הפקולטה למדעי ההנדסה
המחלקה להנדסת מערכות תוכנה ומידע
Faculty of Engineering Sciences
Department of Software and Information Systems Engineering

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

Bounded Suboptimal N-Player Game Tree Search

2021

חיפוש תת-אופטימלי חסום בעץ משחק מרובה משתתפים

2021

מנחה:

דור עצמון

צוות הפרויקט:

שחר מרץ 208240275

פלג ביטון 203842703

אסף זקס 302329693

עומר נגר 307937714

תקציר הפרוייקט

מערכות משחק בהן המחשב מתמודד כנגד בני אדם ואף מביס אלופי עולם קיימות שנים רבות. העיקרון שעומד בבסיס מערכות אלו מבוסס עצי החלטה – המערכת ממפה את מצבי המשחק בצורה של עץ, כל צומת מייצג את מצב המשחק בתורו של אחד המשתתפים, והקשתות מייצגות מהלכים המובילים למצבים חדשים. המערכת תבצע חיפוש בין קודקודי העץ במטרה למצוא את המצב האידאלי ובהתאם תבצע את המהלך הנדרש.

מערכות המבוססות על אלגוריתמים נאיבים המנסים לחקור את כל המצבים לא יעמדו במשימה בזמן פיזיבילי, על כן תוטל מגבלה על מספר הקודקודים אותם מערכת תחקור, וערכו של מצב יקבע באמצעות חישוב יוריסטי שכן לא נגיע למצב סופי. איכות החישוב עולה ככל שהמצב שנעריך רחוק יותר מהמצב ההתחלתי, על כן פותחו טכניקות לגיזום חלקים מן העץ בצורה שלא פוגעת באופטימליות החיפוש, במטרה להעדיף קודקודים עמוקים יותר.

על בסיס טכניקות הגיזום האופטימלי, בפרויקט הגדרנו לראשונה תנאים לביצוע חיפוש תת-אופטימלי חסום בעצי משחקים מרובי משתתפים. בכך שאכפנו תנאים אלו על אלגוריתמים אופטימלים מוכרים, יצרנו שתי וריאציות בעלות פוטנציאל לבצע גיזומים רבים יותר, מכיוון שאינן מחפשות אחר הערך האופטימלי אלא אחר ערך הקטן ממנו עד כדי קבוע מוגדר. המטרה היא להגיע לקודקודים רחוקים יותר ולקבל באמצעות החישוב היוריסטי החלטה איכותית יותר – על אף המחיר שהסכמנו לשלם.

בניסויים שביצענו באמצעות מחולל עצים רנדומלים ובאמצעות סימולטור משחק רוליט לארבעה שחקנים, ניכר כי האלגוריתמים התת-אופטימלים חוקרים מצבים מתקדמים יותר ומשיגים תוצאות טובות יותר בהשוואה לאלגוריתמים האופטימלים מהם פותחו. יתכן ושימוש בתנאים אלו על אלגוריתמים שונים יציגו תוצאות טובות יותר, אך גם לתוצאות שאנו מציגים יכולת להשפיע על זמן ואיכות התגובה של מערכות משחק עתידיות.

Project Summary

For many years game systems were designed to beat world champions. The concept behind these systems is based on decision trees - the system maps the game states as a search tree, where each node represents the game state and edges represent actions between states.

Systems based on naive algorithms that investigate all existing nodes cannot complete the task in a feasible time. Therefore, such systems limit the number of investigated nodes and determine the state value by heuristic calculation. Pruning techniques have been developed to prune nodes in a way that does not affect the search optimality because the calculation quality increases as the nodes we evaluate are farther from the initial node.

In this project, we defined conditions for performing a bounded suboptimal search in multiplayer game trees based on optimal pruning techniques. By setting these conditions on existing optimal algorithms, we have created two algorithms that have the potential to prune more since they settle for a suboptimal solution (within the range of a constant). The aim is to reach deeper nodes and make a better decision, despite the cost we agreed to pay.

In the experiments we performed using a random tree generator and a four-player Rolit game simulator, the suboptimal algorithms investigate deeper nodes and achieve better results than the optimal algorithms.

תוכן העניינים

מבוא

עקרון בסיס עליו מבוססים אלגוריתמי חיפוש בעצי משחק הוא שבכל תור השחקן יבצע את המהלך בעל התועלת הגדולה ביותר עבורו, תוך הנחה כי השחקן השני יעשה בדיוק את אותו הדבר בתורו. מעיקרון זה פותחו אלגוריתמים רבים, אשר לחלקם הנחות שונות לגבי אופי המשחק. עניין מחקרי רב הוקדש לאלגוריתם המינימקס - אלגוריתם נאיבי למשחקים לשני משתתפים המבוסס חיפוש לעומק, פועל על פי עיקרון זה ומוצא את ערך המינימקס של עץ המשחק (נקרא גם ערך שיווי המשקל של העץ ומבטא את התועלת הסופית עבור השחקן שבשורש העץ).

בעוד האלגוריתם מבטיח פתרון אופטימלי שכן הוא חוקר את כל קודקודי העץ, פותחו בעקבותיו אלגוריתמים נוספים, המוכר מבניהם הוא אלגוריתם אלפא בטא – αβ, שעל ידי הגדרת תנאים, מאפשר 'גיזום' של תתי-עצים המייצגים מצבים אפשריים אליהם המשחק יכול להתפתח, שכן חקר המצבים באותם תתי-עצים אינו משנה את תוצאת החיפוש. בהמשך פותחו תנאים שהיוו בסיס לפיתוח אלגוריתמי חיפוש אופטימלים למשחקים מרובי משתתפים, גם כאלו המבצעים גיזומים, וזאת למרות החוקיות הסבוכה יותר של משחקים מסוג זה.

הצורך בגיזום תתי עצים נובע מכך שעצי משחק הם גדולים מאוד – כתלות במספר המהלכים עד לסיום המשחק ובמספר הפעולות האפשריות ששחקן יכול לבצע בתורו. על כן, ביצוע חיפוש נאיבי על כלל הקודקודים במטרה להחזיר את התוצאה האופטימלית אינו תמיד פיזיבילי. ואמנם, הגיזום אינו מספיק ואין זה פיזיבילי לחקור גם את מה שנשאר מהעץ. על כן, בספרות מתועדות שיטות שונות שמטרתן להגדיר לאלגוריתם חיפוש מתי לעצור את פעולתו, ושיטות חישוב היוריסטיות בכדי לקבוע את ערכם של הקודקודים שהאלגוריתם הספיק לבחון, על אף שאינם מייצגים מצבים סופיים. היוריסטיקות אלו נחשבות למדויקות יותר כתלות בקרבת המצב הנחקר למצב סופי, והגיזום למעשה מאפשר לנו לתעדף את הקודקודים אותם נחקור במסגרת החיפוש, לפתח קודקודים הקרובים יותר למצבים סופיים, להשיג היוריסטיקות מדוייקות יותר בסיום החיפוש, ובכך לקבל החלטה איכותית יותר.

המחקר שלנו עוסק בחיפוש תת-אופטימלי חסום בעץ משחק מרובה משתתפים. נרצה להגדיר את התנאים הדרושים על מנת שאלגוריתם חיפוש לעץ משחק מרובה משתתפים יחזיר פתרון תת-אופטימלי חסום - כלומר שונה עד כדי קבוע מהערך האופטימלי. אנו מאמינים שעל ידי התפשרות על הערך האופטימלי יוכלו אלגוריתמים אלו לבצע גיזומים רבים יותר, כתוצאה מכך לחקור מצבים מתקדמים יותר, לקבל החלטות איכותיות יותר ואף לנצח משחקים רבים יותר.

למחקר ערך בתחומים שונים, בניהם תורת המשחקים, קבלת החלטות, סטטיסטיקה, פילוסופיה, כלכלה, רובוטיקה ואבטחה. אף על פי כן, המחקר מהווה המשך ישיר של מחקר של Atzmon (2018) החוקר את אותה הבעיה עבור משחקים לשני שחקנים, שבו מצוטט רק מאמר אחד מחמש השנים האחרונות, כלומר מדובר בנושא שלא נחקר בעבר ולא בוצעו מחקרים בכיוון הזה. מכאן שלתנאים שנגדיר פוטנציאל לחקר קבוצה חדשה של אלגוריתמי חיפוש, ואכן במחקר נציג שני שני אלגוריתמים ראשונים בתחום, המבוססים על האלגוריתמים האופטימליים ShallowPrunningMaxN ו- Paranoid .

בסקירת הרקע נציג רקע על תחום החיפוש בעצי משחק, נבחן עקרונות בסיס המתקיימים בחיפושים אלו ובפתרונות שונים לסיבוכיות הגבוהה של אלגוריתמי חיפוש נאיבים בעצי משחק, בדגש על גיזום קודקודים.

בסקירת ספרות רלוונטית נציג מגוון פתרונות אלגוריתמיים אופטמליים לחיפוש בעצי משחק לשניים או יותר משתתפים ופתרון תת-אופטימלי חסום לשני משתתפים, וזאת במטרה להכיר את התנאים החלים בפתרונות אלו ועל בסיסם להגדיר חדשים למטרותינו.

בהמשך המסמך, נגדיר את מטרות המחקר, נפרט את מתודולוגיית המחקר, ונסקור את הניסויים שביצענו במטרה לבחון את יכולות האלגוריתמים המוצעים. לבסוף נציג את תוצאות הניסויים, נסיק מסקנות ונסכם באתגרים וכיוונים למחקרי המשך בתחום.

סקירת רקע

לדברי Luckhardt and Irani (1986), חיפוש בעצי משחק הוא אחד מהתחומים הראשונים שנחקרו בעולם הבינה המלאכותית. נהוג למדל משחק כסט סופי של מצבים המכיל בתוכו תת-סט של מצבים סופיים בעלי ערך תועלת הנקבע על ידי פונקציית הערכה (מהם לא ניתן להתקדם ולמעשה בהם המשחק מסתיים), וסט נוסף של פעולות למעבר ממצב למצב.

לטענתם, רוב המחקר עוסק במשחקים בהם מספר הפעולות השונות שניתן לבצע בכל מצב הוא סופי, וכך גם מספר הפעולות עד להגעה למצב סופי מכל מצב. הבעיה מאופיינת בצורה של עץ, כאשר כל קודקוד בעץ מתאר את מצב המשחק בתורו של משתתף מסויים, ממנו יוצאות קשתות המייצגות פעולות חוקיות שיכול לבצע המשתתף ויובילו לשינוי המצב במשחק. כל אחד מקודקודי הבן מייצגים מצבים חוקיים, הנגזרים מביצוע הפעולה שבקשת על המצב שבקודקוד האב. במעבר בין קודקוד אב לבן יתקדם התור במשחק לשחקן הבא בסדר.

על מנת להבין את הבסיס למחקר בתחום נסתכל קודם על משחקים סופיים לשני שחקנים עם סכום נקודות קבוע, בדגש על משחקי סכום אפס - משחקים בהם בכל מהלך, הרווח (או ההפסד) של שחקן אחד שווה להפסד (או לרווח) של השחקן השני. היכרות עם עקרונות אלו תסייע לנו בבואנו לעסוק בהמשך במשחקים מרובי משתתפים.

