מבוא לאופטימיזציה- פרויקט סיום

N queens problem

אביב דמרי 206322406

נעמה שריר 318733805

הצגת הבעיה:

N queens או בעיית N המלכות היא בעיה מוכרת בעולם מדעי המחשב.

הבעיה מדברת על הצבת N מלכות בלוח שחמט בגודל NxN כך שאף מלכה אינה מאיימת על מלכה אחרת.

נגדיר איום של מלכה באופן הבא- במשחק השחמט המלכה מאיימת על כל חייל שנמצא איתה באותה שורה, באותה עמודה או באותם אלכסונים.

לדוגמא,

•		•	
	•	•	•
•	•	Q	•
	•	•	•

מסקנה טריוויאלית שנובעת מהגדרת הבעיה היא שבכל עמודה ובכל שורה יכולה להיות בדיוק מלכה אחת.

נסמן פתרון מייצג כווקטור באורך N, $(Q_1, \dots, Q_N)$ שהוא למעשה פרמוטציה של $(1, \dots, N)$. כאשר האיבר הראשון מייצג את שורת המלכה בעמודה הראשונה, האיבר השני מייצג את שורת המלכה בעמודה השנייה וכך הלאה.

באופן זה, צמצמנו את הבעיה למציאת פרמוטציה של וקטור באורך N.

דוגמא לפתרונות אפשריים לבעיית 4 המלכות כלומר הצבת 4 מלכות בלוח שחמט בגודל 4x4,

	Q		
			Q
Q			
		Q	

(3,1,4,2)

		Q	
Q			
			Q
	Q		

(2,4,1,3)

נציין שקיים פתרון לכל N טבעי למעט כאשר N=2,3.

בעיה זו נכללת תחת בעיות CSP, Constraint satisfactions problems כלומר בעיות עם תנאים לסיפוק ושהאילוצים מבטאים את המטרה. כלומר בעיות אלו דורשות מאתנו לספק פתרון אך עליו לספק אילוצים מסוימים.

נגדיר את הבעיה באופן פורמלי,

המשתנים הם $Q_1, \dots Q_N$, כך ש Q_i היא השורה של המלכה שממוקמת על הלוח בעמודה ה i.

התחום עבור כל אחד מהמשתנים הוא $D_i = \{1, \dots, N\}$,

האילוצים הם $Q_i \neq Q_j$ (כלומר, אין שתי מלכות באותה שורה) וכן $|Q_i - Q_j| \neq |i - j|$ (אין שתי מלכות באלכסון אחת לשנייה).

קיימים אלגוריתמים רבים לפתור את בעיית N המלכות.

בפרויקט שלנו נציג אלגוריתם שפותר את הבעיה ב100% מהמקרים, ללא מגבלה של זמן/איטרציות ואז נשווה מול אלגוריתמים היוריסטיים אפשריים לפתרון הבעיה שלא תמיד יבטיחו להגיע לפתרון ונסיק את המסקנות.

האלגוריתמים שאותם נציג הם:

- Branch and bound
- Genetic algorithm
- Hill climbing
- Simulated annealing

בנוסף, נציג בעיית בונוס שקשורה לבעיית n המלכות, בה נגדיל את הלוח ונראה איך מספר הפתרונות האפשריים משתנה בהתאם.

נתחיל עם האלגוריתם שפותר את הבעיה לכל N טבעי למעט N=2,3 שאין פתרון עבורם.

-Branch and bound

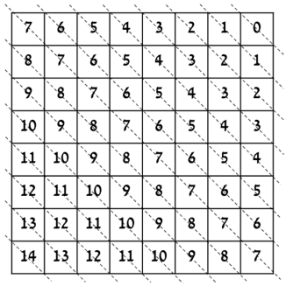
האלגוריתם עובד בצורת הפרד ומשול.
החלק של ה branch מתבצע על ידי חלוקה של כל הפתרונות החוקיים לחתיכות קטנות יותר.
והחלק של ה bound הוא בדיקה האם פתרון הכי טוב של תת בעיה עדיף על המצב הנוכחי.

האלגוריתם branch and bound במקרה שלנו יבצע הצבות של מלכות עמודה אחרי עמודה ובכך מחלק את הפתרון לבעיות קטנות. עבור כל עמודה מחפש שורה חוקית ומייצר פתרון חלקי. במידה ויש פתרון אז יסיים ויחזיר אותו, אחרת (אי אפשר להמשיך עם הפתרון כי יש קונפליקטים בין מלכות בלוח), לא ממשיך להעמיק בפתרון.

