סודוקו

מגישים :

שון פוזנר - 203830930
יניב מדמון - 204293005

מבוא:

משחק הסודקו הינו תשבץ מספרים, הבנוי על גבי לוח משבצות שגודלו 9x9, המורכב מתשעה ריבועים בני תשע משבצות.
מטרת מהשחק הינה למקם ספרות 1 עד 9 על גבי הלוח כך שבכל טור, שורה וריבוע לא תופיע ספרה יותר מפעם אחת.
לוח המשחק מאותחל כך שמספר משבצות מכילות ערך שאינו ניתן לשינוי.
לבעיית סודוקו יכולים להיות מספר פתרונות חוקיים, ואין הבדל או עדיפות לפתרון אחד על האחר, כל עוד הפתרון עומד באילוצי המשחק.

משחק של 9×9 הוא הנפוץ ביותר, אולם אפשריות גם חידות בגדלים שונים, נוסח קשה במיוחד הוא בגודל של 16×16 משבצות עם 16 אזורים, של 4×4. גודל הלוח ניתן לשינוי ובעזרתו ניתן להקשות או להקל על סיבוכיות הבעיה. יש מדדים נוספים לקושי כמו מספר פתרונות אפשרי, מספר נקודות החלטה לא טריוויאליות (משבצות בהם לא ניתן להסיק ישירות את ערכה של המשבצת) וכו'.

בעבודה זו נציג גישות שונות לפתירת בעיית הסודוקו, בלוחות ברמות שונות ובגדלים שונים.
נסביר את תהליך העיבוד של כל שיטת פתרון ונשווה בין הביצועים של מתודות הפתרון השונות.

רקע:

גרסה ראשונה למשחק הסודוקו, הופיעה ב־6 ביולי 1895 בעיתון הצרפתי La France. המשחק במתכונתו הנוכחית הופיע לראשונה ב-1979 באחד המגזינים של דל (Dell). גרסה זו הומצאה על ידי הווארד גארנס, ארכיטקט אמריקאי בן 74 שפרש לגמלאות. גארנס לא רשם פטנט או תבע זכויות יוצרים, בעקבות כך הסודוקו הפך להמצאה ברישיון חופשי ללא צורך בתמלוגים.

שם המשחק "סודוקו", הופיע לראשונה ביפן בשנת 1984 ונקרא: "סואוז'י ווא דוקושין ני קאגירו" שמשמעו - "מוגבל למספר יחיד" . בהמשך קוצר השם ל"סודוקו", שמשמעו ביפנית (נוסף על היותו ראשי תיבות של הביטוי לעיל) "מספר יחיד" ביפנית.

ויין גולד, בעברו שופט בהונג קונג, פיתח תוכנית מחשב המייצרת תשבצי סודוקו, ובשנת 2004 החל לספקם לעיתון Times הבריטי. כיום החידות מפורסמות בכל העולם וכמעט שאין אדם שלא מכיר את משחק "הסודקו".

בקרב קהילת המתמטיקאים משחק הסודוקו נחקר רבות. בעיית משחק הסודקו מוגדרת כבעיה NP שלמה עבור חידות בגודל n2xn2 כאשר במקרה של n=3 הבעיה נפתרת בקלות על ידי המחשב.

דרכי פתרון:

הדרך הנאיבית ביותר לפתור את הסודקו היא על ידי חיפוש נסוג (brute force), נעבור על כל המשבצת בלוח באופן סידרתי, נמקם בה מספר מתוך האופציות האפשריות אשר אינם מפרים את אילוצי החידה, כך נמשיך. במידה והגענו למבוי סתום נחזור לאחור וננסה אפשרות שונה כך עד מציאת הפתרון. הבעיה בדרך זו היא שבמקרה הגרוע (The worst case) נעבור על למעלה מ 600 מיליון אפשריות בסודקו בגודל 9 X 9 .

CSP(Constraint Satisfaction Problem)

מודל CSP הינו מודל לבעיות בעלות שלושה מאפיינים : רשימת משתנים , דומיין (טווח) מוגדר לכל משתנה ורשימה של אילוצים בין המשתנים .
הצגת בעיית הסודוקו כבעיית אילוצים ופתירת בעזרת שיטות שונות.

- Backtracking

פתרון בעזרת אלגוריתם רקורסיבי המתחיל מתא מסוים בלוח (אחד ממשתני בעיה), בוחר לו ערך מהטווח הספציפי לאותו תא (מהדומיין שלו) וממשיך לתא הבא על סמך בחירתו האחרונה (בדומה לחיפוש נסוג), וכך הלאה. במקרה והאלגוריתם יגיע לתא שאין אפשרות למלא לו ערך מהטווח הקיים, הדומיין של תא ריק ולכן לא ניתן לעמוד באילוצי השאלה, האלגוריתם יחזור צעד אחורה וינסה לשנות את בחירתו האחרונה . כך ימשיך עד מציאת פתרון הרצוי. קישור לאנימציית הדגמה

Backtracking ללא פונקציה יוריסטית הינו אלגוריתם לא יעיל ולכן נוסיף לו פונקציות יוריסטיות שונות על מנת לייעל את זמן הפתרון.

- Local Search

מודל של אלגוריתמים Local Search הינו מודל המתייחס לתוצאה ולא לדרך.
הכוונה היא שאלגוריתמים מסוג זה קובעים ערכים מקומיים מסוימים ועל פי הם מתקדמים למציאת הפתרון הרצוי. ההתקדמות לפתרון נעשית על ידי שינוי ערכי המשתנים במטרה למזער את כמות הפרות האילוצים.

