דצמבר 2012

בדיקות פונקציונליות של תוכנה (black box testing) ע"י שימוש באלגוריתם גנטי

עבודה מסכמת תואר שני במדעי המחשב

אוניברסיטה הפתוחה

מגיש: גולן מור 23582695

מנחה: ד"ר מיריי אביגל

תוכן עניינים

תוכן עניינים

רשימת איורים

תקציר

במהלך ביצוע בדיקות קופסא שחורה (Black Box Testing) למערכת יש צורך במגוון רחב ומגוון של מתארי בדיקה לצורך רכישת האמון במערכת ע"י מציאת התקלות החבויות בה. הגדרת מתארי הבדיקות הינן תורה בפני עצמה שכן מגוון המתארים הוא עצום.

עבודה זו בוחנת את השימוש באלגוריתמים גנטיים לצורך ייעול שלב הגדרת מתארי הבדיקות וביצועם כחלק מהבדיקה. לצורך כך נבחנו מספר מאמרים בנושא, הן מאמרים שעוסקים בסוגים שונים של אלגוריתמים גנטיים והן מאמרים שעושים שימוש באלגוריתמים אלה לצורך ביצוע בדיקות תוכנה. סוג בדיקות התוכנה אליהם אני בודק את ההתאמה של אלגוריתמים גנטיים הינם בדיקות נסיגה (regression test) שעליהם יורחב במהלך העבודה.

בעבודה זו יוצגו בפירוט 2 אלגוריתמים גנטיים בעלי פרמטרים משתנים. אלגוריתם הראשון GAVaPS הינו בעל אוכלוסייה משתנה. והאלגוריתם השני FAexGA משלב את עקרונות ה-Fuzzy Logic לצורך קביעה של הסתברות הזיווג (cross-over) באלגוריתם. במהלך העבודה ביצעתי התאמות לשני אלגוריתמים אלה לטובת בדיקות נסיגה. אלגוריתם GAVaPS_R הינו שיפור והתאמת האלגוריתם הראשון, ו- FAexGA_R הינו השיפור וההתאמה של השני.

מעבר להצגה והסבר מפורט של כל אלגוריתם בוצעה בעבודה זו מימוש של האלגוריתמים בקוד. ובחינת השימוש בהם על "פונקציה נבדקת" (system under test) חיצונית, זוהי פונקציה שכתבתי במיוחד על מנת שאוכל לבחון את השימוש של האלגוריתמים בביצוע בדיקת נסיגה על פונקציה זו. בסוף מוצגים המסקנות שעיקרן: השימוש באלגוריתמים גנטיים בעלי גודל אוכלוסייה משתנה מייעל את תהליך בדיקת הנסיגה (regression test)".

1. מבוא:

בדיקות "קופסא שחורה" מבוצעות על פי האפיון הפונקציונלי של המערכת הנבדקת. הם למעשה ההשפעה של הקלט למערכת על הפלט שלה. בבדיקות אלה אין משמעות לדרך שבהם מבוצעת הפעולה ע"י המערכת, אלא רק לערך המוצא שלה על פי הקלט.

מספר האפשרויות לביצוע בדיקת קופסא שחורה (black box test) הוא גדול מאוד. הסיבה לכך היא מספר הצירופים (combinations) של כל משתני הכניסה. אם כך תוכנית לא טריוויאלית שמכילה מספר משתנים יוצרת פריסה של מרחב עצום של אפשרויות בדיקה.

משאבי הבודקים הם תמיד מוגבלים, ועל כן הם חייבים לבחור את מתארי הבדיקה שלהם בקפידה בשני היבטים. הראשון הוא בחירה של מתארי בדיקה שיש להם הסתברות גדולה למצוא תקלות שטרם התגלו. והשני הוא מהירות ביצוע הבדיקה כדי לעבור למרכיבי בדיקה ו/או מערכות אחרות.

אלגוריתמים גנטיים הינם משפחה של אלגוריתמים היונקים את מהותם מתורת האבולוציה, לאור מחקרים רבים הוכח כי אלגוריתם מסוג זה הינם אלגוריתם חיפוש יעיל במרחבי חיפוש עצומים.

החיבור בין אלגוריתם חיפוש יעיל במרחבי חיפוש גדולים ומספר האפשרויות העצום שניתן לבצע בדיקת מערכת, יכול להועיל לצורך בניית מתארי בדיקות המתאימים לאופי הבדיקה והיכולים למצוא את התקלות במערכת בצורה יעילה יותר. האלגוריתם הגנטי מתחיל בכך שהוא בוחר באופן ראנדומלי מתאר בדיקה. מתאר בדיקה זה הינו כרומוזום המייצג וקטור של משתני כניסה. ע"י שימוש בפונקציית התאמה (fitness function) אנו נותנים ציון ליכולתו של מתאר הבדיקה לגלות תקלה. על ידי שימוש במחזורי אבולוציה אנו מעדיפים כרומוזומים "טובים" ובכך בסוף התהליך מוצאים מתארי בדיקה המתאימים לנו.

בעבודה זו אתמקד בשימוש באלגוריתמים גנטיים לצורך ייצור מתארי בדיקות אפקטיביים לבודקי התוכנה, מתארים אשר יכולים, בהתאם לאמצעים המוגבלים של בודקי התוכנה, להעלות את הסיכוי למציאת תקלות במידה ויש במערכת הנבדקת. את הבחינה אני אבצע לגבי התאמת האלגוריתמים לתת משפחה של בדיקות פונקציונליות והיא משפחה בדיקות הנסיגה (regression test) , בדיקות אלו בודקות תאימות של תוכנה לגרסה קודמת לאחר העלאה לגרסה חדשה.

משפחת האלגוריתמים הגנטי שנבדקו בעבודה זו כוללים: אלגוריתם גנטי קאנוני, אלגוריתם גנטי בעל אוכלוסייה משתנה (GAVaPS), ואלגוריתם גנטי בעל אוכלוסייה משתנה המשתמש בהסתברות שחלוף (cross-over) משתנה על פי שיטת ה- Fuzzy Logic.

במהלך העבודה אתן סקירה על עולם בדיקות התוכנה, ואחבר אותו לעולם של האלגוריתמים הגנטיים. כמו כן אסביר בהרחבה על משפחות האלגוריתמים הגנטיים השונים, ודוגמאות לשימוש באלגוריתמים אלו לבדיקת תוכנה.

אתמקד בשני אלגוריתמים המשלבים פרמטרים משתנים – GAVaPS ו- FAexGA. אבצע בהם שיפור והתאמה לבדיקות נסיגה. האלגוריתמים המשופרים ייקראו GAVaPS_R ו- FAexGA_R. את האלגוריתמים הללו אממש בעזרת סביבת visual studio ושימוש בסביבת

הפיתוח #C. אממש פונקצייה אשר תשתמש כ"מערכת הנבדקת" ועליה אריץ את כלל המימושים של האלגוריתמים. את התוצאות אציג בסוף העבודה כולל כלל התובנות.

בסיום העבודה אני אראה כי שימוש באלגוריתמים גנטיים בכלל ואלגוריתמים בעלי פרמטרים משתנים הינם בעלי התאמה גבוהה מאוד, ליצירת מתארי בדיקה מתאימים ויעילים לצורך בדיקות קופסא שחורה בכלל ובדיקות נסיגה (regression) בפרט.

1.1. מטרות העבודה

לעבודת זו שתי מטרות עיקריות:

1.1.1. הכרות עם בדיקות תוכנה ואלגוריתמים גנטיים

הכרות עם עולם בדיקות התוכנה- סוגי בדיקות ושיטות , הבנת הקושי בייצור מתארי בדיקה מספקים. כל זאת לצד הכרות מעמיקה עם אלגוריתמים גנטיים בתצורות שונות אשר יכולים לשמש כשיטה טובה לצורך ייצור מתארי בדיקה

1.1.2. שימוש באלגוריתמים גנטיים בבדיקות נסיגה

הכרות בלתי אמצעית עם אלגוריתמים גנטיים בעלי פרמטרים משתנים. מימוש של חלק מאלגוריתמים גנטיים ובדיקה מעשית שלהם במתארי בדיקות נסיגה.

המימוש של האלגוריתמים הגנטיים והשימוש שלהם בבדיקות נסיגה יאפשרו לי להכיר אותם באופן מעמיק, לבצע בהם התאמות באם צריך, ולכוונן אותם בצורה הטובה ביותר לצורך המטרה – ביצוע בדיקות נסיגה.

1.2. סקר ספרות

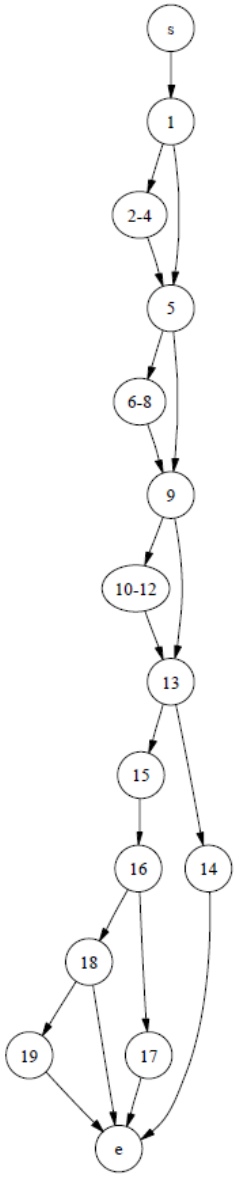
שיטות חיפוש מטה יוריסטיות ובינם חיפוש ע"י אלגוריתם גנטי, משמשות לצורך ייצור אוטומטי של מתארי בדיקה. השימוש נעשה בכמה אזורים של בדיקות תוכנה[18].

א. בדיקות כיסוי לבחינת מבנה של תוכנה, כחלק מבדיקות קופסא לבנה או כחלק מבדיקת תקינות מבנה.
ב. הרצה פונקציונלית של תוכנה בהתאם למפרט שלה.
ג. בדיקה אוטומטית של תקינות תכונות של תוכנה, כגון הרצה של כניסות לא תקינות , או הצבה לא תקינה של נתונים שיכולים להפיל את התוכנה.
ד. בדיקה של תכונות לא פונקציונליות כמו בחינת זמן ההרצה הארוך ביותר.

בספרות ניתן למצוא התייחסויות רבות לשימוש באלגוריתם גנטי לצורך המופיע בסעיף הראשון קרי, בדיקות כיסוי של תוכנה. רבות מהבדיקות מתייחסות לבדיקת גרף זרימת השליטה (control flow graph – CFG), זהו גרף המייצג את הזרימה בין כל שורת קוד אחת לשנייה.

לצורך הבחינה משתמשים בתוכנת בדיקה אחידה (benchmark) של בדיקת משולש. תוכנת הבדיקה מקבלת 3 ערכים של צלעות ומחזירה תשובה באם זהו משולש, ואם כן מאיזה סוג. את תוכנת המשולש וכן את ה- CFG שלה ניתן לראות באיור 1.

CFG Node	
s	int tri_type(int a, int b, int c)
	{
	int type;
1	if (a > b)
2-4	{ int t = a; a = b; b = t; }
5	if (a > c)
6-8	{ int t = a; a = c; c = t; }
9	if (b > c)
10-12	{ int t = b; b = c; c = t; }
13	if (a + b <= c)
	{
14	type = NOT_A_TRIANGLE;
	}
	else
	{
15	type = SCALENE;
16	if (a == b && b == c)
	{
17	type = EQUILATERAL;
	}
18	else if (a == b || b == c)
	{
19	type = ISOSCELES;
	}
	}
e	return type;
	}



איור 1- תוכנית בדיקת משולש וה- CFG שלה [18]

מירב העבודות בוחנות שימוש באלגוריתם גנטי על תוכנת הבדיקה האחודה הזו כמו העבודה של Srivastava ו- Tai-hoon [13] , שהריצו אלגוריתם גנטי על CFG משוקלל כדי למצוא מתארי בדיקה אשר בודקים כיסוי מירבי של הגרף, ניתן למצוא עוד מספר רב של עבודות בתחום[15,16,17].

בתחום הבדיקות הפונקציונליות – בדיקת "קופסא שחורה", נעשתה עבודה פחותה מאשר בתחום הקודם. ככלל רוב העבודה שניתן למצוא מתרכזת בבדיקה של מרכיבים הנדסיים. עבודה לדוגמא היא בדיקה של מערכת לחניה אוטומטית וכן מערכת עזר לבלימה של חברת Diamler's Mercedes. העבודה נעשתה [14] ע"י Buhler and Wegener, אשר הוכיחו את יעליות השימוש באלגוריתם גנטי לצורך בדיקת "קופסא שחורה" של המערכות על פני גישה ראנדומלית וידנית.

העבודה המסכמת היא בתחום ה"הבדיקה השחורה" והיא מתבססת על מספר מאמרים בתחום כאשר המאמר העיקרי הוא [7].

1.3.מבנה העבודה

בתחילת העבודה בסעיף 2 אני אתן סקירה בנושא בדיקות תוכנה, לאחר מכן בסעיף 3 אתן סקירה כללית על אלגוריתמים גנטיים.

בסעיף 4 אציג עבודה בנושא בדיקות אבולוציוניות של סוכני תוכנה אוטונומיים, שזו היא עבודה נוספת הקשורה לבדיקות "קופסא שחורה". בסעיף 5 תוצג בכלליות העבודה לביצוע בדיקות נסיגה באמצעות אלגוריתמים גנטיים, ואז בסעיפים הבאים תבוצע הרחבה לנושא. בסעיף 6 יוצג אלגוריתם גנטי בעל אוכלוסייה בגודל משתנה בשם GAVaPS [8], בסעיף 7 אתייחס לאלגוריתם גנטי נוסף בעל פרמטרים משתנים והוא אלגוריתם המשלב גם יכולות fuzzy Logic בהגדרת הסתברות הזיווג (cross-over) , אלגוריתם זה נקרא – FAexGA[6], ובסעיף 8 אציג בקצרה עבודה קודמת[7] שדנה בשימוש ב- FAexGA לטובת בדיקות קופסא שחורה.

לאחר מכן בסעיף 9 תוצג העבודה לבחינת כלל האלגוריתמים שהוצגו לטובת מימוש בדיקת נסיגה, הסעיף יציג 2 אלגוריתמים חדשים המותאמים לביצוע בדיקות נסיגה, יציג מימוש של אלגוריתמים אלה לצד האלגוריתם הקאנוני, ואת תוצאות העבודה. סעיף 10 הינו הסעיף המסכם את העבודה. בסוף העבודה ניתן למצוא בנספח הסבר קצר על מערכות מבוססות Fuzzy Logic.

2. בדיקות תוכנה - סקירה

האמירה הרווחת היא ש – 50 אחוזים הזמן המוקדש לפיתוח תוכנה, והזמן שנותר לצורך בדיקות התוכנה[1].

אנו משקיעים רבות בבדיקות במהלך הפיתוח, הבדיקות עולות הרבה כסף ולוקחות זמן משמעותי מהפרויקט. ובניגוד למוצר אחר שאנו משקיעים בו זמן וכסף, במקרה של בדיקות תוכנה קשה לנו לענות על השאלות הבאות: בדיקות תוכנה מהם? מה נכלל בין פעולות הבדיקה? ומה בין הבדיקה עצמה לפעולת הפיתוח?

המושג testing מוכר לנו עוד מהיותנו תלמידים. אנו רגילים להבחן (test) אנו מכירים שיש צורך בשאלת שאלות על מנת להבין את מידת ההבנה שלנו בנושא מסויים, מצד שני בדיקת תוכנה (שוב testing) מצריכה סוג אחר לחלוטין של שאלות ולאו דווקא זהות (אנו לא בודקים את מידת ההבנה של התוכנה...). על כן אנו צרכים להגדיר את המושג testing בבדיקות תוכנה בצורה חדשה ומייחדת את עולם בדיקות התוכנה.

2.1. היסטוריה של הגדרות

בעוד בעולם ההנדסה היו רגילים להבין את המושג בדיקה כרצף הפעולות אשר אני עושה כדי להבין שהמערכת שבניתי עושה את מה שהיא אמורה לעשות, בעולם התוכנה הנושאים הללו התחילו להיות מוגדרים רק בסוף שנות ה-50. עד אז המושג בדיקות תוכנה היה שקול לחלוטין למושג Debugging. בין ההגדרות הראשונות שניתן למצוא לבדיקות תוכנה ההגדרה של Hetzel משנת 1973 היא:" בדיקת תוכנה היא התהליך של רכישת אמון בתוכנה שהיא עושה את מה שהיא אמורה לעשות"[2].

החל מתקופה זו התחילו לדון רבות בכנסים ובקבוצות מחקר על המושגים "איכות תוכנה", "אמינות תוכנה", "הנדסת תוכנה". בתקופה יחסית קצרה יצאו מספר ספרי בנושא בדיקות תוכנה כמו: "The art of software testing techniques" [1] של Glen Myers, ו "software tesing techniques" [2] של Boris Beizer.

Glen Myers, טען שצריך להתחיל מההנחה שישנם באגים בתוכנה הנבדקת (להבדיל מהגישה של יצירת אמון בתוכנה). כך הגדיר מחדש את המושג בדיקות תוכנה כ"בדיקות תוכנה הם התהליך בו מריצים את התוכנה לצורך מציאת באגים"[1]. ההגדרה של Myers מצירה את המושג Testing, שכן בדיקות תוכנה הם הרבה יותר מרק ההרצה שלהם לטובת מציאת באגים. אם מקבלים את הגישה של Myers מתחילים לבדוק רק לאחר שמסיימים את כתיבת הקוד. ובנוסף לכך הרעיון של "בדיקות = מציאת תקלות" הינו בעייתי, שכן כמו שאנו בוחנים תלמיד על ידיעותיו ולא על מה שהוא לא יודע, בבדיקות תוכנה אנו צרכים לבדוק גם מה נכון, איך המערכת עובדת, ולא רק מה לא טוב בה.