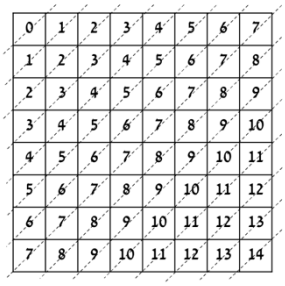
כפי שציינו לעיל בעיית NQueen היא נכללת בבעיות CSP, ולכן אין פתרון אחד יותר טוב מהשני אלא או שמצאת פתרון או שלא ולכן בחלק של bound החסימה היא בינארית.
כלומר במידה ויש פתרון כלשהו נבחר אותו ולא נצטרך להשוות מול ענפים אחרים, אחרת (יש קונפליקטים ואין פתרון) נקטע את הענף.

<u>מימוש האלגוריתם :</u>

- יצירת שלושה מערכים בוליאניים ואתחולם בערכי 0.
  - המערך הראשון יהיה בגודל N ומיועד עבור סימון השורות התפוסות.
  - המערך השני יהיה בגודל 2N-1 ומיועד עבור סימון האלכסונים ה \ התפוסים בלוח.
  - המערך השני יהיה בגודל 2N-1 ומיועד עבור סימון האלכסונים ה / התפוסים בלוח.



$row - column + (N - 1)$



$row + column$

1. נבצע הצבות של מלכות אחת אחרי השנייה בכל עמודה כאשר מתחילים מהעמודה השמאלית ביותר.

1.1. לפני הצבה של מלכה בשורה i נבדוק תחילה בעזרת המערך הראשון אם השורה ה i תפוסה או האם האלכסון השמאלי או הימני תפוסים באמצעות המערכים המיועדים.

1.1.1.במידה והם תפוסים, נבדוק את השורה הבאה.

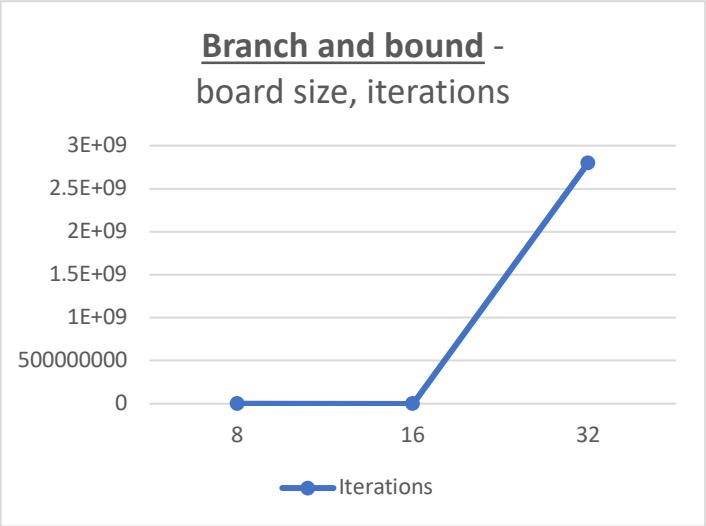
1.1.2.אחרת, כלומר הם לא תפוסים ,אזי נמקם את המלכה במיקום הזה ונמשיך באופן רקורסיבי לעמודה הבאה.

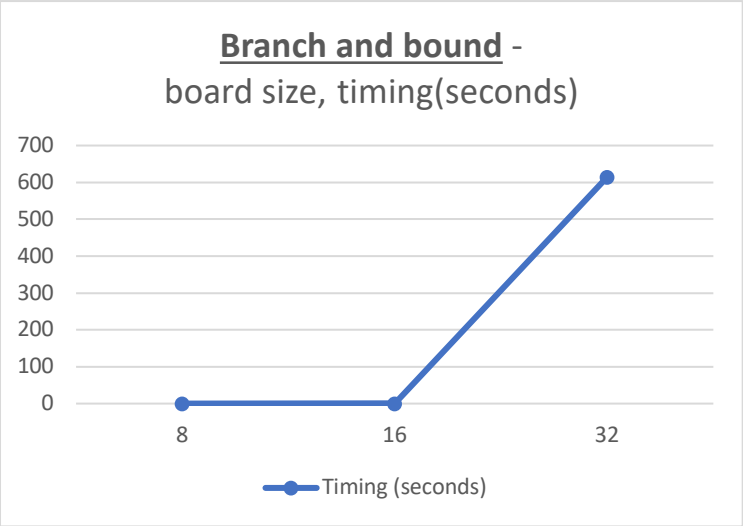
1.1.2.1. אם גילינו בפתרון הרקורסיבי שאין פתרון אז נאפס את המיקום של המלכה ששמנו.

1.1.2.2. אחרת, נחזיר את הפתרון.

בדקנו את הזמני ריצה של האלגוריתם בממוצע של 100+ ריצות שונות עבור גדלי n שונים

N size	Timing (seconds)	Iterations	Success rate
8	0.0006	876	100%
16	0.0574	160712	100%
32	614.28	2799725104	100%
+32	Too much	Too much	100%





מעצם אופי האלגוריתם, האלגוריתם מבטיח למצוא פתרון אופטימלי (במידה וקיים).

