אבי דגמי – 039369061
עדיאל אשרוב - 066595752

עיבוד אילוצים – עבודה מס' 1

תיאור מימוש האלגוריתמים

התחלנו את פתרון התרגיל במימוש בעיות אילוצים כפי שנלמד בכיתה. ראשית, הגדרנו את המחלקה $Problem$ אשר ממדלת לנו בעיית אילוצים. המחלקה בעלת השדות הבאים:

N - המסמל את מספר המשתנים בבעיה.

$\boldsymbol{d}$ - המסמל את גדול דומיין הערכים של כל משתנה (ההנחה כי גודל הדומיין זהה לכל משתנה).

$\boldsymbol{v}$_– ווקטור המשמש להשמות למשתנים (בדומה ל-$values$).

$\boldsymbol{domain}$_ – ווקטור דו מימדי המייצג לכל משתנה i את הדומיין ההתחלתי שלו.

$\boldsymbol{constraints}$_ – מטריצה במימד $n * n$ אשר בכל תא שלה יש $hashmap < pair < int, int >, boolean >$ המבטאת את האילוצים בין משתנים ($C_{i,j}$). עבור זוג ערכים מהדומיין של i, j ישנו $true$ אם ערכים אלו מאולצים (הכוונה ששני הערכים האלה מופיעים באילוץ,דהיינו חוקיים) , ו-$false$ אחרת.

$\boldsymbol{assignments}$_ – שדה בתוך הבעיה אשר סופר את מספר ההשמות אותן ביצענו בפתרון הבעיה.

$\boldsymbol{ccS}$_ – מונה הסופר את מספר הבדיקות אותן ביצענו במהלך הרצת האלגו'.

$\boldsymbol{random}$_ – משתנה השומר את ה $seed$ הרנדומלי אותו אנו שולחים לבעיה, כדי ליצור את אותה בעיה שוב ושוב.

$\boldsymbol{solved}$_ - משתנה בוליאני המסמל האם נמצא פתרון לבעיה או לא.

$\boldsymbol{p1}$_ - כפי שהוגדר בתרגיל, המשתנה הנשמר הוא השכיחות/הסתברות של אילוץ בין משתנה i למשתנה j.

$\boldsymbol{p2}$_ – כפי שהוגדר בתרגיל, המשתנה הנשמר הוא השכיחות של אילוץ בין ערך של משתנה i לערך של משתנה j.

מימשנו את $initConstraints()$ המאתחלת את האילוצים על פי ההסתברויות המתאימות ואת check(int var1, int val1, int var2, int val2) על פי ההגדרה בתרגיל.

לאחר שמידלנו את הבעיות יצרנו את בעיית ה- $N\ queens$ כתת-מחלקה של $Problem$. השוני היחיד בה הוא שפונקציית אתחול האילוצים מאתחלת אותם על פי אילוצי בעיית N המלכות ולא לפי ההסתברויות הנתונות. השלב הבא היה לכתוב את מימושי האלגוריתמים לפתרון בעיות אילוצים.

לאחר קריאת המאמר של פרוסר החלטנו למדל קודם כל את החלקים המשותפים לכלל האלג' המתוארים במאמר, זאת מכיוון שבין האלגו' משתנות $label/unlabel$ ונוספים מבני נתונים, אך גרעין האלגו' לא משתנה. הגדרנו את המחלקה האבסטרקטית CSPAlgorithm – אשר מהווה אב קדמון לכלל האלג'. לה השדות:

problem_ - מופע של הבעיה אותו אנחנו רוצים לפתור.

consistent_ – המשתנה הבוליאני אותו אנחנו משנים תוך כדי ריצת האלגו',הבודק אם ההשמה קונסיסטינטית.

status_ – המשתנה המסמל את שלבי פתרון הבעיה(יש פתרון/אין פתרון/לא ידוע).

currentDomain_- ווקטור דו מימדי המחזיק לכל משתנה i את הדומיין הנוכחי שלו.

הגדרנו את המתודות האבסטרקטיות: label(int i) ו- unlabel(int i) ואת המתודה הלא אבסטרקטית solve() אשר משתמשת במתודות לעיל (מקבעת את הגוף הכללי של אלגו' חיפוש העצים של פרוסר, פרוצדורת ה- $bcssp$).