על פי Bill Hetzel [2] לבדיקות התוכנה ישנם מספר תפקידים:

א. בדיקת התוכנה אל מול המפרט
ב. מציאת באגים – מה לא עובד בתוכנה
ג. הגדרת חווית המשתמש
ד. אשרור תקינות התוכנה לשימוש
ה. צבירת אמון בתוכנה שהיא מבצעת את מה שהיא אמורה לעשות
ו. הוכחה שהתוכנה מתפקדת כראוי
ז. הדגמה שאין יותר באגים
ח. הבנת מגבלות התוכנה
ט. הכרות עם מה שהמערכת לא יודעת לבצע
י. הערכת ביצועי המערכת
יא. בחינת התיעוד
יב. הוכחה לבודק שמשימתו הושלמה

ומכך ההגדרה לבדיקות תוכנה על פיו הם: **"בדיקת תוכנה היא הפעילות המכוונת להערכת הפונקציונליות והביצועים של תוכנה כדי לבחון האם היא עומדת בדרישות שלה".**

2.2. הקשר בין בדיקות תוכנה לאיכות תוכנה

מהי איכות תוכנה? איכות לפי הגדרתה בעולם התוכנה היא "עמידה בדרישות".

אם כך נראה שהקשר בי איכות תוכנה לבדיקות תוכנה היא: "מכיוון שאיכות תוכנה היא מופשטת, מטרת בדיקת התוכנה היא להמחיש את איכות התוכנה, והיא למעשה המדידה של איכות התוכנה".[2]

ניתן להתבונן על 3 מימדים של איכות תוכנה: פונקציונליות (Functionality), הנדסית (Engineering), כושר הסתגלות (Adaptability) .

כל ממד מתפרק למרכיבים קטנים, כך שלמעשה 3 הממדים, על החלוקה הפנימית שלהם, מהווים למעשה את ההגדרה המלאה יותר של איכות תוכנה, ועונים על השאלה האם המערכת עומדת במה שהיא אמורה לעשות.

דוגמא לפירוק שלושת הממדים בצורה פרקטית:

פונקציונליות: תקינות, מהימנות, שימושיות, אמינות

הנדסית: יעילות, בדיקתיות (ניתן בקלות לבדיקה), תיעוד, מבנה

כושר הסתגלות: גמישות, ניתן לשימוש חוזר, תחזוקה

2.3. איך מבצעים בדיקות בארגון ?

קיימות שלוש רמות מקובלות של בדיקות תוכנה:

א. בדיקת יחידה (Unit Testing) – בדיקות של קטעי קוד, בעודם נכתבים ע"י המתכנת.
ב. בדיקת מערכת (System Testing) – בדיקות אינטגרציה בין חלקי התוכנה השונים
ג. בדיקת קבלה (Acceptance testing)– בדיקות קבלה של המערכת , הבודקות את המענה שלה לצורך עליה היא נכתבה.

בהמשך נפרט את רמות הבדיקה השונות

רוב הארגונים מתייחסים לשלושה המרכיבים כמרכיבים שונים, ועל כן מפנים משאבים שונים לצורך ביצועם. המתכנתים בודקים בדיקות "unit testing" . קבוצה של מתכנתים אחרת בודקים את המערכת ע"י ביצוע "system testing" , בסוף השרשרת נבדקת המערכת בסיומה ע"י בדיקות קבלה " acceptance test" כאשר אותם מבצע גוף נפרד מהמתכנתים.

2.3.1.בדיקת יחידה (Unit Testing)

הבחירה מה לבדוק ואיך היא בדרך כלל בבחירתו של המתכנת. ניתן לבצע את הבדיקות מ"חוץ" לקטע הקוד ע"י בדיקות Black Box הבודקות את הפונקציונליות של הקוד, ובדיקות "פנימיות" הבודקות את מבנה הקוד ו"כיסוי" מרכיביו תוך כדי הרצתו.

לדוגמא ניתן לראות נהלי בדיקת unit testing אופייניים:

מטרת הבדיקה – לוודא שהמודול מקודד היטב. **מי מבצע?** – בדרך כלל המתכנת. **מה נבדק?** – בדיקת פונקציונלית הבודקת האם הקוד מבצע את מה שהוא אמור לעשות, בדיקה של מבנה המודול ובדיקה של ערכי סף. **מתי מסתיים?** – כאשר המתכנת חש בנוח עם התוצאות. **כלים** – לרוב לא קיימים. **רישום** – לרוב לא קיים.

2.3.2.בדיקת מערכת (Sytem Testing)

בדיקות אלו מבוצעות כאשר המודולים מורכבים יחד למערכת. שלב מקדים לבדיקה זו הינה בדיקת אינטגרציה הבודקת את ההשפעה שיש לחלק מוגדר במערכת עם חלקים אחרים במערכת ועם מערכות מקבילות ומשיקות.

הבדיקות בודקות את היכולת של המערכת לעמוד בפונקציונליות לה היא תוכננה וכמו כן לבדוק ערכי סף המבטיחים כי המערכת תעבוד גם במקרים קיצוניים.

בגלל אופיה של בדיקת המערכת, השימוש הנפוץ ביותר במהלך הבדיקות הללו הינה שימוש בבדיקות קופסא שחורה Black box testing.

בדיקות המערכת הינם הרבה יותר פורמליות מבדיקות יחידה וניתן בה דגש לתיעוד של הבדיקות ולתנאים לסיומה.

לדוגמא ניתן לראות נהלי בדיקת מערכת אופייניים:

מטרת הבדיקה – בחינת עבודת המערכת הבנויה ממספר מודלים והערכת מוכנות לבדיקות קבלה. **מי מבצע?** – ראשי קבוצות הפיתוח או קבוצת בדיקה נפרדת. **מה נבדק?** – דרישות המערכת, והממשקים. **מתי מסתיים?** – לרוב כאשר רוב הדרישות מתקיימות ואין באגים קריטיים. **כלים** – מאגר תרחישי הבדיקה, מנגנוני השוואה של תוצאות, סימולטורים. **רישום** – מבוצע רישום של התקלות, ומבוצע עדכון של תרחישי הבדיקות במידה וצריך.

2.3.3. בדיקות קבלה(Acceptance Testing)

בדיקות הקבלה מתחילות כאשר בדיקות המערכת מסתיימות. המטרה העיקרית שלהן הינה לתת למשתמש הסופי ו/או לצרכן המערכת בטחון והבטחה שהתוכנה אותה הוא מקבל, עומדת בדרישות והיא מוכנה לשימוש.

תרחישי הבדיקה בבדיקות הקבלה בדרך כלל נגזרים מבדיקות המערכת ומהווים רק חלק קטן מהם.

לדוגמא ניתן לראות נהלי בדיקת קבלה אופייניים:

מטרת הבדיקה – בחינת המוכנות לשימוש. **מי בודק?** – המשתמש הסופי או נציגו. **מה נבדק?**- תרחישי עיקריים, תיעוד ונהלים. **מתי מסתיים?** – כאשר המשתמש מרוצה ו/או כאשר הבדיקה מצליחה. **כלים** – מנגנוני השוואה של התוצאות. **רישום** – נקודתי, הסיכום עצמו מאוד רשמי.

2.4. עקרונות בבדיקות תוכנה:

לאחר צבירת ניסיון מספק בבדיקות תוכנה לאורך השנים ניסח Bill Hetzel 6 עקרונות של בדיקות תוכנה שכל אחד מהם הינו חשוב להבנה ע"י הבודק או המנהל שלו [2].

2.4.1.בדיקה מלאה אינה אפשרית!!

לא ניתן לצפות שאנו נוכל לבצע בדיקה מלאה של תוכנה, הסיבות לכך אינן עצלות או קיצורי פינות אלא חסמים מעשיים. וכמות בלתי אפשרית של נתונים מרבדים שונים שאנו צרכים לבדוק כדי לבצע "בדיקה מלאה".

לכן תפקידו של הבודק הינו לבחור מתוך מרחב אפשרויות הבדיקה את האפשרויות המייצגות ביותר שאיתן הוא יכול לבדוק את המערכת בצורה טובה. אפשרויות אלו צריכות להיות מעשיות לאור המרחב העצום של המידע שניתן לבדוק.

2.4.2.תהליך הבדיקה הינו קשה ויצירתי

הדעה הרווחת שקל להיות בודק תוכנה וכל אחד יכול לעשות זאת, דעה זו אינה נכונה, קשה להיות בודק תוכנה. צריך להכיר היטב את המערכת אותה בודקים,

והמערכת לא תמיד פשוטה... בנוסף הבודק צריך להיות יצירתי לצורך התאמת תרחישי הבדיקה לתרחישים מהעולם היעד של התוכנה הנבדקת, לדוגמא, בתוכנה פיננסית הבודק צריך לדעת עם הריבית שהתקבלה היא נכונה או לאו.

2.4.3.מטרה חשובה של בדיקת תוכנה היא למנוע מתקלות לקרות

רבים רואים בבדיקות את הצורך למצוא תקלות ולא למנוע אותם. צריך להבין שבדיקות תוכנה אינם שלב, אלא הם שזורים בכל תהליך הפיתוח של התוכנה ולכן הן לא נועדו לחפש תקלות אלא למנוע אותן.

2.4.4.בדיקת תוכנה מבוססות על סיכון מושכל

כאמור לא ניתן לבצע "בדיקות מלאות" (סעיף 2.4.1) , על כן נותר תמיד חשש שהבדיקות שבוצעו אינן מספקות. על כן צריך להבין שכמות הבדיקות הנדרשת והסיכון שלוקחים בהכרזה על מערכת תקינה, היא על פי אופי המערכת. למשל מטוס שטס בשמיים צריך בדיקות מקיפות יותר מאשר בדיקות של מכונה לייצור גלידה...

2.4.5.בדיקות תוכנה חייבות להיות מתוכננות

מכיוון שלא ניתן לבדוק בדיקה מלאה (סעיף 2.4.1), חשוב וקריטי שאחראי הבדיקות יתכנן היטב את תהליך הבדיקות. החל מהגדרת מטרת הבדיקה של המערכת דרך יצירת מסמך בדיקות כבר בתחילת הדרך של הפיתוח. היתרונות בתוכנית בדיקה, הם היכולת לבדוק את מה שאנו צרכים ולא לתת לאיש הבדיקות לבדוק את מה שהוא חשב לנכון. יתרון נוסף היא יצירת משמעת בדיקות בכל התהליך.

2.4.6.בדיקת תוכנה זקוקה לעצמאות

במהלך הפיתוח אנו מסגלים לעצמנו "בעלות" על התוכנה אותה אנו מפתחים ואי לכך בבואנו לבדוק אותנו אנו רוצים להוכיח שהיא עובדת. העובדה הזו גורמת לנו להיות מוטים בבדיקה, **ואי לכך יש צורך בגוף חיצוני שלא מרגיש בעלות על התוכנה לצורך בדיקתה**.

2.5. שלבים בבדיקות תוכנה

זווית נוספת שניתן לחלק את עולם בדיקות התוכנה הינה הזווית של משפחות הבדיקה.

קיימות שתי משפחות עיקריות של בדיקות: **בדיקה בקטן**(testing in the small), **בדיקה בגדול** (testing in the large). אלו הן שתי משפחות של בדיקות שיחדיו יוצרות תהליך הבדיקה. בדיקה של מודולים נפרדים של לוגיקה נקראים Tesing in the Small. בדיקה של קבוצת מודולים שיוצרות מערכת נקראת Testing in the large.

המטרות העיקריות של בדיקה בקטן הן לענות על השאלות הבאות: האם הלוגיקה עובדת כמו שצריך? האם הקוד מבצע את מה שהוא אמור לעשות, האם הקוד יכול להיכשל. האם כלל הלוגיקה נמצאת? האם יש פונקציות חסרות? האם

המודול מבצע את מה שהוא יועד לו? שמות נרדפים לבדיקה בקטן הם בדיקות מודולים, ובדיקות יחידה (unit testing).

המטרות העיקריות של בדיקה בגדול הן: מהי איכות המערכת? מה המערכת מסוגלת לבצע? מה המגבלות? האם המערכת תתמוך במתאר המבצעי? האם ישנם נפילות עיקריות? האם האיכות הולמת לשימוש? האם המערכת מוכנה לשימוש?

בדיקות עיקריות בבדיקה בגדול הן: בדיקות אינטגרציה - בדיקת ממשקים בין מודולים, חיבוריות בין מרכיבי המערכת, ביצוע תרחישים ע"י המודולים השונים (ביחד). בדיקות מערכת - ביצועים פונקציונליים, יכולות המערכת. בדיקות קבלה - עמידה בדרישות המערכת, מוכנות לשימוש.

2.6. סיכום חלק בדיקות התוכנה וקישור לעבודת הסיום:

בדיקות תוכנה הם עולם ולמלואו, ומקצוע שלם בפני עצמו.

אחד העקרונות כפי שהוצגו בפירוט (2.4.1) הוא עיקרון שמדבר על המגוון העצום בתרחישי בדיקה האפשריים. כלומר לא ניתן לבדוק את כלל המערכת. אי לכך אנו צרכים לוותר על חלק מהבדיקות, על מנת שנוכל לסיים את הבדיקה בזמן אל מול האיכות הנדרשת.

וויתור על חלק מהבדיקות היא אומנות שנפרסת לפתחו של בודק התוכנה, הרעיון הוא לספק מספיק תרחישים אשר בודקים את מירב האפשרויות אשר איתם מרגישים בטוחים מעבודת התוכנה הנבדקת.

בעבודה אני אתמקד בבדיקות קופסא שחורה ובייחוד בבדיקת תאימות של הפונקציה לגרסה הקודמת, בדיקה זו ידועה גם בשמה בדיקת נסיגה - Regression Test . תוך כדי שימוש באלגוריתמים גנטיים לצורך בניית תרחישי הבדיקות הנותנים את "תחושת הביטחון" בביצוע הבדיקות.

אני אתמקד לצורך זה בשימוש באלגוריתמים גנטיים המשמשים לצורך בדיקות פונקציונליות של תוכנה. חלק מהאלגוריתמים הם בעלי פרמטרים משתנים, כגון אלגוריתם בעל גודל אוכלוסייה משתנה, ואלגוריתם המשלב גודל אוכלוסייה משתנה לצד שימוש בעקרונות ה- Fuzzy Logic לצורך קביעה של הסתברות השחלוף (Cross-over). את האלגוריתמים אני אממש ואתאים לצורך ביצוע בדיקות נסיגה.

3. אלגוריתמים גנטיים סקירה

3.1. מבוא

חלק זה של העבודה מבוסס על המאמר הבא [3]:

Darrell W.: A Genetic Algorithm Tutorial, Statistics and Computing, Vol 4, Nov 2002, 65-85

הסקירה הינה קצרה ומסבירה את העיקרון בלבד, בכדי להעמיק בנושא ניתן לפנות למאמר לצורך הרחבה.

אלגוריתמים גנטיים הינם משפחה של מודלים חישוביים אשר יונקים את השראתם מתורת האבולוציה.

אלגוריתמים אלו מוצאים פתרון אפשרי לבעיה ע"י ייצוג של כרומוזום, ובעזרת פעולות של אבולוציה שהם בחירה, זיווג ומוטציה הם מטייבים את הפתרון עד למתן הפתרון המיטבי.

יישום אלגוריתם גנטי מתחיל בבחירה (לרוב ראנדומלית) של אוכלוסיית הפתרונות (population), כל אחד מהם נבדק אל מול הבעיה, ומקבל ציון על איכות הפתרון שהוא מציע. בשלב הבא ע"י פעולות של בחירה ורביה של כרומוזומים "טובים" (כאלה שקיבלו ציון גבוה בפתרון הבעיה) נוצרים כרומוזומים "טובים יותר" אשר עונים טוב יותר על הבעיה. בסוף התהליך אנו צפויים לקבל אוכלוסייה המהווה פתרון מיטבי לבעיה.

הראשון שהגדיר את המושג אלגוריתם גנטי הנו John Holland (1975) , האלגוריתם שהוא תיאר נקרא בספרות גם האלגוריתם הקנוני ("the Canonial Genetic algorithm"), זהו האלגוריתם הקלאסי ובו גם אתמקד בהמשך סקירה זו.

משפחות נוספות של אלגוריתם נולדו מהאלגוריתם הקאנוני, אך הם שונות ממנו , למעשה כלל המשפחות של האלגוריתם אשר עושות שימוש בפעולות גנטיות (בחירה, שחלוף, ומוטציה) נקראות אלגוריתמים גנטיים, לחלק מאלגוריתמים אתייחס בהמשך. חשוב לציין כי האלגוריתם שעליו אתבסס בכלל העבודה ושימושו בבדיקות תוכנה שייך למשפחה השנייה של האלגוריתמים הגנטיים והוא לא האלגוריתם הקאנוני של Holland.

3.2. האלגוריתם הקאנוני[4]:

הצעד הראשון בכל אלגוריתם גנטי הינו יצירת אוכלוסיית פתרונות (population), אוכלוסייה זו מכילה מספר רב של כרומוזומים.

מהו כרומוזום? כרומוזום במקרה של אלגוריתם גנטי הינו ייצוג של מרחב הפתרונות באמצעות מחרוזת בינרית. אם נתבונן על בעיית המינימום/מקסימום שהאלגוריתם צריך לפתור כ- $F(x1,x2,x3,,,,,,,x_m)$ אזי הכרומוזום הינו הייצוג של

$x1,x2,x3,,,,,,,x_m$. כמובן שיש שיטות רבות איך לייצג את זה כמחרוזת בינארית אך לא נגע בהם כעת.

