פרויקט בקורס מבוא לאופטימיזציה 89589-01

בעיית N-Queens & Knights

אלמוג לב

ספיר דויטשר

רקע:

N-Queens היא בעיה NP-שלמה המוכרת בתחום מדעי המחשב. הבעיה מוגדרת כך שעבור לוח שחמט בגודל N x N צריכים למקם N מלכות כך שאף מלכה לא מאיימת על אחרת (על-פי חוקי התזוזה של כלי המלכה במשחק שחמט). כידוע, ככל ש-N גדול יותר כך מספר האפשרויות גדל למימדים עצומים ועל כן הבעיה היא NP-שלמה.

רעיון הפרויקט- הבעיה אותה אנו מעוניינות לחקור:

בפרויקט זה אנו מעוניינות להרחיב את בעיית N-Queens באופן הבא:

עבור לוח שחמט בגודל N x N נרצה למקם N כלים שהם קומבינציה אפשרית של מלכות ופרשים. למשל עבור A מלכות אותן נרצה למקם על הלוח, נצטרך גם למקם N-A פרשים כך שאף כלי לא מאיים על כלי אחר. נקרא לבעיה שלנו N-Queens & Knights.

המטרה:

בהינתן N כלים הממוקמים בצורה שרירותית על הלוח, נרצה לשפר איטרטיבית את מצב הלוח, כך שלאחר maxIter איטרציות לכל היותר כמות הסתירות בין הכלים תהיה מינימלית (סתירה = כאשר כלי אחד מאיים על כלי אחר).

אופן הביצוע:

אנו נתמקד באלגוריתם האופטימיזציה האיטרטיבי Hill Climbing. נציג 2 וריאציות שלו-

1. First Choice Hill Climbing
2. Stochastic Hill Climbing

נריץ את שתי השיטות על בעיית ה-N-Queens & Knights שהגדרנו עבור לוח בגודל N x N עם קומבינציות אפשריות שונות של כלים: A מלכות, N-A פרשים. נחקור את:

1. <u>מחקר ראשון-</u> נחקור את כל הקומבינציות האפשריות של מלכות ופרשים עבור לוח בגודל קבוע. נבדוק האם יש קשר בין הקומבינציות לזמן הריצה ולכמות האיטרציות בכל אלגוריתם בנפרד.
2. <u>מחקר שני-</u> נשווה בין שני האלגוריתמים ונחקור מתי כדאי להשתמש בכל אחד מהם.

בכל אחד מן המחקרים התמקדנו ב-2 גדלי לוחות: 12 X 12 וב- 15 X 15.

קבענו את גדלי הלוח הללו כיוון שהם כוללים כמות גדולה של קומבינציות אפשריות של מלכות ופרשים.

הרצנו כל קומבינציה 5 פעמים בכל אחד משני האלגוריתמים, סה"כ ביצענו 290 הרצות (130 עבור לוח בגודל 12 X 12 ועוד 160 הרצות עבור לוח בגודל 15 X 15)- פירוט של כל ההרצות ניתן למצוא בלינק:

https://github.com/sapirdeu/Nqueens_And_Knights_Optimization/blob/main/ExperimentsAndGraphs.xlsx

נשים לב: Hill climbing היא שיטת אופטימיזציה השייכת למשפחת החיפוש המקומי (local search) שזוהי שיטה יוריסטית- למחצה. השיטה כוללת שיפור איטרטיבי של פתרון נתון, על ידי בחירת פתרונות קרובים – עד להגעה לפתרון מיטבי מקומית (בזה שאין פתרונות עדיפים בסביבתו). על כן, Hill climbing היא שיטה המתאימה גם לפתירת בעיות NP-קשות נוספות כדוגמת בעיית TSP, פתירת בעיות convex problems (בעיות של אופטימיזציה קמורה) וכן פתירת Constraint Satisfaction Problems (CSP) כלומר בעיות עבורן צריך למצוא השמה מספקת (חוקית)- כמו למשל 3-SAT, 3-COL ועוד.

