קישורים לסיכומים –

* [סיכום מאמר 1 - חיפוש תת אופטימלי מוגבל בעץ משחק](#מאמר1)
* [סיכום מאמר 2 - פתרון אלגוריתמי למשחק מרובה שחקנים](#מאמר2)
* [סיכום מאמר 3 – Best Reply Search למשחק מרובה שחקנים](#מאמר3)
* [סיכום מאמר 4 - חיפוש ערך מינימקס בשיטות Best-First ו Depth-First בפועל](#מאמר4)
* [סיכום מאמר 5 – טכניקות חיפוש במשחקים מרובי שחקנים](#מאמר5)
* [סיכום מאמר 6 – ניתוח גיזום אלפא בטא](#מאמר6)
* [סיכום מאמר 7 – שילוב אסטרטגיות חיפוש עבור משחקי מרובי שחקנים](#מאמר7)
* [סיכום מאמר 8 – טכניקות גיזום עבור משחקים מרובי שחקנים](#מאמר8)
* [סיכום מאמר 9 - אלגוריתמי חיפוש בשיטת Proof-Number](#מאמר9)

סיכום מאמר 1 - חיפוש תת אופטימלי מוגבל בעץ משחק

Bounded Suboptimal Game Tree Search [Atzmon et al., 2018]

הבעיה המוכרת בבסיסו של מאמר זה היא מציאת ערך המינימקס של עץ משחק. תורת המשחקים, קבלת החלטות, סטטיסטיקה, פילוסופיה, כלכלה, רובוטיקה ואבטחה הם רק חלק מהתחומים בהם לפתרון בעיה זו ערך רב, ועל כן מחקרים רבים לאורך השנים עוסקים בפיתוח אלגוריתמים ופתרונות ייעילים העוסקים בה.

כרקע לבעיה נסביר כי עקרון המינימקס הוא עיקרון בסיס בכל הנוגע לקבלת החלטות במשחקי סכום אפס – משחקים בהם בכל מהלך הרווח של שחקן אחד שווה להפסד של השחקן השני. 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 איטרטיבי הפועל על פי עיקרון זה ומוצא את ערך המינימקס על עץ משחק (נקרא גם ערך שיווי המשקל של העץ), המבטא את התועלת הסופית עבור השחקן שבשורש העץ.

במקרים רבים עצי משחק הם גדולים מאוד – כתלות במספר המהלכים עד לסיום המשחק, ובמספר הפעולות האפשריות ששחקן יכול לבצע בתורו, וביצוע חיפוש נאיבי על כלל הקודקודים בשיטה זו במטרה להחזיר את ערך המינימקס לא תמיד מתאפשר. על כן לאורך השנים פותחו אלגוריתמים רבים במטרה להתגבר על קושי זה, כאשר המוכר מבינהם הוא אלגוריתם האלפא בטא המבצע גיזומים בעץ כאשר מזהה צמתים בעץ שאין ערך בפיתוחם – אך במקרים רבים גם אלגוריתם זה או דומים לו אינם מבצעים מספיק גיזומים על מנת להקל מספיק על מציאת הפתרון.

החוקרים זיהו דמיון בין בעיה זו לתחום אחר – פתרון בעיות חיפוש קשות של סוכן בודד, בו הזמן הנדרש למציאת הפתרון האופטימלי לעיתים קרובות אינו פיזיבילי. אחת הדרכים להתגבר על בעיה זו היא להקל על הפתרון האופטימלי ולאפשר פתרונות תת-אופטימליים. תחת הדומיין שבו המאמר עוסק (משחקי סכום אפס לשני שחקנים תלויי מזל עם מידע מושלם), החוקרים מציאים ליצור טרייד-אוף בין זמן הריצה לאופטימליות הפתרון על ידי אפשור של גיזום נרחב יותר מזה המתאפשר בחיפוש אחר פתרון אופטמלי.

המאמר מגדיר תאוריה לתכנון אלגוריתמים תת-אופטימליים מוגבלים – הרעיון הכללי מגדיר כי אלגוריתמים אלו יקבלו מהמשתמש ערך ϵ ויחזירו פתרון תת-אופטימלי לעץ משחק שערכו רחוק בעד ϵ מערך המינימקס האמיתי של העץ. בנוסף מוגדר רעיון כללי 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.