כל כרומוזום נבדק את מול פונקציית כשירות (fitness function) שהיא המדד לטיב הייצוג של הכרומוזום אל מול האופטימיזציה של הבעיה.

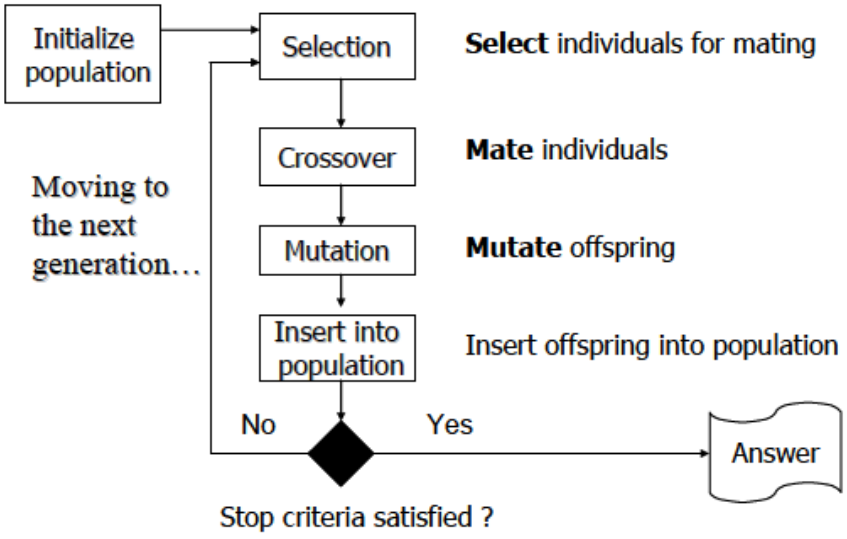
כיצד מחשבים פונקציית כשירות (fitness function)? יש למעשה פונקציית הערכה (evaluation function) שתפקידה לתת ציון אבסולוטי לטיב הפתרון , מכיוון שציון זה אינו מספיק כדי לבצע את פעולת הבחירה, אנו נבצעים המרה של התוצאה , והמרה זו נקראת פונקציית כשירות. החישוב הינו:

ממוצע פונקציות הערכה של כלל הכרומוזומים / פונקציית הערכה = פונקציית כשירות

3.3. פעולה בסיסית של אלגוריתם גנטי

ניתן לראות את פעולת האלגוריתם הגנטי כשני שלבים הראשון הינו בחירה של כרומוזומים מהאוכלוסייה (population) לתוך אוכלוסיית הביניים (intermediate population) ומשם ביצוע של פעולות שחלוף ומוטציה (crossover ו mutation) ליצירת האוכלוסייה החדשה.

ניתן לראות באיור 1 פעולה בסיסית של אלגוריתם גנטי קאנוני:



איור 2 – תרשים זרימה כללי של אלגוריתם קאנוני [3]

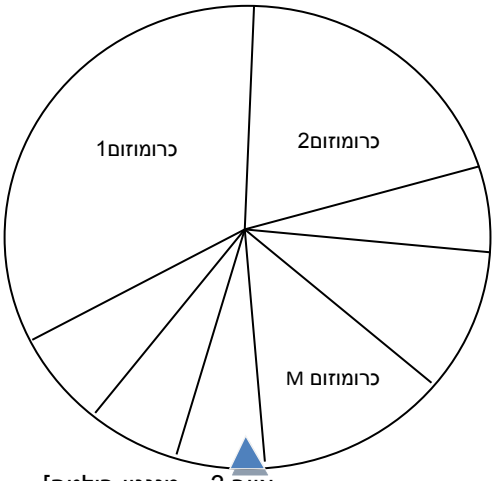
3.3.1.פעולת בחירה (selection):

פעולה זו בחורת מהאוכלוסייה הקיימת את הכרומוזומים העוברים לשלב הבא על פי משוואת הכשירות. ישנן שיטות רבות לבצע בחירה, ארחיב על אחת מהם "שיטת הרולטה"

מהי שיטת הרולטה?

נתייחס לכל בחירה של כרומוזום בודד כסיבוב יחיד של רולטה, כך שמסובבים את הרולטה כמספר הכרומוזומים שקבענו באוכלוסייה (population). היכן שהרולטה עוצרת מצביע על הכרומוזום הנבחר.

כל כרומוזום מקבל יחס שונה ברולטה על פי התוצאה של פונקציית ה- fitness שלו, כך שהכרומוזום בעל פונקציית ה- fitness הגבוהה ביותר יקבל "פרוסה" יותר גדולה ברולטה , כפי שרואים באיור 3, וכך יגדיל את סיכויו להיבחר בשנית.



איור 3 – מנגנון רולטה]

בסיום פעולת הבחירה מסתיים למעשה השלב הראשון של האלגוריתם בסיום ייצור אוכלוסיית הביניים וניתן לעבור לשלב הבא של האלגוריתם שהוא הייצור של אוכלוסיית הבחירה הבאה – next population.

3.3.2.פעולת השחלוף (crossover):

הפעולה הבאה היא זיווג של שני כרומוזומים לצורך ייצור 2 כרומוזומים חדשים.

הפעולה מתחילה בסיום פעולת הבחירה עם שני כרומוזומים שהם ההורים, את זוג הכרומוזומים אנו מזווגים בהסתברות של P_c ע"י פעולת crossover ליצירת שני כרומוזומים חדשים שיוכנסו לאוכלוסייה החדשה.

הדגמה של פעולת שחלוף (crossover):

נניח שניקח את שני הכרומוזומים הבאים שמייצגים פתרון של בעיית אופטימיזציה:

- 1110001110001110
- 0001110001110001

אנו מבצעים crossover מסוג 1 point crossover , בנקודה ראנדומלית נניח במקרה זה 5, אנו מקבלים את חלקי הכרומוזומים הבאים:

- 11100 01110001110
- 00011 10001110001

ע"י פעולת swap בין הכרומוזומים בנקודה 5 נקבל את הכרומוזומים הבאים:

- 1110010001110001
- 0001101110001110

אלו הם הכרומוזומים איתם אנו ממשיכים לשלב הבא של האלגוריתם.

3.3.3.פעולת מוטציה (mutation):

לאחר כלל תהליכי הבחירה והזיווג אנו מבצעים פעולת מוטציה על הביטים, פעולה זו מדמה מוטציות גנטיות הקורות גם בתורת האבולוציה, והיא נועדה לצורך מניעת התכנסות למקסימום לוקלי ע"י האלגוריתם.

בפעולה זו נעשה מעבר על כל הביטים באוכלוסייה, ושימוש בהסתברות של P_m המערכת מחליטה אם לשנות את ערך הביט או להשאירו. ניתן לממש את השינוי ע"י קביעת ערך אבסולוטי בביט, או לבצע היפוך של ערכו על פי המימוש שנבחר.

3.4. המשך האלגוריתם עד לסיומו:

לאחר שנוצרה האוכלוסייה החדשה מתחילים את כל התהליך מהתחלה עד להגעה לקריטריון עצירה. קריטריון זה יכול להיות מספר האיטרציות, ויכול להיות תוצאת evaluation function מספקת כל זאת על פי המימוש הנדרש.

3.5. אלגוריתמים נוספים

האלגוריתם הקאנוני הינו האלגוריתם הגנטי הראשון, המפורסם, והכללי ביותר.

עם השנים הוצגו וריאציות רבות לאלגוריתם זה, למעשה כל שינוי בפרמטרים שהציג האלגוריתם יכול להביא שינוי לדוגמא: שינוי שיטת Crossover: במקום להשתמש ב: 1-point crossover ניתן להשתמש בווריאציות שונות. שינוי שיטת הבחירה: שיטות נוספות הוצגו במקום ה- Roulette Wheel.

בהמשך העבודה אסביר בפירוט מספר אלגוריתמים נוספים במשפחה המשתמשת בפרמטרים משתנים. אלגוריתמים שיוסברו הם אלגוריתם בעלי

אוכלוסייה משתנה וכמו כן אלגוריתם בעל אוכלוסייה משתנה המשלב Fuzzy Logic לצורך קביעת הסתברות השחלוף (cross-over).

4. בדיקות אבולוציוניות של סוכני תוכנה אוטונומיים

4.1. כללי:

כחלק מלימוד נושא בדיקות אוטומטיות בעזרת אלגוריתמים גנטיים, עברתי על מספר רב של מאמרים בחיפוש אחר מאמרים בנושא בדיקות אוטומטיות פונקציונאליות של תוכנה.

רוב המאמרים עוסקים בשילוב של אלגוריתמים גנטיים לטובת בדיקות "קופסא לבנה", אולם ישנם מספר מאמרים שעוסקים גם בשילוב אלגוריתמים גנטיים בבדיקות "קופסא שחורה".

מאמר זה דן בתת ענף של בדיקות פונקציונליות והוא בדיקת סוכני תוכנה אוטומטיים, והוא מהווה רקע מצוין ללימוד שיטות ורעיונות לשילוב אלגוריתמים גנטיים בבדיקות "קופסא שחורה" על כן הוא מופיע בעבודה זו, כחלק מלימוד הנושא.

פרק זה מהווה סיכום של הנכתב במאמר הבא[12]:

S,Miles.M,Harman,M,Luck: Evolutionary Testing of Autonomous Software Agents

המעוניין להעמיק יכול לפנות למאמר עצמו.

4.2. הקדמה:

מערכת אוטונומית איננה זקוקה למשוב מהמשתמש ועל כן בדיקות האיכות שלה צריכות להיות יותר מלאות מאחרות. המערכת מחליטה בכל שלב העבודה עבור עצמה מה לעשות, ומכיוון שההחלטות אינם דטרמיניסטיות , עבודתו של הבודק קשה יותר, שכן עבור מתאר בדיקה זהה ניתן לקבל תשובות שונות.

המאמר טוען שבדיקה של רכיב תוכנה אוטונומי מחייב בדיקה רחבה מאוד של מתארי בדיקה, גם כאלה שאינם ברורים למפתח המערכת. בהמשך המאמר הכותבים מציעים שיטה של בחינה אבולוציונית של מתארי בדיקה, ומציאת המתארים המתאימים ביותר לכך.

4.3. רקע:

הרעיון ברכיב אוטונומי (Autonomos – Auto= self, nomos=Law), הוא בכך שהרכיב חופשי מהשפעות חיצוניות מהחלטות של הנהלה, אלא הוא מחליט את ההחלטות עבור עצמו בהתאם לתנאי הסביבה.

כיום רכיבים אוטונומיים מתרחבים במימדים שונים, ניתן למצוא אותם במכוניות אוטומטיות, במערכות סחר אלקטרוני ואפילו שואבי אבק ביתיים.

בדיקה של תקינות מערכות אלא הינה קריטית עקב העובדה שהם למעשה אחראים על עצמם ובכך יכולים ליצור נזק לא מבוקר. הבדיקה שלהם כבר לא פשוטה משתי סיבות. האחת היא שהמערכת מאפשרת מצבים שהמתכנן שלה לא חשב עליהם שכן היא אוטונמית. ומצד שני מתארי בדיקה זהים יכולים ועלולים לתת תשובות שונות, שוב בגלל האופי הלא דטרמניסטי של המערכת והאוטונומיות שלה.

כדי להבין את הסוגיה ניתן את הדוגמא הבאה: קיימת מערכת לדחיסת לחץ אוויר לא אוטונומית שבודק אמור לבדוק. למערכת יש כניסה אחת ויציאה אחת. הבודק לא מעוניין לדעת את סוג אלגוריתם הדחיסה אלא רק את ערך המוצא מול הכניסה. בסט בדיקות אלו יכול הבודק להזריק ערכים רבים של כניסות, ולקבל אוסף של ערכי יציאה ולבדוק האם המערכת עומדת בקריטריונים שלה.

מצד שני ניקח דוגמא של רכיב אוטונומי לדחיסת נתונים, לרכיב יש כניסה אחת ויציאה אחת אך בתוך המערכת קיימים מספר רב של אלגוריתמים שאותם מחליט הרכיב להריץ על פי

בחירתו בהתאם לתנאי סביבה משתנים. כעת כאשר הבודק מזריק ערך כניסה הוא יכול לקבל ערך ביציאה בתלות האלגוריתם שבחר הרכיב, מכיוון שהרכיב יכול לבחור מספר אלגוריתמים וערכים שונים כדי לדחוס הנתונים ייתכן מאוד שערך המוצא עבור אותו ערך כניסה יהיה שונה. וכך מטלתו של הבודק הופכת קשה יותר כי עליו להזריק מספר מתארים זהים מספר פעמים לקבל תשובות ולבדוק התאמה להגדרות התקינות של הרכיב.

4.4. גישת הפתרון במאמר:

הכותבים מבצעים סקירה מהירה של מספר גישות לבדיקת רכיבים אוטונומיים בעבר, בתחום מערכות בלימה ברכבים וסחר אלקטרוני ומציעים את הגישה שלהם להתמודדות עם הסוגיה.

הם מציעים גישת "גיוס", כלומר על סוכן אוטונומי הינו מועמד ל"גיוס" ע"י בעלי מניות. וכדי להיות כשיר ל"גיוס" עליו לעבור מספר מבחנים ברמת קושי משתנה במשך זמן מסויים, באם עבר הוא מוכרז "קשיר לגיוס" ובכך נחשב רכיב תוכנה אמין.

יעדים רכים (soft goals), הינם יעדים עבור דרישות לא פונקציונליות, כלומר דרישות שהם לא שחור ולבן – לדוגמא יעילות של רכיב. הכותבים מציעים להשתמש במספר יעדים רכים של בעלי המניות כדי לבחון איכות של רכיב אוטונומי. על כן הגישה האבולוציוניות שהם מציעים מכילה 2 שלבים עיקריים:

4.4.1. הצגת היעדים הרכים של בעלי המניות כמדדי איכות:

יעדים רכים רלוונטים מומרים למדדי איכות לבחינת שביעות הרצון של בעל המניות. כמובן שהיעדים הרכים מגיעים ממרחב הבעיה והם לא גנריים.

4.4.2. בדיקה אבולוציונית:

כדי ליצור מגוון בדיקות עם רמות קושי עולות, ההמלצה היא להשתמש באלגוריתם אבולוציוניים. מדדי האיכות מומרים לפונקציית מטרה ובסופו של דבר לפונקציית כשירות (Fitness function), כדי לכוונן את החיפוש וליצור מתארי בדיקה קשים יותר.

לדוגמא ניקח את המקרה הבא: במקרה זה ישנו סוכן ניקיון אשר אמור לנקות שדה תעופה וכמו כן ישנו "בעל מניות" שהוא מנהל הבניין. אשר החליט לתת מטרות לבחינת סוכן הנקיון. במקרה זה הסוכן אמור לעבוד לבד וללא עזרה, כמו כן הסוכן צריך להיות חסון (robust) ויעיל שהם שני היעדים הרכים שהוגדרו.

על פי הגישה שבמאמר היעדים הרכים – חסון ויעיל יכולים להיות המדדים לבחינת איכות הרכיב. כמובן שהרכיב הוגדר להיות אוטונמי, אך אם הוא אוטונומי ולא חסון (מפסיק לעבוד) אז הוא לא תקין.

את היעדים הרכים ניתן לפרוט למדדים לבחינה, במקרה שלנו למשל ניתן ליצור מחסינות (robustness), שני תת יעדים: מצב סוללה (למשל אם הסוללה ירדה מתחת ל- 10 אחוז הרכיב נכשל) והתקלות במכשולים (כל התקלות יכולה להוריד נקודות) באותה שיטה אפשר להגדיר גם את היעילות (efficiency) למשל כמספר הדקות לצורך סיום משימת הניקוי.

4.5. האלגוריתם הגנטי המוצע לבדיקת רכיבים אוטונומיים.

4.5.1. קידוד ערכי הכניסה והערכת התוצאה:

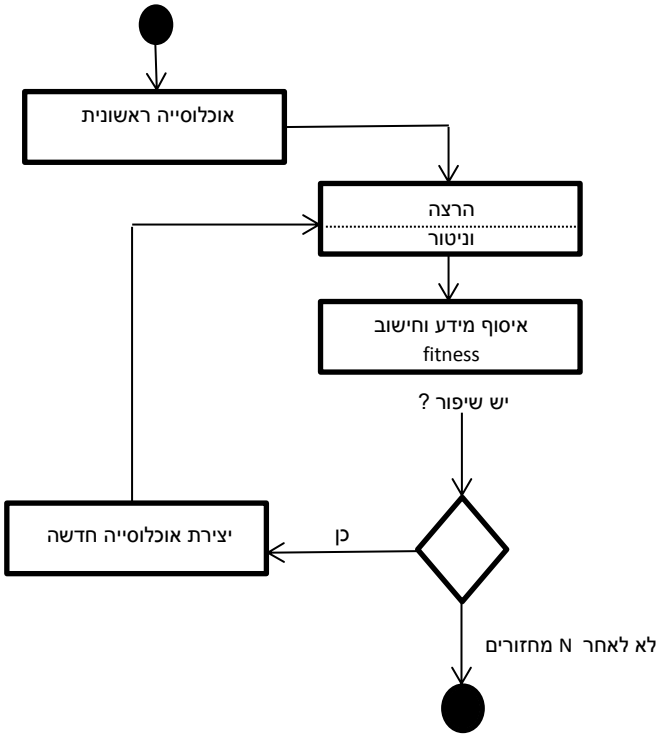
ערכי הכניסה של רכיב אוטונומי הם למעשה ה"עולם החיצון" בראי אותו רכיב. הרכיבים בוחנים באופן רציף את ה"עולם החיצון" (במקרה של הדוגמא הקודמת – את מטעני הסוללה, את המכשולים והלכלוך) ובוחרים לפעול על פי התוצאות שהם מקבלים.