ישנם סוגים שונים של אלגוריתמים מסוג זה לדוגמא אלגוריתם Hill climbing הינו אלגוריתם שמתחיל במילוי הלוח באופן שרירותי ולאחר מכן ישפר מקומית את ערכי התאים על מנת להגיע למצב שבו ערכי הלוח יקימו את אילוצי החידה. הבעיה בדרך זו היא שאנחנו לא מאפשרים מהלכים מקומיים אשר במבט מקומי לא רצויים (מגדילים את מספר הטעויות בלוח) אך רק דרך מעבר מהם ניתן לפתור אץ הלוח. לכן בגישה זו אנו סביר שנגיע לנקודות בהם האלגוריתם נעצר והלוח לא נפתר בשלמותו.

גם את שיטת פתרון זו ניתן ליעל בעזרת מיקום ערכים מקומיים על בסיס הסתברויות, התחלות חדשות ממיקומים שונים (הצבות פתיחה שונות), זיכרון נקודות בדרך וחזרה אליהם וכו'.
בדו"ח זה לא נפרט ונדגים גישות המבוססות על עקרון החיפוש מקומי.

- Liner Programming

מודל של Liner Programming הינו מודל שמתייחס לבעיה כבעיית אופטימיזציה של ביטוי לינארי תחת אילוצים. בעיות תכנות לינארי מורכבות ממשתנים , אילוצים ופונקציה לינארית המכילה את המשתנים שמטרתה להביא למינימום או מקסימום תוצאה . לדוגמא קיימת שיטה בשם "סימפלקס" המחפשת את הפתרון האופטימלי ע"י מעבר בין פתרונות פינתיים במבנה הגיאומטרי (סימפלקס) שנוצר בגרף על ידי אילוצי המשתנים. שיטה זו מסתמכת על כך שלכל פתרון אופטימלי, קיים פתרון פינתי אופטימלי.

בפרויקט זה בחרנו להשוות בין Backtracking בשילוב פונקציות יוריסטיות שונות ובין אלגוריתם במודל Liner Programming תוך תלות ברמת הקושי של הסודקו וגודלו.

רקע לדו"ח:

Backtracking

ישנן שלוש פונקציות יוריסטיות מרכזיות שבחרנו לבדוק בפרויקט:

1. **(MRV) Minimum remaining values**
2. **(FC) Forward checking**
3. **(ARC) Arc consistency**

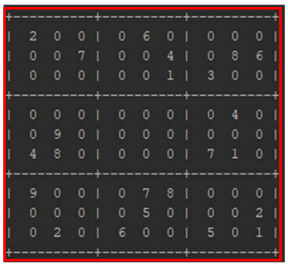
נסביר כל אחת מהן בפירוט.

1. (MRV) Minimum remaining values

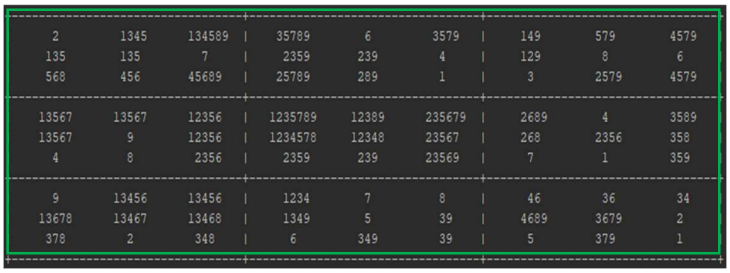
פונקציה יוריסטית זו מנסה ליעל את בחירת המשבצת בה נציב ערך. המוטיבציה היא לנסות לצמצם טעויות בתחילת הדרך ובכך נקטין משמעותית את פתיחת ענפי הרקורסיה באלגוריתם.
דרך פעולה - נבדוק באיזו משבצת יש את מספר אפשרויות הצבה הנמוך ביותר ונבחר בה לפיתוח העץ. בחירה במשבצת זו תביא לכדי פיתוח עץ תקין בסבירות גבוהה וימנע פיתוח ענפים שבסופם נקבל פתרון שגוי.

לדוגמא:

- כאשר נקבל את הלוח האדום נוכל על ידי פעולות חישוב אילוצים, לצמצם את הדומיין שלו ללוח המוצג בירוק.



2	0	0	0	6	0	0	0	0
0	0	7	0	0	4	0	8	6
0	0	0	0	0	1	3	0	0
0	0	0	0	0	0	0	4	0
0	9	0	0	0	0	0	0	0
4	8	0	0	0	0	7	1	0
9	0	0	0	7	8	0	0	0
0	0	0	0	5	0	0	0	2
0	2	0	6	0	0	5	0	1



2	1345	134589	35789	6	3579	149	579	4579
135	135	7	2359	239	4	129	8	6
568	456	45689	25789	289	1	3	2579	4579
13567	13567	12356	1235789	12389	235679	2689	4	3589
13567	9	12356	1234578	12348	23567	268	2356	358
4	8	2356	2359	239	23569	7	1	359
9	13456	13456	1234	7	8	46	36	34
13678	13467	13468	1349	5	39	4689	3679	2
378	2	348	6	349	39	5	379	1

- כעת כדי לבחור באיזו משבצת להתחיל להציב ערכים, נוכל לבדוק איזו משבצת מכילה את הדומיין הקטן ביותר. למשל בבחירה הראשונה נבחר במשבצות עם ערך אפשרי אחד בלבד – נבחר את התא (1,1) המכיל את הערך 2.
- כך נמשיך בבחירה עד שנעבור על כלל התאים שגודל הדומיין שלהם 1.
לאחר מכן נבחר תאים המכילים שני אפשרויות כמו משבצת (7,9) המכילה 3,4. נבחר להציב בתא זה את אחד הערכים ונמשיך ע"פ backtracking.