כמו כן, מוצג במאמר אלגוריתם חיפוש חדש לעצי משחק בשם Bounded Alpha-Beta (BAB) הפועל על פי חוקי גזירה אלו. אלגוריתם זה מבוסס על אלגוריתם האלפא בטא המוכר (האלגוריתם שומר מידע עבור כל קודקוד בעץ בשלושה משתנים - v המבטא את ערכו של הקודקוד, α ו β המבוססים על קודקודים שכבר בוקרו, ומבצע גיזום בקודקוד n כאשר (n) ≤ α(n)β) ועל אלגוריתם נוסף בשם \*-Minimax בו נעשה שימוש בשני ערכים נוספים LוU המשמשים לגיזום במשחקים עם אלמנט של מזל. בדומה לאלגוריתמים אלו, BAB מבצע חיפוש לעומק אך מגדיר סט חוקים חדש להגדרת ערכי המשתנים. בדומה לאלפא בטא – גיזום קודקוד n בעץ יתרחש כאשר יתקיים ϵ (n) ≤ α(n) + β. פלט האלגוריתם עבור קודקוד השורש הינו טווח - טווח 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 האלפא בטא ו.\*-Minimax על מנת לתאר משחקים עם אלמנט של מזל נעשה שימוש בשתי השיטות הללו לבניית העצים כאשר 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 מחישובכאשר 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.

בעזרת אלגוריתמי משחק שהם תת-אופטימליים מוגבלים, ניתן לבצע גיזומים רבים יותר מאלגורתמים אופטימליים. בכך אפשר להשיג אסטרטגית משחק בצורה מהירה יותר. אולם, דבר זה עלול לפגוע באיכות הפתרון החוזר מהאלגוריתם. לכן, קיים trade-off בין איכות הפתרון לזמן חישוב הפתרון.

מחקר זה מקביל למחקר אותו נבצע במסגרת הפרוייקט שלנו. בעוד במחקר זה עוסקים החוקרים בפתרונות תת-אופטימליים מוגבלים בעצי משחק לשני שחקנים – בפרוייקט נרצה להראות פתרון לאותה הבעיה עבור משחקים מרובי משתתפים.

סיכום מאמר 2 - פתרון אלגוריתמי למשחק מרובה שחקנים

An Algorithmic Solution of N-Person Games [Luckhardt et al., 1986]

המאמר מציג לראשונה את אלגוריתם maxN - אלגוריתם לפתרון משחקים עם סכום נקודות קבוע ולא קבוע, המתאים גם למשחקים בהם מתקיים שיתוף פעולה בין שחקנים וגם לכאלו בהם כל שחקן משחק עבור עצמו. ניתן להשתמש בmaxN גם במשחקים עם מזל או ללא מידע מושלם.

כפי שהוכח במאמר - האלגוריתם מוצא את נקודת שיווי המשקל (הנקודה בה אף שחקן לא ירוויח משינוי האסטרטגיה שלו) בהינתן עץ משחק מלא, בדומה לאלגוריתם המינימקס בשני שחקנים.

התאוריה בבסיס האלוגירתם דומה לזו שבאלגוריתם minimax ועל פיה כל שחקן בתורו ינסה לבצע את המהלך בעל התועלת הגבוהה ביותר עבורו. התוצאה של אלגוריתם זה תהיה סט של אסטרטגיות, אחת עבור כל אחד מהשחקנים, עם מידע על המהלך הטוב ביותר בכל מצב חוקי במשחק.

בדומה לאלגוריתמים אחרים לפתרון בעיות בבינה מלאכותית בכלל ובעיות חיפוש בעצי משחק בפרט, האלגוריתם מתאר את הבעיה כעץ ובדומה לעץ minimax בשני שחקנים, כל רמה בעץ מייצגת תור של שחקן מסויים. על מנת לייצג תועלת עבור n שחקנים, נייצג את התועלת בכל צומת בעץ על ידי ווקטור, כאשר הכניסה ה-i בווקטור מייצגת את התועלת עבור השחקן ה-i.

בפרקטיקה, maxN היא גישה רקורסיבית של חיפוש לעומק. בכל איטרציה קודקוד השחקן ה-i יקבל את ערכי הוקטור עם ערך ה-i המקסימלי מבין הוקטורים של צאצאיו. לדוגמה השחקן השלישי יבחר מבין הערכים (1,1,1) ו (4,4,0) בערך (1,1,1).