במאמר הכותבים ממליצים לקודד כל משתנה חיצוני בודד כגן (gene), וכך מקרה בדיקה שמכיל את כלל המשתנים החיצוניים יהווה כרומוזום.

מכיוון שרכיב אוטונומי יכול להתנהג באופן שונה על אותם מתארי כניסה, הרעיון הוא להריץ את אותו מתאר בדיקה (במקרה שלנו כרומוזום) מספר פעמים ולהתייחס לתוצאה הכללית של כלל ההרצות. הרצה בודדת לרוב לא תניב את התוצאה הרצויה.

4.5.2. תהליך הבדיקה:

התהליך מוסבר באופן גרפי באיור 4, ומכיל 4 שלבים:



איור 4 – תהליך בדיקה של סוכן אוטונומי

א. *יצירת האוכלוסייה הראשונה:*

"חבילת" מקרי הבדיקה נקראת "אוכלוסייה", כל מקרה בדיקה הינו יחיד באוכלוסייה ומכיל פריסה יחידה של משתני הכניסה. האוכלוסייה יכולה להיווצר באופן ראנדומלי, או מתוך מקרי בדיקה קיימים.

ב. *הרצה וניטור:*

יש להריץ את הרכיב האוטונומי בסביבה אשר נקבעה ע"י מתאר הבדיקה, ובאותו זמן להתחיל ולנטר את התנהגותו של הרכיב. יש להריץ את התרחיש מספר פעמים כדי לאגור מידע מספק לטובת חישוב פונקיית הכשירות בשלב הבא.

ג. *איסוף המידע וחישוב פונקציית כשירות:*

משתמשים במידע מסוכם מכלל ההרצות לצורך חישוב פונקציית הכשירות. השיטה לחישוב תלויה בבעל המניות שהגדיר יעדים רכים ובמרחב הבעיה. במידה ואין שיפור תוך מספר הרצות צריך להפסיק או לעבור לשלב יצירת האוכלוסייה החדשה.

ד. *יצירת אוכלוסייה חדשה:*

כמו בכל אלגוריתם גנטי יוצרים אוכלוסייה חדשה: שני כרומוזומים נבחרים (עם התייחסות לתוצאת פונקציית הכשירות שלהם), על פי הסתברות מסוימת מבצעים שחלוף כדי ליצור 2 כרומוזומים חדשים, ובסוף מוסיפים mutation בהסתברות מסוימת.

4.6. מקרה בחינה.

מקרה הבחינה שבו השתמשו במאמר הינו מקרה הבחינה של סוכן הניקוי כפי שהוסבר בסעיף 4.4.2. כאשר ההרצה שנעשתה בה שימוש הינה הרצה על סביבה וירטואלית .

הסביבה הווירטואלית הינה ריבוע שהוגדר (נגדיר כ- A) , באיזור זה יכולים להיות: מכשולים, פח אשפה, לכלוך ותחנות טעינה. ומגדירים הצבת משתני סביבה בקונפיגרציה ספציפית של השטח A.

סוכן הניקוי אחראי על ניקוי שטח A ולצורך כך הוא צריך לבצע את המטלות הבאות באופן אוטונומי:

א. חיפוש מיקום של חפצים חשובים
ב. למצוא אשפה ולהביא אותה לפח הקרוב
ג. לשמור על חיי הסוללה על טעינתה
ד. לא לפגוע במכשולים ולשנות מסלול בהתאם לכך
ה. למצוא את הדרך המהירה כל פעם ללא פגיעה בחפצים
ו. לשמור על בטיחות על ידי עצירה בפגיעה בחפץ או כשהסוללה מגיע לרמה נמוכה מידי.

4.6.1. הכנות:

א. : קידוד ערכי הכניסה

משתני סביבה במקרה זה (שהם גם ערכי הבדיקה) מורכבים מכמות ומיקום של מכשולים, פחי אשפה, לכלוך, ותחנות טעינה. כל אחד מהם (מכשול/לכלוך/פח/תחנת טעינה) מקודד לערך גן (gene) על פי הרעיון המודגם באיור 5:

1	0	1	0	1	0
0	1	1	0	1	0
0	1	1	0	1	0
1	1	1	0	1	1
0	1	0	1	0	0
1	0	1	0	0	0

איור 5 – הגדרת גן (gene) למשתנה סביבה

מחלקים את השטח A ל- RXR תאים – Rהוא הרזולוציה.

מניחים את הפריט (מכשול/לכלוך/פח/תחנת טעינה), 0 משמעו אין פריט ו- 1 יש פריט. הרזולוציה לכל סוג משתנה סביבה יכולה להיות שונה. השחלוף שנעשה בין כרומוזומים הינו בין גן אחד לשני, כלומר לא נעשה שחלוף בין גן של פח לגן של לכלוך , אלא רק בין גן של פח לגן של פח.

ב. : חישוב פונקציית כשירות:

הגדרת פונקציית הכשירות נעשתה ע"פ הקרבה של הסוכן למכשולים תוך כדי עבודתו, נעשית מדידה רציפה של המרחק מהסוכן במשך כל משך עבודתו. כמו כן כאמור מבוצעות מספר הרצות בשל היותו אוטונומי והמדדים מחושבים סטטיסטית לצורך קבלת פונקציית הכשירות.

לצורך קביעת מספר הפעמים בהם יש צורך לבצע את ההרצות בצעו במאמר מספר ניסויים . בניסויים הם בחנו את התפלגות התוצאות, וגילו שמספיק 5 הרצות שונות בכדי שנוכל לקבל תוצאת פונקציית כשירות ממוצעת.

ג. : הגדרת פונקציית הכשירות:

נגדיר את D כוקטור של כל המרחקים הקרובים למכשול בכלל 5 הניסויים. נגדיר έ כמרחק המינימלי ממכשול המותר. נקבל את פונקציית הפיטנס הבאה:

Min(D) + ώ1**quartile1*****(D) + ώ3* *quartile3*****(D) / if min(D)>έ**

F=

Min(D)- έ / if min(D)<=έ

+∞ if the agent cannot move and suspend safely

כאשר המשקלים **ώ1** ו- **ώ3** חייבים להיות קרובים ל- 0 כי הם פחות חשובים מ- min(D).

הרעיון הינו ביצירת פונקציה אשר תכוון את החיפוש הגנטי ליצירת מתארים אשר מקרבים את כלל התוצאות קרוב ככל הניתן ל- έ. כאשר המרחק המינימלי כבר שווה או קטן מהמרחק המותר נקבל שגיאה שכן חרגנו מהאיזור המותר.

מכיוון שאנו רוצים לבדוק סוכן, אזי שגיאה משמעותה – כשלון בבדיקה.

4.6.2. ניסוי ותובנות:

המאמר מתארים הכותבים מספר ניסויים שהם בצעו בהתאם לאלגוריתם המוצע. תוצאות הניסוי מוכיחות שהשימוש באלגוריתם גנטי לצורך בדיקת רכיבים אוטונומיים הינו מוצלח ואיכותי.

בהמשך מציעים הכותבים מספר נוסף של בדיקות כדי לבחון את האלגוריתם כמו שימוש בפרמטרים רבים לבחינה (בניסוי הם בחנו רק קרבה למכשול).

5. בדיקות נסיגה (regression) באמצעות אלגוריתמים גנטיים

5.1. כללי:

כל מערכת עוברת שינויים כאלה או אחרים, ובניהם: שינוי תצורה, שינוי תשתית, שינויים בקונפיגורציה ועוד. בדיקה מלאה של כל המערכת לאחר שינוי קוד קטן ו/או שינוי אחר, הינו מחיר גבוה. כאן באים לעזרתנו בדיקות הנסיגה (regression test). שבודקים למעשה רק את העובדה שלא פגענו במערכת לאחר שינוי.

בפרקים הבאים אציג אלגוריתמים שונים אשר נעשה בהם שימוש לטובת בדיקות "קופסא שחורה", אבחן את ההתאמה שלהם לביצוע בדיקות נסיגה, ואציג שינוי קל בהם לצורך התאמה טובה יותר לסוג זה של בדיקות.

5.2. פירוט האלגוריתמים הנבדקים:

האלגוריתמים אותם אבחן הם:

א. אלגוריתם גנטי בעל אוכלוסייה משתנה (GAVaPS) – האלגוריתם יוצג בפירוט בפרק 6.
ב. אלגוריתם גנטי בעל אוכלוסיה משתנה, והסתברות שחלוף (cross-over) משתנה על פי עקרונות ה- Fussy Logic (fl)- האלגוריתם יוצג בפירוט בפרק 7.
ג. התאמה של שני האלגוריתמים בסעיפים הקודמים לבדיקות נסיגה, GAVaPS_R (שיפור של האלגוריתם GAVaPS לשימוש בבדיקות נסיגה) ו- FAexGA_R (שיפור של האלגוריתם FAexGA לשימוש בבדיקות רגרסיה), ובחינה של כל האלגוריתמים לטובת בדיקת נסיגה של "פונקציה תחת בדיקה "(system under test) . את עבודה זו אציג בפרק 8.

6. GAVaPS – אלגוריתם גנטי בעל אוכלוסייה משתנה[8]

6.1. כללי:

לצורך הבנת אלגוריתם זה פניתי למאמר בשם:

Arabas J.,Michalewicz Z.,Milawka J.: GAVaPS – a Genetic Algorithm with varying population size

סעיף זה מתבסס על המאמר [8] לצורך הרחבה ניתן לפנות למאמר העיקרי

גודל האוכלוסייה באלגוריתם גנטי הינה קריטית ליכולתו להגיע לתוצאה הרצויה. במידה והאוכלוסייה קטנה מידי האלגוריתם עלול להתכנס מהר מידי, ובמידה והוא גדול מידי, האלגוריתם יצרוך משאבי חישוב ובכך עלול לגרום לאיטיות במתן פתרון.

פרמטר נוסף וחשוב המשפיע מאוד על יכולתו של האלגוריתם הגנטי, הינו שיטת הבחירה (selection), שיטת הבחירה משפיעה על יכולתו של האלגוריתם להתכנס מהר או לאט, וזאת על פי "חוזק" הבחירה שנוקטת בה השיטה. בעוד שימוש בשיטת בחירה מאוד "חזקה" (שיטה שנותנת דגש חזק לתוצאות הטובות באוכלוסייה), מבטלת את השונות שבאוכלוסייה וגורמת להתכנסות מהירה. שיטת בחירה "חלשה" יותר מנצלת את השונות שבאוכלוסייה אך יכולה לקחת זמן רב ולהתבדר.

האלגוריתם המוצג (GAVaPS) מבטל כליל את שיטת הבחירה, ועובר לבחירה ראנדומלית, מצד שני הוא מתייחס ל"גילו" של הכרומוזום שבאוכלוסייה , שהוא למעשה מספר המחזורים בו הכרומוזום "חי". שיטה זו שתוסבר בפירוט בהמשך, מעבירה את ההשפעה של ה- fitness של כל כרומוזום על כמות המחזורים שהוא יחיה ("גילו") ובכך גם יוצרת אוכלוסייה בת מספר כרומוזומים משתנה.

גודל אוכלוסייה משתנה, גם מחקה בצורה טובה יותר את ההתנהגות האבולוציונית האמיתית , בה האוכלוסיות אב הם לא בגודל קבוע.

סוג זה של אלוגוריתם גנטי מתווסף לאלגוריתמים גנטיים אחרים בהם נעשה שינוי מסתגל של פרמטרים שונים של האלגוריתם הגנטי כמו הסתברויות של crossover ושל mutation.

6.2. מבנה האלגוריתם:

האלגוריתם כמובן מתבסס על מבנה אלגוריתם גנטי אך עם מספר שינויים:

- כל כרומוזום מכיל פרמטר נוסף והוא Age, פרמטר זה מציין את גילו של הכרומוזום, שהוא מספר מחזורי האלגוריתם מאז שהוא נולד.
- בכל מחזור של האלגוריתם נוספים כרומוזומים נוספים לאוכלוסייה.
- בכל מחזור של האלגוריתם יכולים "למות" חלק מהכרומוזומים ובכך האוכלוסייה קטנה.

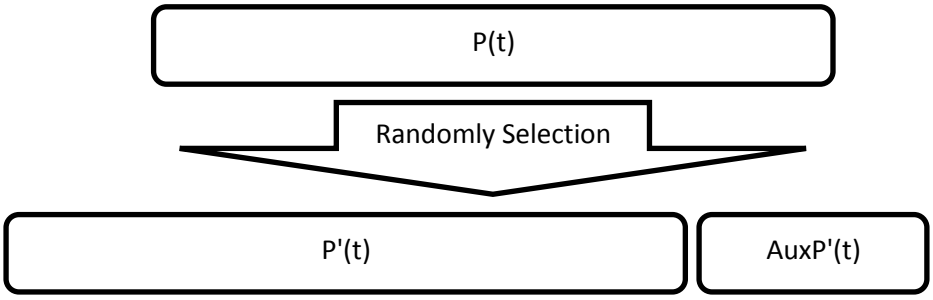
במחזור t ישנה אוכלוסייה P(t) בגודל מסוים. בשלב השחלוף ליצירת P'(t), נוצרת קבוצה נוספת של בנים לצד האוכלוסייה הקיימת, גודל אוכלוסייה זה הינו פרופורציונלי לאוכלוסייה הראשונית. אוכלוסייה חדשה זו מתווספת לאוכלוסייה שנבחרה, כך שאנו מקבלים אוכלוסייה גדולה יותר.

גודל האוכלוסייה הנוספת נקבע ע"י הנוסחה:

AuxPopulation(t)=P(t)*p

כאשר p הינו יחס הנקבע מראש (reproduction ratio).

כל כרומוזום מהאוכלוסייה הראשית יכול להיבחר לאוכלוסייה הנוספת. למעשה אפשר לתאר את גודל אוכלוסיית הבנים על הסכימה הבאה:



איור 6 – GAVaPS יצירת אוכלוסיית השלב הבא

שלב הבחירה (selection) הינו ראנדומלי ולא מתחשב בערך פונקציית הכשירות של כל כרומוזום האלגוריתם, על כן משתמש בשני פרמטרים נוספים Age ו- Lifetime של הכרומוזום שיעזרו בקביעת הכרומוזומים באוכלוסייה.

Lifetime הינו פרמטר אשר מחושב פעם אחת עבור כרומוזום בזמן ה-evaluation stage, או בזמן יצירת האוכלוסייה הראשונית של הכרומוזומים. ונשאר קבוע כל "חיי" הכרומוזום. כלומר מרגע שהוא נולד נקבע לו Lifetime, הוא לא מחושב שוב אלא הפרמטר מלווה אותו כל ימי "חייו" באוכלוסייה עד מותו.

גילו של כרומוזום (Age) הנו משתנה אשר גדל ב- 1 כל מחזור של האלגוריתם, הגיל הראשוני הינו 0. כרומוזום "מת" ברגע שגילו (Age) , גדול מהפרמטר Lifetime.

אם כך הפרמטר Lifetime למעשה קובע את הזמן המכסימלי בו יכול כרומוזום להיות בתוך האוכלוסייה, וכאמור הוא נקבע פעם אחת בלידתו של הכרומוזום. פונקציית הכשירות משפיע באלגוריתם זה על רק על פרמטר ה-Lifetime וכך קובעת את הכרומוזומים בתוך האוכלוסייה – יוסבר בהמשך.

למעשה בסיום מחזור אחד גודל האוכלוסיה מתנהג על פי הנוסחה הבאה:

P(t+1)=p(t)+AuxP(t)-D(t)

כאשר D(t) מציין את מספר הכרומוזומים שמתו במהלך מחזור האלגוריתם.

6.3. חישוב Lifetime:

כאמור זהו הפרמטר היחידי אשר מאפשר לנו להוריד כרומוזומים מהאוכלוסייה ולהתאים אותה למצב האלגוריתם. לכן חשוב למצוא שיטה טובה לצורך חישובו. ברור כי הצבה של מספר קבוע בתור Lifetime יגרום לגידול קבוע של האוכלוסייה מצד אחד, ומצד שני מכיוון ששלב הבחירה באלגוריתם הינו ראנדומלי, לא תהיה השפעה של ערך הכשירות של כל כרומוזום על הפרמטר ובכך ביצועי האלגוריתם יהיו גרועים.

מה נדרש כי לבצע חישוב טוב של הפרמטר Lifetime ?

1. אנו נדרשים לתגבר כרומוזומים בעלי כשירות גבוהה, לצד הורדה של כרומוזומים עם כשירות נמוכה.
2. כוונן טוב של גודל האוכלוסייה שיתאים למצב האלגוריתם ולא יגדל מצד אחד מעבר לנדרש, אך מצד שני ייתן יותר אפשרויות שונות (diversity) לאוכלוסייה ע"י הגדלתה במידת הצורך.

תגבור של כרומוזמים בעלי כשירות גבוהה יעשה ע"י מתן ערך Lifetime גבוה יותר עבורם ובכך נאפשר זמן חיים ארוך יותר לכרומוזומים חזקים, ובכך נגדיל את התפוצה שלהם באוכלוסייה (בדומה למה ששיטות בחירה גנטיות עושות).

מצד שני אנו צרכים להתייחס לכלל הכרומוזומים כקבוצה ולא להתייחס לכל אחד מהם כפרט, שכן כשירות טובה היא לאו דווקא כשירות גבוהה כי אם כשירות הגבוהה יותר משאר הכרומוזומים. לכן נחשב גם פרמטרים נוספים כמו MaxFit שהוא הכשירות המקסימלית באוכלוסייה, AbsFitMax שהיא הכשירות המקסימלית עד כה, ובדומה את ה- MinFit ו- AbsFitMin, וכמובן גם AvgFit שהוא ממוצע הכשירות באוכלוסייה.

כמובן שאנו צרכים למצוא גם נוסחה למציאת Lifetime שתהיה קלה ופשוטה לחישוב.