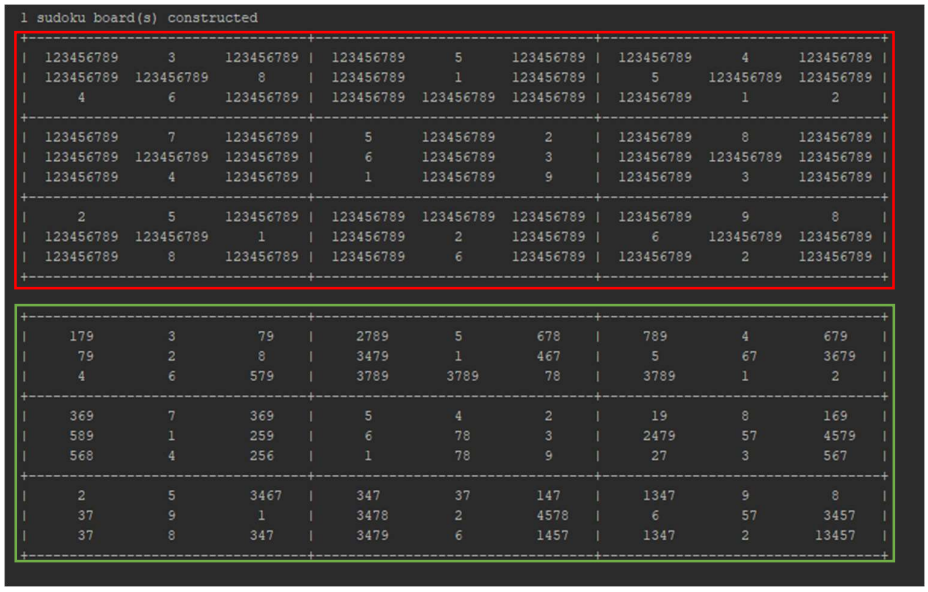
לסיכום **MRV** הינה יוריסטיקה המגדירה את סדר בחירת המשבצות אליהם נעבור. כך שתחילה נבחרות משבצות עם דומיין נמוך ובסוף משבצות עם אפשרויות הצבה רבות יותר.

2. (ARC) Arc consistency

יוריסטית זו תאפשר לנו כבר בתחילת התהליך להקטין את הדומיין שניתן להציב בכל תא. מספר האפשרויות הבסיסי של כל תא בסודוקו הינו 9 אך ניתן בעזרת אילוצי המשחק להקטין את מספר האפשרויות. היוריסטיקה תבדוק אילו ערכים ניתן להשמיט מהדומיין של כל תא ובכך תקטין מראש את גודל העץ שנוצר במהלך הbacktracking.

דרך פעולה – נפעיל על הלוח את האלגוריתם AC-3, אשר יעבור על האילוצים השונים ועל פי הגדרתם מקטין את הדומיין של המשבצת. כל משבצת שהשתנה לה הדומיין משפיעה כעת על תאים נוספים. לכן לאחר כל צמצום של דומיין נוסיף בדיקות נוספות של כל התאים המושפעים מהשינוי (מומש בקוד בעזרת מחסנית המכילה זוגות שיש לבדוק).
כך נעדכן את דומיין התאים, עד התייצבות, מצב בו לא ניתן לצמצם עוד את האפשרויות בלוח.

נראה דוגמא:



```
1 sudoku board(s) constructed
+-----------------------------+-----------------------------+-----------------------------+
|  123456789     3     123456789 |  123456789     5     123456789 |  123456789     4     123456789 |
|  123456789  123456789     8     |  123456789     1     123456789 |     5     123456789  123456789 |
|     4          6     123456789 |  123456789  123456789  123456789 |  123456789     1          2     |
+-----------------------------+-----------------------------+-----------------------------+
|  123456789     7     123456789 |     5     123456789     2     |  123456789     8     123456789 |
|  123456789  123456789  123456789 |     6     123456789     3     |  123456789  123456789  123456789 |
|  123456789     4     123456789 |     1     123456789     9     |  123456789     3     123456789 |
+-----------------------------+-----------------------------+-----------------------------+
|     2          5     123456789 |  123456789  123456789  123456789 |  123456789     9          8     |
|  123456789  123456789     1     |  123456789     2     123456789 |     6     123456789  123456789 |
|  123456789     8     123456789 |  123456789     6     123456789 |  123456789     2     123456789 |
+-----------------------------+-----------------------------+-----------------------------+

+-----------------------------+-----------------------------+-----------------------------+
|    179         3         79    |    2789        5        678    |    789         4        679    |
|     79         2          8    |    3479        1        467    |     5         67       3679    |
|      4         6        579    |    3789       3789       78    |    3789        1          2    |
+-----------------------------+-----------------------------+-----------------------------+
|    369         7        369    |      5         4          2    |     19         8        169    |
|    589         1        259    |      6        78          3    |   2479        57       4579    |
|    568         4        256    |      1        78          9    |     27         3        567    |
+-----------------------------+-----------------------------+-----------------------------+
|      2         5       3467    |    347        37        147    |   1347         9          8    |
|     37         9          1    |   3478         2       4578    |      6        57       3457    |
|     37         8        347    |   3479         6       1457    |   1347         2      13457    |
+-----------------------------+-----------------------------+-----------------------------+
```

בריבוע האדום ניתן לראות את הלוח עם כל דומיין של כל משתנה כאשר הוא מודפס במקום של המשתנה לפני שצמצמנו את הטווח לכל תא על פי אילוצי הבעיה.