בדומה לאלגוריתם אלפא בטא בשני שחקנים, גיזומים מתאפשרים גם בmaxN.Simple Shallow Pruning גוזם כניסות לא רלוונטיות בווקטורי העלים. מאחר ובכל צומת נשווה רק על פי כניסה אחת בווקטור, ניתן לחשב את ערכי כל שאר הכניסות רק עבור הווקטור עם הכניסה המקסימלית (הצומת אותה האלגוריתם יבחר). Shallow Pruning חוסך אף יותר חישובים ומחשב רק כניסות נחוצות בווקטור בכל עומקי העץ. בDeep Pruning נוכל לגזור קודקודים תחת שורש מסויים לאחר שסיימנו לחקור את נכדי הבן הראשון שלו. לאחר שהבן הראשון יקבל את ערכו נוכל לגזור נכדים מבנים אחרים במידה והתועלת האפשרית מהם קטנה יותר.

החוקרים השוו בין מספר החישובים הנדרשים למציאת פתרון בשימוש בגיזומים במקרה הטוב ביותר והגרוע ביותר. תחת הנחות המקרה הטוב ביותר Deep Pruningהוא הגיזום היעיל ביותר ותחת הנחות המקרה הגרוע ביותר הוא הגיזום הכי פחות אפקטיבי. בShallow Pruning אמנם מתבצעים פחות חישובים מאשר ב Simple Shallow Pruning אבל נעשה שימוש נרחב יותר בפוינטרים.

חשיבות המאמר לפרוייקט שלנו היא גבוהה. המאמר מציג לראשונה את אלגוריתם maxN – אלגוריתם חיפוש מוכר למשחקים מרובי משתתפים, המוזכר רבות בספרות ונעשה בו שימוש נרחב בפרקטיקה. במאמר הקודם ראינו כיצד החוקרים השתמשו בעקרון המינימקס ובגיזומי אלפא בטא על מנת לתאר חיפוש אחר פתרון תת-אופטימלי מוגבל במשחקים לשני שחקנים. מאחר ואנו מחפשים אחר אלגוריתם דומה עבור משחקים מרובי משתתפים, ולאור הדמיון בין האלגוריתם לאלגוריתם הminimax, והדמיון בין הגיזומים שמבצע לאלו של אלפא בטא – סביר מאוד להניח שאלגוריתם זה יקח חלק בפתרון שנציע בפרוייקט שלנו.

סיכום מאמר 3 – **Best Reply Search** למשחק מרובה שחקנים

Best-Reply Search for Multi-Player Games [Schadd et al., 2011]

מאמר זה מציע אלגוריתם בשם Best-Reply Search (BRS) למשחקים דטרמיניסטיים מרובי שחקנים עם מידע מושלם, כאשר יחודיות האלגוריתם היא שרק המתמודד עם פעולת הנגד החזקה ביותר רשאי לעשות צעד. בניסויים עליהם נסביר בהמשך האלגוריתם הציג יכולות מרשימות ביחס לאלגוריתמים מוכרים אחרים.

אלגוריתמים שונים מניחים הנחות לא ריאליות לגבי המשחק. בmaxN מניחים שיוויון בין מהלכים בעלי ערך שווה, אך שינוי חוק שבירת השוויון יכול להשפיע משמעותית על התוצאה. כמו כן, האלגוריתם במידה רבה אופטימי מדי שכן לא מתייחס לכך שהאויבים ישתפו פעולה. בנוסף מאפשר מאט מאוד גיזומים ביחס לאלגורימים אחרים.

בParanoid ההנחה היא שכל השחקנים משחקים בקואליציה כנגד שחקן השורש. ההנחה אמנם מאפשרת רדוקציה לעץ משחק עם שני שחקנים ובכך ניתן לבצע גיזום אלפא בטא, אבל כאמור אינה מציאותית.

האלגוריתם The Comixer מניח שיש שיתופי פעולה נגד השחקן החזק ביותר בכל צומת. האלגוריתם יעבוד טוב אם יש שחקן מוביל, ויעילותו תפגע אם אין באמת שיתופי פעולה. החוקרים מסכמים שלאלגוריתם חסרונות רבים, ובהרבה מקרים שחקנים המשתמשים בו לא צוברים נקודות במשחק.

לBRS הנחה משלו – 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. בכל צומת MIN יופיעו כל אפשרויות המשחק של כל היריבים של השורש (שחקן הMAX) והאלגוריתם יבחר מבין המהלכים של כל השחקנים את המהלך שימזער את הניקוד של שחקן השורש. יתרונו של אלגוריתם BRS על פני maxN הוא שבכל בכל חישוב בעל עומק קבוע מפותחים יותר מהלכים של שחקן השורש מה שמאפשר תכנון ארוך טווח. בדומה לParanoid, ההנחה של אלגוריתם BRS מבצעת רדוקציה לעץ משחק לשני שחקנים ולכן ניתן לבצע גיזומי אלפא בטא ויש שימצאו הנחה זו הגיונית יותר מזו שמבצע Paranoid.