בכדי לענות על כלל הצרכים שהוצגו לצורך חישוב Lifetime נקבעו 3 שיטות שונות לחישוב פרמטר זה:

Proportional allocation:

Lifetime(i)= Min(MinLT+ ή*(Fitness(i)/AvgFit),MaxLT)

Where:

MinLT= Minimal allowed Lifetime

MaxLT=Maximal allowed Lifetime

ή=1/2*(MaxLT-MinLT)

הרעיון בשיטה זו דומה לשיטת הבחירה roulette wheel בכך שהערך של ה- Lifetime הינו פרופורציונלי לערך הכשירות שלו אל מול הממוצע. ועל כן כרומוזומים בעלי כשירות גבוהה באוכלוסייה יקבלו ערך Lifetime גבוה.

הבעיה בשיטה זו שהיא לא לוקחת בחשבון את טיב הכרומוזום אל מול כרומוזומים שכבר לא נמצאים באוכלוסייה ולהם היה ערך כשירות גבוה יותר.

Linear allocation:

Lifetime(i)=MinLT+2*ή((Fitness(i)-AbsFitMin)/(AbsFitMax-AbsFitMin))

Where:

MinLT= Minimal allowed Lifetime

MaxLT=Maximal allowed Lifetime

ή=1/2*(MaxLT-MinLT)

הרעיון פה הוא שיפור של השיטה הראשונה בכך שהיא מתחשבת בערכים האבסולוטיים שנמצאו גם באוכלוסיות קודמות, הבעיה כאן שבמידה וישנם הרבה כרומוזומים אשר ה- Fitness שלהם קרוב למקסימלי האבסולוטי, הרבה כרומוזומים יקבלו ערכי Lifetime גבוהים ובכך יגדילו משמעותית את האוכלוסייה.

השיטה האחרונה מנסה למצוא פשרה בין שתי השיטות הראשונות.

Bi-Linear allocation

Lifetime(i)=

- **MinLT+ ή((Fitness(i)-MinFit)/(AvgFit-MinFit)) if AvgFit>=Fitness(i)**
- **0.5(MinLT+MaxLT)+ ή((Fitness(i)-AvgFit)/(MaxFit-AvgFit)) if AvgFit<Fitness(i)**

Where:

MinLT= Minimal allowed Lifetime

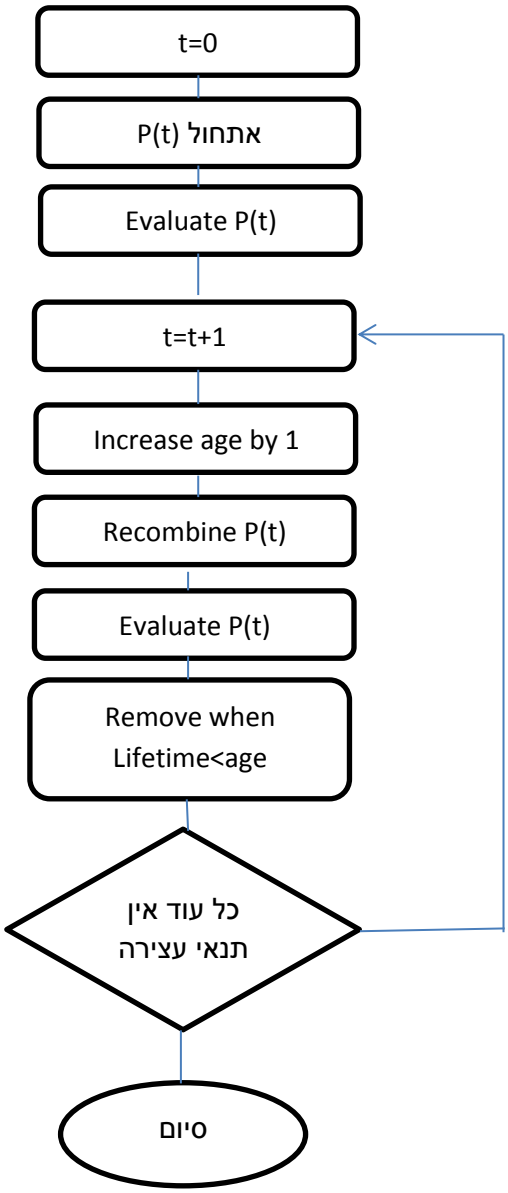
MaxLT=Maximal allowed Lifetime

ή=1/2*(MaxLT-MinLT)

הרעיון כאן הוא התייחסות לכרומוזומים עם כשירות נמוכה מהממוצע בדומה לשיטה הראשונה, בעוד לכרומוזומים עם ערכי כשירות גבוהים, היא מתחשבת לא רק בממוצע אלא גם בערך המכסימלי הנוכחי של הכשירות.

6.4. קיצור מבנה האלגוריתם

באיור 7 מוצג תרשים זרימה של האלגוריתם



איור 7 – GAVaPS – תרשים זרימה של האלגוריתם

7. FAexGA – אלגוריתם המשלב Fuzzy Logic ואוכלוסייה משתנה [6]

7.1. רקע

פרק זה נכתב לאורו של המאמר הבא:

Last M., and Eyal , S.: A Fuzzy-Based Lifetime Extension of Genetc Algorithms, Fuzzy Sets and Systems, Vol. 149, Issue 1, January 2005, 131-147

שימוש בפרמטרים מסתגלים באלגוריתמים גנטיים יכול לשפר את האלגוריתם ע"י התאמת פרמטרים אלה לסטטוס ריצת האלגוריתם ובכך עוזרים במניעה בהתכנסות מהירה שלו [9].

ניתן להשתמש בפרמטרים מסתגלים בבחירת הסתברות ה- crossover (Pc) , הסתברות ה- Mutation (Pm) ואפילו שיטת השחלוף עצמה.

אלגוריתם שנותן תוצאות טובות הינו אלגוריתם המשתמש באוכלוסייה משתנה, ע"י שימוש בפרמטרים נוספים של Age ושל LifeTime, והוא אלגוריתם ה- GAVaPS כפי שהוסבר במסמך בסעיף 6.

לצד המגמה של פרמטרים משתנים יש התעניינות במימוש שיטות Fuzzy Logic (FL) לצורך מימושם.

אלגוריתם ה- FAexGA הינו אלגוריתם אשר מוסיף לאלגוריתם ה- GAVaPS יכולת שימוש ב- FL לצורך בחירה מושכלת של הסתברות ה- Crossover (Pc).

7.2. אלגוריתמים גנטיים המשתמשים ב- FL (FGA- Fazzy logic GA)

אלגוריתמים אלה משתמשים ב- FL בכדי לכוונן את האלגוריתם. החל בהתאמה של הסתברויות (Pm,Pc), דרך בחירת שיטת השחלוף ועד לבחירת נקודת העצירה (stop criteria).

השימוש יכול להיעשות מכמה סיבות, כל אלגוריתם לפי הייעוד שלו. ניתן לקבוע את האורך המכסימלי של הכרומוזום, וזאת על מנת שהאלגוריתם יהיה יעיל ומהיר. ניתן לשנות את הסתברות ה- mutation לטובת מניעה של התכנסות מהירה והאצת מהירות האלגוריתם. וניתן לקבוע מועד עצירה של האלגוריתם כדי לעזור במציאת אופטימום גלובלי ולא לוקלי.

7.3. Fuzzy Logic Controller (FLC)

ה- FLCהוא למעשה המנגנון אשר מאפשר לנו לכוונן את הפרמטרים בצורה מסתגלת, ה- FLC מממש את הגדרות ה- FL אשר הוחלט עליהם באלגוריתם.הסבר מפורט לדרך פעולת ה – FLC והסבר כללי על הרעיון מאחורי ה- FL נמצא בנספח העבודה בסעיף 11.

7.4. הרעיון מאחורי האלגוריתם

האלגוריתם משתמש באלגוריתם מסוג GAVaPS [8] כפי שהוסבר עליו מקודם, ובנוסף משתמש ב- FL בכדי לכוונן את הסתברות השחלוף (CrossOver) שלו (Pc).

המבנה הרעיוני של האלגוריתם הוא (השינוי מהאלגוריתם הקודם מודגש):

```
Procedure LifetimeExtendedGA {
        t = 0
        Random_initialize P(t)
        Evaluate P(t)
        While (not termination condition) {
                t= t+1
                        Increase the age of each individual by 1
                Recombine P(t) {
                    Select parents from P(t)
                    For each pair of parents {
                        Assign crossover probability (Pc) {
                                Call Fuzzy Logic Controller (parameters)
                        }
                        Perform crossover with probability Pc
                        Perform mutation with probability Pm
                        Insert offsprings into P(t)
                      }
                }
                Evaluate P(t)
                Remove from P(t) all individuals with age greater than their
                lifetime
        }
}
```

7.4.1. FLC

מרכיבי ה- FLC תוכננו בדרך הבאה:

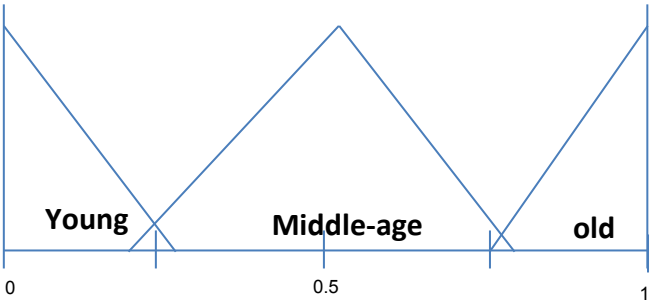
- משתני מצב: Age ו- Lifetime של כל כרומוזום המיועד לזיווג (crossover) וממוצע ה- Lifetime של כלל האוכלוסייה (ערכים בדידים)
- משתני בקרה (control parameters): הסתברות שחלוף (crossover) בתור משתנה בדיד (crisp).

7.4.2. משתני ה- fuzzy:

לצורך מימוש ה- Fuzzification השתמשנו במשתני השפה הבאים:

- Young
- Middle-Age
- Old

כל אחד מההורים מחושב עבורו ה- degree of truth לפי הדיאגרמה הבאה באיור 5:



איור 8 – FAexGA פונקציית החברות של הגילאים

כאשר הציר התחתון הינו גרף הגילאים הקיימים באוכלוסייה המנורמל בין 0 ל- 100

7.4.3. שימוש בגיל הכרומוזומים לקביעת Pc

ה- FLC קובע את ה- Pc של שני ההורים על פי משתני המצב לפי הרעיון הבא:

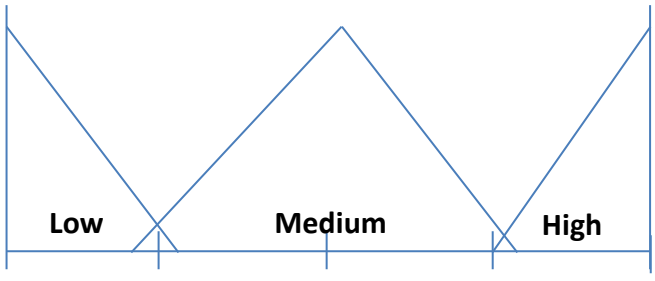
גיל ההורים קובע את הסתברות ה- Pc . להורה צעיר נקבע הסתברות שחלוף נמוכה, וזאת בכדי לאפשר לו להתפתח (Exploration) באוכלוסייה ולא ל"היזרק" מוקדם מידי, מצד שני הורה בגיל הביניים אנו ניתן לו הסתברות גבוהה לשחלוף בכדי לנצל (Exploitation) את ההורה ליצירת "ילדים" הדומים לו. הורה מבוגר יותר באוכלוסייה בהסתברות שחלוף נמוכה עד אשר הוא "ימות".

לסיכום אלו הם משתני השפה (Linguistic Parameters) של ה- FLC:

Age ∈[Young, Middle-age, Old]

*P*c ∈ [Low, Medium, High]

פונקציית החברות של ה- Pc שהוצגה במאמר נראית כך:



איור 9 – FaexGA פונקציית החברות של הסתברות השחלוף

כאשר הציר התחתון הוא בין 0 ל- 1 לערכי ה- Pc.

7.4.4.חוקי ה- FL

לצורך השלמת השימוש ב- FL יש צורך בקביעת חוקי השפה. החוקים מופיעים בטבלה הבאה, כאשר הם נקבעו על פי העיקרון של גיל הכרומוזומים כפי שהוצג מקודם:

Old	Middle-Age	Young	גיל כרומוזום אב 1 גיל כרומוזום אב 2
Low	Medium	Low	**Young**
Medium	High	Medium	**Middle-Age**
Low	Medium	Low	**Old**

כאשר הערכים בטבלה (High,Medium,Low) הינם ערכי ה- Pc לכל זוג הורים על פי קבוצת שייכות הגיל שלהם.

כל חוק מקבל relevance degree השווה לערך המינימלי של התנאים שלו, לאחר מכן נקבע לכל משתני פלט (Low,Medium,High) ערך אחד שהוא הערך המקסימלי של כלל החוקים. השיטה הזו נקראית MAX-MIN (הסבר ניתן למצוא בנספח סעיף 11).

7.4.5.שיטת Defuzzification

בסוף התהליך יש צורך במציאת ערך בדיד של הסתברות השחלוף, תהליך זה (Defuzzification) מקבל את ערכי משפחות הגילאים של ההורים. על פי חוקי ה- FL שהוצגו בסעיף 7.4.4 וביחד עם פונקציית החברות של Pc מחשב ערך בדיד.

באלגוריתם זה משתמשים בשיטת ה- Center of gravity שעיקרה הינו מציאת הערך שמהווה את ה- center of gravity שנמצא מתחת לפונקציית החברות של ה- Pc. גם נושא זה יוסבר בקצרה בפרק ההסבר על שיטת ה- FL (סעיף 11 בנספח)

7.4.6.חישוב Lifetime

המשתנה האחרון שיש להחליט על צורת חישובו באלגוריתם זה הינו שיטת חישוב משתנה ה- lifetime.

בהסבר על אלגוריתם ה- GAVaPS הוצגו 3 שיטות לחישוב המשתנה.
במימוש האלגוריתם הזה הוחלט להשתמש בחישוב הבא:

Bi-Linear allocation

Lifetime(i)=

- **MinLT+ ή((Fitness(i)-MinFit)/(AvgFit-MinFit)) if AvgFit>=Fitness(i)**
- **0.5(MinLT+MaxLT)+ ή((Fitness(i)-AvgFit)/(MaxFit-AvgFit)) if AvgFit<Fitness(i)**

Where:

MinLT= Minimal allowed Lifetime

MaxLT=Maximal allowed Lifetime

ή=1/2*(MaxLT-MinLT)

הרעיון כאן הוא התייחסות לכרומוזומים עם כשירות נמוכה מהממוצע, בעוד לכרומוזומים עם ערכי כשירות גבוהים, היא מתחשבת לא רק בממוצע אלא גם בערך המכסימלי הנוכחי של הכשירות.

לדוגמא, באם ניקח כרומוזום עם רמת כשירות הנמוכה מהמומצע, אזי החישוב שייבחר (האופצייה הראשונה) יתייחס למשקל הכשירות שלו אל מול המומצע כך שהוא יקבל ערך Lifetime נמוך יחסית לשאר הכרומוזומים ולכן "ימות" מהר.

מצד שני באם לכרומוזום ערך כשירות גבוה מהממוצע הוא ייבדק אל מול ערך הכשירות המקסימלי שנמצא עד כה, כך ששככל שערך הכשירות קרוב לערך המכסימלי הוא יקבל Lifetime יותר גבוה, במידה וערך הכשירות גבוה מהממוצע אך קטן מהמקסימלי הוא יקבל ערך Lifetime גבוה, אך נמוך מהכרומוזום בעל ערך הכשירות המקסימלי.

8. בדיקת קופסה שחורה בעזרת אלגוריתם FAexGA

סעיף זה מתבסס על המאמר [7]:

Last M.,Eyal, S. Kandel,A :Effective Black-Box Testing with Genetic Algorithms,Lecture Notes in Computer Science,2006,vol 3875. 2006, 134-148

בסעיף הבא אני אציג שינויים שביצעתי הן באלגוריתמים והן בפונקציית הבדיקה כדי לבחון השפעות של כלל האלגוריתמים במאמר: קאנוני, GAVaPS_R, FAexGA_R. על בדיקת רגרסיה ואצביע על האלגוריתמים היעילים ביותר לצורך כך.

במאמר זה הכותבים בוחנים שימוש של אלגוריתם גנטי עם אוכלוסייה משתנה המשתמש בעקרונות ה- Fuzzy Logic. למעשה הם בוחנים את אלגוריתם ה- FAexGA אל מול אלגוריתמים גנטיים אחרים כמו הקאנוני וה- GAVaPS שהוצג בעבודה זו והן הוצג במאמר.

המטרה העיקרית שלהם היא למצוא שיטה אפקטיבית לייצור משתני בדיקה לפונקציה הנבדקת. ייצור משתנים אלה בצורה ידנית הינם קשים ביותר שכן מרחב האפשרויות למשתני כניסה הוא עצום, ועל כן הבודק לא יכול לבדוק את כולם. הרעיון הוא ליצור מספר משתני בדיקה יעילים ובמהירות. כך שבמידה וישנה תקלה פונקציונלית במערכת הנבדקת, תוכנית הבדיקה המבוססת על האלגוריתם הגנטי, תגלה אותה.

בתחילת המאמר מבצעים הכותבים סקירה על משפחות של אלגוריתמים גנטיים וכמו כן הסבר על אלגוריתם ה- FAexGA וה- GAVaPS.