עקרון המינימקס שהציג Wald (1945) הוא עיקרון בסיס בכל הנוגע לקבלת החלטות במשחקי סכום אפס. על פי עקרון זה, בכל תור השחקן יבצע את המהלך בעל התועלת הגדולה ביותר עבורו, תוך הנחה כי השחקן השני יעשה בדיוק את אותו הדבר בתורו. אלגוריתם המינימקס הוא אלגוריתם נאיבי מבוסס חיפוש לעומק, הפועל על פי עיקרון זה ומוצא את ערך המינימקס של עץ משחק (נקרא גם ערך שיווי המשקל של העץ), ומבטא את התועלת הסופית עבור השחקן שבשורש העץ. תוצאת החיפוש הינה האסטרטגיה האופטימלית עבור שחקן השורש למיקסום הרווח שלו. האלגוריתם מבטיח פתרון אופטימלי שכן הוא חוקר את כל קודקודי העץ.

על פי Knuth and Moore (1975), במקרים רבים עצי משחק הם גדולים מאוד – כתלות במספר המהלכים עד לסיום המשחק ובמספר הפעולות האפשריות ששחקן יכול לבצע בתורו. על כן, ביצוע חיפוש נאיבי על כלל הקודקודים במטרה להחזיר את ערך המינימקס אינו תמיד פיזיבילי. לאורך השנים פותחו טכניקות שונות במטרה להתגבר על קושי זה, בניהן טכניקת הαβ, שמטרתה להאיץ את תהליך החיפוש ללא איבוד של מידע. במילים אחרות, הערך שיוחזר ע"י שימוש בטכניקה זו זהה לערך שיחזור כתוצאה משימוש באלגוריתם המינימקס. בפועל, אלגוריתמים מבוססי αβ חוקרים את העץ באופן דומה לאלגוריתם המינימקס ומבצעים גיזום לקודקודים בעץ כאשר ניתן לדעת באופן ודאי שאין צורך בפיתוחם, שכן קיימת פעולה חוקית אחרת אשר תוביל את השחקן שבשורש העץ למצב עדיף מבחינתו. ע"י גיזום קודקוד מסויים האלגוריתם למעשה מדלג ואינו חוקר את תת-העץ שקודקוד זה מהווה השורש שלו, ומכך מתקיים חיפוש מהיר יותר מבלי לפגוע באופטימליות הפתרון.

גם לאחר החלת חוקי הגזירה הללו, טכניקה זו או אחרות הדומות לה אינן מבצעות מספיק גיזומים על מנת להקל מספיק על מציאת הפתרון. אחת הדרכים להתגבר על בעיה זו היא להקל על הפתרון האופטימלי ולאפשר פתרונות תת-אופטימליים. Atzmon (2018) מציע ליצור טרייד-אוף בין זמן הריצה לאופטימליות הפתרון על ידי אפשור של גיזום נרחב יותר מזה המתאפשר בחיפוש אחר פתרון אופטמלי. מחקרו מגדיר תנאים לתכנון אלגוריתמים תת-אופטימליים חסומים, אשר יקבלו מהמשתמש ערך ϵ ויחזירו פתרון תת-אופטימלי לעץ משחק שערכו רחוק בעד ϵ מערך המינימקס האמיתי של העץ. בפועל, המאמר מציע שיטה להרחבת חוקי גזירה של אלגוריתמים מוכרים כך שיחזירו פתרונות תת-אופטימליים מסוג זה. על פי עקרונות אלו ככל שערכו של ϵ יהיה גדול יותר תתאפשר גזירה נרחבת יותר בעץ.

כך, בעזרת אלגוריתמי משחק תת-אופטימליים חסומים, ניתן לבצע גיזומים רבים יותר מאלגורתמים אופטימליים וכפועל יוצא להשיג אסטרטגית משחק בצורה מהירה יותר. אולם, דבר זה עלול לפגוע באיכות הפתרון החוזר מהאלגוריתם ומכאן שקיים טרייד-אוף בין איכות הפתרון לזמן חישוב הפתרון. בעוד המחקר עוסק בפתרונות תת-אופטימליים חסומים בעצי משחק לשני שחקנים, בפרוייקט נרצה להציע פתרון דומה עבור משחקים מרובי משתתפים, המבוסס על פתרונות אופטמליים מוכרים.

Luckhardt and Irani (1986) הציעו את אלגוריתם max^n, המהווה המקביל האלגוריתמי של אלגוריתם המינימקס למשחקים מרובי משתתפים. האלגוריתם מתאים למשחקים עם סכום נקודות קבוע ולא קבוע, בהם מתקיים שיתוף פעולה בין שחקנים וגם לכאלו בהם כל שחקן משחק עבור עצמו. ניתן להשתמש בmax^n גם במשחקים עם אלמנט של מזל ובכאלו ללא מידע מושלם.

בדומה לאלגוריתם המינימקס, בכל קודקוד לאורך החיפוש נניח כי השחקן שזהו תורו יבחר את הפעולה שתמקסם את הרווח עבורו. בשונה מאלגוריתם המינימקס המיועד למשחקי סכום אפס בשני שחקנים, לא נוכל לייצג את הרווח הצפוי לכלל השחקנים באמצעות מספר בודד ונעבור לייצוג ווקטורי. האלגוריתם שמציגים החוקרים בסיסי ונאיבי (שכן על אף שמאפשר "לגזום" כניסות בווקטור הרווח של קודקוד מסויים, הוא עדיין מחוייב לחקור את כל הקודקודים) ותוצאתו היא סט של אסטרטגיות עבור כל שחקן, המוכח כווקטור שיווי המשקל, שכן כל חריגה של שחקן מהאסטרטגיה שהתקבלה עבורו תוביל לתוצאה קטנה או שווה לזו שיקבל תחת האסטרטגיה האופטימלית שהחזיר האלגוריתם. האלגוריתם מהווה אבן דרך בתכנון טכניקות חיפוש לא נאיביות למציאת פתרון אופטימלי לבעיות חיפוש בעצי משחק מרובי משתתפים, בניהן טכניקות המאפשרות גיזום קודקודים שלמים.

לטובת הפרוייקט המחקרי אותו נבצע, נסקור ספרות בדבר אלגוריתמים מבוססי max^n המאפשרים גיזום קודקודים (מבלי לפגוע באופטימליות הפתרון). בנוסף, נסקור אלגוריתמים נוספים המבצעים תחת הנחות שונות רדוקציה מבעיית חיפוש למשחקים מרובי משתתפים, לעצי חיפוש בעלי מבנה זהה לאלו שנפרשים על ידי אלגוריתם המינימקס במשחקים בשני משתתפים, עליהם נוכל לבצע חיפוש בשיטת αβ. באמצעות הידע שנרכוש ננסה להגדיר תנאים באמצעותם נתכנן אלגוריתמים חדשים המבצעים חיפוש תת-אופטימלי חסום עבור משחקים מרובי משתתפים.

<u>סקירת ספרות</u>

על מנת להגדיר תנאים לביצוע גיזום תת-אופטימלי חסום בעצי משחק מרובי משתתפים, נרצה ראשית ללמוד על אסטרטיות שונות הקיימות באלגורתמים אופטימליים שונים, ללמוד על ההנחות שחייבות להתקיים במשחק על מנת להשתמש באלגוריתם ועל התנאים הנדרשים לביצוע גיזום אופטימלי של קודקדים תחת הנחות אלו במטרה להגדיר חוקים אלו מחדש עבור גיזום תת-אופטימלי חסום.

נסתכל ראשית על גיזום אופטימלי בעצים מסוג max^n. Sturtevant and Korf (2000) סקרו כיצד תחת הנחות שונות ניתן לבצע חלק מגיזומי αβ על עצים אלו. על פי ממצאי החוקרים, אם מאופי המשחק ידועים לנו סך הנקודות המקסימלי עבור כל השחקים יחד (maxsum) וערך הנקודות המקסימלי עבור שחקן בודד (maxp) נוכל לגזום.

ראשית נוכל לבצע גיזום מיידי (immediate pruning) – גיזום טריוויאלי, נגזום כאשר השחקן הנוכחי מקבל את מספר הנקודות המקסימלי האפשרי. נוכל גם לבצע גיזום רדוד (shallow pruning) – גיזום קודקוד המתאפשר בזכות מידע מקודקוד "דוד" שפותח קודם לכן. עוד קודם לכן, Korf (1991) פיתח אלגוריתם בשם Shallow (אליו נתייחס בשם ShallowPrunningMaxN) המבצע את שני סוגי גיזום אלו בחקר עצים מסוג max^n. סוג נוסף של גיזום המתקיים בטכניקת הαβ אך אינו מתאפשר בגרסה רבת המשתתפים הוא גיזום עמוק (Deep pruning) – גיזום של קודקוד כלשהו על סמך מידע מאב קדמון שלו. החוקרים מראים כי טכניקה זו אינה מתאפשרת בעצים מסוג max^n מאחר וקודקוד שנגזר על ידי גזירה זו אמנם איננו מכיל את ערך שיווי המשקל של העץ, אך גזירתו יכולה להוביל לזיהוי קודקוד שגוי כתוצאת האלגוריתם.

טכניקה נוספת שהציגו החוקרים ועושה שימוש בmaxsum וmaxp היא Depth-First Branch-and-Bound (DFBnB), המניחה כי בחוקי המשחק ניתן למצוא פונקציה יוריסטית מונוטונית לחישוב המרחק מניצחון (ניתן למצוא כזו במשחקי קלפים רבים). על פי הטכניקה נוכל לבצע גיזומים דומים לגיזום מיידי וגיזום רדוד על ידי השוואת ערך היוריסטיקה למידע שאספנו עד כה ולדלג על קודקודים שנדע שבוודאות שאין ערך בפיתוחם.

מהטכניקות הללו בונים החוקרים אלגוריתם חדש בשם Alpha-Beta Branch-and-Bound pruning (ABBnB). על פי האלגוריתם, לכל קודקוד יש את גבולות הαβ שלו, שהגיעו מגילוי קודקודים בשלב מוקדם יותר, וכן גבול שמגיע מפונקציית היוריסטיקה, ובאמצעותם ניתן לחשב את ערך השחקן בקודקוד מסוים כחיסור של הנקודות שבטוח יש לשחקנים אחרים (בחיתוך שתי הגבולות) מתוך כלל הנקודות האפשריות בסך הכל לכלל השחקנים, ולבצע יותר חיתוכים מאשר יכלו כל אחד מהאלגוריתמים בנפרד – אך גם במקרה זה לא מתאפשר גיזום עמוק.

בהשוואה שבוצעה בין האלגוריתמים על בסיס מספר הקודקודים שכל אלגוריתם מפתח כאשר הוא משחק Hearts או Sergeant Major, התוצאות מראות כי האלגוריתם המשולב (ABBnB)

אכן מפתח פחות קודקודים מכל אחת מהשיטות בנפרד (αβ וDFBnB) מאחר ומאפשר את הגזירות שהן מבצעות וגזירות נוספות המתאפשרות באמצעות המידע המשותף משתיהן.

על ידי שינוי התנאים לגיזום בטכניקה זו, או בכל אחת מטכניקות הבסיס, נוכל להגדיר מחדש תנאים לאלגוריתמים המבצעים גיזום תת-אופטימלי בעצי max^n. נסייג בכך שפתרון זה מתאים רק למשחקים בהם נדע את ערכי maxsum וmaxp, ובשימוש ב Branch-and-Bound נידרש גם למשחקים בהם מתקיימת פונקציה מונוטונית.