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 וmaxN. 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 מmaxN ומגיעים לעומקים דומים. BRS מנצח את Paranoid בשחמט, אך בפוקוס ורוליט הם כמעט שווים. במצב בו שלושת האלגוריתמים שיחקו יחד maxN 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 קודקודי MAX בזמן קצוב ובכך מתאפשר תכנון ארוך טווח.

בעוד במחקר זה עוסקים החוקרים בפתרון אופטימלי, בפרוייקט נרצה להראות פתרון תת אופטימלי מבוסס פתרון אופטימלי. אופציה אחת תהיה באמצעות BRS – בזכות הרדוקציה לעץ משחק לשני שחקנים נוכל לבחון זאת בקלות באמצעות הרעיון שהוצג במאמר הראשון.

סיכום מאמר 4 - חיפוש ערך מינימקס בשיטות **Best-First** ו **Depth-First** בפועל

Best-First and Depth-First Minimax Search in Practice [Plaat et al., 1996]

המאמר סוקר אלגוריתמים שונים לחיפוש ערך המינימקס בעץ משחק, ומבצע השוואה בין אלגוריתמים מסוג Depth-First לאלגוריתמים מסוג Best-First.

אלגוריתמי הDepth-First ובראשם אלגוריתם אלפא בטא 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 אלפא בטא, תחת הגדרות זיכרון זהות יעילותו אינה גבוהה יותר באופן ניכר וכבר היום קיימים אלגוריתמים מבוססי אלפא בטא דוגמת NegaScout המציגים ביצועים טובים יותר.

יתרה מכך, החוקרים מצאו שתחת אותו אלגוריתם גנרי ניתן לבצע שיטות חיפוש מוכרות אחרות דוגמת C\* וDUAL\* מאחר וגם הן מהוות מקרה פרטי שלו. המשותף לאלגוריתמים אלו הוא היותם אלגוריתמי Null window בהם החלון בו נעשה שימוש באלגוריתמי אלפא בטא 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 מנוסח בצורה כזו שמבצע קריאות אלפא בטא ומעמיק איטרטיבית עד שמגיע לערך המינימקס, ולמעשה ניתן לבצע באמצעותו חיפושים השקולים לחלוטין לחיפושים אחרים - יש להחליף רק את הערך שמתקבל באתחול. חיפוש שקול לSSS\* (AB-SSS\*) נשיג על ידי אתחול ב+∞, חיפוש שקול לDUAL\* (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 אלפא בטא המשמש מקור להשוואות רבות, אך אלגוריתם MTD(f) מציג את התוצאות הטובות ביותר בכל המדדים שנבדקו. החוקרים מצאו בנוסף שייעילותו של אלגוריתם הSSS\* ביחס לאלגוריתם Alpha-Beta אינה מובהקת, ש MTD(f)משיג תוצאות טובות ממנו ועל כן אינם חושבים שיש להמשיך במחקר בעניינו.

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

סיכום מאמר 5 – טכניקות חיפוש במשחקים מרובי שחקנים

An Overview of Search Techniques in Multi-Player Games [Schadd et al., 2011]

המאמר עוסק בבעיית חיפוש בעץ משחק מרובה משתתפים ובמרכזו השוואה בין ביצועי אלגוריתמי אלפא בטא שונים - maxN, Paranoid, Best Replay Search (BRS), ובהשוואה בניהם בשילוב שיטת חיפוש בשם Monte-Carlo Tree Search (MCST) על העצים שהאלגוריתמים השונים פורסים.

חיפוש MCST שונה מחיפוש מבוסס אלפא בטא 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 maxN, אך ניתן לבצע שימוש גם בעצים שפורשים Paranoid וBRS תוך שימוש בפונקציית הערכה שמטרתה להחליש את השחקן הנגדי (בשונה מ maxNשבכל 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 maxN, Paranoid וBRS. החוקרים מדדו גם את עומק החיפוש הממוצע של כל אחד מהאלגוריתמים. על פי התוצאות maxN 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 - maxN, Paranoid וBRS. על פי התוצאות MCTS-maxN 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 מ MCTS-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 - BRSו MCTS-maxN (במשחק מרובה משתתפים) ותוצאותיה משתנות - תחת משחקים שונים וקונפיגורציות שונות היתרון עובר צד, אך החוקרים מצאו כי MCTS-max מתחזק ככל שהזמן המוגדר לתור ארוך יותר.