אבי דגמי – 039369061
עדיאל אשרוב - 066595752

השלב הבא היה מימוש BT, CBJ - מחלקות אשר יורשות מ - CSPAlgorithm.

BT הינה מחלקה היורשת מCSPAlgorithm ,ופונקציות ה $label/unlabel$ שלה הן הבסיסיות ביותר על פי המאמר.

אחרי שהיה לנו אלגוריתם בדוק ופותר בעיות אילוצים , המשכנו למימוש CBJ. המחלקה הנ"ל יורשת מ-BT.

ובה מבני הנתונים הבאים:

_confSets - שהוא ווקטור של $sortedSet$ (לנוחיות שלנו לקחנו קבוצה ממוינת כדי לשלוף את המקסימום בקלות מהמבנה נתונים). לכל i הווקטור מכיל את ה $Conflict\ set$ של המשתנה ה-i .

מימשנו את התוספות הנדרשות לBT, מכיוון שכעת צריך לעדכן את מבני הנתונים של ה $Conflict\ set$ ולעדכן את הצורה בה אנחנו שולפים את המשתנה אליו אנו קופצים($getHFromI$,$labelAddition, UnlabelAddition$).

לאחר מכן יצרנו את המחלקה FCCBJAlgorithm אשר מממשת את האלגו' (יורשת מ CBJ).

נוספו המבני נתונים:

_reductions – כפי שהוגדר במאמר לכל משתנה i מחסנית של קבוצות של ערכים אשר הורדו מהדומיין הנוכחי שלו.

_pastFc – ווקטור של מחסניות של *integer*. לכל משתנה i ,שמורים המשתנים אשר הורידו ממנו ערכים ב$checkForward$ במחסנית.

_futureFc - ווקטור של מחסניות של *integer*. לכל משתנה i ,שמורים המשתנים אשר הוריד מהם ערכים ב$checkForward$ במחסנית.

הוספנו את הפונקציה checkForward(int i, int j).

ואת כל הפונקציות הדרושות לטיפול עם המבני נתונים ($undoReductions, updateCurrentDomain$).

לאחר מכן, יצרנו את המחלקה $FCCBJDACAlgorithm$ אשר יורשת מהקודמת ובנוסף אליה מבצעת dac תוך פעולת ה-$label$ שלה. שוב דרסנו את המתודות הדרושות כדי להרחיב את הפעילות של האלגו'. הגענו למסקנה כי אין צורך להשתמש במבני נתונים נוספים. החלטנו כי אנו מטפלים בערכים שהורדו עם Dac באותו אופן בו אנו מטפלים בערכים שהורדו ב $forward\ checking$. התבוננו בהגדרות של המבני נתונים וראינו כי שימוש במבני הנתונים הנוכחים יכולים לתת לנו את התפקוד הרצוי עבורנו בשביל שחזור הבעיה לאחר ביצוע Dac לא מוצלח, ולכן אין צורך להוסיף עוד. טיפלנו בביטול השמה עם המתודה undoAssignment.

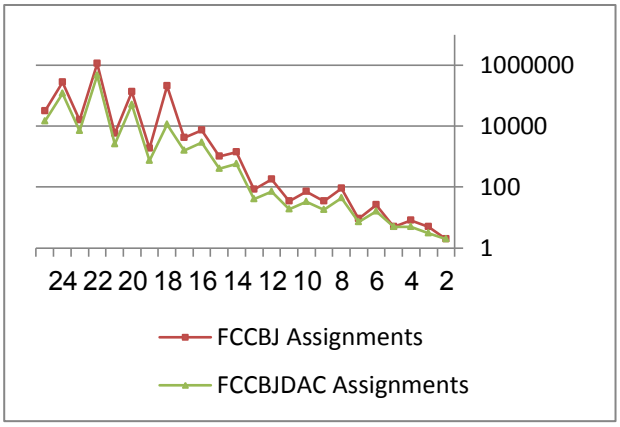
לצורך שמירת הסטטיסטיקות יצרנו את המחלקה ProblemsSetStats אשר בה אנו משתמשים כדי ליצור ממוצעים של מספרי השמות/בדיקות בשביל 50 בעיות. עבור $FCCBJDACAlgorithm$/FCCBJAlgorithm .