פתרון למגבלה זו ניתן למצוא באלגוריתם נוסף בשם Paranoid המוזכר גם הוא במאמר לטובת השוואת תוצאות האלגוריתם ABBnB במבחן מספר הקודקודים שפותחו. האלגוריתם, בדומה לאלגוריתמים אחרים לפתרון משחקים מרובי משתתפים, מציע חיפוש בדמות עץ משחק לשני שחקנים, תחת ההנחה שכלל השחקנים שאינם שחקן השורש משחקים בקואליציה נגד שחקן השורש. באמצעות הנחה זו ניתן למשקל את עלי העץ בתועלת של שחקן השורש בלבד, שכן הוא פועל למקסם אותה ומנגד הם פועלים למזער אותה מבלי להתעניין בהישגייהם האישיים, ונקבל מבנה של משחק סכום אפס. נייצג את המצב לאחר כל פעולות הנגד שמבצעים השחקנים בתורם תחת קודקוד אחד ומכאן שנקבל עץ משחק הזהה לבעיית המינימקס. בהשוואה שביצעו החוקרים, אלגוריתם הParanoid פיתח משמעותית פחות קודקודים מאלגוריתמי הmax^n שכן האלגוריתם פורס עצים קטנים משמעותית בעקבות הרדוקציה ולמעשה מאפשר תכנון ארוך טווח שכן הוא חוקר בעומק קבוע צעדים רבים קדימה במשחק ביחס לשחקן השורש. החוקרים מציינים כי ההנחה שהאלגוריתם מבצע מובילה למשחק תת-אופטימלי שכן היא אינה ריאלית ותוביל את שחקן השורש להעדיף מצבים בהם יפגע בצורה מינימאלית במידה ואכן קיימת קואליציה, על פני מצבים אופטימליים.

Schadd and Winands (2011) התייחסו גם הם לבעייתיות שבאלגוריתם הParanoid והציגו במאמר אלגוריתם בשם Best-Reply Search (BRS) שעושה רדוקציה לבעיה בשני שחקנים תחת הנחה אחרת. על פי האלגוריתם, בכל סיבוב רק שחקן אחד מנסה למזער את שחקן השורש, ורק הוא ישחק לאחר שחקן השורש. בפועל בכל תור שני שחקן השורש משחק ומתקבל עץ משחק זהה במבנהו לעץ משחק לשני שחקנים וניתן לחפש בצורה דומה אחרי ערך הMinmax. אחר כל תור של שחקן השורש יחקרו כל אפשרויות המשחק של כל היריבים של השורש והאלגוריתם יבחר מבין המהלכים של כל השחקנים את המהלך שימזער את הניקוד של שחקן השורש. גם כאן, בכל חישוב בעל עומק קבוע מפותחים יותר מהלכים של שחקן השורש ביחס לעצי max^n, ובכך מתקיים תכנון ארוך טווח. מנגד, במשחקים רבים דילוג על תורם של שחקנים מסויימים (למשל במשחקים עם מספר שחקנים קבוע או trick-based) עלול להוביל למצבי משחק לא חוקיים. כמו כן תורות של שחקנים שיתרמו לשחקן השורש לא ילקחו בחשבון.

החוקרים השוו את יכולות BRS אל מול Paranoid וmax^n. האלגוריתמים ששיחקו זה מול זה את המשחקים שחמט יפני, פוקוס ורוליט והחוקרים מדדו את אחוזי הניצחון של כל אלגוריתם. על פי התוצאות BRS ופרנואיד מציגים תוצאות טובות מmax^n ומגיעים לעומקים דומים. BRS מנצח את Paranoid בשחמט, אך בפוקוס ורוליט הם כמעט שווים. במצב בו שלושת האלגוריתמים שיחקו יחד max^n היה החלש ביותר וBRS היה החזק ביותר. יכולתיו של BRS מובהקות יותר כאשר ניתן לו זמן חישוב ארוך יותר. במסקנותינם החוקרים קבעו כי BRS הוא האלגוריתם המיטבי וכוחו בכך שהוא מחפש את היריב עם הצעד החזק ביותר ובכך מתקבל אלגוריתם זהיר הבוחן את כל היריבים. בכך שלא מאפשר לכל השחקנים לשחק, האלגוריתם חוקר יותר קודקודים של שחקן השורש בעומק קצוב ובכך מתאפשר תכנון ארוך טווח.

בעוד במחקר זה עוסקים החוקרים בפתרון אופטימלי, בפרוייקט נרצה להראות פתרון תת-אופטימלי חסום מבוסס פתרון אופטימלי. אופציה אחת תהיה באמצעות BRS – לאחר רדוקציה לעץ משחק לשני שחקנים נוכל לבחון זאת באמצעות האלגוריתם התת-אופטימלי החסום שהציג Atzmon (2018). במאמר מוצג אלגוריתם חיפוש חדש לעצי משחק בשם Bounded Alpha-Beta (BAB).

אלגוריתם זה מבוסס על אלגוריתם αβ ועל אלגוריתם נוסף בשם Minimax-* בו נעשה שימוש בשני ערכים נוספים L וU המשמשים לגיזום במשחקים עם אלמנט של מזל. בדומה לאלגוריתמים אלו, BAB מבצע חיפוש לעומק אך מגדיר סט חוקים חדש להגדרת ערכי המשתנים. בדומה לαβ – גיזום קודקוד n בעץ יתרחש כאשר יתקיים β (n) ≤ α(n) + ϵ. פלט האלגוריתם עבור קודקוד השורש n_1 הינו טווח $[L(n_1), U(n_1)]$ בגודל שאינו עולה על ϵ, וכל פתרון בטווח הינו פתרון תת-אופטימלי (כזה שאינו רחוק מערך המינימקס ביותר מ ϵ).

על מנת לבחון את ייעילות האלגוריתם החדש (BAB), החוקרים ביצעו ניסויים על עצים רנדומלים ועל משחק מוכר שנקרא שוטרים וגנבים, ולמעשה השוו את ביצועי האלגוריתם אל מול אלגוריתמי הαβ וMinimax-*. על מנת לתאר משחקים תלויי מזל נוסף אלמנט של הסתברות לביצוע המהלך הטוב ביותר בקודקוד מסויים אל מול ההסתברות לבצע כל מהלך אחר.

את תוצאות הניסויים בחרו החוקרים להציג באמצעות השינוי בערכם של שני מדדים –

- ER – מספר הקודקודים שפותחו על ידי אלגוריתם BAB חלקי מספר הקודקודים שפותחו על ידי האלגוריתם האופטימלי הרלוונטי. ככל שהערך של ER יהיה קטן יותר – BAB מבצע חיפוש ייעיל יותר.
- AMB – מתקבל מחישוב $|L(n_1) - U(n_1)|$ כאשר n_1 הוא שורש העץ. חסום על ידי ϵ מהגדרתו של BAB.

התוצאות במאמר חושבו על בסיס יותר מ50 חזרות בכל אחת מההגדרות עבורן הוצגו ערכי ER וAMB , ומציגות דפוסים דומים לגבי משחקים דטרמיניסטים ותלויי מזל. על פי התוצאות, ערכו של ER קטן ככל שערכו של ϵ גדל בהתאם לציפיות החוקרים כי יתכנו גיזומים רבים יותר בעץ כתלות במרחק הפתרון התת-אופטימלי המקובל מערך המינימקס. בנוסף לעצים עמוקים יותר ערכי ER נמוכים יותר מאחר ומתאפשרים חיתוכים עמוקים יותר בעלי יותר קודקודים. ערכו של AMB גדול יותר ככל שהערך של ϵ גדול יותר – תוצאה הגיונית מאחר ו ϵ מהווה חסם עבור AMB. כמו כן ערך AMB גדול יותר כתלות בעצים עמוקים יותר בעלי ערך ϵ זהה מאחר ומתאפשרים חיתוכים רבים יותר לעץ ועל כן הזדמנויות גדולות יותר להגדלת מרחב הערכים התת-אופטימליים.

במחקר נוסף שביצעו Schadd and Winands (2011), מבוצעת השוואה נוספת בין האסטרטגיות Paranoid, Best Replay Search (BRS), max^n בשילוב שיטת חיפוש בשם(Monte-Carlo Tree Search (MCST על העצים שנוצרים בשיטות השונות.

חיפוש MCST שונה מחיפוש מבוסס αβ מאחר ואינו מצריך שימוש בפונקציה היוריסטית לחישוב התועלת של קודקוד מסויים. הערכת קודקוד מסויים מתבצעת על ידי שימוש בפונקציה המבוססת על משתנים כמו מספר הביקורים בקודקוד האב, מספר הביקורים בקודקוד הבן, סטטיסטיקת הניצחון של הקודקוד הבן וסטטיסטיקת הניצחון של הפעולה המבוצעת. בנוסף האלגוריתם מסוגל לחשב את ערך הmixed equilibria של העץ בשונה מהאלגוריתמים האחרים שהוזכרו.

שיטת MCST מורכבת מארבעה שלבים - בחירה , התרחבות , סימולציה והחזרה מעלה. על ידי חזרה על ארבעת השלבים האלו בצורה איטרטיבית עץ החיפוש נבנה הדרגתית. על פי השיטה המקורית, MCST עושה שימוש במבנה עץ הזהה לזה של חיפוש של max^n, אך ניתן לבצע שימוש גם בעצים שפורשים Paranoid וBRS תוך שימוש בפונקציית הערכה שמטרתה להחליש את השחקן הנגדי (בשונה מmax^n שבכל תור נבחר במהלך בעל התועלת המקסימלית עבור השחקן שזהו תורו).

בהשוואות השונות שבוצעו במאמר החוקרים השוו את אחוזי הניצחון של האלגוריתמים תחת הגדרות זמן שונות, כאשר שיחקו זה מול זה דמקה סינית, פוקוס, רוליט ובלוקוס.

בסט הראשון של הניסויים שיחקו זה נגד זה האלגוריתמים max^n, Paranoid וBRS. החוקרים מדדו גם את עומק החיפוש הממוצע של כל אחד מהאלגוריתמים. על פי התוצאות max^n הוא האלגוריתם החלש ביותר עם אחוזי ניצחון נמוכים משמעותית המהאחרים, דבר שניתן להסביר בכך שמדובר באלגוריתם שמבצע מעט גיזומים ועל כן בזמן קצוב מגיע לעומקים נמוכים יחסית. בנוסף מצאו החוקרים כי תחת רוב המשחקים והקונפיגרציות BRS מציג תוצאות טובות יותר גם מParanoid.

בסט השני של הניסויים השוו החוקרים את ביצועי שלושה שחקני MCST, כאשר לכל שחקן עץ משחק שנבנה על ידי אחד מבין האלגוריתמים - max^n, Paranoid וBRS. על פי התוצאות MCTS-max^n משיג את התוצאות הטובות ביותר בכך שניצח את מספר המשחקים הגבוה ביותר תחת רובן המוחלט של הקונפיגורציות. החוקרים מסבירים תוצאות אלו בכך שיתרונם של אלגוריתמי הParanoid והBRS הוא הגיזומים הרבים שהם מאפשרים, אך אינם מבוצעים תחת MCST. החוקרים גם מצאו כי MCTS-Paranoid משיג תוצאות טובות יותר מMCTS-BRS - הסבר לך ניתן למצוא בכך שהנחת הBest Reply של BRS יוצרת מצבי משחק לא אפשריים שפוגעים בביצועי האלגוריתם מבלי להשיג חיפוש עמוק יותר תחת MCTS.

הסט האחרון של הניסויים כלל השוואה בין המנצחים בהשוואות הקודמות - BRS ו -MCTS max^n (במשחק מרובה משתתפים) ותוצאותיה משתנות - תחת משחקים שונים וקונפיגורציות שונות היתרון עובר צד, אך החוקרים מצאו כי MCTS-max^n מתחזק ככל שהזמן המוגדר לתור ארוך יותר.