אופן המימוש:

Variables:

גודל הלוח	$N \times N$
A מלכות	$Q_1, \dots, Q_A$
N-A פרשים	$K_1, \dots, K_{N-A}$
כמות איטרציות מקסימלית	$maxIter$

Objective:

$$\min\big(f(maxIter)\big)$$

$$s.t:$$

$$f(maxIter) = number\ of\ chess\ pieces\ that\ contradict\ each\ other\ after\ maxIter\ iterations\ at\ most.$$

Constraints:

נגדיר P כלי (Piece) כלשהו במשחק (מלכה או פרש), בצורה פורמלית:

$$P \in \{Q_1, \dots, Q_A, K_1, \dots, K_{N-A}\}$$

אילוצי המלכה:

המלכה לא באותה שורה עם אף כלי אחר	$\forall\ Q_i \neq P_j :\ Q_i.x \neq P_j.x$
המלכה לא באותה עמודה עם אף כלי אחר	$\forall\ Q_i \neq P_j :\ Q_i.y \neq P_j.y$
המלכה לא באותו אלכסון עם אף כלי אחר	$\forall\ Q_i \neq P_j :\ \lvert Q_i.x - P_j.x \rvert \neq \lvert Q_i.y - P_j.y \rvert$

אילוצי הפרש:

חוקי תזוזה של הפרש $A = \{[2,1],[2,-1],[-2,1],[-2,-1],[1,2],[1,-2],[-1,2],[-1,-2]\}$

הפרש לא באותה משבצת עם אף כלי אחר $\forall K_i : K_i \neq P_j$

אין פרש שמאיים על כלי אחר $\forall K_i \neq P_j : [P_j.x , P_j.y] \notin \{[K_i.x + X , K_i.y + Y] \mid X,Y \in A\}$

מימוש אילוצי המלכה בקוד:

```
super(Queen, self).__init__(x, y)
self.__name = "Q"
self.__rules = [lambda i: {'x': self.get_x() + i, 'y': self.get_y()},
                lambda i: {'x': self.get_x() - i, 'y': self.get_y()},
                lambda i: {'x': self.get_x(), 'y': self.get_y() + i},
                lambda i: {'x': self.get_x(), 'y': self.get_y() - i},
                lambda i: {'x': self.get_x() + i, 'y': self.get_y() + i},
                lambda i: {'x': self.get_x() + i, 'y': self.get_y() - i},
                lambda i: {'x': self.get_x() - i, 'y': self.get_y() + i},
                lambda i: {'x': self.get_x() - i, 'y': self.get_y() - i}]
```

מימוש אילוצי הפרש בקוד:

```
super(Knight, self).__init__(x, y)
self.__name = "K"
self.__rules = [lambda: {'x': self.get_x() - 2, 'y': self.get_y() + 1},
                lambda: {'x': self.get_x() - 1, 'y': self.get_y() + 2},
                lambda: {'x': self.get_x() + 1, 'y': self.get_y() + 2},
                lambda: {'x': self.get_x() + 2, 'y': self.get_y() + 1},
                lambda: {'x': self.get_x() + 2, 'y': self.get_y() - 1},
                lambda: {'x': self.get_x() + 1, 'y': self.get_y() - 2},
                lambda: {'x': self.get_x() - 1, 'y': self.get_y() - 2},
                lambda: {'x': self.get_x() - 2, 'y': self.get_y() - 1}]
```

הסבר על האלגוריתמים ומימושם:

First Choice Hill Climbing- בוחר את הstate הראשון שמשפר

הפונקציה היוריסטית:

הפונקציה מחזירה את מספר הסתירות בין הכלים בלוח.

האלגוריתם:

.1 *best_heuristic* <- None

.2 כל עוד האיטרציה הנוכחית קטנה מ-*maxIter*:

2.1 חשב את היוריסטיקה הנוכחית -> *current_heuristic*

2.2 *local_maximum* <- false

2.3 כל עוד *current_heuristic* > 0 וגם לא נתקענו במקסימום מקומי (*local_maximum* == false):

2.3.1 *better_state* <- false

2.3.2 <u>כל עוד לא הגענו לstate משופר (*better_state* == false) וגם לא עברנו על כל הכלים:</u>

2.3.2.1 בחר כלי

2.3.2.2 <u>עבור כל צעד אפשרי של הכלי הנוכחי:</u>

2.3.2.2.1 חשב ערך יוריסטי

2.3.2.2.2 אם הוא טוב יותר מ-*current_heuristic*:

2.3.2.2.2.1 שים ב-*current_heuristic* את הערך היוריסטי המעודכן

2.3.2.2.2.2 better_state <- true

2.3.2.2.2.3 break

2.3.3 אם עברנו על כל הכלים ולא מצאנו state טוב יותר (*better_state* == false):

2.3.3.1 *local_maximum* <- *true*

2.4 אם *best_heuristic* < *current_heuristic*:

2.4.1 *best_heuristic* <- *current_heuristic*

2.5 אם *current_heuristic* == 0:

2.5.1 break

<u>יתרונות:</u>

- האלגוריתם בוחר את הstate הראשון הטוב ביותר. אסטרטגיה זו טובה כאשר ל-state הנוכחי יש המון שכנים.
- עבור maxIter גדול מספיק נמצא פתרון אופטימלי. במהלך המחקר האלגוריתם תמיד מצא פתרון אופטימלי חוקי בפחות מ-maxIter איטרציות.
- בכל שלב של האלגוריתם ניתן לעצור אותו ולקבל את המצב הכי טוב עד כה.
- רץ מהר.

<u>חסרונות ומגבלות:</u>

- האלגוריתם חמדני, לוקח את השיפור הראשון שמוצא (כלומר אם יש בסביבה שיפור טוב יותר הוא עלול כלל לא להגיע אליו), לכן זמן הריצה וכמות האיטרציות גדול מזו של ה- Stochastic Hill Climbing.
- מוצא מקסימום מקומי ועלול להיתקע שם ולא למצוא את הגלובלי.
- כאשר נמצאים במישור אין עדיפות ברורה לאן להתקדם.

<u>לינק לקוד (מתודה בשם first_choice בשורה 41):</u>

https://github.com/sapirdeu/Nqueens_And_Knights_Optimization/blob/main/algorithm/HillClimbing.py

Stochastic Hill Climbing- מבין כל הstates שמשפרים, נבחר את הstate בצורה רנדומלית

הפונקציה היוריסטית:

הפונקציה מחזירה את מספר הסתירות בין הכלים בלוח.

האלגוריתם:

.1 *best_heuristic* <- None

.2 כל עוד האיטרציה הנוכחית קטנה מ-*maxIter*:

2.1 חשב את היוריסטיקה הנוכחית -> *current_heuristic*

2.2 *local_maximum* <- false

2.3 כל עוד *current_heuristic* > 0 וגם לא נתקענו במקסימום מקומי (*local_maximum* == false):

2.3.1 *better_states* <- [ ]

2.3.2 כל עוד לא מצאנו את כל המצבים המשופרים (len(*better_states*) == 0) וגם לא עברנו על כל הכלים:

2.3.2.1 בחר כלי

2.3.2.2 עבור כל צעד אפשרי של הכלי הנוכחי:

2.3.2.2.1 חשב ערך יוריסטי

2.3.2.2.2 אם הוא טוב יותר מ-*current_heuristic*:

2.3.2.2.2.1 הוסף את הכלי הנוכחי ל- *better_states*.

2.3.2.3 אם מצאנו states טובים יותר (len(*better_states*) == 0):

2.3.2.3.1 בחר כלי רנדומלי מתוך רשימת ה-*better_states*.

2.3.2.3.2 עדכן את *current_heuristic*.

2.3.3 אם עברנו על כל הכלים ולא מצאנו state טוב יותר (*better_state* == false):

2.3.3.1 *local_maximum* <- *true*

2.4 אם *best_heuristic* < *current_heuristic*:

2.4.1 *best_heuristic* <- *current_heuristic*

2.5 אם *current_heuristic* == 0:

2.5.1 break

יתרונות:

- האלגוריתם מוצא רשימת שיפורים ובוחר רנדומלית אחד מהם (בהתפלגות אחידה), כלומר הוא פחות חמדני מ- First Choice Hill Climbing ולכן בסבירות גבוהה מוצא את הפתרון האופטימלי מהר יותר. הזמן וכמות האיטרציות קטנים יותר.
- עבור maxIter גדול מספיק נמצא פתרון אופטימלי. במהלך המחקר האלגוריתם תמיד מצא פתרון אופטימלי חוקי בפחות מ-maxIter איטרציות.