לסיכום, מבין שלושת טכניקות החיפוש מבוססות אלפא בטא שהוצגו במאמר – BRS מציג את התוצאות הטובות ביותר (מנצח יותר משחקים) וכוחו בכך שחוקר עמוק יותר בכל תור ובכך מפתח יותר צמתי MAX בכל חיפוש. תחת שיטת MCTS אלגוריתם maxN משיג את התוצאות הטובות ביותר - הסבר לכך ניתן למצוא בכך שגיזומי אלפא בטא 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 מוגבל בעץ משחק מרובה שחקנים.

סיכום מאמר 6 – ניתוח טכניקות גיזום אלפא בטא

An analysis of alpha-beta pruning [Knuth et al., 1975]

המאמר סוקר את אלגוריתם אלפא בטא, מציג ניתוח כמותי לביצועי האלגוריתם ועוסק בחולשתו של האלגוריתם כאשר משחק כנגד שחקן לא אופטימלי.

המחקרים הראשונים בתחום עסקו באלגוריתמים נאיביים לפתרון משחקים ובהמשך פורסמו מאמרים העוסקים בגיזומים חד צדדיים. מאוחר יותר פורסמו מאמרים בהם מבצעים גיזומים דו צדדיים, אך באלו חסרו הוכחות מתמטיות לנכונות האלגוריתם ובנוסף התעלמו מכך שהאלגוריתם מחזיר פתרון אופטימלי במקרה הפרטי שבו אלפא ובטא מאותחלים למינוס אינסוף ואינסוף. במחקרים האחרונים הוצגה טכניקת האלפא בטא במלואה וחלקם עוסקים גם בחיתוכים לעומק. המשותף לרוב המאמרים הוא הקושי לנסח את הבעיה בצורה ברורה טרם הצגת הפתרונות לראשונה כאלגוריתם המשתמש בסימונים אלפא ובטא. לאורך הזמן פותחו מספר ורסיות לאלגוריתם האלפא בטא והחוקרים מוכיחים במאמר כי שלוש מהן שקולות - איטרטיבית, רקורסיבית וכזו המתייחסת לשחקן המינימום והמקסימום כשחקנים זהים.

במקרים רבים הזמן הדרוש לחישוב ערך המינימקס אינו פיזיבילי גם לאחר ביצוע גיזומי אלפא בטא ועל כן החוקרים מציעים שלוש דרכים להתמודד עם בעיה זו. השיטה הראשונה היא לקבוע עומק מקסימלי עד אליו נחפש ופונקציה יוריסטית לחישוב ערך המצבים בעומק זה על אף שאינם מצבים סופיים. בשיטה השנייה נקבע סבירות לכל מהלך וערך סף על פיהם נבצע גיזומים כאשר הסבירות לרצף של פעולות יורד מערך הסף. השיטה השלישית היא ייעול של גיזום אלפא בטא על ידי שימוש באלגוריתם בגבולות הקרובים לערך המינימקס האמיתי, במקום לאתחל בין אינסוף למינוס אינסוף.

בהמשך מבצעים החוקרים ניתוח כמותי וקובעים חסם תחתון למספר המינימלי של קודקודים טרמינליים שיש לפתח על ידי אלגוריתם האלפא בטא ו/או על ידי כל אלגוריתם אחר שפותר את אותה הבעיה הכללית, על ידי שימוש בהנחה כי בכל קודקוד נפתח את המהלך הטוב ביותר ראשון.

כמו כן החוקרים מנסחים תנאים כללים לחסם עליון לחיפושים אלו וקובעים אחד כזה בצורה פשוטה יחסית (ביחס למחקרים קודמים) על ידי ניתוח המקרה הגרוע עבור אלגוריתם בעל גבול בודד (על אף שלאלפא בטא גבול עליון ותחתון). הנחה זו נכונה מאחר והחיתוכים שמתבצעים בשיטה זו מוכלים באלו שמבצע אלגוריתם האלפא בטא ומאחר ולא מבוצעים חיתוכים לעומק.

בסיכום המאמר מציינים החוקרים כי חוזקו של אלגוריתם אלפא בטא הוא בכך שהנחות המקרה הגרוע אינן מתקיימות, אך גם תחת הנחות אלו האלגוריתם עדיין יעיל ביחס לחיפוש נאיבי. כמו כן מראים החוקרים כי האלגוריתם יעיל יותר במצב בו קיימת תלות בין מהלכים אל מול אי תלות מוחלטת.