לסיכום, מבין שלושת טכניקות החיפוש שהוצגו במאמר – BRS מציג את התוצאות הטובות ביותר (מנצח יותר משחקים) וכוחו בכך שחוקר עמוק יותר בכל תור ובכך מפתח יותר צמתי max^n בכל חיפוש. תחת שיטת MCTS אלגוריתם max^n משיג את התוצאות הטובות ביותר - הסבר לכך ניתן למצוא בכך שגיזומי αβ הם כוחם של האלגוריתמים האחרים ואינם מבוצעים בMCTS. בהשוואה בין שתי השיטות העדיפות – לא נרשם יתרון מובהק לאף שיטה.

המאמר סוקר שיטות נוספות לחיפוש בעצי משחק ומציג יתרונות אלגוריתמים אחדים על פני אחרים. תוצאות מחקר זה יוכלו למקד אותנו בבואנו לבחור באלו אלגוריתמים אופטימליים כדאי להתמקד לפיתוח אלגוריתמים לחיפוש תת-אופטימלי חסום בעץ משחק מרובה שחקנים.

Zuckerman (2009) מציע דרך שונה מBRS להתמודד עם הקשיים שבאסטרטגיית הParanoid - אלגוריתם משולב המבצע חיפוש בעץ משחק מרובה משתתפים למשחקים בהם מנצח בודד ואין מדרוג בין המפסידים. האלגוריתם משלב מספר אסטרטגיות שונות ופועל בצורה כזו שמנצל את ההנחות עליהם כל אלגוריתם בנוי בשלב במשחק בו הנחות אלה אכן מתקיימות.

על פי האלגוריתם החדש - MP Mix, בכל תור שחקן המקסימום יבחר באחת משלושת האסטרטגיות- max^n, Paranoid או Directed Offensive – על פי האסטרטגיה המוצגת לראשונה במאמר זה, שכאשר שחקן מסוים חזק בהפרש ניכר משאר השחקנים ("שחקן המטרה"), יבחר שחקן השורש במהלך המזיק בצורה מקסימלית לשחקן המטרה על מנת למנוע את נצחונו במשחק. על פי האסטרטגיה, שאר השחקנים בתורם ינסו לחזק עצמם כפי שהיו פועלים באסטרטגיית max^n.

האלגוריתם המשולב מקבל ערכי To,Td -threshold, מחשב את היוריסטיקת הניקוד של כל משתתף ואת פער הנקודות בין שני השחקנים המובילים. אם השחקן המוביל הוא שחקן השורש וגם הפער שחושב גדול מTd השחקן יבחר באסטרטגיית הParanoid. אם השחקן המוביל אינו שחקן השורש וגם הפער שחושב גדול מTo השחקן יבחר באסטרטגיית Directed Offensive, אחרת - יפעל על פי אסטרטגיית max^n. כאשר To=0 וגם Td>0 השחקן יבחר באסטרטגיית האופנסיב בכל פעם שלא יוביל. כאשר Td=0 וגם To>0 השחקן יבחר באסטרטגיית הפרנואיד בכל פעם שיוביל. כאשר שניהם יהיו גדולים מערך המקסימום שהפונקציה היוריסטית מחזירה, השחקן ישחק תמיד על פי אסטרטגיית max^n.

המאמר בוחן את יכולות האלגוריתם המשולב אל מול max^n וParanoid תחת שני דומיינים – משחק לבבות ומשחק ריסק, תוך שימוש בהגדרות שונות לפרמטרים אותם האלגוריתם מקבל. תחת רוב הקונפיגרציות האלגוריתם המשולב משיג אחוזי ניצחון גבוהים יותר מהאלגוריתמים הקיימים.

תוצאות האלגוריתם מרשימות יותר עבור ריסק מאשר עבור לבבות ועל מנת להסביר זאת ניסחו החוקרים את Opponent impact factor – מדד לגודל השפעה שיש לשחקן אחד על שאר השחקנים שמתמודדים מולו. ערכו של מדד זה נקבע על ידי חישוב מספר המצבים במשחק שמוגדרים כ-InfluentialStates – כאלו שבהם פעולה של שחקן אחד משפיעה על ערך היוריסטיקה של שחקן אחר, חלקי כלל המצבים האפשריים. ניתן לראות כי בכל עומק של העץ, למשחק ריסק ערך Opponent impact גבוה משמעותית משל לבבות.

לסיכום, מסיקים החוקרים שאלגוריתם החדש מנצח יותר מצבים מהאלגוריתמים המוכרים שמולם נמדד, ומשיג תוצאות טובות יותר ככל שערך הOpponent impact factor של המשחק גבוה יותר. בבואנו לבחור אלגוריתם המבצע חיפוש אופטימלי במטרה לבצע על בסיסו חיפוש תת-אופטימלי חסום, חשוב שניקח בחשבון את תוצאות מחקר זה שכן האלגוריתם המשולב מציג תוצאות טובות אל מול האלגוריתמים המוכרים, שכן מנצל את ההנחות שמבוצעות במסגרתם בשלבים המתאימים במשחק.

בטרם נחליט באילו טכניקות נרצה להשתמש, נרצה לבחון חלופות לטכניקות מבוססות αβ. דוגמה לכך ניתן לראות בסקירה של Plaat (1996), המשווה בין אלגוריתמים מסוג Depth-First לאלגוריתמים מסוג Best-First.

אלגוריתמי הDepth-First ובראשם אלגוריתם αβ נמצאים בשימוש נרחב יותר מאשר פתרונות מבוססי Best-First, ובראשם כאלו שעושים שימוש בTransportation tables וIterative deepening. מחקרים קודמים הראו כי אלגוריתמי Best-First בעלי פוטנציאל לבצע חיפושים יותר יעילים, דוגמת *SSS המפתח באופן כמעט מוחלט עצים קטנים יותר, אך היותם סבוכים יותר והצורך שלהם בהקצאות זיכרון נרחבות הופכים אותם לשמישים פחות.

החוקרים מצאו כי ניתן לנסח מחדש את אלגוריתם *SSS כמקרה פרטי של אלגוריתם Depth-First גנרי, ובמחקרים שביצעו זאת מצאו כי תחת ניסוח זה האלגוריתם צורך את אותה הכמות זיכרון כמו αβ, אך תחת הגדרות זיכרון זהות יעילותו אינה גבוהה יותר באופן ניכר וכבר היום קיימים אלגוריתמים מבוססי αβ דוגמת NegaScout המציגים ביצועים טובים יותר.

עוד מצאו החוקרים שתחת אותו אלגוריתם גנרי ניתן לבצע שיטות חיפוש מוכרות אחרות דוגמת *C ו*DUAL מאחר וגם הן מהוות מקרה פרטי שלו. המשותף לאלגוריתמים אלו הוא האופציה להמירם לאלגוריתמי Null window בהם החלון בו נעשה שימוש באלגוריתמי αβ הוא מינימאלי וגודלו שווה ל1, תחת הגדרה זו מתאפשר גיזום של יותר צמתים לאורך החיפוש. על מנת להתמודד עם צרכי הזיכרון האלגוריתם מבצע שימוש בtransposition table – טבלאות המרכזות מידע על מהלכים שנאסף בקריאות קודמות לאלגוריתם בחיפוש זה (בקודקודים קודמים) ובכך נחסכים חישובים חוזרים של מידע.

האלגוריתם הגנרי מנוסח בצורה כזו שמבצע קריאות αβ ומעמיק איטרטיבית עד שמגיע לערך המינימקס, ולמעשה ניתן לבצע באמצעותו חיפושים השקולים לחלוטין לחיפושים אחרים - יש להחליף רק את הערך שמתקבל באתחול. חיפוש שקול ל*SSS (*AB-SSS) נשיג על ידי אתחול ב∞+, חיפוש שקול ל*DUAL (*AB-DUAL) נשיג על ידי אתחול ב∞-. החוקרים מציעים במאמר קונפיגרציה חדשה בשם MTD(f) לאלגוריתם זה שלטענתם משיגה תוצאות טובות יותר משאר האלגוריתמים שהוזכרו קודם. MTD(f) הוא שימוש באותו האלגוריתם הגנרי על ידי אתחול כל איטרציה על פי תוצאת האיטרציה הקודמת ובכך מצפים החוקרים להקטנה משמעותית של מספר הקריאות עד להתכנסות לערך המינימקס.

החוקרים השוו בין ביצועי האלגוריתמים Alpha-Beta, NegaScout, *AB-SSS, AB-*DUAL, וMTD(f) כאשר שיחקו שחמט, אוטלו ודמקה, על בסיס כמות העלים שפיתחו וסך הצמתים הכולל שפיתחו. על פי התוצאות MTD(f) מפתח משמעותית פחות עלים משאר האלגוריתמים ואחריו בסדר *SSS, *DUAL, NegaScout ולבסוף Alpha-Beta. בהשוואה על פי מספר הצמתים הכולל MTD(f)
מפתח את מספר הצמתים הקטן ביותר, אחריו NegaScout, Alpha-Beta, *DUAL ולבסוף *SSS. כאשר החוקרים ביצעו השוואה בין MTD(f) לNegaScout על בסיס זמן עיבוד ומספר עלים הם מצאו כי MTD(f) מציג שיפור של כ10% בשני המדדים.

מסקנות החוקרים הן שאלגוריתם NegaScout מציג יכולות טובות מאלגוריתם αβ המשמש מקור להשוואות רבות, אך אלגוריתם MTD(f) מציג את התוצאות הטובות ביותר בכל המדדים שנבדקו. החוקרים מצאו בנוסף שייעילותו של אלגוריתם ה*SSS ביחס לאלגוריתם Alpha-Beta אינה מובהקת, שMTD(f) משיג תוצאות טובות ממנו ועל כן אינם חושבים שיש להמשיך במחקר בעניינו.

לסיכום, המאמר מציג טכניקה גנרית לביצוע שיטות חיפוש מתקדמות - הן מסוג-Depth First והן מסוג Best-First, על ידי שילוב טכניקות נפוצות להתמודדות עם צרכי זיכרון – על ידי חיפוש בשיטת Iterative Deepening ושימוש מאגרי מידע כמו Transposition tables נוכל להשיג חיפושים יעילים יותר.

סוג נוסף של אלגוריתמים מסוג Best-First הוא אלגוריתמי Proof-Number Search (PN) אותם סקרו Herik and Winands (2008) במטרה לבצע השוואה בינהם לבין אלגוריתם αβ הנמצא בשימוש ברוב המערכות המבצעות חיפוש בעץ משחק, תוך ביקורת על יכולות אלגוריתם הαβ ב End-Game Positions - הבעיה עליה קבוצת אלגוריתמי הPN באים להתגבר.

PN- אלגוריתם הנועד לחיפוש ערכים בעצי משחק, ומטרתו היא הוכחת ערך השורש כtrue (לעץ ישנם 3 ערכים אפשריים - true המסמל ניצחון, false המסמל הפסד או תיקו, unknown אם לא ידוע). העץ מתואר כבעל קודקודי AND/OR, שלכל אחד מהם שני ערכים pn ו dpn(ערכים המייצגים את המספר המינימלי של קודקודים אותם צריך להוכיח בכדי להוכיח את הקודקוד הנוכחי או להפריכו בהתאמה). ערכיהם של pn, dpn נקבעים בהתאם לערכים של

ילדיהם ובהתאם לסוג הקודקוד (AND/OR). האלגוריתם בוחר את הקודקוד הבא לפיתוח על פי מבנה העץ (כמות הילדים לכל קודקוד פנימי) והערכים בעלים, ומבצע חיפוש עבור "הקודקוד המוכיח ביותר", דבר הדורש ממנו זיכרון רב וקיימים מקרים בהם האלגוריתם קורס מעקבות מגבלת זיכרון הנובעת מכך שכאלגוריתם Best-First הוא נדרש לשמור את כל העץ. על כן פותחו האלגוריתמים $PN^2, PN^*, PDS, df-pn$ שמטרתם להתגבר על בעיה זו.