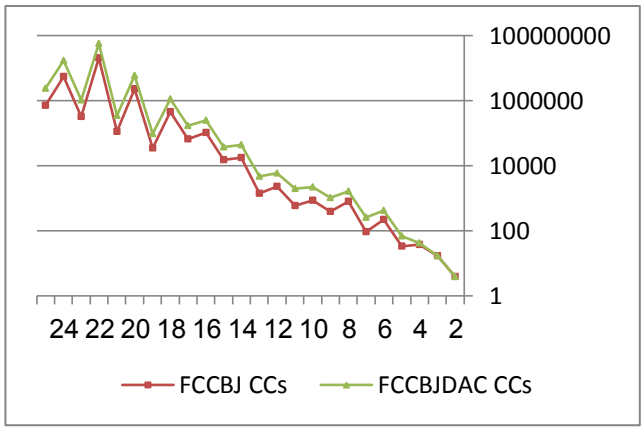
לבסוף יצרנו את המחלקה $main$ אשר יוצרת הבעיות בצורה רנדומלית (עבור אותו n נקבל את אותה בעיה). היא מריצה את האלגו' המתאימים על פי הדרישות (50 בעיות לכל $p1, p2$ שהוגדרו בתרגיל = סה"כ 1050 בעיות). וממלאת את הסטטיסטיקות.

אבי דגמי – 039369061
עדיאל אשרוב - 066595752

תיאור הניסויים

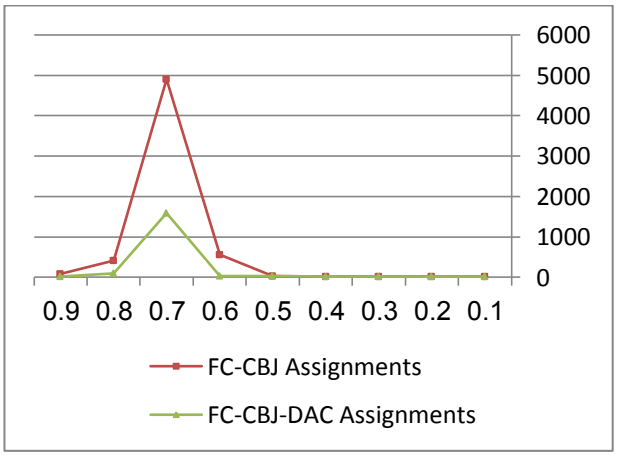
בשלב הראשון, ייצגנו את בעיית N-המלכות כבעיית אילוצים והרצנו עליהם את ארבעת האלגוריתמים שמימשנו כדי לבדוק את נכונות האלגוריתמים. חישבנו את מס' ההשמות ומס' הבדיקות של כל אחד מהאלגוריתמים על הבעיה הנ"ל עבור $N \in [2, 25]$. התוצאות שהתקבלו (מס' ההשמות ומס' הבדיקות כתלות ב- N):