- בכל שלב של האלגוריתם ניתן לעצור אותו ולקבל את המצב הכי טוב עד כה.
- רץ מהר.

חסרונות ומגבלות:

- מוצא מקסימום מקומי ועלול להיתקע שם ולא למצוא את הגלובלי.
- כאשר נמצאים במישור אין עדיפות ברורה לאן להתקדם.

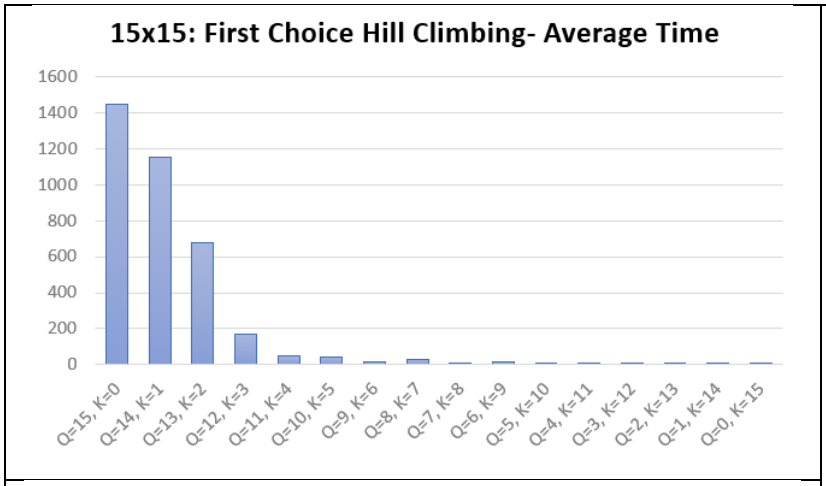
לינק לקוד (מתודה בשם stochastic בשורה 159):

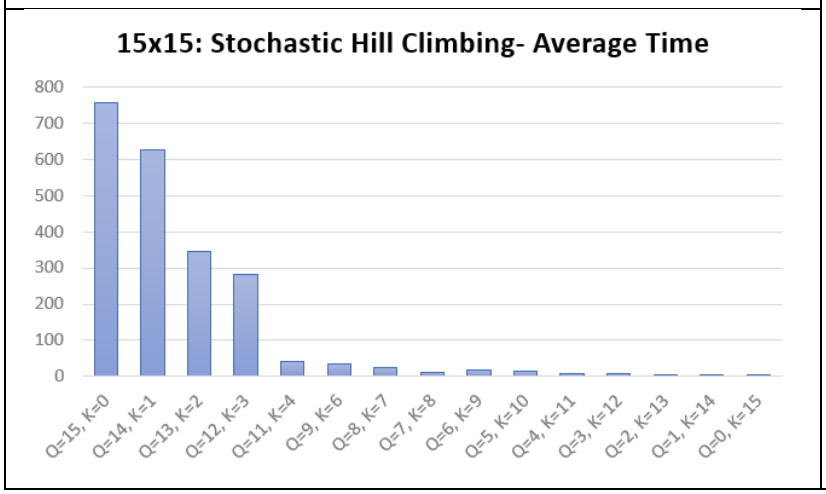
https://github.com/sapirdeu/Nqueens_And_Knights_Optimization/blob/main/algorithm/HillClimbing.py

תוצאות המחקר:

המחקר הראשון

- המשתנה הנבדק: כל הקומבינציות האפשריות של N כלים (שילוב של מלכות ופרשים).
- גדלי לוח:
  - תחילה, בדקנו עבור לוח מקובע לגודל 12 X 12.
  - לאחר מכן, בדקנו עבור לוח מקובע לגודל 15 X 15.