PN^2- אלגוריתם שנועד להתמודד עם בעיית הזכרון של PN, אלגוריתם בעל שני שלבים– בשלב PN_1 מופעל PN על שורש העץ, כאשר אלגוריתם PN משתמש גם כפונקציית ההערכה של הבנים של השורש וזהו שלב PN_2 . בהערכת הבנים מוטלת מגבלת זיכרון כפונקציה של גודל העץ בPN_1. כאשר מושגת מגבלת הזיכרון בשלב PN_2 ישמר המידע שהושג על הבנים של שורש כל אחד מתתי העצים ושאר המידע של תתי העצים ימחקו. הילדים של קודקוד השורש מPN_2 לא מפותחים בהכרח אלא רק של הקודקוד הmost proving ותהליך זה נקרא Delayed evaluation.

PN^* - אלגוריתם זה פותח בסיס הרעיון של MTD(f) הממיר אלגוריתמי best לתצורת depth, מוכיח את שקלותם ומבטל את התלות בזיכרון רב. זהו האלגוריתם האיטרטיבי המבצע חיפוש לעומק הראשון שמבצע שימוש בmultiple-iterative deepening, וממנו פותחו אלגוריתמים נוספים המתגברים על חוסר יעילותו במקרים בהם יש לבצע disproof לעץ מאחר ומשתמש בערך threshold בודד שמגביל את ערך הproof number אך לא את ערך ה disproof number.

PDS – אלגוריתם זה מתגבר על הבעיה שהוצגה ב PN^*, על ידי שימוש בערכי disproof לצד ערכי proof (ובשני ערכי threshold בהתאמה) ומבצע שימוש בmultiple-iterative deepening בכל קודקוד על ידי שימוש בטבלאות המרה.

df-pn - האלגוריתם מהווה וריאציה של PDS שלא מבצעת iterative deepening בשורש העץ על ידי הגדרת שני ערכי הthreshold לאינסוף. וריאציה זו מוכחת כמחזירה תמיד את הmost proofing node בניגוד לPDS ולעיתים מהירה ממנה, אך סובלת מיותר מקרים של בעיית Graph-History Interaction - בעיה אותה מזכירים במאמר אך לא שמים עליה דגש.

המאמר מציג לראשונה אלגוריתם PDS-PN, הנועד לשלב את היתרונות של 2PN ו PDS. בשלב הראשון, מתבצע חיפוש PDS, כך שברוב הגדול של המקרים הוא מפתח את "הקודקוד המוכיח ביותר" לא רק עבור קודקוד השורש אלא עבור כל קודקוד פנימי. את הקודקודים שכבר פיתחנו נשמור בשתי טבלאות המרה לשימוש חוזר. נגדיר קודקוד כ proof-like אם סביר יותר שנצליח להוכיח אותו כproof מאשר כdisproof על ידי בדיקה מתמטית של מספר תנאים, ובמקרה זה נעלה את ערך הסף שלו בהתאם (עבור dislike-proof הגדלת ערך הסף המתאים לו). ניתן לראות כי קל יותר להוכיח עבור קודקוד מסוג or שהוא proof-like וכן להוכיח עבור קודקוד מסוג and שהוא disproof-like.

בשלב השני, מתבצע אלגוריתם PN שממשיך לחקור את העץ כל עוד מגבלת הזיכרון לא מפריעה, וכל עוד העץ עוד לא הוכח כproof או disproof. לאחר סיום השלב השני כל תת העצים שנוצרו בשלב זה ימחקו, ורק השורש של תת העץ בשלב זה ישמר בטבלת ההמרה. החוקרים ביצעו tuning לפרמטרים a וb שמקבל PDS-PN. מסקנת החוקרים היא שעבור כל ערך של a, מספר המצבים שהאלגוריתם פתר גדל ככל שb גדל – עד ערך מסוים, וציינו את ערכי הפרמטרים עבורם התקבל מספר המצבים הפתורים המקסימלי.

מסקנות החוקרים הן שאלגוריתמי הPN השונים השיגו תוצאות טובות מ$\alpha\beta$ במדדים שנבדקו, שאלגוריתם PN חלש בבעיות קשות בעקבות צרכי זיכרון גבוהים ושבאופן כללי PDS ו PN^2 פותרים מגוון בעיות גדול יותר מ $\alpha\beta$ וPN. כמו כן קבעו החוקרים כי PDS-PN מציג יכולות טובות יותר מ$\alpha\beta$ בשלב הendgame, כי הוא מהיר כמעט כמו PN^2 תחת ההגדרות המתאימות, פותר יותר מצבים קשים מ PN^2 ויעיל יותר מPN ואף מPN^2 מאחר ואיננו מגיע לסף מגבלת הזיכרון ומתפקד בצורה טובה גם במצבי זיכרון קשים. מסקנה נוספת היא ש Df-pn יכול להוות אלטרנטיבה איכותית לPDS-PN בעקבות תוצאותיו הטובות מול PDS.מכאן שאלגוריתמים ממשפחת PN הם כלי חשוב לפתרון בעיות בשלב הendgame ובפרט PDS-PN היעיל יותר מ PDS מPN^2 כendgame solver עבור משחקים קשים.

תיאור מטרות המחקר

הגדרת התנאים הדרושים על מנת שאלגוריתם חיפוש, בעץ משחק מרובה שחקנים, יחזיר פתרון תת-אופטימלי חסום, כלומר כזה הרחוק עד כדי קבוע מהערך האופטימלי, ופיתוח של אלגוריתם חיפוש על בסיס תנאים אלו.

מתודולוגיית המחקר

המחקר מהווה המשך לAtzmon (2018) – בעוד הוא עוסק בפתרונות תת-אופטימלים חסומים למשחקים בשני משתתפים, אנו נרצה כתוצר סופי להציע פתרונות דומים למשחקים מרובי משתתפים. במאמר, החוקרים עשו שימוש באלגוריתם αβ (אלגוריתם אופטימלי המבצע גיזום) כתשתית על גביה הוגדרו תנאים נוספים המאפשרים גיזום תת-אופטימלי חסום, כך שלבסוף הציגו אלגוריתם חדש בשם Boundedαβ, אשר αβ מהווה מקרה פרטי של אותו האלגוריתם עם אפיסילון (קבוע המרחק מהתוצאה האופטימלית) שווה לאפס. באופן דומה ולאחר סקירת אלגוריתמים שונים למשחקים מרובי משתתפים, בחרנו באלגוריתם ShallowPrunningMaxN – אלגוריתם חיפוש למשחקים מרובי משתתפים בעל אותן התכונות, כלומר אופטימלי ומבצע גיזום, לשמש כתשתית לפיתוח תנאים לגיזום תת-אופטימלי חסום –

```
ShallowPrunningMaxN(Node, Player, Bound):
IF Node is terminal, Return static value
Best = ShallowPrunningMaxN(first Child, next Player, Sum)
FOR each remaining Child:
        IF Best[Player] ≥ Bound, RETURN Best
        Current = ShallowPrunningMaxN (Child, next Player, Sum – Best[Player])
        IF Current[Player] > Best[Player], Best = Current
RETURN Best
```

בדרכנו להגדיר את התנאים כנדרש, ולאור פתרונות שונים ומגוונים אותם סקרנו, בחנו מספר אפשרויות שאמנם לא עמדו בדרישות, אך חקר כישלונותיהם הוביל לאלגוריתם אותו נציג. על כן, אנו מוצאים ערך בתיעודם –

```
Version1(Node, Player, Bound, Parent, Uparent, Lparent, ε):
FOR index In Players.length():
        U[index] = Sum
        L[index] = 0
IF Node is terminal, Return static value
Best = Version1(first Child, next Player, Sum, Player, U, L , ε)
FOR each remaining Child:
        IF Best[Player] ≥ Bound, RETURN Best
        Current = Version1(Child, next Player, Sum – Best[Player], Player, U, L, ε)
        IF Current[Player] > Best[Player]:
                Best = Current
                L[Player] = Best[Player]
                For index In Players.length():
                        If index ≠ Player
                                U[index] = Sum – Best[Player]
        IF Lparent[Parent] + ε ≥ U[Parent] , RETURN Best
RETURN Best
```

הפתרון שומר לכל קודקוד את הערך המקסימלי והמינימאלי שכל שחקן יוכל להשיג תחתיו, על סמך מידע שאסף מצאצאיו שנחקרו עד לאותו השלב. הנחנו שניתן לבצע גיזום תחת אותו הקודקוד ערכים אלו שכן אנו מבטיחים שהקודקודים שנגזום אינם מכילים ערך הגבוה ביותר מאפסילון עבור השחקן שזו קודקוד בחירה שלו. כך, אם יצטרך לשחק את המהלך הזה יוכל לבחור בערך המובטח מבלי להפגע ביותר מאפסילון. בפועל, קיימת בעייתיות בביצוע גיזומי עומק בעצים מסוג max^n הנובעת מהעובדה שהגיזום מונע מאיתנו לדעת כיצד השחקנים האחרים שתורם קודם לתור המדובר יבחרו לפעול במידה ויחקרו את תת-העץ שגזמנו - אם נגזום את הערך האופטימלי עבורם, נניח בטעות שיבצעו מהלך אחר בתור קודם ונחזיר ערך שגוי לשורש העץ.

מסקנתנו מכשלון התנאים שהגדרנו היא שניתן להבטיח ערך מסויים לשחקן רק אם אותו הערך עלה עד לשורש העץ, שכן בכל קודקוד מתחת לשורש קיים תור אחד לפחות של שחקן אחר שלא נוכל לקבוע כיצד יפעל עד שלא נחקור את כל האפשרויות העומדות לפניו. נרצה לבחון האם בעדכון הערך המובטח רק עבור שחקן השורש ורק בשורש העץ נוכל לבצע גיזומים תת-אופטימליים חסומים גם בעומקו

```
Version2(Node, Player, Bound, Promised, ε):
IF Node is terminal, RETURN static value
Best = Version2(first Child, next Player, Sum, Promised, ε)
FOR each remaining Child:
        IF Node is root, Promised = Max{Promised, Best[Player]}
        IF Node is root AND Promised + ε ≥ Sum, RETURN Best
        IF Player is Player2 AND Promised + ε ≥ Sum – Best[Player], RETURN Best
        IF Best[Player] ≥ Bound, RETURN Best
        Current = Version2 (Child, next Player, Sum - Best[Player], Promised, ε)
        IF Current[Player] > Best[Player], Best = Current
RETURN Best
```

גם במקרה הזה גיזום בעומק העץ עלול להוביל לתמונת מצב לא נכונה לגבי מהלך שאחד המתמודדים האחרים יבחר לבצע, ואכן במהרה קיבלנו דוגמה הסותרת את נכונות האלגוריתם.