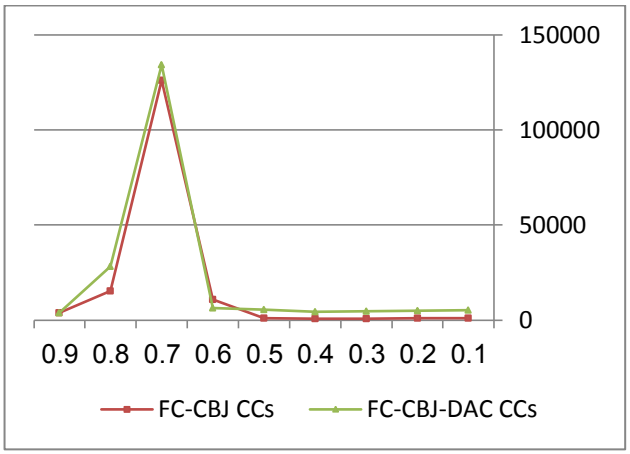




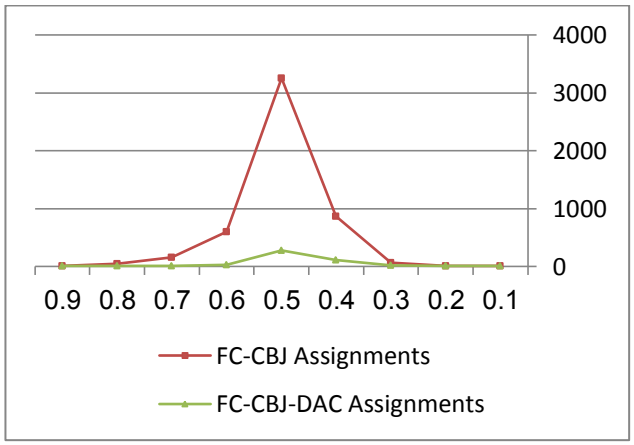
בשלב השני, עבור כל $P1 \in \{0.2, 0.5, 0.8\}$ וכל $P2 \in [0.1, 0.7]$ יצרנו 50 בעיות והרצנו עליהם את FC-CBJ ו- FC-CBJ-DAC. חישבנו את ממוצע ההשמות וממוצע הבדיקות של כל אחד מהאלגוריתמים על כל קבוצה כזו של 50 בעיות. להלן התוצאות (מס' ההשמות ומס' הבדיקות כתלות ב- P2):

