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.

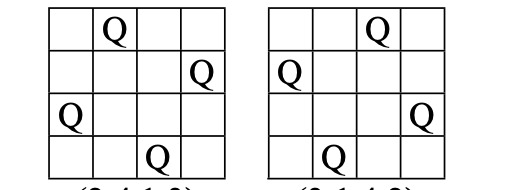
Square

Description automatically generatedלדוגמא,

מסקנה טריוויאלית שנובעת מהגדרת הבעיה היא שבכל עמודה ובכל שורה יכולה להיות בדיוק מלכה אחת. נסמן פתרון מייצג כווקטור באורך N, שהוא למעשה פרמוטציה של .כאשר האיבר הראשון מייצג את שורת המלכה בעמודה הראשונה, השני מייצג את שורת המלכה בעמודה השנייה וכו׳.

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,



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

נגדיר את הבעיה באופן פורמלי,

המשתנים הם , כך ש היא השורה שבה המלכה שממוקמת על הלוח בשורה ה .

התחום עבור כל אחד מהמשתנים הוא ,

האילוצים הם (כלומר, אין שתי מלכות באותה שורה) וכן (אין שתי מלכות באלכסון אחת לשנייה).

קיימים אלגוריתמים רבים לפתור את בעיית N המלכות.

בפרויקט שלנו נציג במספר אלגוריתמים היוריסטיים אפשריים לפתרון הבעיה ונשווה בניהם.

האלגוריתמים שאותם נציג הם,

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

Genetic algorithm-

אלגוריתם גנטי מבוסס על תורתו של דאווין- החזק שורד.

פסאודו קוד של האלגוריתם הגנרי,

* בחר אוכלוסייה התחלתית
* הערך בעזרת פונקציית הערכה את החוזק של כל אינדיבידואל באוכלוסייה
* כל עוד לא הגענו לתנאי העצירה (פונקציית המטרה),
  + בחר את האינדיבידואלים החזקים ביותר על פי פונקציית הערכה
  + הפגש זוג אינדיבידואלים באקראיות (זווג זוג פתרונות באקראיות).
  + עבור כל זוג החל עליהם את האופרטור של התרבות (crossover).
  + עבור כל צאצא החל את האופרטור של המוטציות
  + הערך את החוזק של כל אינדיבידואל (פתרון צאצא חדש).
* החזר את הפתרון הכי טוב.

נגדיר פונקציית הערכה שמקבלת פתרון ומחזירה מספר שמייצג כמה הפתרון הוא טוב.

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.

נגדיר פונקציית crossover שמקבלת שני פתרונות,

פונקציה זו מייצרת 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 שההפרש בניהם הוא 1 או 0, אזי היא תחליף את האיבר השני 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 צידי הווקטור.

מימוש האלגוריתם,

* הגדר קבוצת פתרונות )אוכלוסייה) בגודל 100 שכל אחד מהם מייצג פרמוטציה שונה של וקטור .
* 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 10 ריצות שונות עבור גדלי n שונים

|  |  |  |
| --- | --- | --- |
| N= 8 | 0.0093 sec | 83.4% |
| N=16 | 0.1541 sec | 80% |
| N=32 | sec 1.9953 | 100% |
| N=64 | 60.1977sec | 100% |
| N=128 | 389.4987 sec | 100% |

ניתן לראות שעבור n=8,16 האלגוריתם לא מחזיר תשובה אופטימלית תמיד .

אבל מביא את הפתרון האופטימלי ביותר מהר מbranch &bound

Branch and bound-

האלגוריתם עובד בצורת הפרד ומשול.

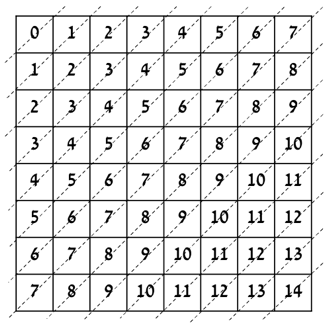
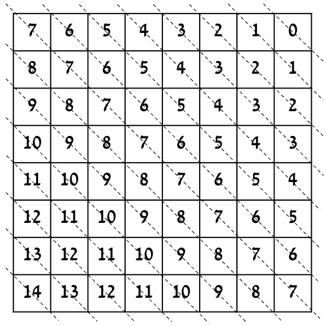
החלק של ה branch מתבצע על ידי חלוקה של כל הפתרונות החוקיים לחתיכות קטנות יותר.

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.

מימוש האלגוריתם:

* 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.



1. נבצע הצבות של מלכות אחת אחרי השנייה בכל עמודה כאשר מתחילים מהעמודה השמאלית.
   1. לפני הצבה של מלכה בשורה נבדוק תחילה בעזרת המערך הראשון אם השורה ה תפוסה או האם האלכסון השמאלי או הימני תפוסים באמצעות המערכים המיועדים.
      1. במידה והם תפוסים, נבדוק את השורה הבאה.
      2. אחרת, כלומר הם לא תפוסים אז נמקם את המלכה במיקום הזה ונמשיך באופן רקורסיבי לעמודה הבאה.
         1. אם גילינו בפתרון הרקורסיבי שאין פתרון אז נאפס את המיקום של המלכה ששמנו.
         2. אחרת, נחזיר את הפתרון.

בדקנו את הזמני ריצה של האלגוריתם בממוצע של 10 ריצות שונות עבור גדלי n שונים

|  |  |
| --- | --- |
| N=8 | 0.0006sec |
| N=16 | 0.0838 sec |
| N=32 | 614.28 sec |
| N>32 |  |

מעצם אופי האלגוריתם, בסוף בוודאות הוא יחזיר את הפתרון האופטימלי עם אותו סדר גודל של זמני ריצה מכיוון שהוא דטרמיניסטי ועובר על כל האפשרויות אך עושה זאת בזמנים מאוד גדולים.

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 במקרה שלנו יגריל פתרון לבעיית N המלכות.

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.
   1. אם קיים פתרון כזה, עבור אליו.
   2. אחרת, אם קיים פתרון עם ערך זהה בפונקציית ההערכה, הגרל פתרון כלשהו מהפתרונות השכנים ועבור אליו.
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.

או הגרל פתרון אקראי לבעיה.

Top of Form

Bottom of Form

Top of Form

Bottom of Form

|  |  |
| --- | --- |
| 0.000829 sec | N=8 |
| 0.019 sec | N=16 |
| 2.846 sec | N=32 |
| 15.2 sec | N=64 |
| sec 2.4516 | N=128 |

Simulated annealing

אלגוריתם זה יעבוד בצורה דומה לhill 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.

פסאדו קוד:

* הגרל פתרון התחלתי
* חשב פונקציית הערכה עבור הפתרון ההתחלתי
* כל עוד temperature>0:
  + 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 50000 איטרציות עם decay\_rate=0.99

ֿ

|  |  |  |
| --- | --- | --- |
| 90.2% | 0.0284 sec | N=8 |
| 96.9% | 0.0916 sec | N=16 |
| 100% | 0.1843 sec | N=32 |
| 100% | 0.5707 sec | N=64 |
| 100% | sec 2.2045 | N=128 |
| 100% | 13.8630 sec | N=256 |
| 100% | 48.1771 sec | N=512 |

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