למשל במקום (0,0) אנו רואים את המספרים "123456789" כלומר במקום זה אין ערך קבוע שניתן בנתוני הבעיה ולכן נוכל כביכול בהתחלה להציב את טווח המספרים האפשרי. דוגמא נגדית נראה במקום (0,1) אנו רואים את המספר 3 בלבד שכן ישנו מספר יחיד שהגיע עם נתוני הבעיה.

בריבוע הירוק ניתן לראות את הלוח עם הדומיין של כל משתנה כאשר הוא מודפס במקום של המשתנה לאחר שצמצמנו את הטווח לכל תא על פי אילוצי הבעיה.

למשל במקום (0,0) אנו רואים את המספרים 1,7,9 כלומר במקום זה הדומיין שלנו מוגבל לשלושה מספרים לאחר שיישמנו את האילוצים על פי נתוני הבעיה. נשים לב ששלושת המספרים הנ"ל הינם היחידים שלא מופיעים פעמיים באותה שורה, עמודה או ריבוע .

דוגמא לשלבי אלגוריתם AC-3:

(1)

	A	B	C	D	E	F	G	H	I
A		3			5			4	
B			8		1		5		
C									
D		7		5		2		8	
E				6		3			
F		4		1		9		3	
G	2	5						9	8
H			1		2		6		
I		8			6			2	

- תחילה נסתכל על הלוח ונגדיר כי כל תא ריק הינו מכיל את כל אפשרויות ההצבה מספרים בין 1 ל 9.
- נעבור על האילוצים השונים ונצמצם את מספר האפשרויות (הדומיין) בכל תא.
אילוצי שורה, עמודה וריבוע.

(2)

	A	B	C	D	E	F	G	H	I
A		3			5			4	
B		2,9	8		1		5		
C									
D		7		5		2		8	
E				6		3			
F		4		1		9		3	
G	2	5						9	8
H			1		2		6		
I		8			6			2	

- נסתכל על התא בכתום (B,B) לפני הקטנת הדומיין יש בו את כלל הטווח
- כעת נריץ בדיקת אילוצים של תא זה:
לפי אילוצי שורה - 8,1,5 אינם השמה טובה
לפי עמודה – 3,7,4,5,8
לפי ריבוע – 3,8
לכן ניתן לשים בתא זה רק 2,9.
- מבדיקת האילוצים התגלו קשרים למשבצות שונות, משבצות אלו צריכות להיבדק מחדש.

(3)

	A	B	C	D	E	F	G	H	I
A		3			5			4	
B		2,9	8		1		5		
C									
D		7		5		2		8	
E				6		3			
F		4		1		9		3	
G	2	5						9	8
H		9	1		2		6		
I		8			6			2	

נעדכן את הדומיין של כל התאים עד שנגיע לתא HB

- נסתכל על תא HB ונבדוק את האילוצים:
לפי אילוצי שורה – 1,2,6 אינם השמה טובה
לפי עמודה – 3,7,4,5,8
לפי ריבוע – 1,5,8
- **לכן ניתן לשים בתא זה רק 9.**

(4)

	A	B	C	D	E	F	G	H	I
A		3			5			4	
B		2,9	8		1		5		
C									
D		7		5		2		8	
E				6		3			
F		4		1		9		3	
G	2	5						9	8
H		9	1		2		6		
I		8			6			2	

- מבדיקת האילוצים התגלו קשרים למשבצות שונות באפור, אותם נבדוק שוב (למשל BB)
- נעדכן את הדומיין של כל התאים עד שנגיע לתא BB שוב ונגיע לערך 2

(5)

	A	B	C	D	E	F	G	H	I
A		3			5			4	
B		2	8		1		5		
C									
D		7		5		2		8	
E				6		3			
F		4		1		9		3	
G	2	5						9	8
H		9	1		2		6		
I		8			6			2	

נשים לב כי בדוגמא הצלחנו למצוא את הערך של תא (B,B) ע"י כך שעברנו על האילוצים וצמצמנו את הדומיין ללא צורך בפעולות רקורסיביות ו"הימור" על ערך.

לסיכום נרצה להפעיל ARC על הלוח עם קבלתו ובכך לצמצם את גודל העץ שנוצר בזמן ה backtracking.

