הבעיה עצמה צריך לפרמל/להתאים ביחס לאלגוריתם, שתממש את ה- interface) ובפרט אינו תלוי בבחירה של n. החסרון של האלגוריתם הוא כנראה השימוש במקום. כאמור, האלגוריתם מתחזק רשימה של צמתים שביקר בהם בעבר. אנו הסתכלנו על בעיה לא כל-כך גדולה - queen-8. אך שימוש באלגוריתם זה בבעיות עצומות כמו למשל ניתוב מסלולי נסיעה או queen-1000 יכול להיות כי שמירת הרשימ+ה תעמיס מאוד על הזיכרון וכך שימוש באלגוריתם זה לא יהיה יעיל ואולי אפילו לא אפשרי.

לסיכום, הכרנו אלגוריתם חיפוש מקומי יעיל מאוד יחסית מבחינת זמנים, מדויק ובבעיות לא גדולות השימוש בזיכרון אינו מגביל. האלגוריתם קל להבנה, לתפעול, שלם ואופטימלי.

איתמר לרדו 311547087
עמית טויטו 208445866

הקוד:

https://github.com/laredo77/N-Queens



דרישות: פייתון 3. נריץ pip install plotly.

נריץ את קובץ main.py עם ארגומנטים -

python main.py <is_run_algorithm> <number of queens> <number_of_iterations> <is_print_all_boards>

הראשון:האם נריץ run_algorithm על כל האלגוריתמים והדפסת הלוחות - 1 או 0. אם 0, תופעל פונקציה שמגנרטת את הגרפים של הדוח.

השני:מספר המלכות (גודל הלוח). השלישי: מספר האיטרציות והרביעי האם להדפיס את כל הלוחות של כל האיטרציות לכל אלגוריתם (אם הוכנס ערך 1 לארגומנט הראשון) - 1 או 0. אם 0 - יבחר פתרון אחד רנדומלית להדפסה.

```
if __name__ == "__main__":
    algorithms = [tabu_search, hill_climbing, random_restart, simulated_annealing]
    names = ["Tabu Search", "Hill Climbing", "HC Random Restart", "Simulated Annealing"]

    if sys.argv[1] == '1':
        n = int(sys.argv[2])
        iterations_num = abs(int(sys.argv[3]))
        is_print_all_boards = int(sys.argv[4])
        if (n < 1 or n == 2 or n == 3) or is_print_all_boards not in [0, 1]:
            raise "Please provide the correct arguments"

        run_algorithms_and_show_boards(n, iterations_num, is_print_all_boards)
    elif sys.argv[1] == '0':
        generate_graphs(algorithms, names)
    else:
        raise "Please provide the correct arguments"
```

מקורות וחומרי עזר:

- https://github.com/vitorverasm/ai-nqueens n - queens and algorithms
- https://en.wikipedia.org/wiki/Tabu_search Tabu Search Pseudocode
- https://www.youtube.com/watch?v=tIDhFPhrCbU , https://www.youtube.com/watch?v=saNk8h2KuVE Tabu Search explanation
- https://leeds-faculty.colorado.edu/glover/tabusearchvignettes.html Various problems that are solved with Tabu Search
- Course learning material