בחינה נוספת של המאמרים שסקרנו, תוך הבנה מעמיקה יותר של התחום, הובילה למסקנה שהתנאים שנציע מוגבלים לעומק הגיזום שמבצע האלגוריתם ShallowPrunningMaxN בלבד, כלומר נוכל לגזום רק "נכדים" של השורש. כמו כן, במידה ונגזום תת-עץ מסויים נרצה להמנע מלבחור בו על סמך הקודקודים שחקרנו בו עד כה, שכן הוא לא נחקר עד הסוף - דבר שיכול לפגוע בנכונות האלגוריתם, על כן נחזיר לשורש ערך שלילי על מנת שימנע מלהמשיך בחיפוש באותו הכיוון, שכן כבר מובטח לנו ערך מבן אחר של השורש הקטן בעד אפסילון מכל ערך בתת העץ הזה. מסקנות אלו הובילו אותנו לאלגוריתם BoundedSuboptimalMaxN -

```
BoundedSuboptimalMaxN(Node, Player, Bound, Promised, ε):
IF Node is terminal, RETURN static value
Best = BoundedSuboptimalMaxN(first Child, next Player, Sum, Promised, ε)
FOR each remaining Child:
    IF Node is root, Promised = Max{Promised, Best[Player]}
    IF Node is root AND Promised + ε ≥ Sum, RETURN Best
    IF Player is Player2 AND Player.Parent is root
      AND Promised + ε ≥ Sum – Best[Player], RETURN [-1] +[0]*(NumPlayers-1)
    IF Best[Player] ≥ Bound, RETURN Best
    Current = BoundedSuboptimalMaxN(Child, next Player, Sum - Best[Player],
Promised, ε)
    IF Current[Player] > Best[Player], Best = Current
RETURN Best
```

אלגוריתם נוסף שהחלטנו לממש ולבחון את איכות ביצועיו הוא BoundedSuboptimalParanoid. מטרת מימוש אלגוריתם זה היא לבצע השוואה בין התנאים לשני שחקנים שהוגדרו על ידי Atzmon(2018) לאלו שהחלנו על ShallowPrunningMaxN במטרה ליצור את BoundedSuboptimalMaxN . האלגוריתם למעשה אינו מהווה לגמרי אלגוריתם חדש שכן נעשה שימוש באלגוריתם Boundedαβ. השוני נובע מכך שנחשב את ערכו של V(n) על ידי חיסור הכניסה בוקטור התועלת שמייצגת את השחקן הParanoid בסכום שאר הכניסות בוקטור ובכך למעשה נשיג את הרדוקציה לשני שחקנים הדרושה על מנת שנוכל להשתמש בBoundedαβ כמעט כלשונו (במקור יש התייחסות גם לקודקודי מזל בהם לא עסקנו במחקר ועל כן הורדו מהפסאודו קוד) –

```
BoundedSuboptimalParanoid(Node, ε):
IF Node is a terminal node, RETURN (V(Node); V(Node))
L(Node) = vmin
U(Node) = vmax
IF Node is a MAX node, bestU = vmin
ELSE IF Node is a MIN node, bestL = vmax
IF Node is root, α(Node) = vmin; β(Node) = vmax
ELSE, α(Node) = α(p(Node)); β(Node) = β(p(Node))
FOR each child c of Node:
        (L(c), U(c)) = BoundedSuboptimalParanoid(c, ε)
        IF Node is a MAX node:
                bestU = max(bestU, U(c))
                L(Node) = max(L(Node),L(c))
        ELSE:
                bestL = min(bestL, L(c))
                U(Node) = min(U(Node),U(c))
        α(Node) = max(L(Node), α(Node))
        β(Node) = min (U(Node), β(Node))
        IF β(Node) ≤ α(Node) + ε and c is not the last chile of Node:
                 RETURN (L(Node), U(Node))
IF Node is a MAX node, U(Node) = bestU
ELSE, L(Node) = bestL
RETURN (L(Node), U(Node))
```

פירוט הניסויים

ניסוי ראשון – עצים רנדומליים

מטרה –

לבחון כיצד שינוי ערכו של אפסילון באלגוריתם BoundedSuboptimalMaxN משפיע על ה Expansion Rate(ER) – מדד המבטא את היחס בין מספר הקודקודים שפיתח אלגוריתם חיפוש ביחס למספר הקודקודים בעץ השלם. בנוסף, בנוסף נרצה לבחון שינוי ערכו של אפסילון משפיע על השינוי בניקוד הסופי שמקבל השחקן התת-אופטימלי בסיום החיפוש, ביחס לתוצאה האופטימלית.

מהלך הניסוי –

הניסוי מדמה משחק בשלושה משתתפים – השחקן הנבדק ושני שחקני ShallowPrunningMaxN . באמצעות מחולל העצים הרודומליים ייצרנו 150 עצים בשלושה עומקים שונים – 4, 7, ו10, כך שהתקבלו 50 עצים בכל עומק. הגדרנו branching קבוע של 5. לכל עלה הוגרל וקטור רנדומלי עם 3 כניסות, כאשר סכום כל וקטור תועלת בכל עלה הוגדר להיות 64 (בהתאמה לניסויים הבאים בהם נממש משחק רוליט עם היוריסטיקה מקסימלית של 64). כל עץ נחקר על ידי ShallowPrunningMaxN ועל ידי 5 חיפושים של BoundedSuboptimalMaxN עם אפסילונים שונים – 4,8,16,24,32. החיפוש רץ עד סיום מבלי שהוטלה מגבלת קודקודים או זמן.

מימושים –

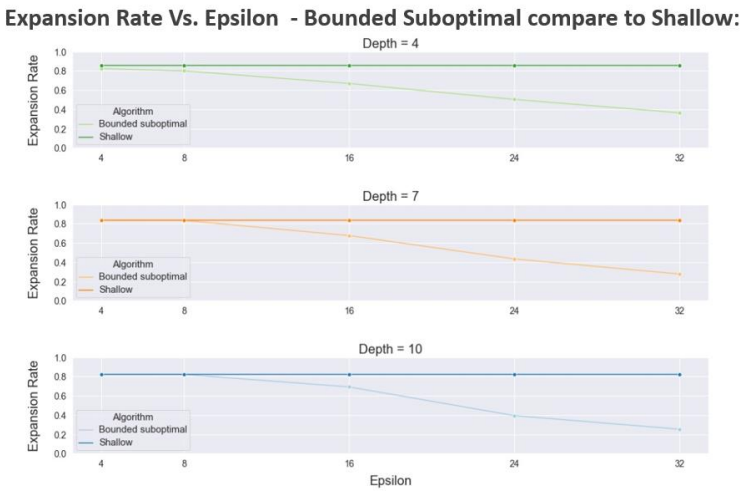
לטובת הניסוי מימשנו מחולל עצים רנדומליים וסקריפט המדמה סימולציית משחק. בנוסף מומשו האלגוריתמים המשתתפים בהשוואה – BoundedSuboptimalMaxN וShallowPrunningMaxN (אשר זהה לBoundedSuboptimalMaxN כאשר אפסילון שווה ל0).

תוצאות הניסוי –

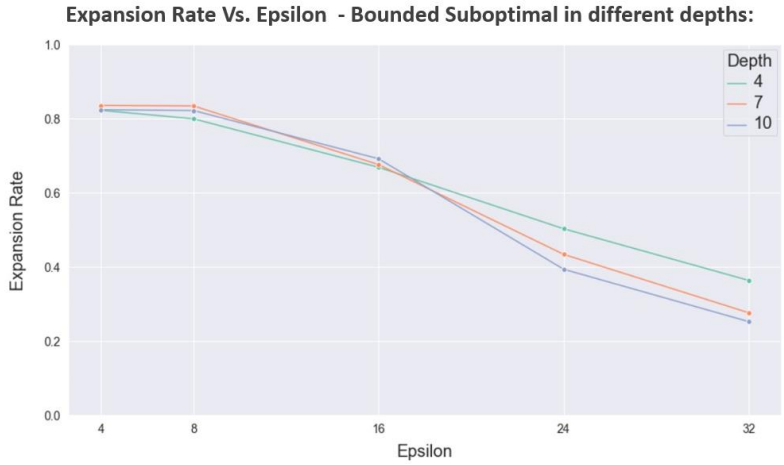
טבלה 1

Epsilon	Mean ER Bounded Suboptimal	Mean ER Shallow	Result Suboptimality
4	0.827991	0.838354	0.000000
8	0.819230	0.838354	0.000000
16	0.679456	0.838354	0.233333
24	0.443992	0.838354	1.733333
32	0.297895	0.838354	3.240000

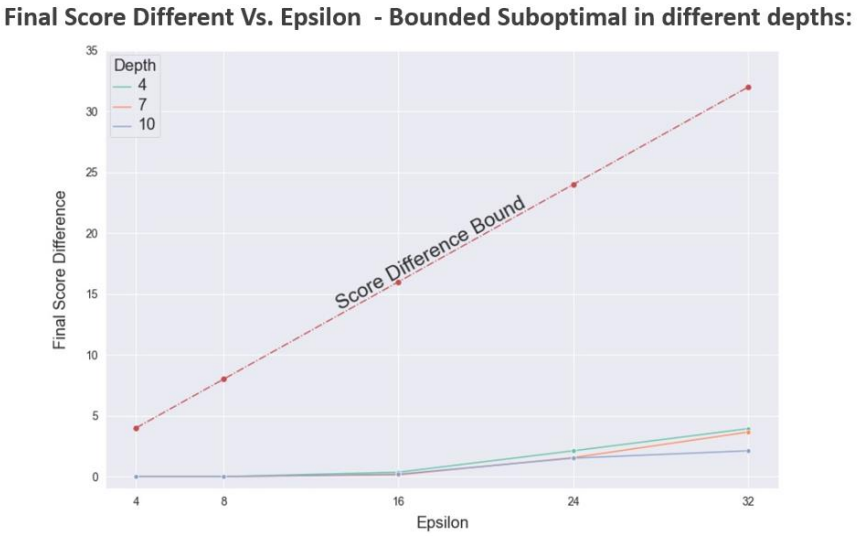
גרף 1



גרף 2



גרף 3



ניתן לראות בגרף 1 כי בעוד ShallowPrunningMaxN פיתח כ80% מקודקודי העץ בכל אחד משלושת העומקים, BoundedSuboptimalMaxN פיתח כצפוי פחות קודקודים ככל שערכו של אפסילון גדל (באפסילון 24 מפתח כבר פחות ממחצית מהעץ).

מגרף 2 מתקבלת מגמה צפויה נוספת והיא שככל שהעץ עמוק יותר, לאלגוריתם BoundedSuboptimalMaxN פוטנציאל לפתח באופן יחסי חלק קטן יותר ממנו, שכן כבר בבנים הראשונים של השורש חוקר יותר מצבים ומעלה את סיכויו לעדכן את הערך המובטח לו בערך גבוה יותר (שכן הערכים בעץ מתפלגים בצורה אחידה).

בגרף 3 ניתן לראות שבאופן גורף התוצאה הממוצעת שהחזיר האלגוריתם קרובה מאוד לאופטימלית ורחוקה מאוד מהמינימאלית שיכל להחזיר.

<u>מסקנות –</u>

מצאנו כי קיים קשר חזק בין ערכו של אפסילון לExpansion Rate – ככל שנגדיל יותר את אפסילון נקטין בהתאמה את החלק היחסי בעץ אותו נחקור.

מצאנו כי קיים קשר עומק העץ לExpansion Rate – בהנחה שערכי העלים מתפלגים אחיד, נצפה לראות כי נחקור חלקים קטנים יותר באופן יחסי מעצים עמוקים יותר.

בכל שלושת העומקים שחקרנו מצאנו כי הגדלת ערכו של אפסילון אינה משפיעה באותה עוצמה על על התוצאה הסופית.

ניסוי שני – מציאת אפסילון אופטימלי

מטרה –

מציאת האפסילון האופטימלי עבור BoundedSuboptimalMaxN ועבור BoundedSuboptimalParanoid בדומיין משחק רוליט. נרצה לבחון כיצד שינוי ערכו של אפסילון משפיע על העומק הממוצע אליו יחקור כל אלגוריתם, וכיצד השינוי משפיע על התוצאה הסופית אותה יחזיר, כל זאת בהשוואה לאלגוריתם האופטימלי ממנו הוא פותח.