מכיוון שהוא דטרמיניסטי ועובר על כמעט כל האפשרויות (רק באופן יעיל).

החיסרון כמובן שעושה זאת בזמנים מאוד גדולים.

עבור כמה ריצות שונות נקבל אותו סדר גודל של זמני ריצה ובדיוק אותם כמות איטרציות.

נציין שבחירת הערך בענף כלומר ניסיון הצבה של מלכה בשורה i תתבצע באופן סדרתי שורה אחרי שורה, ללא יוריסטיקה עבור branching (זה פחות הנושא שנרצה להתמקד בפרויקט).

כעת שמנו לב באופן לא מפתיע ,שהאלגוריתם B&B לוקח המון זמן להחזיר תשובה ובמיוחד כאשר N מטפס למספרים גבוהים N>=32. כפי שניתן לראות בגרף, העלייה היא מאוד דרמטית ככל שמגדילים את הלוח. לכן, כעת נציג אלגוריתמים היוריסטיים שמקרבים לפתרון האופטימלי ומוצאים אותו ברוב מוחלט של הפעמים בזמנים קטנים משמעותית.

נתחיל עם אלגוריתם גנטי אלגוריתם any time שמקרב אותנו לפתרון כאשר נשתמש בפונקציית מטרה **מונוטונית** שתייצג עבורנו מספרית עד כמה אנו מתקרבים למטרה.

-Genetic algorithm

אלגוריתם גנטי מבוסס על תורתו של דאווין- החזק שורד.
נקבל אוכלוסייה כללית כלומר כמה פתרונות אפשריים לבעיה (פתרונות שלא בהכרח מקיימים את כל האילוצים) ונעריך את החוזק של כל פתרון ונבחר את הפתרונות הטובים ביותר "להתרבות". כלומר נבצע זיווגים בין פתרונות לייצר פתרונות חדשים עד שנגיע לפתרון האופטימלי. יש לציין שחלק מהפתרונות לא ימשיכו בתהליך וייעלמו. נשיב לב שהפתרון האופטימלי הוא הפתרון שמקיים את כל האילוצים.

בהקשר שלנו בבעיית nqueen נגדיר את הפונקציות הרלוונטיות עבור האלגוריתם :

נגדיר פונקציית הערכה שמקבלת פתרון ומחזירה מספר שמייצג כמה הפתרון הוא טוב (חזק).
נגדיר שהמספר ייצג את כמות הקונפליקטים בין המלכות בלוח. כמובן, ככל שיש פחות קונפליקטים אזי הפתרון הינו יותר טוב ופתרון עם 0 קונפליקטים מייצג את פונקציית המטרה כלומר הפתרון האופטימלי.

נגדיר פונקציית crossover שמקבלת שני פתרונות ומייצרת קומבינציה בין שני פתרונות (בניסיון לשמר את הטוב מכל פתרון ולקבל פתרון טוב יותר) יש לציין שקיימים מספר אפשרויות לממש פונקציה זאת.
פונקציה זו תעבור על כל אחד מהווקטורים ובמידה ומצאה שני איברים סמוכים $(i, i+1)$ שההפרש בניהם הוא 1 או 0, אזי היא תחליף את האיבר השני $(i+1)$ עם האיבר המקביל בווקטור השני.
ההיגיון מאחורי זה הוא שבמידה וההפרש בין שני איברים סמוכים (2 עמודות סמוכות בלוח) הוא 0 או 1 כלומר, שורה זהה או שורה סמוכה הם בוודאות מייצגים קונפליקט בלוח ולכן נרצה לנסות לשנות אותם.

נגדיר פונקציית mutation שמקבלת פתרון (לאחר crossover) ומטרתה לשפר אותו.
יש לציין שקיימים מספר אפשרויות לממש פונקציה זאת אנחנו בחרנו כאן לשלב 2 גישות.

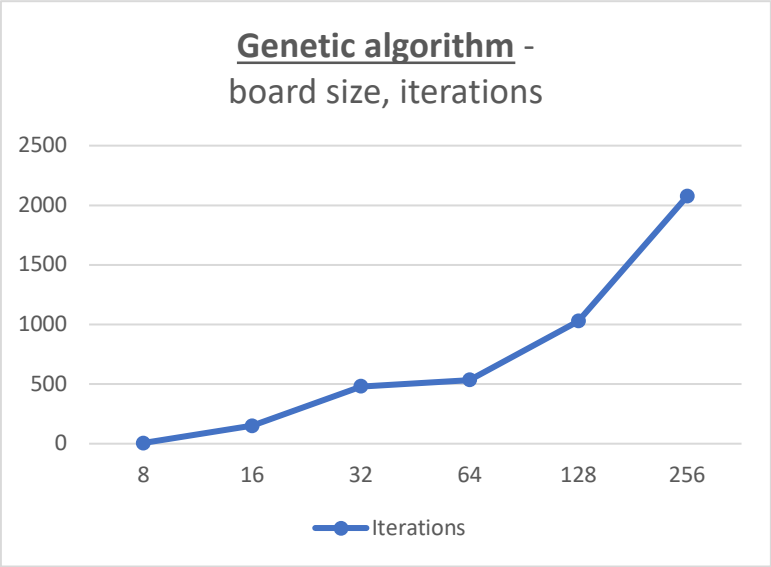
הפונקציה תעבור על הפתרון ותוריד כפילויות שעשויות לקרות כתוצאה מביצוע crossover.
כמובן שכפילויות מייצרות בוודאות קונפליקט (2 מלכות באותה שורה).
כמו כן, נחליף בין שני איברים רנדומליים משני צדי הווקטור.