הכרת טכניקות הגיזום השונות הכרחית במסגרת הפרוייקט אותו נציג. תחת הנחות שונות נוכל לבצע גיזומים שונים ובכך להקל על החיפוש במשחקים מרובי שחקנים. פתרון איכותי לחיפוש תת-אופטימלי מוגבל יהיה כזה שיאפשר גיזומים רבים יותר גם מבלי להגדיל את הגבול.

סיכום מאמר 7 – שילוב אסטרטגיות חיפוש עבור משחקים מרובי שחקנים

Mixing Search Strategies for Multi-Player Games [Zuckerman et al., 2009]

המאמר עוסק בפיתוח אלגוריתם משולב לבעיית חיפוש בעץ משחק מרובה משתתפים בעל מנצח בודד וללא מדרוג בין המפסידים. האלגוריתם בנוי ממספר אלגוריתמים שונים מוכרים לבעיה זו 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 משלוש האסטרטגיות הבאות-

MaxN- ערכו של כל קודקוד בעץ המשחק מיוצג באמצעות וקטור של ערכים באורך מספר המשתתפים כך שכל כניסה בוקטור מייצגת את ערך המצב הנתון עבור המשתתף, ומניח שכל משתתף בתורו מנסה למקסם את הניקוד של עצמו.

Paranoid- ערכו של כל קודקוד בעץ מיוצג על ידי ניקוד שחקן השורש המניח שכל שחקן אחר ינסה לפגוע בניקוד שלו בצורה מקסימלית (קואליציה נגד שחקן השורש). הנחה זו מאפשרת רדוקציה לעץ משחק של שני שחקנים ובכך ניתן לבצע חיתוכים עמוקים.

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 האלגוריתם שאר השחקנים בתורם ינסו לחזק עצמם כפי שהיו פועלים באסטרטגיית MaxN.

האלגוריתם המשולב מקבל ערכי trashold- To,Td, 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 הפרנואיד, אם השחקן המוביל אינו שחקן השורש וגם הפער שחושב גדול מTo השחקן יבחר באסטרטגיית האופנסיב, אחרת יפעל על פי אסטרטגיית MaxN. 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 MaxN.

המאמר בוחן את יכולות האלגוריתם המשולב אל מול MaxN 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 השחקים שמתמודדים מולו. ערכו של מדד זה נקבע על ידי חישוב מספר המצבים במשחק שמוגדרים כInfluentialStates- – 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 מוגבל, 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.

סיכום מאמר 8 – טכניקות גיזום עבור משחקים מרובי שחקנים

On Pruning Techniques for Multi-Player Games [Sturtevant et al., 2000]

המאמר עוסק בטכניקות גזירה במשחקים רבי משתתפים - לדעת כותביו טכניקות הגזירה הקיימות באלגוריתמים MaxN ו branch-and-boundאינן יעילות מספיק ביחס ליכולות הגזירה של אלגוריתם אלפא בטא בשני שחקנים. המאמר משווה בין שני פתרונות לבעיה זו, האחד הוא אלגוריתם היברידי המשלב את שתי השיטות שהוזכרו, והשני הוא ביצוע רדוקציה מעץ משחק רב-משתתפים לעץ לשני משתתפים על ידי אלגוריתם Paranoid.

טכניקות הגזירה המתאפשרות באלגוריתם MaxN הן גזירה מיידית (immediate pruning) – גיזרה טריוויאלית על פיה נגזור כאשר השחקן הנוכחי מקבל את מספר הנקודות המקסימלי האפשרי, וגזירה רדודה (shallow pruning) – גזירת קודקוד המתאפשרת בזכות מידע מקודקוד "דוד" שפותח קודם לכן. המאמר מגדיר באופן גנרי את התנאים הנדרשים לביצוע גזירה זו. גזירה עמוקה(Deep pruning) - גזירה של קודקוד כלשהו על סמך מידע מאב קדמון שלו, אינה מתאפשרת באלגוריתם MaxN 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.

אפשרות נוספת להגדלת מספר הגזירות המבוצעות במהלך החיפוש היא ביצוע רדוקציה לעץ משחק של שני שחקנים תחת ההנחה של אלגוריתם Paranoid. כעת, כאשר מדובר בעץ משחק לשני שחקנים ניתן יהיה לבצע גם גזירות עמוקות, אך צורת המשחק לא תהייה אופטמלית בגלל ההנחה השגויה שמבצע האלגוריתם (קואליציה של כלל השחקנים כנגד השורש).