הכותבים ממשים פונקציה חיצונית המשמשת כפונקציית הנבדקת (system under test), מבנה הפונקציה הוא: משתנה בוליאני בן 100 איברים, 3 אופרטורים AND,OR,NOT המפרידים בין כל משתנה בוליאני. הפונקציה נבנתה בתוכנה חיצונית והיא נקראת Correct Expresion.

בכדי ליצור פונקציה עם שגיאה, נבחר אקראית אחד מהאופרטורים AND/OR , ובוצעה החלפה (FLIP) של AND ב- OR או להיפך, כך נוצר ביטוי שגוי שיקרא Erroneous Expression. שגיאה כזו היא מאוד קשה למציאה שכן מרחב הכניסות הוא 2^{100}.

אורך הכרומוזום הוא 100 ביטים (ביט עבור כל איבר), ועל מנת למצוא שגיאה אנו למעשה צרכים למצוא כרומוזום שהרצתו על הפונקציה התקינה נותן תוצאו אחרת והרצתו על הפונקציה השגויה.

על כן הגדירו פונקציית הפיטנס הבאה:

$$F(t)=\begin{cases}1, & \text{if } Eval\_correct \neq Eval\_erroneous \\ 0, & \textit{respectively}\end{cases}$$

כלומר פונקציית הפיטנס מחזירה ערך בוליאני, "1" עבור תוצאה "טובה" (מצאנו שגיאה) "0" עבור תוצאה לא טובה (לא מצאנו שגיאה)

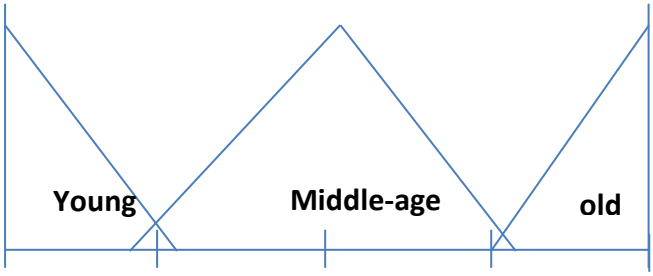
הבחינה נעשתה על 3 סוגי אלגוריתמים:

א. אלגוריתם קאנוני
ב. אלגוריתם GAVaPS
ג. אלגוריתם FAexGA בשלושה סוגי ערכי Fuzzy.

את הערכים ניתן לראות בטבלה הבאה:

פרמטר	FAexGA	GAVaPS	קאנוני
גודל אוכלוסיה	100	100	100
אורך כרומוזום	100	100	100
מספר מחזורים (generation size)	200	200	200
Pc	משתנה FL	0.9	0.9
Pm	0.01	0.01	0.01
שיטת בחירה (selection)	אקראי	אקראי	גלגל רולטה
שיטת crossover	1-point	1-point	1-point
שיטת mutation	Flip	Flip	Flip
חישוב Lifetime	Bi-Linear	Bi-Linear	-
Reproduction ratio	0.3	0.3	-
MaxLT	7	7	-
MinLT	1	1	-

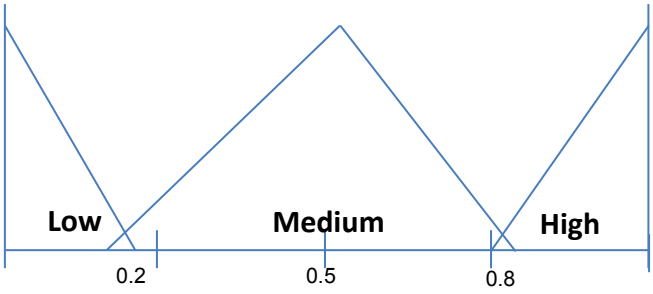
הם בדקו את אלגוריתם ה- FAexGA בשלושה ערכי קונפיגורציה של FL. את הערכים ופונקציית ההתאמה ניתן לראות באיור 10.



פרמטר	FAexGA#1	FAexGA#2	FAexGA#3
"young" upper limit	30	25	20
"old" lower limit	70	75	80

איור 10 – FaexGA פונקציית החברות גילאים בשלושת הקונפיגורציות

ובנוסף הם השתמשו בפונקציית ההתאמה לערכי ה- Pc באיור 11:



איור 11 – FaexGA, פונקציית חברות הסתברות שחלוף

את תוצאות הניסוי הם ריכזו באיור 12:

Measure\Algorithm	SimpleGA	GAVaPS	FAexGA#1	FAexGA#2	FAexGA#3
% Runs with solution	70%	60%	95%	70%	75%
% Solutions in the final population	1.267%	81.519%	99.934%	98.857%	87.183%

איור 12 – תוצאות ניסוי לבדיקת קופסא שחורה [7]

ניתן לראות בסיכום כי אלגוריתם FAexGA בכיוונן יעיל מגיע לתוצאות מעולות בבחירת מתארי בדיקה מתאימים לבדיקות "קופסא שחורה". הוא מגיע לתוצאות ב- 96% מהמקרים, וכמו כן מעל 99 אחוז מהכרומוזומים באוכלוסייה יכולים לשמש כקלט לבדיקת הקופסא השחורה.

אלגוריתם ה- GAVaPS מגיע גם הוא לתוצאות טובות עם 70 אחוזי הצלחה ומעל 82 אחוז מהאוכלוסיה ניתנת לשימוש. לעומתם אלגוריתם הקאנוני, מגיע אמנם לתוצאות ב- 70 אחוז מהמקרים אך פחות מ- 2 אחוז מהאוכלוסיה הם כרומוזומים בהם ניתן להשתמש כקלט למערכת הנבדקת.

9. מימוש והרצה של אלגוריתמים לצורך בדיקת תאימות (Regression test)

9.1 מטרות המימוש והבדיקה

המטרות שלי בבחינת האלגוריתם, ע"י קידוד והרצתו הי:

א. הבנת האלגוריתם בצורה טובה ע"י ניסיון בהרצתו.
ב. בחינת התאמתו בעולם האמיתי של בדיקת תאימות (regression), וביצוע שינויים נדרשים.
ג. הבנת המשמעות בכוונון הפרמטרים ומציאת הפרמטרים היעילים.

9.2 שינויים בין הבדיקה בעבודה זו למאמר המוזכר[7]

העבודה ינקה את מהותה מהמאמר שהוצג :

Last M.,Eyal, S. Kandel,A :Effective Black-Box Testing with Genetic Algorithms,Lecture Notes in Computer Science,2006,vol 3875. 2006, 134-148

אולם תוך כדי עבודה נערכו מספר שינויים והתאמות הן לאלגוריתמים עצמם והן לסביבת הבדיקה , וזאת כדי לטייבם לצורך בדיקות נסיגה (regression). את האלגוריתמים עצמם שדרגתי לצורך התאמת לסוג זה של בדיקות. שני האלגוריתמים הם:

GAVaPS_R – התאמה של האלגוריתם GAVaPS לעולם בדיקות הנסיגה.

FAexGA_R – התאמה של האלגוריתם FAexGA לעולם בדיקות הנסיגה.

לצורך כך בוצעו מספר שינויים קלים באלגוריתמים המקוריים. כלל השינויים יוצגו בסעיפים הבאים. מעבר לכך נעשו גם שינויים בסביבת הבדיקה עצמה של האלגוריתמים וזו על מנת להתאים את הסביבה לצורך הרצה ובחינת האלגוריתמים לביצוע בדיקות נסיגה.

9.2.1 ערכי פונקציית הפיטנס

במאמר המקורי [7] נעשה שימוש בפונקציית פיטנס בעלת שני ערכים בלבד 1 לתאימות גבוהה לפתרון ו- 0 לאין פתרון.

שהתחלתי לממש הבנתי שלא ניתן להשתמש בערך 0 לפונקציה מהסיבה הבאה והיא חישוב של ה- Lifetime של הכרומוזום.

חישוב ה- Lifetime הוא:

Lifetime(i)=

- **MinLT+ ήּ((Fitness(i)-MinFit)/(AvgFit-MinFit)) if AvgFit>=Fitness(i)**

- **0.5(MinLT+MaxLT)+ ή((Fitness(i)-AvgFit)/(MaxFit-AvgFit)) if AvgFit<Fitness(i)**

במחזור הראשון של הריצה כנראה שאין כלל התאמה לאף אחד מהכרומוזומים ולכן

AvgFit=MinFit=0

וכך קיבלנו חלוקה ב- 0 בחישוב.

בכדי למנוע את הבעיה עברתי לפונקציית פיטנס שמחזירה 0.2 במקום 0, ו- 0.8 במקום 1.

9.2.2. קריטריון סיום

כאשר קריטריון הסיום הוא מספר מחזורי האלגוריתם (generation Size) , התוצאות שמתקבלות לא מניחות את הדעת.

בעוד באלגוריתם הגנטי הקאנוני אין בעיה עם קריטריון סיום כזה, בשני האלגוריתמים האחרים הוא הרה אסון.

הסיבה לכך הינה פרמטר ה- Lifetime, מכיוון שכל כרומוזום יכול לחיות עד MaxLT מחזורים ואחריו הוא מת, לא מובטח ש"ילדיו" המידיים יתנו ערכי כשירות גבוהים, וזאת בגלל העובדה שפונקציית הכשירות היא בינרית (0.8/0.2).

על כן שהרצתי את האלגוריתמים ללא פרמטר עצירה נוסף למעט Generation Size קיבלתי תוצאות לא טובות.

מכיוון שמטרת בדיקת התאימות (Regression Test), היא מציאת ערך ש"מפיל" את הבדיקה, כלומר ערך שבהרצתו אנו מוצאים "באג". אזי אין משמעות להרצת האלגוריתם עד סופו ומספיק לעצור ברגע שאנו מזהים פתרון.

כאשר הוספתי פרמטר נוסף לסיום הריצה , קיבלתי תוצאות מעולות. **אי לכך באלגוריתמים המותאמים לבדיקות נסיגה GAVaPS_R,FAexGA_R , קריטריון העצירה הנו מציאת התאמה לפונקציית הכשירות, אין טעם וזה גם מזיק לאלגוריתם להמתין עד למספר מחזורי הבדיקה (Generation size).**

9.2.3. פרמטרי הבדיקה

במאמר [7] נבחנים מספר רב של פרמטרים, בהתאמת האלגוריתמים לבדיקות קופסא שחורה. במקרה של בדיקות נסיגה מספיק לדעתי לבחון 2 פרמטרים בלבד והם:

- מספר ההרצות (מתוך 100 הרצות של כל אלגוריתם) בהם קיבלנו תוצאות טובות (כלומר מצאנו את ה"באג" בתוכנה- אי התאמה של הגרסה החדשה לגרסה הישנה)
- מתוך ההרצות אשר הצליחו, מהו ממוצע המהירות בהן הפסיק האלגוריתם לרוץ – נמצא "באג".

9.2.4. פונקציית הבדיקה

פונקציית הבדיקה שבה השתמשתי היא שונה. לדעתי פונקציית הבדיקה שהשתמשו במאמר הינה תיאורטית מידי ולא מדמה את העולם האמיתי, על כן השתמשתי בפונקציית בדיקה אחרת. פונקציה מסוג זה ייתכן ונמצא גם בעולם האמיתי. השינוי של הפונקציה גם יכול לדמות תקלה שמופיעה לאחר שדרוג גרסה.

לטובת דימוי של פונקציית בדיקה מימשתי פונקציה פשוטה המקבלת 100 מספרים בערך [1,255] , הפונקציה מחזירה את הערך הגבוה ביותר. כדי לייצר מצב של "באג" בפונקציה (שאותו אמורה הבדיקה של המערכת לגלות- בדיקת נסיגה), יצרתי פונקציה תאומה לפונקציה הראשית עם "באג" מושתל בתוכה.

הפונקצייה ה"תקולה" החזירה תוצאה לא תקינה ברגע שסכום המספרים הראשון והאחרון לחלק לסכום המספרים השני ולפני האחרון קטן מ- 0.06:

$$(A1+An)/(A2+An-1)<0.06$$

מדובר על תקלה בהסתברות נמוכה מאוד שקשה מאוד לשחזר אותה בבדיקת נסיגה.

ובכך יצרתי שתי פונקציות שעליהן אני מריץ את אותם 100 איברים בכניסה: אם התוצאה זהה לא נמצא "באג" – הגרסה החדשה לא מקלקלת פונקציה שמומשה בגרסה הראשונה. אך אם היא שונה מצאתי ערכי כניסה אשר מוצאים את התקלה, שכן הגרסה החדשה משנה את פונקציונאליות הגרסה הישנה.

תקלות מהסוג שסימלצתי הינם תקלות מתרחשות בדרך כלל שמוסיפים לוגיקה לפונקציה קיימת שמוסיפה יכולת. לדוגמא תוספת של שדה תאריך לטופס רישום קיים, הטופס המקורי מחשב את גילו של המועמד לפי תאריך הלידה שלו. אנו רוצים לוודא שהפונקציה החדשה לא "מקלקלת" את הפונקציונליות של הפונקציה המקורית , בדוגמא שלנו שהתוספת של התאריך בטופס הרישום לא מקלקלת את חישוב גיל הנרשם על פי תאריך לידתו.

9.2.5. מספר חלוקות Fuzzy Logic

לצורך הבנת עומק אלגוריתם ה- FAexGA_R בצעתי משחק רחב בטבלאת ההצבה שקובעת את ערכי ה- FL, במאמר לא נעשתה בחינה למשל של ערכים שונים של ה- PC לפי גילאים, אני ביצעתי שינוי גם שם בכדי להבין היטב ולכוונן את התוצאות המתאימות של האלגוריתם.

9.3. שיטת הבדיקה

לצורך בדיקת האלגוריתם נעזרתי בסביבת פיתוח Visual C#.

בסביבה זו פיתחתי מימוש ל- 3 אלגוריתמים כפי שגם נעשה במאמר שפירטתי בסעיף הקודם: מימוש של אלגוריתם גנטי עם שיטת 1-point Crossover שיטת mutation מסוג FLIP, ושיטת בחירה Roulette Wheel.מימוש של אלגוריתם GAVaPS_R, ומימוש של אלגוריתם FAexGA_R.

לצורך ההבנה מימשתי את שתי פונקציות הבדיקה: האחת זהה לחלוטין לפונקציה הבוליאנית אשר הוצגה במאמר. השנייה פונקציה שונה שמתאימה יותר לדעתי לבדיקת נסיגה (Regression Test) ומוסברת בסעיף 9.2.4

את כלל הבדיקות בפונקציה מהסוג הראשון (זו שהוצגה במאמר) בצעתי בצורה מהירה רק לצורך ההבנה, ואת מירב הבדיקות בצעתי על הפונקציה החדשה אותה מימשתי.

9.4. שיטת הבדיקה והשימוש באלגוריתם גנטי:

כאמור למצוא את התקלה מימשתי 3 אלגוריתמים שונים:

הרעיון בכלל האלגוריתמים דומה:

1. קבע כרומוזום של 100 מספרים [1,255].
   - כל כרומוזום מורכב מ- 100 מספרים, כל אחד יכול להיות [1,255].
   - בחירת המספרים נעשית ראנדומלית ע"י התוכנה.
   - כל כרומוזום מחושב מחדש – אין העתקה של כרומוזומים בשלב היצירה שלהם.
2. הפעל את האלגוריתם- מהאפשרויות: קאנוני, GAVaPS_R, FAexGA_R:
   - כל אלגוריתם רץ לבד, אין עבודה במקביל.
   - קאנוני – האלגוריתם מומש על פי ההסבר בפרק 3.
   - GAVaPS_R – מימוש האלגוריתם נעשה על פי המוסבר בפרק 6 עם השינויים וההתאמות לאלגוריתם שפורטו בסעיף 9.2.
   - FAexGA_R – מימוש האלגוריתם נעשה על פי המוסבר בפרק 7 עם השינויים וההתאמות לאלגוריתם שפורטו בסעיף 9.2.
3. פונקציית הכשירות היא:

```
If CorrectFunction(chromosome)==FaultyFunction(chromosome)

Return 0.2;// not 0 like in the original paper [7]

Else

  Return 0.8;// not 1
```

כלומר ביצענו שימוש בפונקציית כשירות בינרית. במידה ושתי הפונקציות זהות יתקבל ציון כשירות נמוך (0.2) שכן לא נמצאה שגיאה בבדיקת הנסיגה. ובמידה ויש שוני בתוצאה תתקבל כשירות גבוהה (0.8) שכן נמצאה שגיאה וזו מהות בדיקת הנסיגה.

4. בצע את האלגוריתם 100 פעמים בצורה מלאה, ושמור את התוצאות לטובת סטטיסטיקה. הכוונה ביצוע כלל האלגוריתמים עד עצירה מהווה פעם אחת. יש לבצע 100 פעמים כאלה , כל פעם תוצאות הריצה יישמרו לדיסק, כך שבסוף כלל הריצות יהיה ניתן לנתח את התוצאות שנשמרו בקובץ.