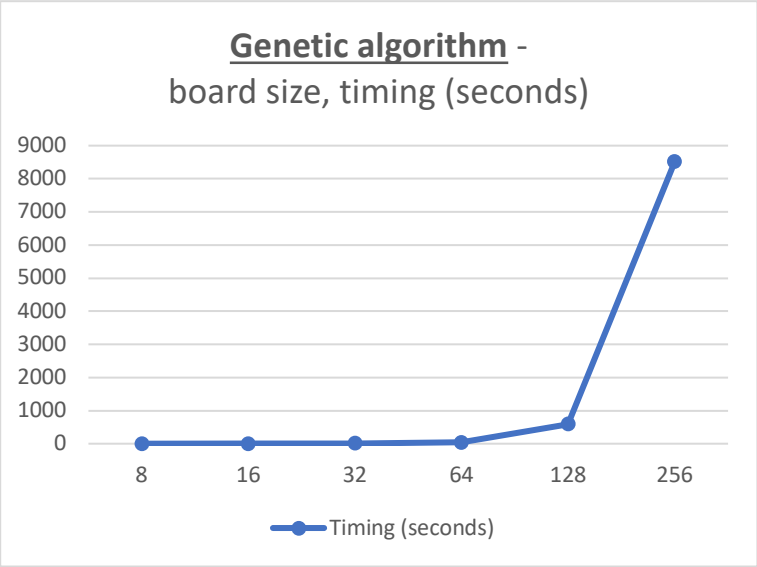
מימוש האלגוריתם:

- הגדר קבוצת פתרונות(אוכלוסייה) בגודל 100 שכל אחד מהם מייצג פרמוטציה שונה של וקטור $(1, \dots, N)$.
- הערך את הפתרונות באמצעות פונקציית הערכה.
- בדיקה האם קיים פתרון חוקי בקבוצת הפתרונות. במידה וכן, החזר אותו.
- כל עוד לא הגענו לפתרון חוקי (פונקציית המטרה),
  - בחר את שני הפתרונות הטובים ביותר עפ"י פונקציית ההערכה.
  - הפעל עליהם את פונקציית ה crossover שמייצרת שני פתרונות חדשים.
  - הפעל על שני הפתרונות החדשים את פונקציית ה mutation.
  - הוסף את שני הפתרונות לאוכלוסייה.
  - הערך את הפתרונות באמצעות פונקציית הערכה.

בדקנו את הזמני ריצה של האלגוריתם בממוצע של 100+ ריצות שונות עבור גדלי n שונים.
יש לציין בחרנו אוכלוסייה התחלתית בגודל 100(ניתן לשנות גם את הגודל שלה).

N size	Timing (seconds)	Iterations	Success rate
8	0.0093	6	78.7%
16	0.03545	151	84%
32	5.9194	481	90%
64	40.7662	535	100%
128	589.7288	1028	100%
256	8514.8957	2075	100%





ניתן לראות שעבור $n \leq 32$ האלגוריתם לא תמיד מחזיר תשובה אופטימלית.
כלומר בערך ב20 אחוז לא נקבל את הפתרון האופטימלי עבור המספרים הקטנים האלו אלא נתקע במינימום לוקאלי(אחרי הגבלה סבירה של איטרציות).
אבל עבור $n > 32$ האלגוריתם מביא את הפתרון האופטימלי בהסתברות מאוד מאוד גבוהה . (בהרצות שלנו לא היה כישלונות אבל סטטיסטית זה יכול לקרות).

כעת נציג 2 אלגוריתמים איטרטיביים שמתחילים מפתרון ומשפרים אתו עם הזמן לכמה שפחות קונפליקטים במקרה שלנו.

-Hill climbing

קיימות מספר גרסאות לאלגוריתם hill climbing, נתחיל מלהסביר את הבסיסית ונמשיך לפתח ולשפר את האלגוריתם.