מהלך הניסוי –

יצרנו בנק של 30 לוחות משחק רוליט רנדומליים לארבעה שחקנים, כך שכל לוח מייצג מצב אפשרי של המשחק לאחר 16 מהלכים חוקיים שבוצעו בצורה רנדומלית. כל אחד מארבעת האלגוריתמים – ShallowPrunningMaxN, BoundedSuboptimalMaxN, Paranoid, BoundedSuboptimalParanoid, התמודד בכל אחד מ30 הלוחות כשחקן רביעי נגד Paranoid, Paranoid, ShallowPrunningMaxN בסדר הזה. נחדד כי עבור כל אפסילון שבדקנו עבור כל אחד מהאלגוריתמים התת-אופטימלים הרצנו 30 משחקים חדשים. עבור BoundedSuboptimalMaxN בחנו את ערכי אפסילון 2,4,8,16,24,32,36,40,44,48 ועבור BoundedSuboptimalParanoid בחנו גם את הערכים 1 ו3. את בחירת האפסילונים ביצענו בצורה הדרגתית ובהתאם לתוצאות שהתקבלו מאפסילונים קודמים מתוך ציפייה שערכים קרובים יחזירו תוצאות יחסית דומות. חקרנו את העץ בשיטתIterative Deepening והגבלנו את כלל האלגוריתמים לחקור מקסימום 20,000 קודקודים בסך כל האיטרציות שהן מבצעות, כך שניתן יהיה לבחון לאיזה עומק הגיע כל אלגוריתם תחת המגבלה שהוטלה.

מימושים –

לטובת הניסוי מימשנו דומיין משחק רוליט והיוריסטיקה על מנת לחשב ערך לקודקודים שאינם מצבים סופיים, אותו הטמענו בממשק הסימולטור שמימשנו בניסוי הקודם. הוספנו אפשרות להטיל מגבלת קודקודים על תור מסויים בכך שמימשנו אופציה לחיפוש בצורת Iterative Deepening. כמו כן מימשנו את האלגוריתמים Paranoid וBoundedSuboptimalParanoid המשתתפים בהשוואה.

תוצאות הניסוי –

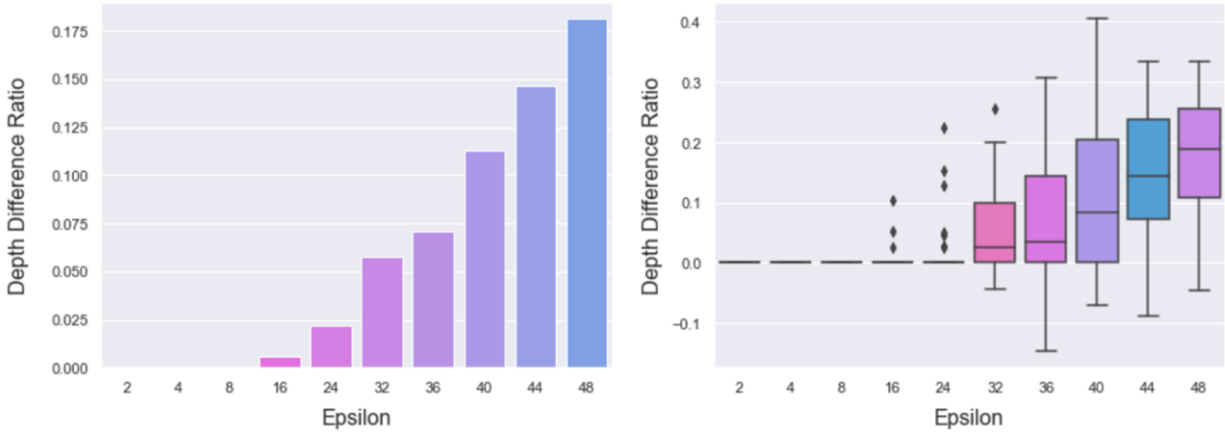
BoundedSuboptimalMaxN

טבלה 2

Epsilon	Final Score Difference Ratio	Depth Difference Ratio
2	0.000000	0.000000
4	0.000000	0.000000
8	0.000000	0.000000
16	0.012346	0.006028
24	-0.025803	0.021899
32	0.027469	0.057730
36	0.249699	0.071143
40	0.137707	0.112935
44	0.029993	0.146458
48	-0.078389	0.180969

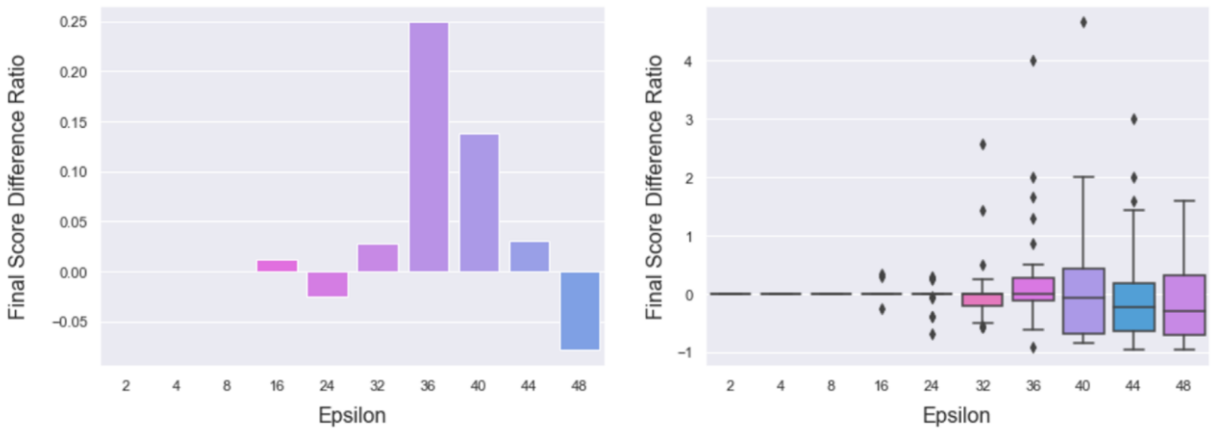
גרף 4

Final Score Difference Ratio Vs. Epsilon - Bounded Suboptimal Final Score compare with Shallow:



גרף 5

Depth Ratio Vs. Epsilon - Bounded Suboptimal average searches depth compare with Shallow:

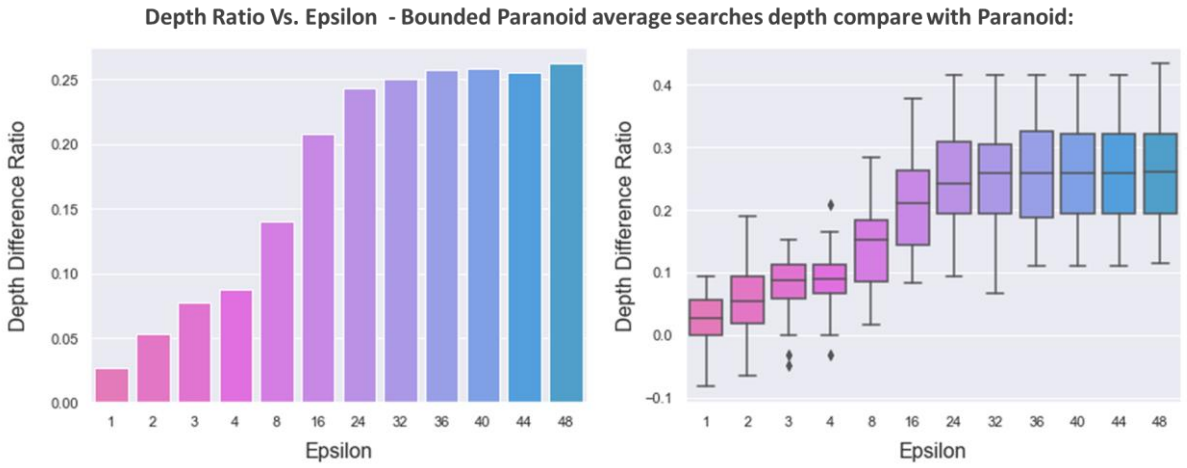


BoundedSuboptimalParanoid

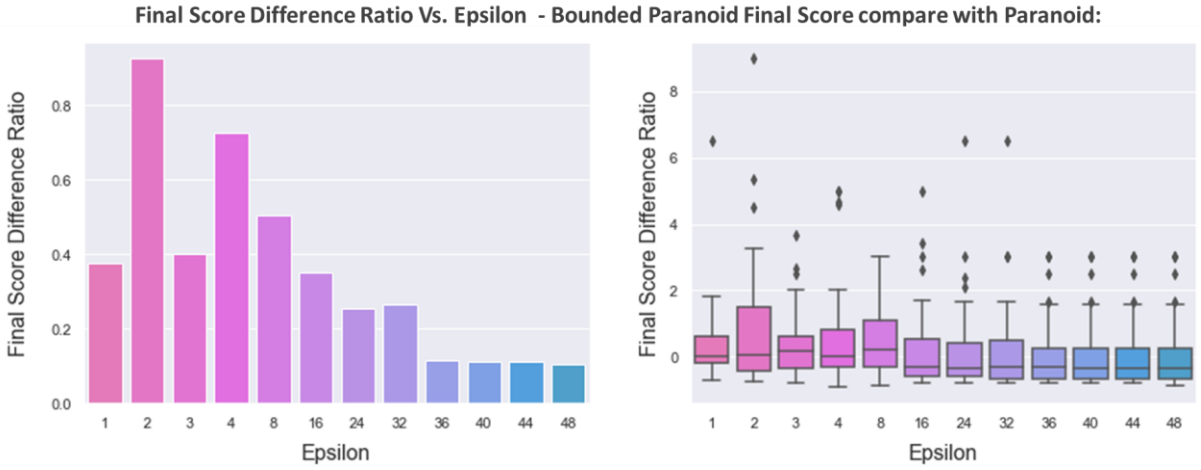
טבלה 3

Epsilon	Final Score Difference Ratio	Depth Difference Ratio
1	0.376060	0.026242
2	0.923168	0.052957
3	0.398913	0.077186
4	0.726368	0.086970
8	0.502990	0.140313
16	0.349662	0.207757
24	0.252841	0.243770
32	0.264444	0.249911
36	0.115905	0.257484
40	0.111738	0.258152
44	0.113253	0.255625
48	0.105678	0.262070

גרף 6



גרף 7



גרפים 4 ו6 מצביעים על מגמה צפויה – כשם שמצאנו בניסוי הראשון שהגדלת ערכו של אפסילון מקטינה את מספר הקודקודים שנפתח בריצה בודדת של האלגוריתם, התוצאות מראות שככל שאפסילון גדל העומק אליו הגיע האלגוריתם גדל - שכן הוא הספיק לבצע איטרציות רבות יותר.

מגרפים 5 ו7 אנו למדים שעבור רוב ערכי האפסילון שהוגדרו, האלגוריתמים התת אופטמליים השיגו תוצאה טובה יותר בעד 25% בסיום המשחק ממקביליהם האופטימליים (עבור Paranoid כל האפסילונים הציגו שיפור). מנגד לא ניכרת מגמה המצביעה על הקשר בין שינוי ערכו של אפסילון לפער שבין הפתרון שיתקבל לאופטימלי.