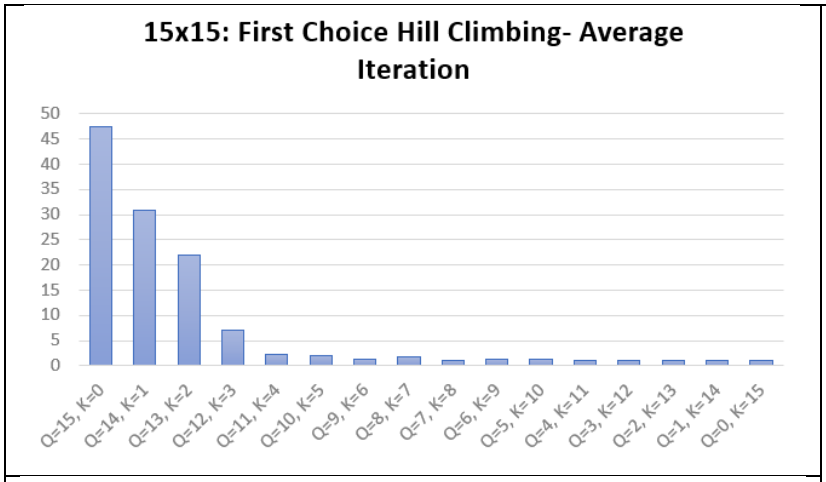


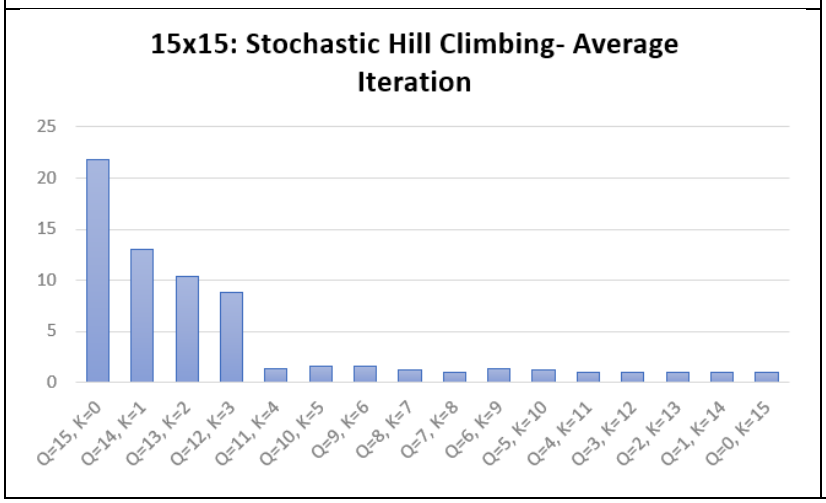
ציר ה-X: קומבינציות אפשריות של מלכות ופרשים.

ציר ה-Y: זמן ריצה בשניות עד מציאת פתרון אופטימלי.

מסקנה מהגרפים:

בשני האלגוריתמים ניתן לראות שבכל שכמות המלכות יורדת כך זמן הריצה יורד משמעותית ונמצא פתרון אופטימלי במהירות.





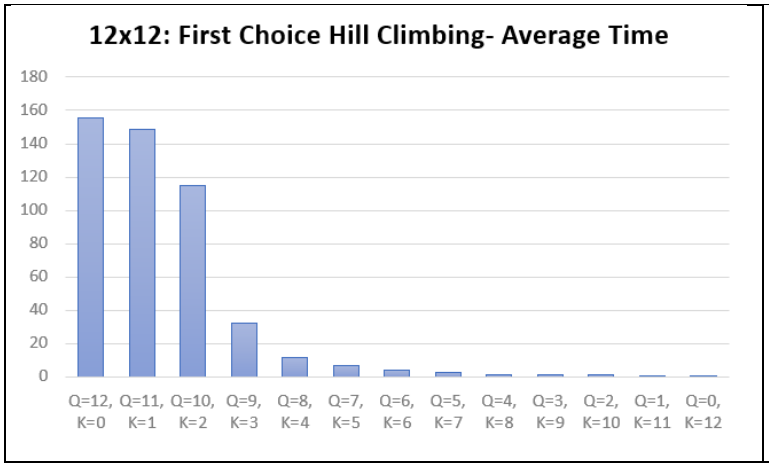
<u>ציר ה-X:</u> קומבינציות אפשריות של מלכות ופרשים.