האלגוריתם הוא איטרטיבי המתחיל מפתרון אקראי לבעיה ולאחר מכן מנסה לשפר את הפתרון. האלגוריתם מחפש שכנים של הפתרון הקיים כלומר, פתרונות אחרים עם שינוי של צעד אחד ומדרג את הפתרונות הללו.
אם הוא מוצא פתרון טוב יותר מהפתרון שבו הוא נמצא כעת, הוא מגדיר אותו כפתרון הנוכחי וממשיך כך עד שאין שינויים נוספים שמשפרים את הפתרון.

נשים לב שבמקרה זה האלגוריתם עשוי לעצור לפני שהוא מגיע לפתרון האופטימלי עקב הגעה לאופטימום מקומי במקום לאופטימום הגלובלי.
לכן, על מנת להימנע מבעיה זו נשדרג את האלגוריתם ונוסיף לו אלמנט של רנדומיות.
כאשר נגיע לאופטימום מקומי, נגריל מחדש פתרון ונמשיך ממנו.

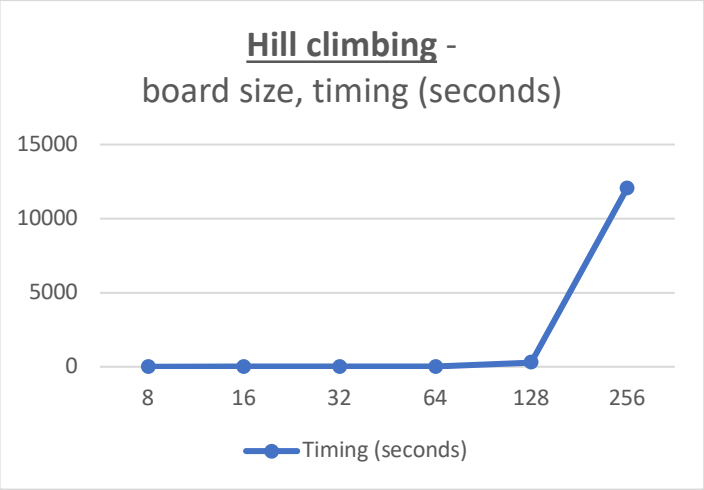
האלגוריתם עבור הבעיה שלנו nqueen יגריל פתרון התחלתי.
לאחר מכן ימצא את הפתרונות השכנים לבעיה זו כלומר, פתרונות עם שינוי של צעד אחד בלוח. יפעיל פונקציית הערכה על הפתרונות השכנים שמצא ואם קיים פתרון טוב יותר מהפתרון שבו אנחנו נמצאים כעת, כלומר פתרון עם פחות קונפליקטים בלוח, נעבור אליו.

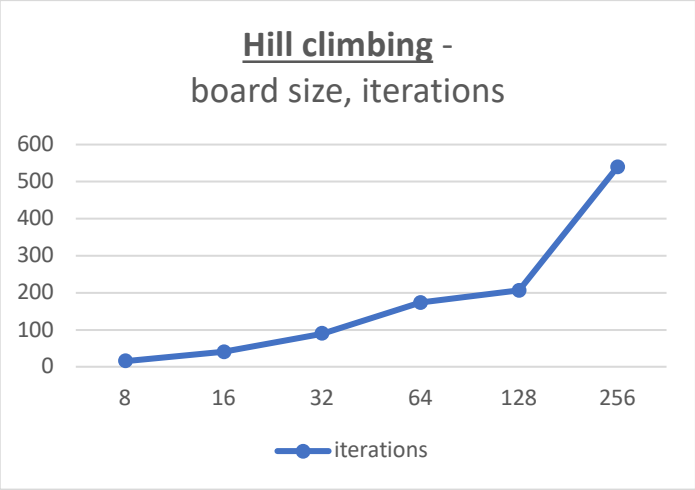
במידה ולא נצליח להגיע לפתרון ללא קונפליקטים, נגריל פתרון חדש וננסה שוב להגיע למטרה. האלגוריתם לא שומר את כל עץ הפתרונות אלא מספיק לו לשמור את הפתרון שאנחנו נמצאים בו כעת ואת ערך המטרה של פונקציית ההערכה.

מימוש האלגוריתם:

1. התחל מהגרלת פתרון אקראי לבעיה.
2. סרוק את כל הפתרונות השכנים של הפתרון שאנחנו נמצאים בו כעת וחפש שכנים שפונקציית ההערכה שלהם טובה יותר מהפתרון הנוכחי כלומר כאלו עם מספר התנגשויות נמוך יותר.
   2.1. אם קיים פתרון כזה, עבור אליו (המשך בטיפוס ההר אל עבר הפתרון האופטימלי).
   2.2. אחרת, אם קיים פתרון עם ערך זהה בפונקציית ההערכה, הגרל פתרון כלשהו מהפתרונות השכנים ועבור אליו (הימנעות ממינימום מקומי).
3. חזור על צעד 2 עד שתמצא פתרון שטוב יותר מכל שכניו. כשתמצא כזה, עבור לשלב 4.
4. אם הפתרון שמצאנו הוא הפתרון האופטימלי כלומר, בעל 0 התנגשויות- החזר אותו.
אחרת, כלומר הגענו לאופטימום מקומי. הגרל שכן רנדומלי ועבור אליו.