3. (FC) Forward checking

יוריסטיקה זו תעזור לנו לבדוק האם ההצבה שבחרנו חוקית עוד לפני הצבתה בפועל. כך נחסוך הצבות חסרי סיכוי שיעמיקו את עץ הרקורסיה לחינם.
דרך פעולה – לאחר שבחרנו משבצת וערך מסוים נבדוק את השפעת הצבה על הלוח.
נכניס את הערך לתא ונבדוק האם כל האילוצים מסופקים, כמו כן נבצע צמצום לדומיין על בסיס הערך שהצבנו. לאחר תהליך זה אם קיבלנו הפרה של האילוצים (תא ריק ללא אפשרויות הצבה חוקיות) נוותר על ההצבה שנבדקה, אחרת נכניס את הערך לתא ונמשיך בתהליך.
במימוש שלנו השתמשנו באלגוריתם Ac-3 (מוסבר בArc) כדי לבצע את צמצום הדומיין.
לסיכום לפני כל הצבת ערך נריץ forward checking נבדוק האם ההצבה הגיונית אם כן נציבו אחרת ננסה הצבה אחרת.

Liner Programming

ננתח את בעיית סודוקו על ידי בעיית סיפור אילוצים:
בכדי לעשות זאת נבצע רדוקציה לבעיה דומה אשר קל יותר לכתוב לה אילוצים אך שקולה לבעיה המקורית.

רדוקציה- נמיר את לוח המשחק אשר במקור דו ממדי, להיות תלת ממדית. תוכן כל תא (הערך שלו) יוגדר להיות הממד הנוסף. הבעיה החדשה הינה למצוא השמה כך שסכום כל שורה, סכום כל עמודה וסכום עומק שווה ל1 בדיוק. כמו כן בכל סכום הערכים בריבוע קטן של (3x3x3) יהיה 1.
השמה חוקית בתא הינה 0 או 1 בלבד (הדומיין של כל תא). את בעיה זו נציג כבעיית אילוצים.

ניתוח המשחק כבעיית סיפוק אילוצים בשלמים:

- **פרמטרים:** נגדיר מטריצה תלת ממדית כך שכל פרמטר יקבל תא.
  התאים יוגדרו ע"פ מיקום על גבי הלוח (שורה ועמודה) והערך במיקום זה. ולכן נקבל סך הכל $9^3 = 729$ משתנים (תאים במטריצה) אשר נסמנם כך $x_{r,c,v}$.
- **אילוצים:**
  - אילוצי שורה - $\sum_{r=1}^{9} x_{r,c,v} = 1 \; for \; v, c \; \epsilon \; [1,9]$
  - אילוצי עמודה - $\sum_{c=1}^{9} x_{r,c,v} = 1 \; for \; v, c \; \epsilon \; [1,9]$
  - אילוצי ערכים - $\sum_{v=1}^{9} x_{r,c,v} = 1 \; for \; v, c \; \epsilon \; [1,9]$
  - אילוצי ריבועים - $\sum_{r=3p-2}^{3p} \cdot \sum_{c=3q-2}^{3q} x_{r,c,v} = 1 \; for \; v \; \epsilon \; [1,9] \; and \; p, q \; \epsilon \; [1,3]$
  - אילוצי נתונים – כל ערך ידוע על הלוח יקבל ערך 1 בתא המתאים במטריצה.
  - אילוץ טריוויאלי – כל תא יקבל רק אחד או אפס.

את בעיית אילוצים זו נכניס לפותר בעיות מוכר, את התשובה שנקבל נמיר בחזרה לפתרון של לוח סודוקו חוקי. במימוש השתמשנו בחבילה PuLP ובפותר CBC.

מהלך הניסוי:

מימשנו תוכנה הפותרת לוחות סודוקו בגדלים שונים באמצעות backtracking (לוחות מהצורה n^2xn^2). מימוש הbacktracking הבסיסי שיצרנו אינו יעיל מבחינת זמן הפתרון ועוצב במטרה לאפשר הוספת יוריסטיקות שונות בצורה פשוטה. בכדי להשוות בין יעלות הפותרים השונים (היוריסטיות השונות) בצורה טובה, נספור את כמות האינטראקציות החוזרות (ניסיונות כושלים) עד הפתרון.
היוריסטיקות אותם מימשנו הם- Forward checking, Arc, Mrv המוסברים למעלה. אלו פועלים על גבי האלגוריתם הבסיסי של הbacktracking.
בנוסף מימשנו פותר סודקו נוסף המתבסס על Liner Programming פותר זה משתמש ב cbc solver.
לשם הניסוי השתמשנו במספר רב של קבצי לוחות סודוקו שונים, המחולקים לפי רמות וגודל.

הניסוי כלל 3 חלקים:

1. השוואה בין שימוש נאיבי בbacktracking לבין שימוש בו עם היוריסטיקות השונות הן בהיבטי זמן והן בכמות האינטראקציות.
2. השוואה בין שילובים של היורסטיקות כתלות בקושי הלוח.
3. השוואה בין פותר המשתמש בbacktracking לבין פותר המבוסס Liner Programming (השוואת זמנים בלבד)

כיצד להריץ:

הפעלת התוכנה ומשם לבחור:

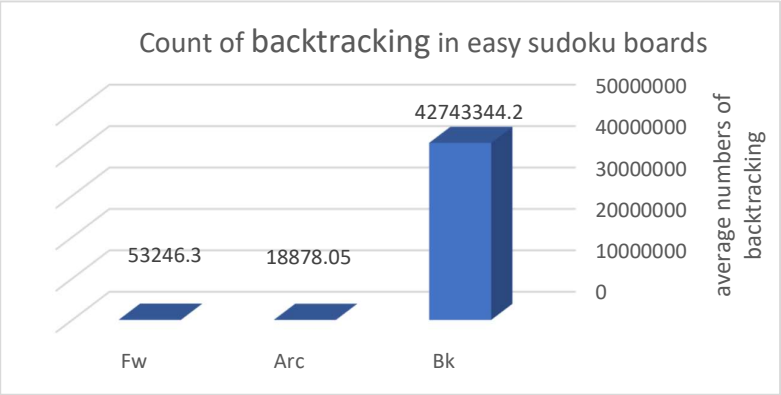
1. בחירת "1" - פתירת לוח מטריצה בהכנה אישית:
   a. בוחרים איזה לוח רוצים לפתור - קל, קשה, גדול (16x16) או ענק (25x25).
   b. בוחרים באיזה פותר רוצים להשתמש - בעזרת backtracking או liner Programming
      i. כאשר ניתן להגדיר איזה יורסטיקות נרצה להשתמש
2. בחירת "2" - הכנת הדוח:
   a. מריץ את כלל הבדיקות המפורטים בדו"ח זה, מייצר מסמכי טקסט עם התוצאות.
   b. תהליך זה לוקח זמן רב! (עשרות שעות לבדיקה מלאה).
   c. כלל התוצרים של הבדיקות שלנו מפורסמים בקישור בתקיה report כולל קבצי האקסל.

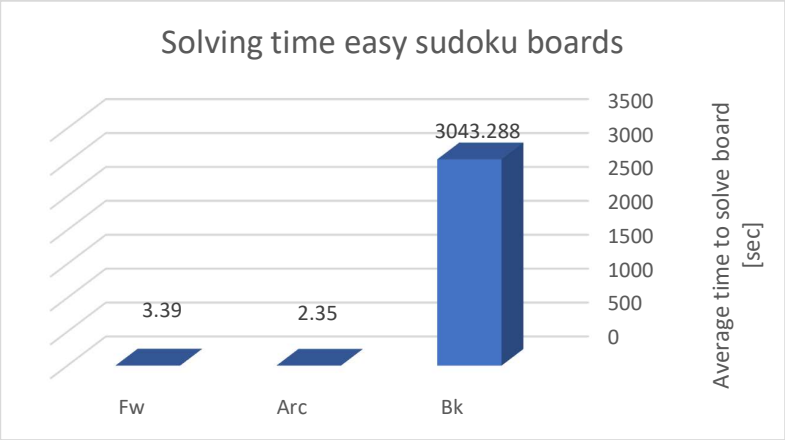
קישור לקוד בקישור

ממצאים:

1. השוואה בין שימוש נאיבי בbacktracking לבין שימוש בו עם היוריסטיקות השונות, תוך השוואת משך זמן החישוב וכמות האינטראקציות.
   - נבדקו 20 לוחות ברמת קל, הלוחות נלקחו מהאינטרנט ומצורפים עם הקוד.
   - הוגדר חסם עליון של מספר האינטראקציות (100 מיליון).

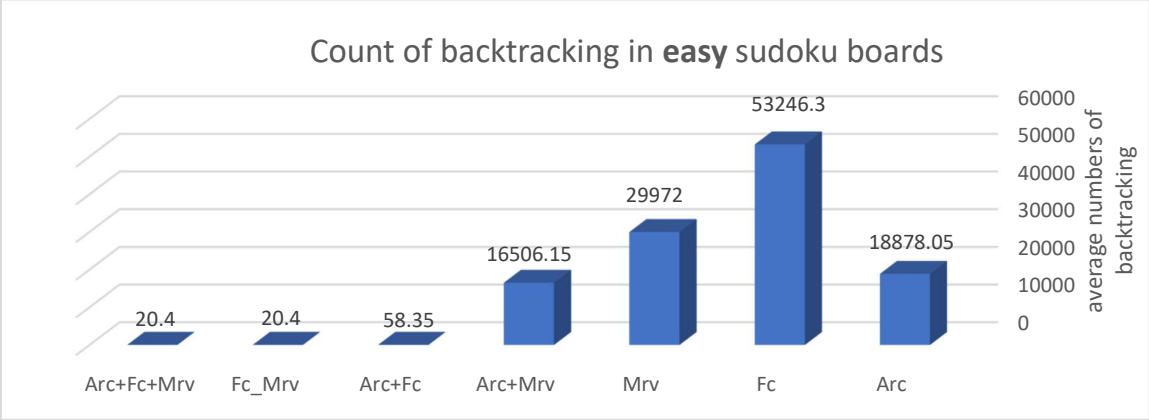
Time to solve [sec]	Count of backtracking	Heuristics
3043.287988	42743344.2	backtracking
2.35171262	18878.05	Arc
3.390064251	53246.3	Fc