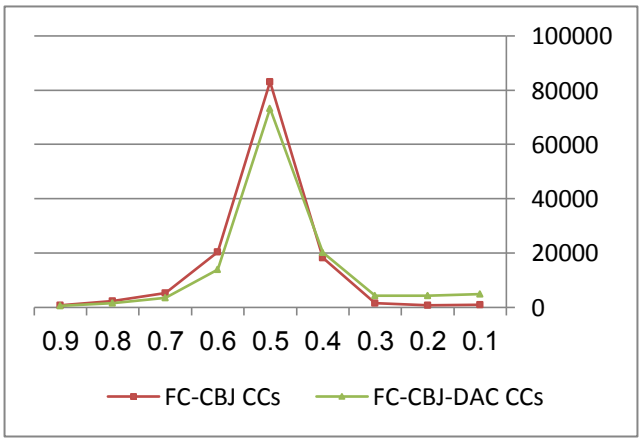
$P1 = 0.2$





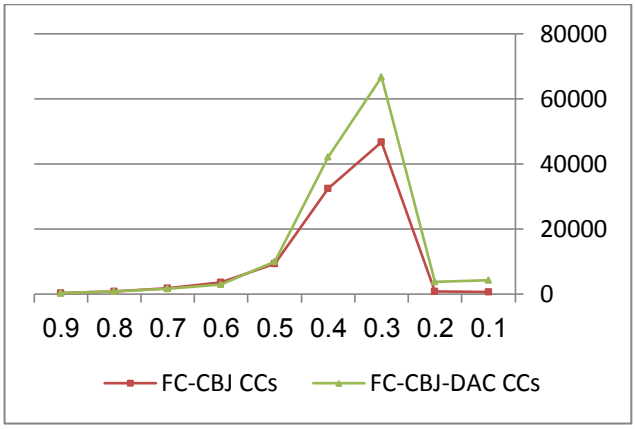
$P1 = 0.5$

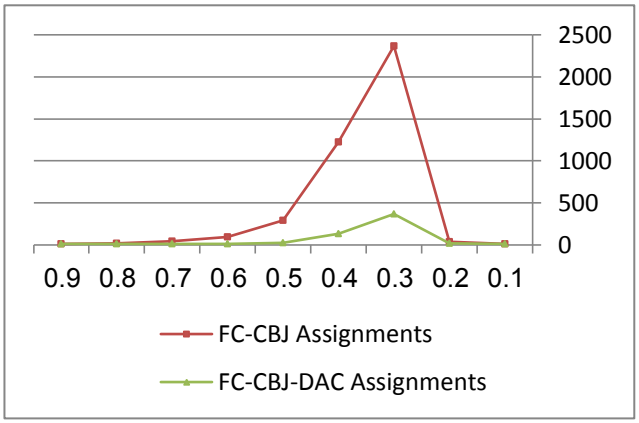




אבי דגמי – 039369061
עדיאל אשרוב - 066595752

$P1 = 0.8$

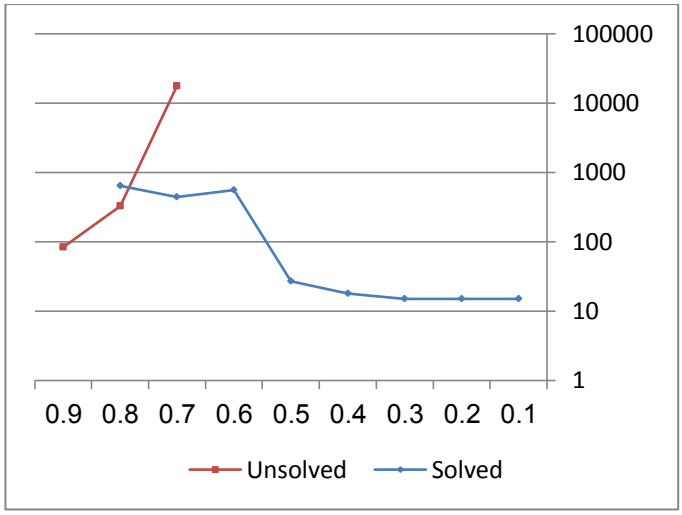




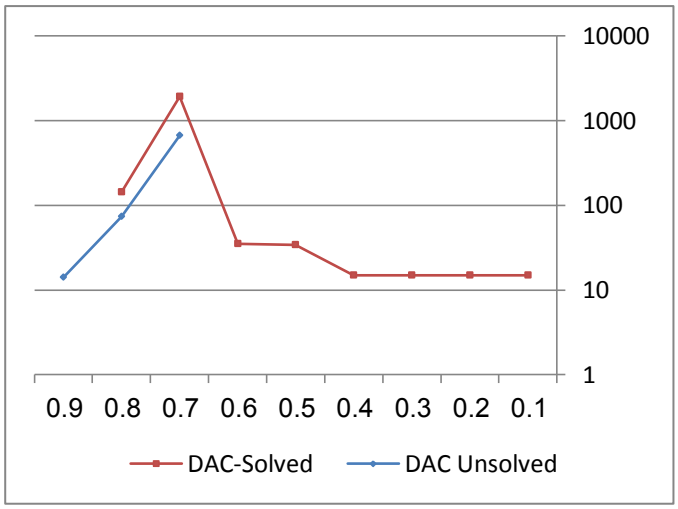
בנוסף, על מנת להשתכנע שהאלגוריתמים עובדים בצורה תקינה, בדקנו שכל בעיה נפתרת ע"י כל ארבעת האלגוריתמים או לא נפתרת ע"י כולם (על חלק מהבעיות לא הרצנו את אלגוריתם BT מכיוון שזמן הריצה של האלגוריתם עליהן היה ארוך מדי ולא ראינו בכך תועלת).

בדיקה נוספת שביצענו הייתה להפריד בין בעיות שנפתרו ע"י האלגוריתמים ובין בעיות שלא נפתרו על ידם. התוצאות שהתקבלו עבור למשל $p1 = 0.2$ מראות על חלוקה ברורה לתחומים, כך שעבור ערכים נמוכים של $p2$ יש פתרונות לבעיות, עבור ערכים גבוהים של $p2$ אין פתרונות לבעיות וישנו תחום ערכים ($P2 \in [0.7, 0.8]$) שבו ישנה חפיפה, כלומר, ישנן בעיות שיש להן פתרון וכאלו שאין להן (מס' ההשמות של כל אחד מהאלגוריתמים כתלות ב- $p2$):

FC-CBJ



FC-CBJ-DAC



אבי דגמי – 039369061
עדיאל אשרוב - 066595752

מסקנות:

כפי שנראה במאמר של פרוסר על אלגו' היברדיים לפתרון בעיות אילוצים ("Hybrid algorithms for the constraint satisfaction problem"), האלגו' $FCCBJ$ הינו היעיל ביותר לפתרון בעיית אילוצים על פי המדדים של מספר הבדיקות הממוצע ומספר ההשמות הממוצע אשר האלגו' מבצע. השאלה אותה באנו לבדוק האם שילוב של DAC לאלגו' ישפרו את ביצועיו. נזכיר כי DAC הינו אכיפה של קונסטנטיות מכוונת על פי טאסנג, אותה אנו מבצעים על הבעיה המושרית לפנינו בכל איטרציה.

באופן תיאורטי אנו מצפים כי האלגו' המשולב יבצע יותר בדיקות ופחות השמות מאשר ה $FCCBJ$. זאת מכיוון שבכל השמה אותה אנו בודקים, DAC מבצע עוד בדיקות קונסיסטנטיות אל מול שאר המשתנים אשר אין להם השמה בבעיה. לאחר מכן הוא מקצץ בתחומי המשתנים אשר אינם קונסיסטנטיים ולכן נצפה גם לקבל פחות השמות בהמשך הטיול על עץ החיפוש. למעשה אנו גוזמים תתי עצים עתידיים עליהם נטייל בהמשך תהליך החיפוש אחר פיתרון. ניתן לראות שזוהי התוצאה שהגענו אליה בניסוי זה.