N size	Timing (seconds)	iterations	Success rate
8	0.000463	15	100%
16	0.013495	40	100%
32	0.428879	89	100%
64	15.6245	173	100%
128	289.238	206	100%
256	12037	539	100%





האלגוריתם תמיד מחזיר תשובה אופטימלית (במידה ולא מגבילים את כמות האיטרציות).

-Simulated annealing

אלגוריתם זה יעבוד בצורה דומה ל hill climbing with restart רק במקום להגריל פתרון חדש במצב שהפתרון של השכן פחות טוב מהפתרון הנוכחי, נאפשר לקחת את הפתרון של השכן (סוג של ירידה לצורך עלייה בשביל להימנע ממקסימום גלובלי) בהסתברות מסוימת.

נגדיר 2 משתנים: temperature ,decay_rate

-Temperature - הטמפרטורה שאותה נקטין ככל שעוברים יותר צעדים ומתכנסים לפתרון.

-decay_rate היחס שמכפילים ב temperature בשביל להקטין אותה.

נגדיר את פונקציית ההסתברות:

פונקציית ההסתברות יורדת ככל שעוברים הצעדים בשביל התכנסות לפתרון אופטימלי.

כמו כן פונקציית ההסתברות יורדת ככל שהפתרון של השכן גרוע יותר מהפתרון הנוכחי

פונקציית ההערכה לא תשתנה ותייצג את כמות הקונפליקטים בלוח בין המלכות.

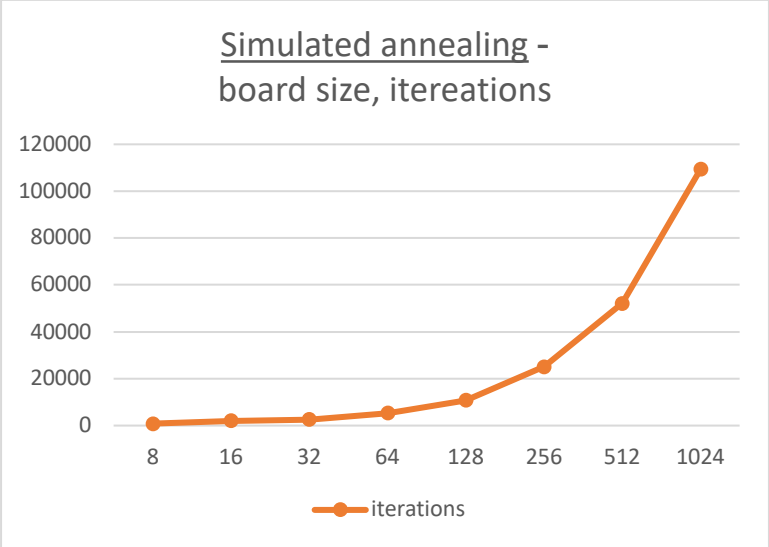
<u>מימוש האלגוריתם:</u>

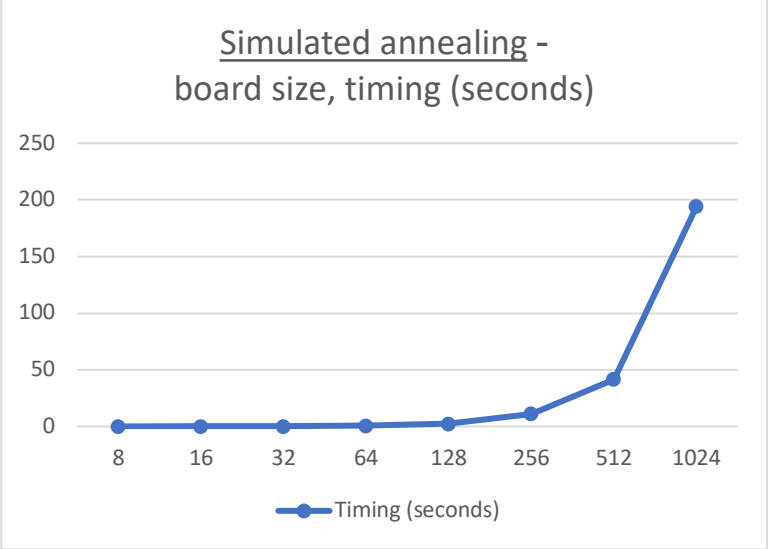
- הגרל פתרון התחלתי
- חשב פונקציית הערכה עבור הפתרון ההתחלתי
- כל עוד 0<temperature:
  - הקטן את temperature פי decay_rate
  - בחר שכן רנדומלי ע"י החלפת 2 אינדקסים רנדומליים בווקטור הפתרון.
  - חשב פונקציית ההערכה עבור השכן.
  - אם פונקציית הערכה של השכן קטנה יותר משל הפתרון הנוכחי או בהסתברות של:
    - אם פתרון השכן הוא הפתרון האופטימלי (פונקציית הערכה =0) החזר אותו.
    - אחרת, הגדר את הפתרון של השכן כפתרון הנוכחי והמשך בלולאה לצעד הבא.
  - אחרת (הפתרון הנוכחי טוב יותר משל השכן), המשך בלולאה לצעד הבא.