9.5. פרמטרים לאלגוריתמים:

9.5.1.אלגוריתם קאנוני:

הפרמטרים שנעשה בהם שימוש באלגוריתם הגנטי הם:

Pc=0.7

Pm=0.01

Crossover – 1-point

Mutation – Flip

Population size- 120

Generation size - 200

Selection method – Roulette wheel

9.5.2.אלגוריתם עם אוכלוסייה משתנה מותאם נסיגה GAVaPS_R

Pc=0.7

Pm=0.01

Crossover – 1-point

Mutation – Flip

Population size- 120

Generation size - 200

Selection method – Random

Reproduction ratio – 0.3

MinLT- 1

MaxLT- 20

9.5.3.אלגוריתם עם אוכלוסייה משתנה וערך FL של Pc המותאם לבדיקות נסיגה FAexGA_R

Pc – changes with fuzzy logic

Pm=0.01

Crossover – 1-point

Mutation – Flip

Population size- 120

Generation size - 200

Selection method – Random

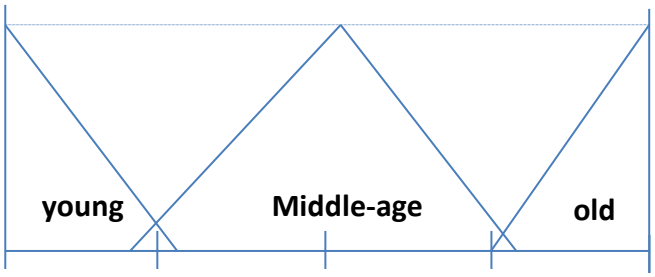
Reproduction ratio – 0.3

MinLT- 1

MaxLT- 20

9.6. חלוקה לגילאים: "young" "middle-age", "old":

לצורך מימוש אלגוריתם ה- FAexGA_R השתמשתי בפונקציית ההתאמה הבא:



איור 13 – FAexGA_R פונקציית החברות לגילאים

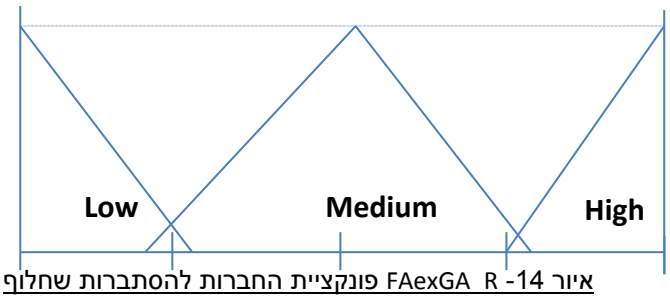
לכל גיל יש סף עליון ותחתון ,נוכל להשתמש בטבלה הבאה ולתאר את הגרף בצורה פשוטה וברורה יותר.

Young		Middle-age		old	
0	20	15	70	65	100

כאשר 0-20 ערכי young, 15-70 ערכי middle-age ו- 65-100 ערכי old.

9.7. חלוקה להסתברויות Pc: "Low" "Medium", "High":

לצורך מימוש אלגוריתם ה- FAexGA_R יש צורך להגדיר גם את פונקציית ההתאמה להסתברות השחלוף (Pc) ועל כן השתמשתי בגרף החלוקה הבא:



איור 14- FAexGA_R פונקציית החברות להסתברות שחלוף

כאשר הציר התחתון הוא בין 0 ל- 1 לערכי ה- Pc.

גם כאן אפשר לעזר באותה טבלה:

Low		Medium		High	
0	0.2	0.15	0.7	0.65	1

כאשר 0-0.2 לערכי Low, 0.15-0.7 לערכי Medium ו 0.65-1 ל High.

9.8. ערכים להרצה באלגוריתם FAexGA_R:

באלגוריתם זה השתמשתי ב- 12 קנפוגים שונים שהם וריאציות של 3 ערכי טבלה של גיל ו- 4 ערכי טבלה של הסתברות.

ערכי גיל:

Young		Middle-age		Old	
0	20	15	85	80	100
0	25	20	80	75	100
0	30	25	75	70	100

ערכי הסתברות:

Low		Medium		High	
0	0.50	0.45	0.8	0.75	1
0	0.3	0.25	0.8	0.75	1
0	0.3	0.25	0.85	0.80	1
0	0.32	0.15	0.85	0.80	1

9.9. סיכום הרצות:

כאמור כל הרצה היא למעשה 100 הרצות מלאות של כל אלגוריתם למציאת סטטיסטיקה.

הערכים שמדדתי הינם:

א. כמות ההרצות שמצאו פתרון מתוך 100 הרצות – למעשה ניתן ללמוד על יעילות האלגוריתם למציאת פתרון מפרמטר זה, ככל שהערך גבוה יותר כך האלגוריתם יותר יעיל למציאת פתרון.

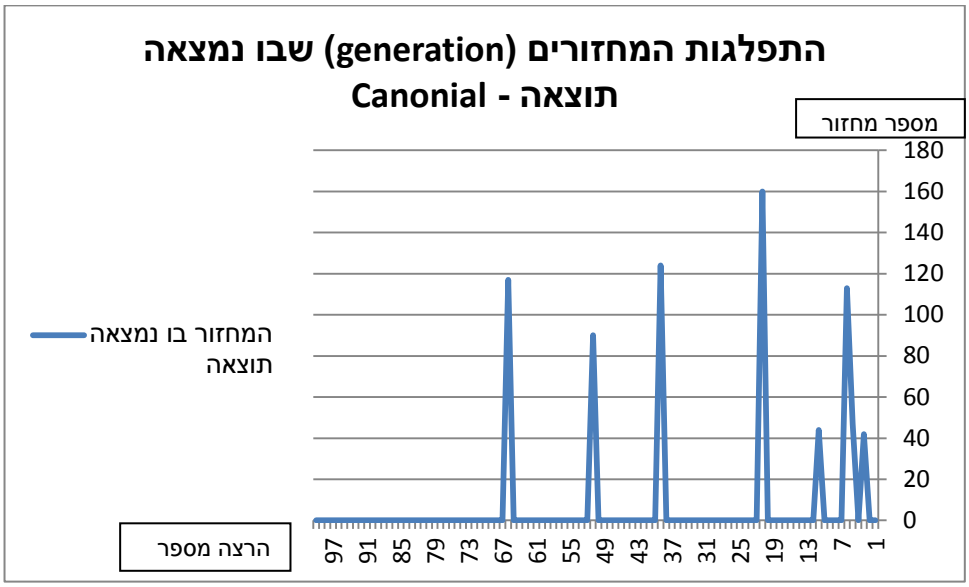
ב. מתוך ההרצות שמצאו פתרון- ממוצע המחזורים למציאת התשובה. ניתן ללמוד על מהירות ההתכנסות של האלגוריתם, ככל שהתוצאה נמוכה יותר האלגוריתם מתכנס מהר יותר ויותר יעיל מבחינת מהירות הריצה

ג. גודל האוכלוסייה הסופי בכל הרצה של האלגוריתם (מתוך 100 הרצות). המטרה היא להבין כמה זיכרון צורך כל אלגוריתם והאם הערכים יציבים או לא.

9.9.1.אלגוריתם קאנוני:

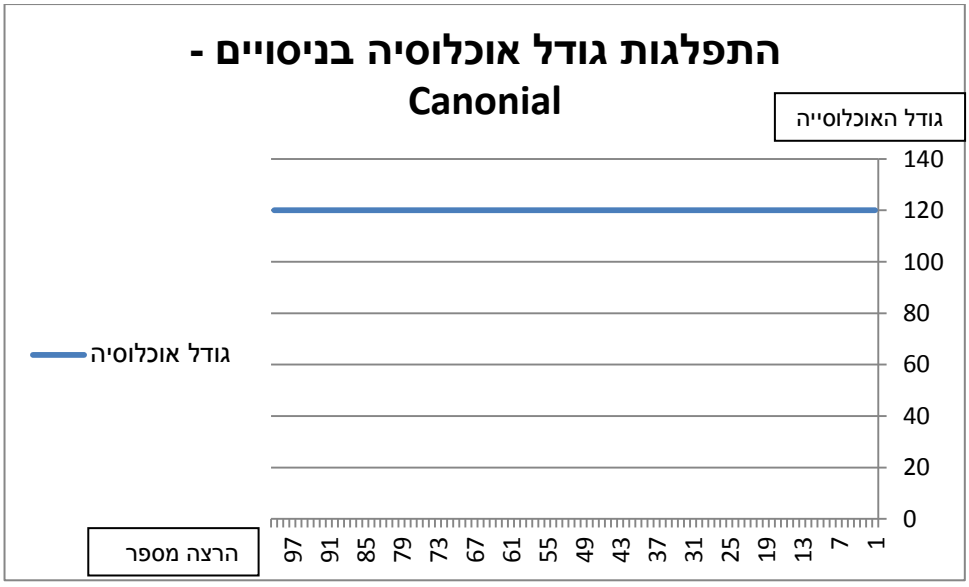
מתוך 100 הרצות האלגוריתם הצליח למצוא רק ב- 8 מהן פתרון, וזאת בממוצע של המחזור ה- 92 ב- 8 הרצות אלה (למציאת התשובה הנכונה).

מעבר לכך ניתן לראות באיור 15 את התפלגות המחזורים שבהם נמצאה תוצאה. ניתן לראות שבמקרים הבודדים שנמצאה תוצאה באלגוריתם זה, רובם היו מעבר למחזור ה- 100.



איור 15 – אלגוריתם קאנוני גרף התפלגות מחזורים למציאת תוצאה

גודל האוכלוסייה באלגוריתם זה הינו קבוע. מכיוון שהתחלנו עם גודל של 120 , ניתן לראות באיור 16 כי אכן אין שינוי בגודלה של האוכלוסייה בין ההרצות השונות.



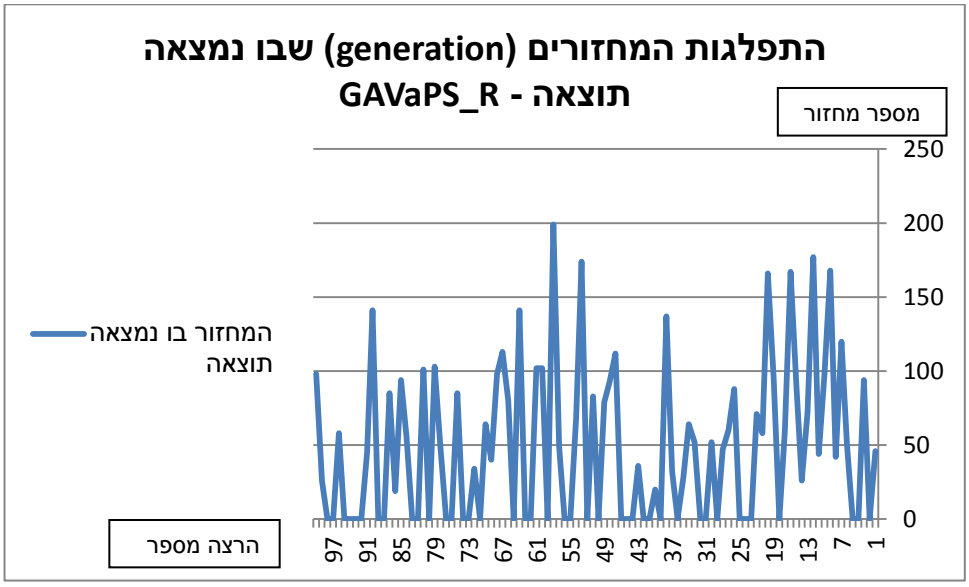
איור 16 - אלגוריתם קאנוני גרף התפלגות גודל אוכלוסיה

9.9.2.אלגוריתם עם אוכלוסייה משתנה המתואם לבדיקות נסיגה :GAVaPS_R

מתוך 100 הרצות – הצליח למצוא ב- 59 מהן פתרון (59 אחוז הצלחה!!).

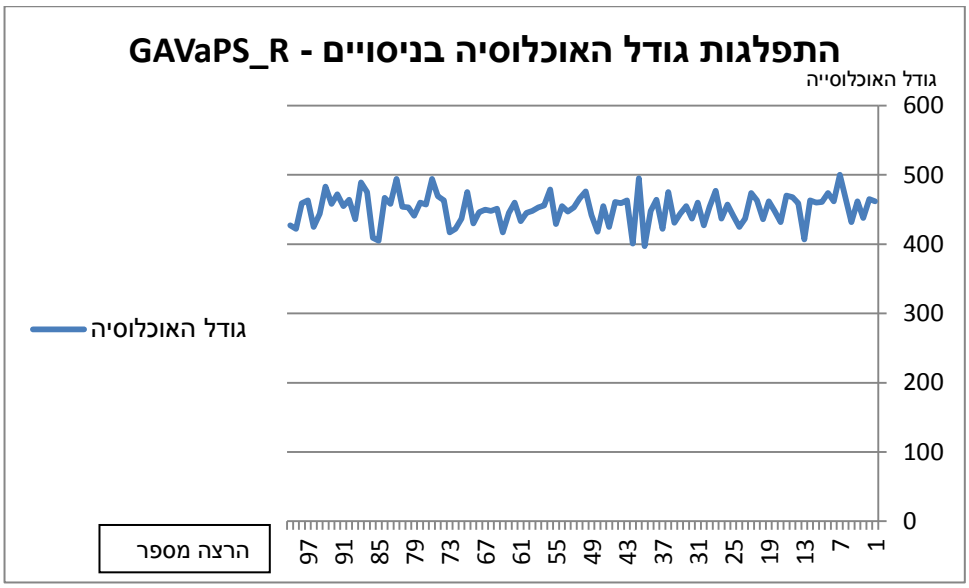
את הפתרון בהרצות שנמצא בהם פתרון הוא מצא במחזור ה- 82 בממוצע.

מעבר לכך ניתן לראות באיור 17 את התפלגות המחזורים שבהם נמצאה תוצאה. ניתן לראות שבמרבית המקרים אכן התקבלה תוצאה, וכן היא התקבלה לרוב מתחת למחזור ה- 100 של אותה הרצה.



איור 17 - אלגוריתם GAVaPS-R גרף התפלגות מחזורים למציאת תוצאה

גודל האוכלוסייה הינו משתנה כאמור, ניתן לראות שבאופן קבוע האלגוריתם שומר על גודל אוכלוסייה בין 400 ל- 500



איור 18 - אלגוריתם GAVaPS-R גרף התפלגות גודל אוכלוסייה

9.9.3.אלגוריתם ה- FAexGA_R:

כאמור באלגוריתם זה השתמשתי ב- 12 קנפוגים שונים שהם וריאציות של 3 ערכי טבלה של גיל ו- 4 ערכי טבלה של הסתברות.

ערכי גיל:

Young		Middle-age		Old		ערך מספר
0	20	15	85	80	100	1
0	25	20	80	75	100	2
0	30	25	75	70	100	3

ערכי הסתברות:

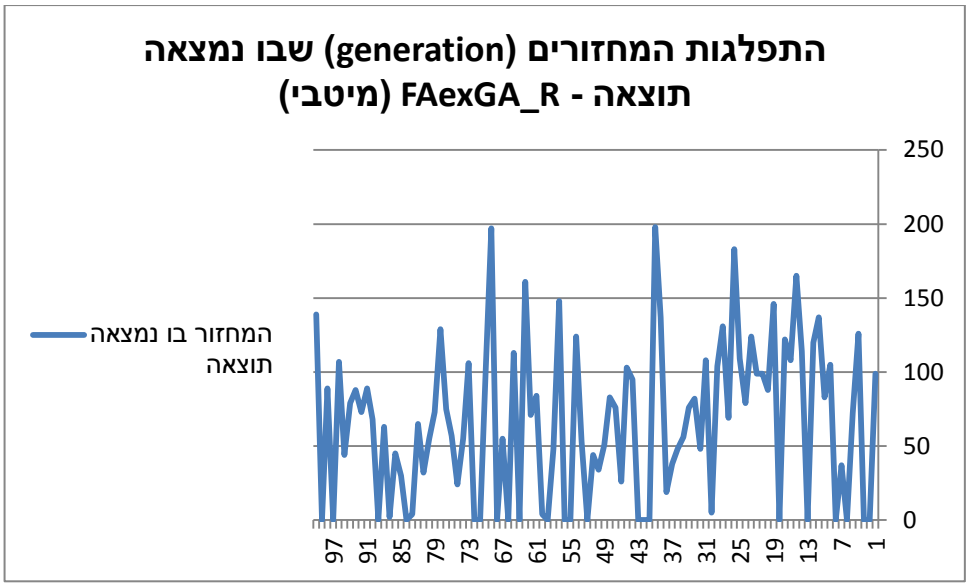
Low		Medium		High		ערך מספר
0	0.50	0.45	0.8	0.75	1	א
0	0.3	0.25	0.8	0.75	1	ב
0	0.3	0.25	0.85	0.80	1	ג
0	0.32	0.15	0.85	0.80	1	ד

בטבלה הבאה, איור 19,מסוכמות כלל הריצות עם התייחסות לממוצע המהירות בה נמצא פתרון , ולאחוזי ההצלחה במציאת הפתרון מכלל 100 ההרצות. ניתן לראות בקנפוג מוצלח ניתן להגיע לתוצאות מעולות.

		ערכי גיל 1	2	3
ערכי הסתברות	א	**78**	65	65	אחוז הצלחה במציאת פתרון
		84.5	75.98	79.5	ממוצע מהירות מציאת הפתרון
	ב	37	40	45	אחוז הצלחה במציאת פתרון
		87.5	88.75	79.9	ממוצע מהירות מציאת הפתרון
	ג	42	26	53	אחוז הצלחה במציאת פתרון
		95.9	94.1	87.04	ממוצע מהירות מציאת הפתרון
	ד	41	49	46	אחוז הצלחה במציאת פתרון
		78	87.5	77.3	ממוצע מהירות מציאת הפתרון

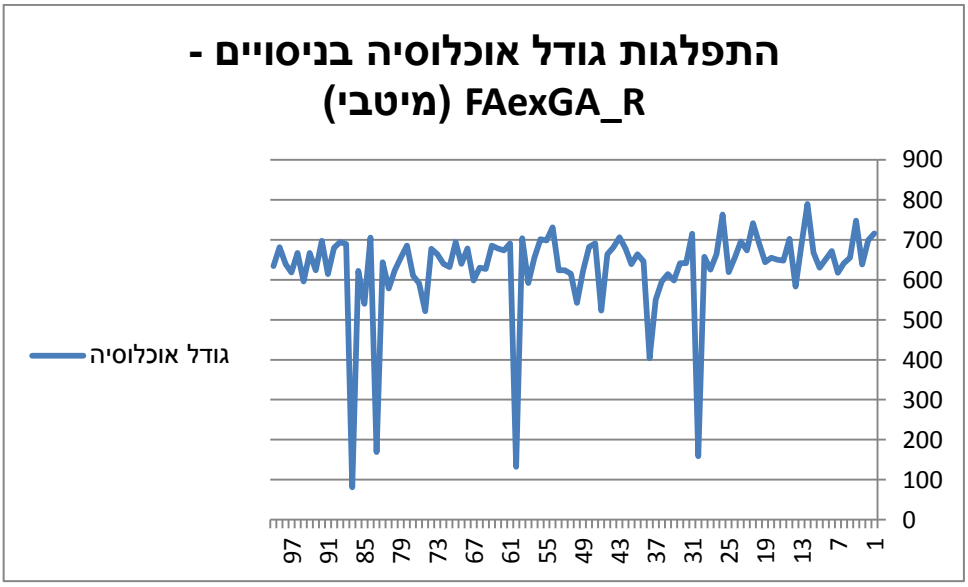
איור 19 – סיכום תוצאות הרצה אלגוריתם FAexGA-R

באיורים 20 ו-21 ניתן לראות את הפרמטרים הנוספים שנבדקו גם באלגוריתמים הקודמים. במקרה זה מצורף רק הגרף של ההרצה המוצלחת המודגש בטבלה הקודמת.