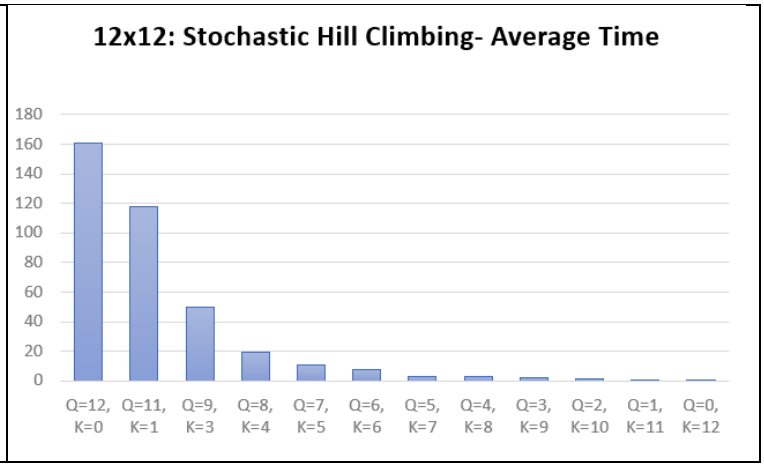
<u>ציר ה-Y:</u> כמות איטרציות עד מציאת פתרון אופטימלי.

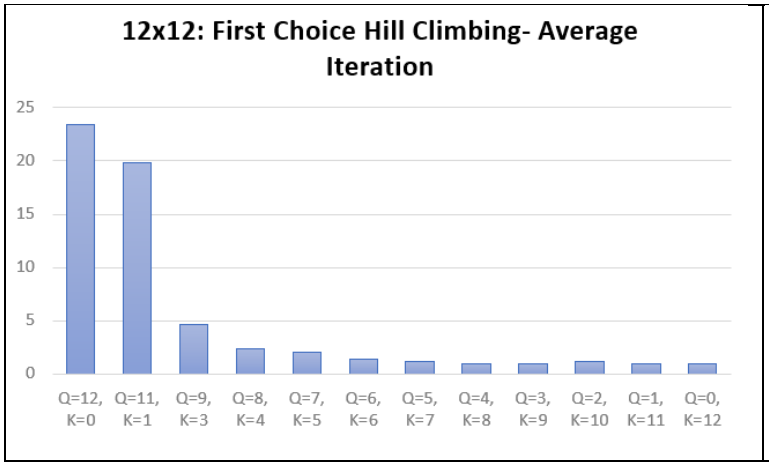
<u>מסקנה מהגרפים:</u>

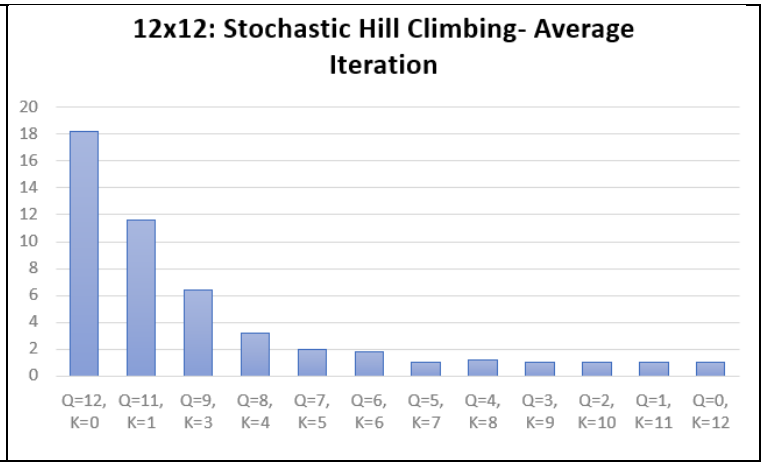
בשני האלגוריתמים ניתן לראות שככל שכמות המלכות יורדת כך כמות האיטרציות הנדרשת עד מציאת פתרון אופטימלי יורדת גם היא, עד כדי איטרציה אחת.

ניתן לראות תוצאות דומות עבור לוח בגודל 12 X 12:









<u>לאחר הרצת הניסוי על לוח בשני גדלים שונים עם כל הקומבינציות האפשריות, עולה המסקנה:</u>

כאשר כמות המלכות > כמות הפרשים, לוקח זמן רב למצוא פתרון אופטימלי (ביוון שצעד של המלכה מכסה יותר שטח מצעד של פרש, לכן יותר קשה למקם אותן ללא קונפליקט עם יתר הכלים).