תוצאות : עבור temperature=10000,והגבלה של 150000 איטרציות עם decay_rate=0.99

N size	Timing (seconds)	iterations	Success rate
8	0.0266	840	86%
16	0.0934	1982	97%
32	0.1886	2535	100%
64	0.6459	5345	100%
128	2.4886	10745	100%
256	11.2402	25137	100%
512	41.7745	52097	100%
1024	194.1722	109292	100%

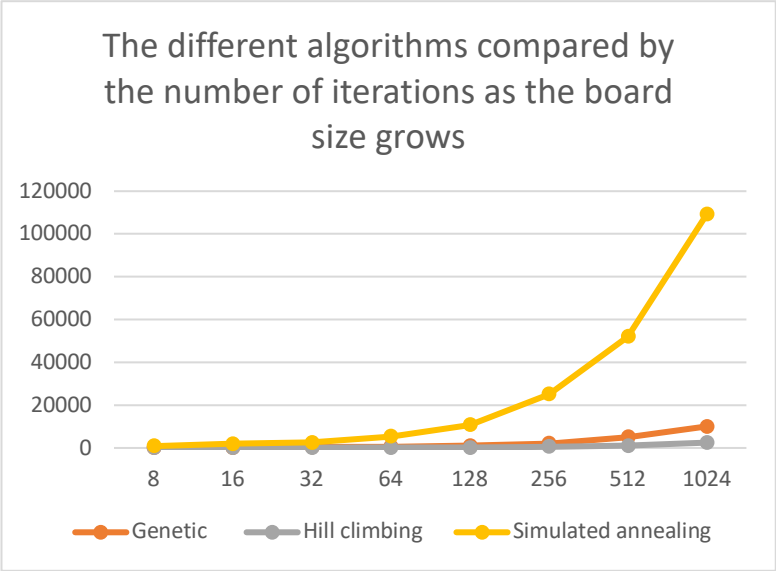
גם כאן באלגוריתם ניתן לראות שעבור n=8,16 האלגוריתם לא תמיד מתכנס לפתרון האופטימלי,
אבל מעבר לזה הוא מגיע ל100 אחוז דיוק.
נשים לב שהאלגוריתם מביא תוצאות מאוד מהירות ביחס לאלגוריתמים האחרים שראינו.

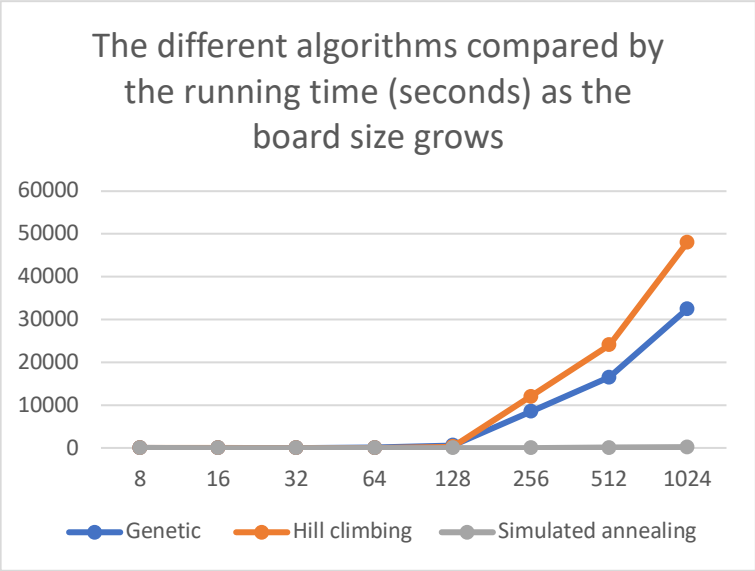




דו"ח הסקה על פי הנתונים שקיבלנו:

את המסקנות נוכל להסיק באמצעות שני הגרפים הבאים:





ההפרדה בין הגרפים נבעה מכך שיהיה קשה להשוות בין סדרי הגודל של הזמנים לסדרי הגודל של מספר האיטרציות.