הבעיות אותן באנו לבדוק נוצרות באותה צורה בה נוצרות הבעיות במאמר *"An empirical study of phase transitions in binary constraint satisfaction problems"* של פרוסר. מדובר על n, k אשר מסמלים את מספר המשתנים וגודל הדומיין שלהם בהתאמה. ובנוסף הבעיות נוצרו עם שימוש בעוד שני פרמטרים,$p1$ המסמל את צפיפות המשתנים, ו- $p2$ המסמל את שכיחות ($tightness$) המשתנה אל מול משתנה אחר.

כפי שנראה במאמר ישנה תופעה מקשרת בין הערכים הנ"ל הנקראת מעבר פאזה. עבור ערך $p1$ מסוים קיים ערך קריטי $p2$ אשר עד אליו הבעיות "קלות" ויש להן פתרון. בסביבתו הבעיות הן קשות ולא ברור אם יש פתרון. ולערכים גדולים ממנו בעיות חוזרות להיות קלות כתוצאה מכך שאין פתרון לבעיה. בניסויים שערכנו עבור הערך $p1 = 0.2$ גילינו כי הנקודה הקריטית היא $P2 \in [0.7, 0.8]$, וככל ש $p1$ גדל מיקומה של הנקודה הקריטית קטן, מכיוון שכעת דרושים פחות אילוצים בין הערכים כדי להפוך את הבעיה לקשה יותר. לדוג' עבור $p1 = 0.2$ אנו רואים את הנקודה הקריטית בסביבות $P2 \in [0.25, 0.35]$. נסביר כי כאשר אנחנו מדברים על בעיות "קשות" הכוונה היא לבעיות עליהן האלג' מבצעים הרבה יותר השמות ובדיקות כדי להגיע למסקנה שיש או אין להן פתרון. תופעה זו של מעבר פאזה נראית בבירור בגרפים שקיבלנו, כמו כן ניתן לראות בגרפים את הנדידה של הנקודה הקריטית מימין לשמאל כתלות ב- $p1$ ואת העובדה שלבעיות עם $p2$ קטן מה- $p2$ הקריטי יש פתרון ולכאלו עם $p2$ גדול מה- $p2$ הקריטי אין פתרון. כמו כן, התופעה מתקבלת בשני האלגוריתמים.

לסיכום, התוצאות אותן קיבלנו דומות לתוצאות אותן קיבל פרוסר בניסויים שלו כפי שנראה במאמריו.

שאלה מעניינת בה לא עסקנו היא מהי העלות של שחזור הבעיה בזמן ריצה בשתי הבעיות ($Undo\ Reductions\ etc'$). מכיוון שהשתמשנו באותו מנגנון לניהול ההשמות, ניתן לבדוק מי מבצע יותר שחזורים של הבעיה. להערכתנו שני האלגו' מבצעים מספר זהה של שחזור הבעיה לאחר כישלון בהשמה למשתנה או כישלון ב- DAC . זאת מכיוון שבסופו של דבר אם ל- DAC נגמר הדומיין עבור משתנה i, גם ל- $FCCBJ$ יגמר עבורו הדומיין והוא יצטרך לקפוץ אחורה למשתנה h (עבור השמה שאינה עובדת). ההבדל הוא שאנו מגלים שההשמה למשתנה h מרוקנת את הדומיין של i כבר אחרי הרצת ה-DAC ב- $label$ של h.

שאלה מעניינת בה כן עסקנו היא האם יש שוני בין ריצת האלגו על מקרה פרטי של בעית אילוצים. הבעייה אותה פתרנו היא ה- $N\ Queens$ הזכורה לטובה. התוצאות אותן קיבלנו מראות כי לכל N אלג' DAC מבצע פחות השמות אך מבצע יותר בדיקות. ההבדל איננו משמעותי וחד כפי שאנחנו רואים בבעיות הרנדומליות(למרות שעשרה אחוזים של שיפור הם שיפור משמעותי בAI).אנו מעריכים כי זה כך מכיוון שבעיית המלכות היא בעיה סימטרית.