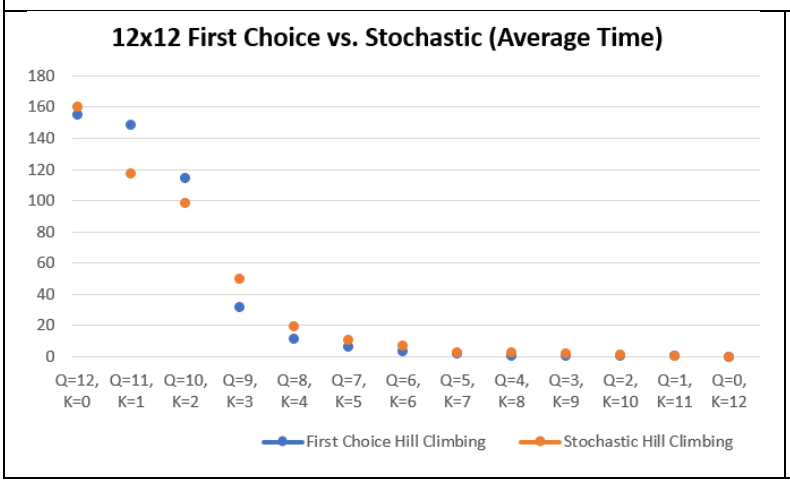
ככל שהלוח גדול יותר, זמן הריצה גדל.

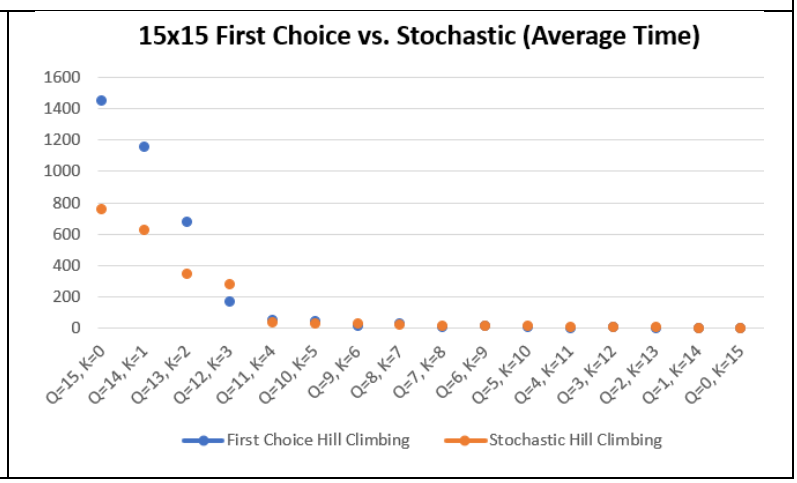
<u>המחקר השני</u>

- השוואה בין First Choice Hill Climbing לבין Stochastic Hill Climbing:
  - השוואת זמן ריצה ממוצע עד למציאת פתרון אופטימלי.
  - השוואת כמות איטרציות ממוצעת עד למציאת פתרון אופטימלי.
- גדלי לוח:
  - תחילה, בדקנו עבור לוח מקובע לגודל 12 X 12.
  - לאחר מכן, בדקנו עבור לוח מקובע לגודל 15 X 15.

<u>השוואת זמני ריצה:</u>

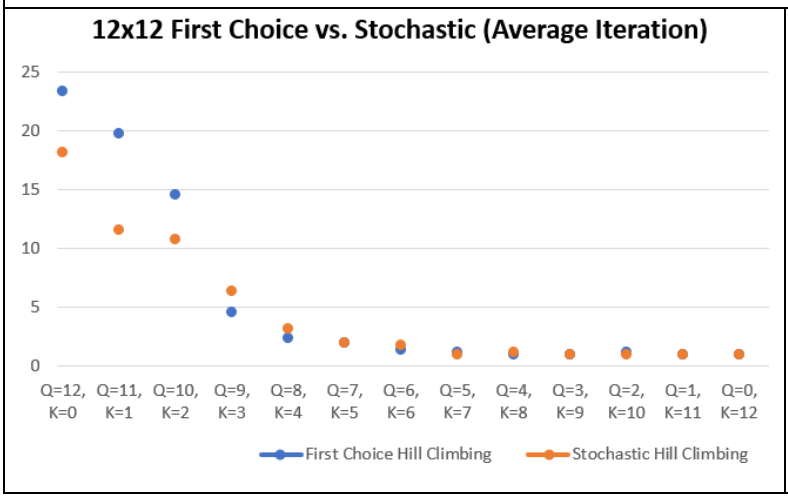
ציר ה-X: קומבינציות אפשריות של כלים. ציר ה-Y: זמן הריצה הממוצע בשניות.