- Branch & Bound- זהו אלגוריתם הדטרמיניסטי היחיד מבין האלגוריתמים שהצגנו הוא פתרון אופטימלי ב100% מהמקרים ועושה זאת באותו סדר גודל של זמן ריצה ובדיוק באותם כמות איטרציות בגלל הדטרמיניסטיות ומעבר במקרה הגרוע על כל הפתרונות.
  החיסרון העיקרי שלו כמובן זה זמן הריצה שמרקיע שחקים כבר עבור לוח בגודל 32 (10 **דקות**) בזמן ששאר האלגוריתמים הרנדומליים עושים זאת בקושי בשנייה אחת בודדת! ולכן גם מספר האיטרציות שלו גם עבור מספרים קטנים של n מאוד גבוהים 1000+.

- Gentic Algorithm – האלגוריתם הביא ביצועים סבירים לעומת שאר האלגוריתמים הרנדומליים כמו כן במספרים הקטנים n=8,16,32 הוא מתקשה למצוא פתרון ונתקע במינימום מקומי (לאחר הגבלה לא סבירה של איטרציות). זה יכול לנבוע מבחירה חוזרת ונשנית של הפתרונות הטובים ביותר שמאוד קשה לבצע בהם מוטציה כי הווקטור הוא מאוד קטן וניתן לראות באמת שכאשר n>32 האלגוריתם מביא תוצאות כמעט מושלמות מבחינת אחוזי דיוק.

- Hill climbing – עבור האלגוריתם ללא השכלולים מקבלים תוצאות מאוד לא טובות בגלל שהוא נוטה מהר מאוד להיתקע על מינימום מקומי.
  לעומת זאת האלגוריתם שהצגנו לעיל עם הרנדומיות והקפיצות לפתרון אחרונים במצב של מינימום מקומי מביא תוצאות כמעט מושלמות מבחינת אחוזי דיוק.
  נשים לב שהוא גם מוצא את הפתרון האופטימלי עם מספר האיטרציות הנמוך היותר מבין כל האלגוריתמים מכיוון שהוא עובר על כל השכנים ובוחר את הכי טוב.

כמו כן כמות האיטרציות הם יותר טובות מהאלגוריתם genetic ולכן מביא גם תוצאות יותר מהירות ממנו.

- Simulated Anneaning- זה הוא האלגוריתם שמביא את הביצועים הכי טובים מבחינת זמני ריצה. התוצאות שלו הרבה יותר מהירות פי כמה משאר האלגוריתמים ככל ש n יותר גדול. גם הוא כמו הgenetic עבור n=8,16 מתקשה להביא אחוזי דיוק מושלמים אבל הוא גם במקרים אלו מביא אחוזים מאוד גבוהים (97%).

מבחינת איטרציות הוא האלגוריתם שמבצע הכי הרבה וזה נובע מהרנדומיות שלו בתחילת הריצה שהטמפרטורה גדולה ולא עברו מספיק איטרציות אז הוא עובר המון בין מרחב הפתרונות ולכן אחרי המון איטרציות הוא מתחיל להתכנס ככל שהטמפרטורה קטנה ומספר האיטרציות עולה.

היתרון שלו על שאר האלגוריתמים הוא כמובן זמן הריצה וזה נובע מכך שהוא מחשב רק פעם אחת בכל איטרציה את פונקציית התועלת(כמות הקונפליקטים בלוח) עבור פתרון כלשהו, לעומת genetic שלפחות פעמיים + חישוב פונקציות אחרות כמו mutation crossover או לעומת hill climbing שמחשב עבור כל השכנים.

בעיית בונוס-

הסוגיה הבאה שנעלה מראה איך מספר הפתרונות האפשריים בבעיית n המלכות משתנה בהתאם לגודל הלוח.
עבור לוח בגודל 1x1 ישנו פתרון יחיד.
עבור לוחות בגדלים 2x2 ו 3x3 אין פתרון בכלל.
עבור לוח בגודל 4x4 ישנן 2 פתרונות.
אם נמשיך באופן זה, נקבל את הסדרה הבאה: 1, 0, 0, 2, 10, 4, 40, 92, וכו'..
לסדרה זו יש שם- A000170.

גילינו כי אין פתרון אנליטי מדויק לחישוב איברי סדרה זו, אך לא מזמן גילו פתרון מקורב לבעיה- כלומר, פונקציה שמקבלת את גודל הלוח ומחזירה את הכמות המקורבת של פתרונות אפשריים.
בפרויקט זה, באמצעות אלגוריתם backtracking חישבנו את מספר הפתרונות האפשריים לפי גדלי הלוח- האלגוריתם יעיל מלוחות בגודל 1x1 עד לוחות בגודל 25x25 מכיוון שלאחר מכן מספר הפתרונות הוא עצום.

קישור לגיט:

https://github.com/avivdimri/N_Queen_Problem