טכניקת גזירה נוספת היא Depth-First Branch and Bound (DFBnB) הדורשת פונקציה יוריסטית מונוטונית, דבר שקל למצוא במשחקי קלפים רבים. על פי טכניקה זו הפונקציה היוריסטית משערת את הגבול התחתון והעליון של הניקוד בכל סיבוב. כמו אלפא בטא, יש רמות עומק שונות של גזירה המתאפשרות, אך גם במקרה זה לא ניתן לבצע גזירה עמוקה.

המאמר לבסוף מציג אלגוריתם חדש בשם Alpha-beta Branch-and-bound pruning (ABBnB). על פי האלגוריתם, לכל קודקוד יש את גבולות האלפא בטא 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 (ב חיתוך שתי הגבולות) מתוך כלל הנקודות האפשריות בסך הכל לכלל השחקנים, ולבצע יותר חיתוכים מאשר יכלו כל אחד מהאלגוריתמים בנפרד – אך גם במקרה זה לא מתאפשרת גזירה עמוקה.

התוצאות במאמר מבוססות על המשחקיםHearts ו-Sergant Major. בוצעה השוואה בין האלגוריתמים DFBnB, ABBnB, Paranoid וShallow pruning, כאשר המדד הוא מספר הקודקודים שפותחו בעץ.

על פי התוצאות אלגוריתמי הParanoid (עם ובלי היוריסטיקה) פיתחו משמעותית פחות קודקודים, אחריהם ABBnB, Shallow pruning ולבסוף DFBnB. כצפוי, התוצאות מראות כי האלגוריתם המשולב (DFBnB) אכן מפתח פחות קודקודים מכל אחת מהשיטות בנפרד (Shallow pruning ו (ABBnBמאחר ומאפשר את הגזירות שהן מבצעות וגזירות נוספות המתאפשרות באמצעות המידע המשותף משניהן.

המאמר מציג טכניקת גזירה משולבת למשחקים רבי משתתפים שביכולתה להקטין את מספר הצמתים שמפותחים במהלך חיפוש בעץ. שימוש באלגוריתם זה בפתרון שנציע בפרוייקט יכול להיטיב עם תוצאות המחקר שנבצע.

סיכום מאמר 9 - אלגוריתמי חיפוש בשיטת **Proof-Number**

Proof-Number Search and its Variants [Jaap Van Den Herik et al., 2008]

המאמר עוסק באלגוריתם Proof-Number Search, חיפוש מסוג Best-First, בווריאציות שונות שלו ומבצע השוואה בינהן לבין אלגוריתם אלפא בטא הנמצא בשימוש ברוב המערכות המבצעות חיפוש בעץ משחק, תוך ביקורת על יכולות אלגוריתם האלפא בטא בEnd-Game Positions.

אלגוריתמים מסוג Best-First ידועים כאלגוריתמים הצורכים זיכרון רב מאחר ונדרש לשמור את כל עץ המשחק. על כן, מוצגות במאמר וריאציות שונות (כולן מוזכרות במאמרים קודמים למעט אחת – PDS-PN המוצאת פה לראשונה) להתמודדות עם בעיה זו תוך השוואה בניהן ובין אלגוריתם אלפא בטא.

הבעיה עליה קבוצת אלגוריתמי הPN באים להתגבר היא חולשתם של אלגוריתמים מבוססי אלפא בטא בשלב הEndGame. פתרון נוסף לבעיה זו הוא שימוש במאגרי מידע – אך לא בכל המשחקים האופציה פיזיבילית. על כן הוצע אלגוריתם הPN המקורי - אלגוריתם best-first מבוסס היוריסטיקה המעדיף פיתוח של תתי עצים רזים על פני סבוכים, אך כאמור אלגוריתם זה כ best-first נתקל בקשיי זיכרון בשל הצורך לשמור את כל העץ. על כן פותחו האלגוריתמים שמטרתם להתגבר על בעיה זו.

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.

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

- 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, מבצע 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 הdisproof number.

– אלגוריתם זה מתגבר על הבעיה שהוצגה ב , 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.

החוקרים ביצעו מספר מספר ניסויים המשווים בין האלגוריתמים השונים ממשפחת PN, כאשר אלגוריתםמשתתף גם הוא לצורך פרופורציית התוצאות בהיותו אלגוריתם מוכר. ראשית בין האלגוריתמים ומסקנותיה העיקריות הן ש פותרים משמעותית יותר יותר מצבים מהאחרים, שבונה עצים גדולים משמעותית משאר האלגוריתמים ואחריו בגודל נמצאים שהעלות של יכולתם לפתור יותר מצבים היא עצים רחבים יותר ועל כן מתאימים לבעיות קשות יותר מאשר PN. בהמשך נחקרו לעומק ההבדלים בין , והושג יתרון מהירות של - על פי התוצאות על אף ש מפתח פי 2.6 יותר קודקודים, הוא עדיין משיג ביצועים טובים פי 3 מPDS מאחר והשני עובד עם delayed evaluation, ונצפה כי לוקח לו פי 7-8 יותר זמן לפתח קודקוד. החוקרים סיכמו כי תחת ביצועים אלו קיימת עדיפות ל בשימוש תחרותי תוך הסתייגות מפתרון בעיות קשות מאוד בגלל מגבלות זיכרון שקיימות באלגוריתם.