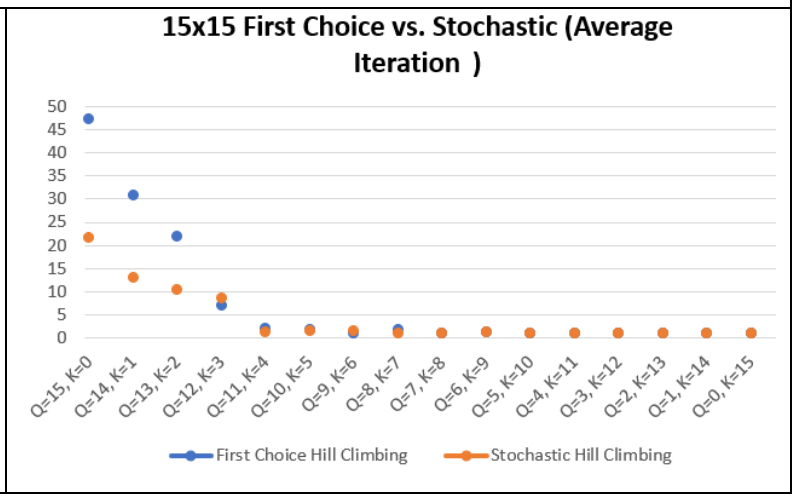




השוואת כמות האיטרציות:

ציר ה-X: קומבינציות אפשריות של כלים. ציר ה-Y: כמות האיטרציות הממוצעת.





לאחר הרצת שני האלגוריתמים על לוח בשני גדלים שונים, עולות המסקנות:

שני האלגוריתמים מוצאים פתרון אופטימלי חוקי.

ככל שהלוח גדל הפערים בין האלגוריתמים הולכים וגדלים לטובת ה-Stochastic Hill Climbing:

- זמן הריצה הממוצע של Stochastic Hill Climbing מהיר יותר מ-First Choice Hill Climbing.
- כמות האיטרציות הממוצעת של Stochastic Hill Climbing קטנה יותר מ-First Choice Hill Climbing.

ככל שכמות המלכות קטנה, לאלגוריתמים יש ביצועים דומים.

דיון:

מתוצאות המחקר עולה ששני סוגי אלגוריתמי ה-Hill Climbing שהצגנו פותרים את בעיית ה-N-Queens & Knights בצורה יעילה.

- האלגוריתמים מצאו פתרון חוקי עבורו כמות הסתירות היא 0 בכל ההרצות שביצענו, שכן בחרנו את maxIter להיות גדול מספיק (אנו קיבענו אותו להיות 180). עבור maxIter קטן האלגוריתמים לא תמיד ימצאו פתרון חוקי, אך בהכרח ימצאו את הפתרון הטוב ביותר עד אותה איטרציה, כלומר האלגוריתם מבטיח שיפור.
- קיימת קורלציה בין זמן הריצה של שני האלגוריתמים לבין המשחק הפנימי של כמות מלכות והפרשים- ככל שיש יותר מלכות כך זמן הריצה גדל.
- עוד עולה מתוצאות המחקר כי עבור לוח גדול ו\או כמות מלכות גדולה נעדיף להשתמש ב-Stochastic Hill Climbing על פני First Choice Hill Climbing, כיוון שהוא מוצא תוצאה אופטימלית מהר יותר.

Fun Facts:

- 290 הרצות סה"כ (130 הרצות על גבי לוח בגודל 12X12. 160 הרצות על גבי לוח 15X15).
- 1,631 איטרציות סה"כ.
- זמן ריצה כולל של 34,123 שניות שהן 9.48 שעות.
- ההרצה הארוכה ביותר ארכה 4,854 שניות שהן 81 דקות, ולקחה 161 איטרציות.

לינק לקוד:

https://github.com/sapirdeu/Nqueens_And_Knights_Optimization