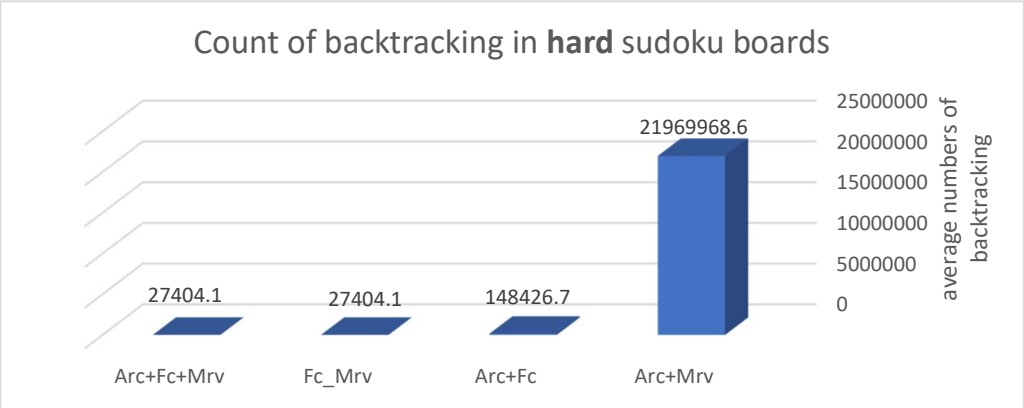


2. השוואה בין שילובים של היורסטיקות כתלות בקושי הלוח וגודלו.
   - נשים לב שמספר האינטראקציות של Fc+Mrv וArc+Fc+Mrv זהה וזאת משום שהמימוש של שניהם מתבסס על Ac-3 ולמעשה בFc הראשון תתבצע פעולת הArc ולכן כאשר יש אותו סדר בחירת משבצת אין משמעות בביצוע Arc .
   - בבדיקת לוחות ברמת קושי גבוה נבדקו רק שילובי יורסטיקות, ללא שילובים התקבלו תוצאות נמוכות (לרוב מחוץ לרף התחתון שהוגדר).
   - נבדקו 10 לוחות קשים, ו20 לוחות קלים.

Time to solve[sec]	Count of backtracking	Heuristics
2.35171262	18878.05	Arc
3.390064251	53246.3	Fc
4.975152087	29972	Mrv
2.364226282	16506.15	Arc+Mrv
0.082680404	58.35	Arc+Fw
0.015210915	20.4	Fc+Mrv
0.069314516	20.4	Arc+Fc+Mrv

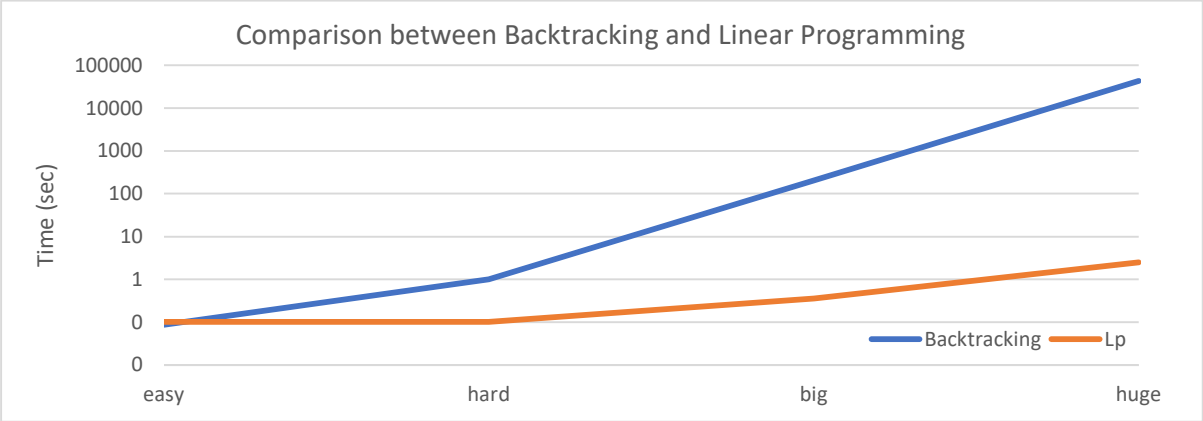


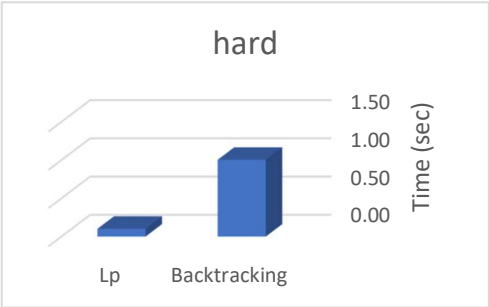
Time to solve[sec]	Count of backtracking	Heuristics
7268.247409	21969968.6	Arc+Mrv
18.61218624	148426.7	Arc+Fc
4.24531312	27404.1	Fc+Mrv
3.876014161	27404.1	Arc+Fc+Mrv

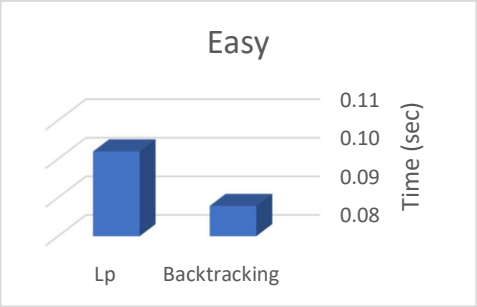


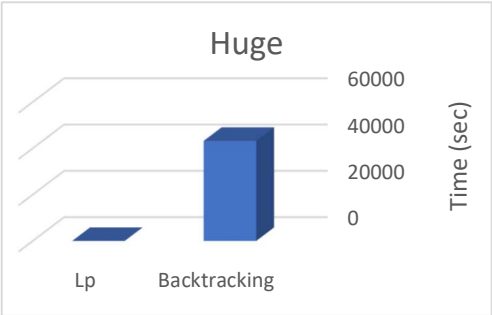
3. השוואה בין פותר המשתמש בbacktracking לבין זה מבוסס Liner Programming (השוואת זמנים בלבד)
   - השוואה נעשתה על:
     - 1000 לוחות ברמה קלה.
     - 95 לוחות ברמה קשה.
     - 3 לוחות גדולים (16x16) .
     - לוח ענקי אחד (25x25).
   - כדי להמחיש את המגמה של הגרף ציר הזמן מוצג על פי קנה מידה לוגריתמי.
   - זמן ההמתנה המקסימלי שהוגדר להמתנה לפתרון הינו 12 שעות, ניתן לראות כי בלוח הענק לא התקבל פתרון בזמן.