בהמשך המאמר מוצג לראשונה אלגוריתם PDS-PN, הנועד לשלב את היתרונות של PN2 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). קל 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.

החוקרים ביצעו מספר השוואות בין אלגוריתם לאלגוריתמים על בסיס מספר הendpoint positions שכל אלגוריתם פותר, מספר הקודקודים הכללי שעליו לפתח וזמן שימוש ב.CPU כאשר נבדקו האלגוריתמים על אותו הסט של מצבים ונמדד אחוז המצבים שכל אלגוריתם פתר, נמצא כי לא פיתח שום מצב שאף אלגוריתם אחר לא פיתח, ומציג תוצאות טובות פחות בכ18 אחוז מהאלגוריתמים האחרים, מפתח משמעותית יותר קודקודים לאורך החיפוש ודורש גם משמעותית יותר זמן עיבוד. בהשוואה בה נלקחים בחשבון רק מצבים שכל האלגוריתמים הצליחו לפתור - PDS מפתח באופן ניכר פחות קודקודים מכולם בדרך לפתרון. באופן כללי ניתן לראות כי PDS-PN משיג תוצאות טובות מ, וכי מספר המצבים שהוא פותר קרוב לאלו שמשיגים .

כאשר החוקרים הקטינו את מספר הקודקודים המקסימלי אותו האלגוריתמים מרחיבים במטרה לבדוק האם מגבלת הזיכרון אכן פוגעת ביעילות האלגוריתמים - PDS-PN פותר יותר מצבים מ אך לצורך כך פיתח יותר קודקודים וצרך יותר זיכרון. הניסויים מראים כי מפתח פי 2.6 יותר קודקודים מPDS מאחר ועבור בעיות קשות הוא מפתח תתי עצים גדולים, ועדיין מהיר ממנו פי 4 מבחינת ביצועי מעבד. PDS-PN מפתח פי 3.7 קודקודים מPDS ועדיין מהיר פי 3 מPDS מבחינת ביצועי מעבד. בסך הכל משיג תוצאות טובות יותר מ PDS-PN וPDS-PN יעיל יותר מPDS. בהשוואה שביצעו החוקרים בין PDS-PN ל עבור בעיות קשות, כלומר עצים בעלי עומק גדול יותר, פותר פחות מצבים מPDS-PN ומסקנת החוקרים היא שהפער נובע מגודל זיכרון העבודה ש דורש. תחת תוצאות אלו המלצתם היא כי PDS-PN יעיל יותר במצבים סופיים בבעיות קשות. החוקרים הבחינו בצורך הרב של בזיכרון עבודה ואת מספר המצבים הנמוך שהוא פותר בהשוואה לPDS-PN וגם לעצמו ככל שגודל זיכרון העבודה קטן, בעוד PDS-PN נשאר יציב בכמות המצבים אותם הוא פותר. לבסוף החוקרים השוו בין Df-pn לPDS והשיגו כי Df-pn מציג יעילות גבוהה יותר – חיפוש מהיר פי 4 מPDS, בעוד בהשוואות קודמות PDS-PN הציג מהירות גדולה פי 3 מPDS. החוקרים ציינו כי נעשה שימוש בMobility וDeleting (dis)proved trees המגדילים עת מהירות האלגוריתמים ו ומשפר את תוצאותיהם בפתרון מצבי endgame.

מסקנות החוקרים הן שאלגוריתמי הPN השונים השיגו תוצאות טובות מ במדדים שנבדקו, שאלגוריתם PN חלש בבעיות קשות בעקבות צרכי זיכרון גבוהים ושבאופן כללי PDS ו פותרים מגוון בעיות גדול יותר מPN. כמו כן קבעו החוקרים כי PDS-PN מציג יכולות טובות יותר מ הendgame, כי הוא מהיר כמעט כמו תחת ההגדרות המתאימות, פותר יותר מצבים קשים מ ויעיל יותר מPN ואף מ 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 מPDS ו כendgame solver עבור משחקים קשים.