<u>מסקנות –</u>

האלגוריתם BoundedSuboptimalMaxN השיג את השיפור הממוצע הטוב ביותר עבור אפסילון 36, בעוד עבור BoundedSuboptimalParanoid היה זה אפסילון 2. השוני מצביע על כך שאכם קיים צורך לחפש אפסילון אופטימלי לכל אלגוריתם. נשתמש באפסילונים אלו על מנת לבחון את יכולות האלגוריתמים בניסוי הבא.

מהתוצאות נסיק כי הגדלת ערכו של אפסילון מגדילה ביחס ישר את העומק אליו מגיעים אלגוריתמי החיפוש התת-אופטימליים.

BoundedSuboptimalParanoid מציג שיפור גדול ומובהק יותר מאשר BoundedSuboptimalMaxN כאשר השוו לאלגוריתמים האופטימלים מהם פותחו.

ניסוי שלישי – בדיקת איכות האלגוריתמים

מטרה –

להשוואת את יכולות האלגוריתמים התת-אופטימלים שניסחנו אל אלגוריתמים שונים.

מהלך הניסוי –

בעוד בניסוי הקודם בחנו כל פתרון תת-אופטימלי בהשוואה לאלגוריתם האופטימלי ממנו פותח, בניסוי זה נגדיר נרצה להשוות כל אחד מהאלגוריתמים המשתתפים לשאר באופן חופשי. לשם כך ייצרנו 96 לוחות רנדומליים, כך שכל לוח מייצג מצב אפשרי של המשחק לאחר 16 מהלכים חוקיים שבוצעו בצורה רנדומלית. לכל לוח הגרלנו 4 משתתפים מתוך 6 אפשריים (בלי החזרות) – Random, Greedy, ShallowPrunningMaxN, BoundedSuboptimalMaxN, Paranoid, BoundedSuboptimalParanoid. Random בוחר מהלך בצורה רנדומלית וGreedy מחפש בצורה נאיבית ומשתמש בהיוריסטיקה חלשה שלא מבצעת חיפוש לעומק. האלגוריתמים התת-אופטימליים עשו שימוש באפסילונים המיטביים מהניסוי הקודם. השוונו את התוצאה הסופית ההמוצעת שהחזיר כל אחד מהאלגוריתמים בכלל המשחקים שלו, וכן בדקנו מהו יחס הזכיות שלו. גם כאן חקרנו את העץ בשיטת Iterative Deepening והגבלנו את כלל האלגוריתמים לחקור מקסימום 20,000 קודקודים בכל תור.

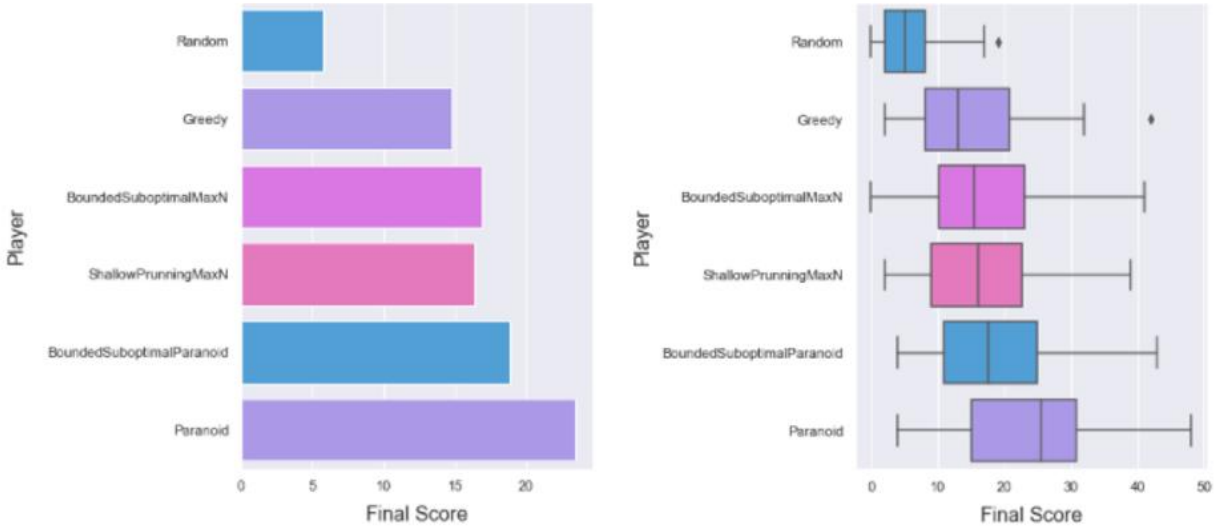
מימושים –

לטובת הניסוי מימשנו את השחקנים Greedy וRandom וכן מחולל שיבוצים רנדומאלי הוגן לשיבוץ השחקנים ללוחות בכמות דומה.

תוצאות הניסוי –

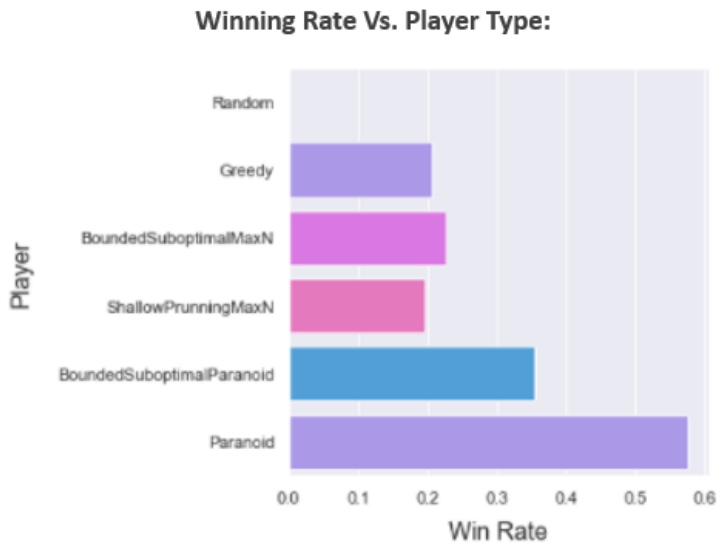
גרף 8

Final Score Vs. Player Type:



גרף 8 מציג את ההבדל בתוצאה הסופית הממוצעת שהשיג כל אלגוריתם בשכלול תוצאותיו במשחקים בהם השתתף, בעוד בגרף 9 ניתן לראות את יחס הזכיות של כל אחד מהאלגוריתמים. ניתן לראות כי אלגוריתם Paranoid הוא המנצח בטורניר בבחינת שני המדדים, ואחריו BoundedSuboptimalParanoid. מידת הצלחתם של BoundedSuboptimalMaxN וShallowPrunningMaxN דומה, עם יתרון לחלופה התת-אופטימלית בשני המדדים, אך שניהם באופן כללי לא הציגו תוצאות טובות, במיוחד לאור העובדה שהשחקן החלש, Greedy, השיג תוצאות דומות.

גרף 9



<u>מסקנות</u>

הדימיון בין ShallowPrunningMaxN לGreedy עלול להצביע על פגם בהיוריסטיקה שבה השתמשנו. מקור החישוב ההיוריסטי שביצענו הוא מניסויים שבוצעו על חלופה של המשחק לשני שחקנים ויתכן וזו גם הסיבה להצלחתם של אלגוריתמי הParanoid שכן הם מבצעים רדוקציה לבעיה בשני שחקנים.

העובדה שBoundedSuboptimalMaxN השיג תוצאה ממוצעת ויחס זכיות טובים יותר מהאלגוריתם האופטימלי המקביל מצביעים על כך שלאלגוריתמים מסוג זה אלו פוטנציאל לבצע חיפוש איכותי יותר שכן הם חוקרים מצבים מתקדמים יותר.

<u>מסקנות, סיכום ואתגרים לעתיד:</u>

בהסתכל על תוצאות שלושת הניסויים נוכל להסיק כי בכוחם של האלגוריתמים התת-אופטימליים לשפר את תוצאותיו של חיפוש בהשוואה לתוצאות שהיו מתקבלות על ידי אלגוריתם אופטימלי. הסבר לכך נוכל לזקוף לזכות גיזום הקודקודים הנרחב שהם מאפשרים, כך שביכולתם לחקור מצבים מתקדמים יותר בעץ ולקבל החלטות שקולות יותר על סמך חישובים היוריסטים מדוייקים יותר.

ובכל זאת, התוצאות אינן חד משמעיות. יתכן ובקביעת ערכו של אפסילון יש לבודד משתנים נוספים מסוג המשחק והאלגוריתם, כגון – שלב המשחק, השחקנים מולם מתמודד, מצבו של השחקן ביחס למתמודדים האחרים וכדומה. על כן, כמחקר המשך נמליץ לבחון האם ניתן לבחור את אפסילון בצורה דינאמית לאורך ריצת החיפוש, למשל בתחילת תור או טרם כל איטרציה של Iterative Deepening שמבוצעת.

נקודה נוספת נוגעת לאופן שבו עושים ואלידציה להיוריסטיקה. בסיום הניסוי השלישי הבחנו כי השחקן החלש, Greedy, והשחקן האופטימלי, ShallowPrunningMaxN, משיגים תוצאות דומות ביותר, הן מבחינת התוצאה הסופית הממוצעת והן ביחס הזכיות בטורניר. מתוצאות אלו הנחנו כי השתמשנו בהיוריסטיקה חלשה למדי עבור האלגוריתמים שלנו שכן על אף שהם חוקרים לעומק אלפי קודקודים הניחוש הרדוד משיג תוצאה דומה. נמליץ לבחון בצורה דומה את יכולת ההיוריסטיקה בשלב מוקדם יותר של מחקרי המשך על מנת שחולשתה לא תגרום להטעיית תוצאות.

גרפים וטבלאות

ביבליוגרפיה

1. Atzmon, D., Stern, R., & Saffidine, A. (2018). Bounded suboptimal game tree search. Proceedings International Symposium on Combinatorial Search, Sweden, 11, 10-18.

2. Luckhardt, C.A., & Irani, K.B. (1986). An algorithmic solution of n-person games. Proceedings of the AAAI National Conference on Artificial Intelligence, 5, 158–162.

3. Schadd, M.P.D., & Winands, H.M. (2011). Best Reply Search for multiplayer games. IEEE Transactions on Computational Intelligence and AI in Games, 3(1), 57-66.

4. Plaat, A., Schaeffer, Schaeffer, J., Pijls, W., & de Bruin, A. (1996). Best-First and Depth-First Minimax Search in practice. Artificial Intelligence, 84(2), 299-337.

5. Nijssen, J.A.M., & Winands, M.H.M. (2011). An Overview of Search Techniques in Multi-Player Games. Computer Games Workshop at ECAI 2012, France, 50-61.

6. Knuth, D.E., & Moore, R.W. (1975). An analysis of alpha-beta pruning. Artificial Intelligence, 6(4), 293-326.

7. Zuckerman, I., Felner, A., & Kraus, S. (2009). Mixing Search Strategies for Multi-Player Games. Proceedings of the international joint conference on Artificial intelligence, USA, 21, 646-651.

8. Sturtevant, N.R., & Korf, R.E. (2000). On pruning techniques for multi-player games. Proceedings of the Conference on Innovative Applications of Artificial Intelligence, 12, USA, 201–207.

9. Van der Herik, H.J., & Winands, M.H.M. (2008). Proof-Number Search and its Variants. Oppositional Concepts in Computational Intelligence, 155, 91-118.

10. Wald, A. (1945). Statistical Decision Functions Which Minimize the Maximum Risk. Annals of Mathematics, 46(2), 265-280.

11. Korf, R.E. (1991). Multi-player alpha-beta pruning. Artificial Intelligence, 48(1), 99-111.