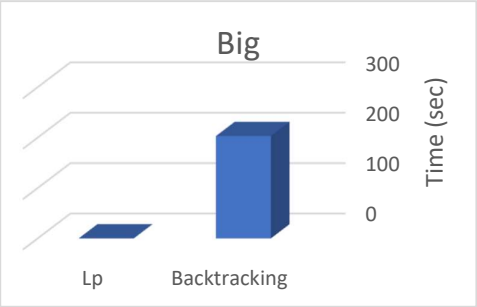
Huge	big	hard	easy	
43200	203.2847	1.004635	0.087826132	**Backtracking**
2.484592	0.353958	0.102392	0.101938328	**Liner Programming**











מסקנות:

1. ניתן לראות מהממצאים כי פתירת סודקו בעזרת backtracking ללא יוריסטיקות לקחה מספר אינטראקציות גבוהה וזמן. נוסף על כך, ככל שרמת הקושי עלתה לקח יותר זמן כדי להגיע לפתרון אם בכלל. מטריצות קשות לא נפתרו כלל (מעל מאה מיליון אינטראקציות) וכך גם חלק מהמטריצות הקלות. נציין שוב שבמימושים אחרים שנוסו למימוש בסיסי של backtracking, זמן הפתרון היה קצר בהרבה מזה המוצג בדו"ח, אך גם בהם לוחות קשים לקחו זמן רב (כמה שעות).
2. הוספת כל אחת מהיוריסטיקות משפרת את ביצועיו של backtracking. יתרה מכך, הרכבה של היורסטיקות אחת על השנייה משפרת עוד יותר את הביצועים כך שנעדיף להשתמש בשילוב כמה יורסטיקות יחד.
3. המימוש של הFc ושל Arc נעזרים ב AC-3 ולכן בצרוף של Mrv , המכוון אותם לאותו סדר משבצות יוצא שאנחנו מבצעים בדיוק את אותה כמות אינטראקציות.
4. מבחינת הפתרון עצמו, כל יוריסטיקה מאפשרת לנו להגיע לפתרונות שונים (למעט הרשום בסעיף 3).
5. ככל שהלוחות קשים וגדולים יותר כך הרצון שלנו להשתמש במספר יורסטיקות במקביל יגדל. למשל פתירת לוח גדול ללא שילוב היורסטיקות לקח מספר שעות ואילו בעזרת כל היורסטיקות נפתור במספר דקות.
6. מסקנה מניסיונות המימוש השונים מגלה שבמקרה של סודקו קל אין צורך להשתמש במבנים מורכבים ויורסטיקות, אלא להיפך האלגוריתם הפשוט נותן ביצועים טובים.
אך כאשר ניסינו לפתור מטריצות קשות אלגוריתם פשוט ללא מבנים מורכבים או יורסטיקות גילנו יעילות נמוכה ורק בעזרת מימוש מורכב ויורסטיקות הצלחנו להגיע לפתרון בזמן סביר.
7. פתרון בעזרת Liner Programming יעיל מאוד מבחינת זמן חישוב הפתרון בעיקר בבעיות מסובכות.
ככל שהקשנו את הבעיה גדל פער הביצועים בין הגישות. בלוחות הקלים נמצא כי השימוש בbacktracking עדיף (בפער קטן), אך ככל שהבעיה מורכבת יותר ומספר האפשרויות גדל, התוצאות שהתקבלו הראו בברור כי גישת Liner Programming עדיפה משמעותית.
8. הבדיקות בניסוי הוגבלו לגודל המטריצות אותם מצאנו 25x25 , בעיות בסיבוך זה עדין נפתרו בקלות ע"י Liner Programming אך בלא הצליחו להיפר בעזרת backtracking (במגבלת הזמן שניתנה).

סיכום:

משחק הסודקו הינו מורכב יותר משנראה, וטומן בו בעיות מתמטיות רבות. בדו"ח זה סקרנו רק נדבך קטן מאוד מעולם זה.
מצאנו כי פתרונות פשוטים עשויים לעבוד בצורה טובה לבעיות מסוימות, אך עם זאת בבעיות מורכבות אין מנוס מלהשתמש בכלים מתוחכמים ויעילים יותר.
כמו כן ראינו את כוח החישוב שיש בשימוש בפונקציות יוריסטיות וLiner Programming .

מקורות:

https://sandipanweb.wordpress.com/2017/03/17/solving-sudoku-as-a-constraint-satisfaction-problem-using-constraint-propagation-with-arc-consistency-checking-and-then-backtracking-with-minimum-remaning-value-heuristic-and-forward-checking/

https://www.cs.huji.ac.il/~ai/projects/2017/csp/Suduko_3/Sudoku%20report.pdf

https://www.youtube.com/watch?v=4cCS8rrYT14&t=383s