איור 20 - אלגוריתם FAexGA-R גרף התפלגות מחזורים למציאת תוצאה

ניתן לראות בטבלה את התפלגות המחזורים שבהם נמצאה תוצאה. ניתן לראות שבמרבית המקרים אכן התקבלה תוצאה, וכן היא התקבלה לרוב מתחת למחזור ה-100 של אותה הרצה. מאוד דומה לתוצאות שהתקבלו מהאלגוריתם ה- GAVaPS.



איור 21 - אלגוריתם FAexGA-R גרף התפלגות גודל אוכלוסייה

באלגוריתם זה גודל האוכלוסייה שהתקבל הינו יותר תזזיתי מאשר בשאר האלגוריתמים. ולרוב הוא בין 600 ל- 700 כרומוזומים באוכלוסייה. שזו כמות גדולה משמעותית מאשר באלגוריתמים האחרים

9.10 סיכום:

נרכז את כל ההרצות בטבלה אחת כאשר ניקח את התוצאה הטובה ביותר של אלגוריתם ה- FAexGA_R ולא את כל ה- 12.

נקבל את הטבלה באיור 22:

שם האלגוריתם	אחוזי הצלחה במציאת פתרון (יעילות)	ממוצע מחזור מציאת התשובה (מהירות)
קאנוני	8	92
GAVaPS_R	59	82
FAexGA_R – הקנפוג המיטבי	78	84.5

איור 22 – סיכום הרצות כלל האלגוריתמים בבדיקת נסיגה

ניתן לראות בבירור שהאלגוריתם המשלב אוכלוסייה משתנה וכמו כן הסתברות שחלוף (crossover) משתנה (FAexGA_R), הינו בעל ביצועים הטובים ביותר הן ביעילות והן במהירות מציאת הפתרון.

אם נשווה מצד שני את גודלי האוכלוסייה, אזי בעוד האלגוריתם הקאנוני מקבל ציון טוב של גודל אוכלוסיה יציב וקטן אך עם ביצועי מערכת גרועים. אלגוריתם ה- GAVaPS_R הנו בעל ביצועי זיכרון (גודל אוכלוסייה קטן יותר) מאשר מאלגוריתם ה- FAexGA_R.

10.סיכום העבודה

לעבודת הסיום היו מספר מטרות, הכרות עם עולם בדיקות התוכנה ,אלגוריתמים גנטיים, והשילוב האפשרי ביניהם.

בתחילת העבודה ביצעתי סקירה מקיפה על אלגוריתמים גנטיים, ובדגש על אלגוריתמים המשלבים פרמטרים משתנים. כל זאת לצד הבנת עולם הבעיה של "בדיקות תוכנה" והשפעה של אלגוריתמים גנטיים על פתרונות בעולם זה ובדגש על בדיקות פונקציונאליות – "קופסא שחורה".

חלק זה של העבודה הכיר בפניי את עולם בדיקות התוכנה על כלל מרכיביו והקשיים בהם הוא נתקל. בעיה קשה בעולם בדיקות התוכנה הינה ייצור מתארי בדיקה אשר יכסו את מירב האפשרויות, לטובת "הרגשת בטחון" עם איכות הבדיקה, שכן כפי שהוסבר לא ניתן להבטיח תקינות תוכנה ב- 100 אחוז.

עולם האלגוריתמים הגנטיים, נותן פתרון טוב לכל עולם האופטימיזציה ונראה כי הוא מתאים מאוד בעבור יצירת מתארי בדיקה, שכן מתארי הבדיקה בדרך כלל הינם בעלי מרחב עצום של אפשרויות והחוכמה היא לבצע בכמות יחסית מועטה של מתארי בדיקה, כיסוי מירבי והעלאת "הרגשת הביטחון" בתקינות התוכנה

במהלך העבודה בחרתי לבצע הכרות בלתי אמצעית עם אלגוריתמים בעלי פרמטרים משתנים ע"י מימושם והבנת מנגנון הריצה שלהם בעולם המעשי. כמו כן בחנתי התאמה של אלגוריתמים: קאנוני, FAexGA ו- GAVaPS לבדיקות נסיגה (regression test). אופטימיזציה של הפרמטרים המשפיעים על הרצת אלגוריתמים אלה למתן פתרון יעיל וטוב.

לאחר מימוש ולימוד של 2 אלגוריתמים עיקריים GAVaPS ו- FAexGA , ביצעתי בהם שינוי כדי להתאימם לעולם בדיקות הנסיגה ובכך יצרתי 2 אלגוריתמים משודרגים המותאמים לעולם של בדיקות נסיגה והם: GAVaPS_R ו- FAexGA_R.

על פי הניסיון שצברתי במימוש שני האלגוריתמים המוסבים לצד האלגוריתם הקאנוני, בביצוע בדיקת נסיגה (Regression) הגעתי למסקנות הבאות:

- אלגוריתמים בעלי פרמטרים משתנים יעילים יותר בפתרון בעיות אופטימיזציה הקשורות לייצור מתארי בדיקות בעולם "בדיקות התוכנה"
- אלגוריתם ה- GAVaPS_R הנו אלגוריתם יעיל מאוד, לצד פשטות יחסית בקנפוג הפרמטרים שלו. בבדיקה שערכתי אלגוריתם זה השיג יעילות של 59 אחוז במציאת פתרון לצד יעילות של 8 אחוז בלבד באלגוריתם קאנוני.
- אלגוריתם ה- FAexGA_R מורכב יותר למימוש ולקנפוג, שכן דיאגרמות ה- Fuzzy משפיעות מאוד על יעילות האלגוריתם. נאלצתי להריץ מגוון רחב של אפשרויות לקבלת תשובה איכותית. אם כי קנפוג טוב שלו מעלה את היעילות ל 78 אחוז במציאת הפתרון !!

- לצד יעילות במציאת הפתרון באלגוריתם ה- FAexGA_R הוא משתמש בגדלי אוכלוסייה גדולים משמעותית מזה של האלגוריתמים האחרים (בפרמטרים אשר גרמו לו להיות יעיל יותר), ועל כן הינו "בזבזן" משאבים גדול יותר.
- מהירות הריצה של אלגוריתם ה- FAexGA_R הינה הנמוכה מכלל האלגוריתמים, לא השקעתי במימוש בייעול של מהירות הריצה, אך בכל מקרה האלגוריתם מבצע פעולות נוספות של Defuzzication , פעולות שלוקחות זמן.

המלצתי לאחר מגוון ההרצות שעשיתי על בחינת בדיקות רגרסיה הייתי בוחר דווקא באלגוריתם ה- GAVaPS_R שכן הפשטות והמהירות שלו, גוברות על היעילות אך המסובכות בקנפוג והאיטיות בריצה של אלגוריתם ה- FAexGA_R.

מכיוון שהאלגוריתם כאמור פשוט יותר לכוונון ניתן פשוט להריצו מספר מחזורים גדול יותר ובכך לקבל תוצאה דומה לזו של ה- FAexGA_R עם פשטות גדולה יותר בקנפוג.

בדיקות תוכנה זהו מקצוע שמכיל בתוכו הרבה התמחויות ובמגוון נושאים. אחד מהקשיים הגדולים של בדיקות תוכנה הינם בייצור מתארי בדיקה אשר בסופו של תהליך יחזקו את "תחושת הביטחון" של הבודק בתקינות התוכנה. המגוון העצום של אפשרויות מתארי הבדיקה השונים, מרמזים על כך שניתן להשתמש באלגוריתמים גנטיים לצורך בניית מתארי בדיקה אלה.

בעבודה זו פרסתי מגוון רחב של אלגוריתמים כולל מימושם ובחינתם לצורך עזרה בגיבוש מתארי בדיקות אשר יענו על הצורך של הבודק. טבען של מערכות תוכנה היא בייצור גרסאות תוכנה כל תקופת זמן לצורך תיקוני באגים וגיבוש של תכולות נוספות. חלק מהבדיקות שנערכים אליהם בסיום גרסא הינם בדיקות נסיגה (Regression) שבודקות את תקינות התכולות הקודמות של המערכת- או במילים אחרות, האם הגרסה החדשה נאמנה לגרסה הקודמת ולא פגמה בה עבור תכולות קיימות. בעבודה זו ניסיתי לחקות סוג זה של בדיקה ולבחון התאמה של אלגוריתמים גנטיים בעלי אוכלוסייה משתנה ליצירת מתארי בדיקה מתאימים.

בסוף הבדיקה ניתן לאמור כי אלגוריתם גנטי בעל אוכלוסייה משתנה מתאים מאוד עבור משימת ייצור מתארי בדיקות לבדיקת רגרסיה. אולם הבדיקה גם קבעה כי האלגוריתם הפשוט יותר ללא השימוש ב- Fuzzy Logic הוא גם זה היעיל יותר מבחינת כוונון ופשטות.

העבודה עסקה בייצור מתאר בדיקות עבור פונקציה פשוטה וסינטטית. בעולם האמיתי המערכות הנבדקות הן הרבה יותר סבוכות מהפונקציה הפשוטה אותה מימשתי כמערכת תחת בדיקה (system under test). אי לכך ניתן בעתיד ולא בעבודה זו להרחיב את הבחינה לבדוק אותה עבור מערכת תוכנה סבוכה יותר. על מערכת או תת-מערכת מהעולם האמיתי.

לדוגמא אפשר לבדוק את האלגוריתם על דף אינטרנט המבצע רישום לשירות, הדף מקבל פרמטרים ועל פיהם מבצע רישום של אדם לשירות. פעם בכמה שבועות משנים דף זה – מוסיפים לו פרסומות, שינוי עיצוב ועוד. המטרה היא לבצע בדיקת

נסיגה של הדף החדש מול הדף הישן. לצורך כך ניתן לבנות פונקציית כשירות אשר בודקת את הפלט של דף זה לפני ביצוע הרישום בפועל, אם הפלט זהה מחזירים 0.2 אם לאו 0.8, וכך ממשים את האלגוריתם GAVaPS_R שמוסבר בעבודה.

אפשרות נוספת היא לבצע כיוונון נוסף של אלגוריתם ה- FAexGA_R, ע"י שימוש במשפחות בצורות שונות של FL, למשל מעבר ממשולש לטרפז, ייתכן ושינויי כזה יועיל על ביצוע הכיוונון של האלגוריתם ובכך יהפוך אותו פשוט יותר וקל למשתמש.

נספח

11. סקירה קצרה Fuzzy Logic

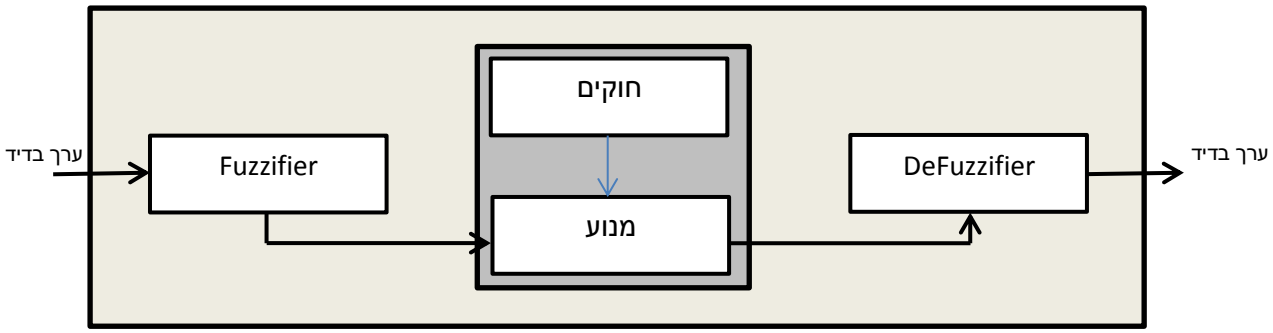
בסקירה זו אתן הסבר על עקרונות ה- Fuzzy Logic (FL) , הסקירה מבוססת על שני המאמרים [10] [11], לצורך העמקה ניתן לפנות למאמרים אלה ישירות.

11.1. כללי:

מערכת המבוססת FL, בשמה הלועזי Fuzzy Logic System(FLS) , יכולה להיות מוגדרת כמיפוי של כניסה לא לינארית למוצא בדיד.

המערכת מכילה את המרכיבים הבאים: Fuzzifier, חוקים (rules), מנוע מסקנות (inference engine), ו- defuzzifier.

מבנה המערכת נראה על פי התרשים באיור 23:



איור 23 – מבנה סכימתי של מערכת FLS

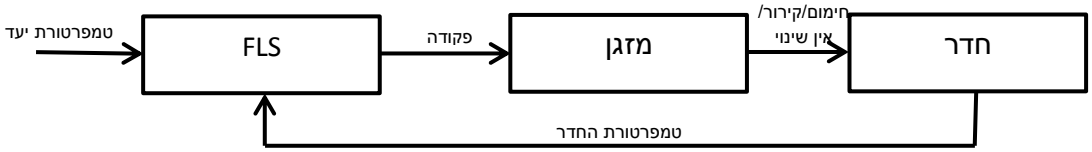
השלבים הכללים בחישוב ה- FL במערכת זו יהיו:

א. הגדרת המשתנים הלשוניים (Linguistic variables) והמושגים (terms) בתוך המשתנים - אתחול
ב. בניית פונקציית החברות (membership functions) - אתחול
ג. בניית בסיס החוקים (אתחול)
ד. המרת ערך בדיד בכניסה לערך מעורפל (Fuzzy), ע"י שימוש בפונקציית החברות (Fuzzifier)
ה. חישוב תוצאות החוקים (מנוע החוקים)
ו. איחוד תוצאות החוקים (מנוע חוקים)
ז. המרת המידע שהצטבר לערכים בדידיים (Defuzzifier)

במהלך ההסבר נשתמש בדוגמא של מיזוג אוויר, על פי הגדרת המערכת הבאה:

קיימת מערכת מיזוג אוויר הנשלטת ע"י FLS. המערכת מכוננת את הטמפרטורה בחדר ע"י קבלת טמפרטורת הכניסה ומיזוג על פי טמפרטורת היעד. ה- FLS באופן קבוע מנתר את הטמפרטורה ובהתאם נותן פקודות של "חימום" או "קירור למזגן.

ניתן לראות את פעולת המערכת באיור 24:



איור 24 – מערכת מיזוג מבוססת FLS

11.2. משתני ה- FL והשימוש בהם:

כעת נעבור בפירוט על כלל המשתנים והדגמתם על ה- FLS שהגדרנו בסעיף הקודם.

11.2.1.משתנים לשוניים (Linguistic variables):

אלו הם משתנים של הכניסה ו/או היציאה, אשר מוגדרים בשפה לשונית רגילה ולא בשפה מספרית/מתמטית.

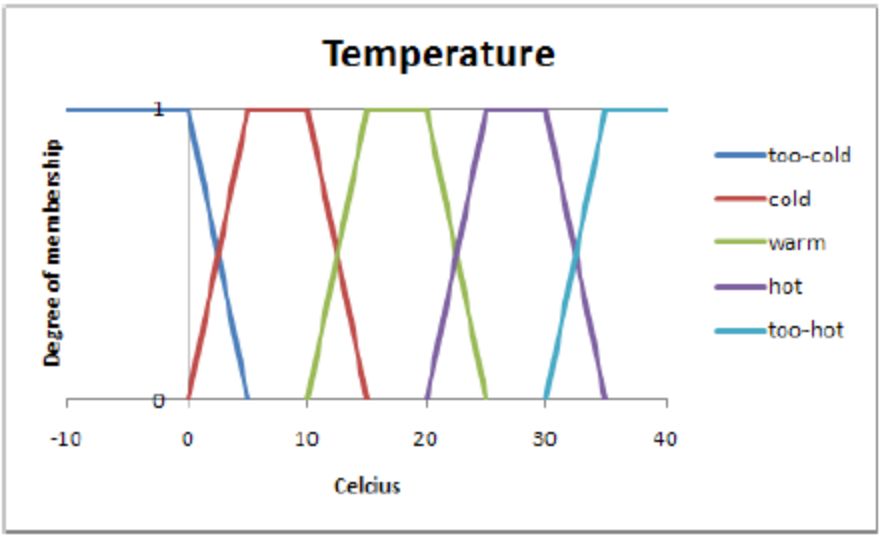
במשפחה של משתנים לשוניים מגדירים תת מושגים (terms) אשר מפרטים את הפריסה של המשתנה הלשוני.

דוגמא', בהמשך להגדרת מערכת מיזוג האוויר, אנו נגדיר טמפרטורה t כמשתנה לשוני שמציין את הטמפרטורה בחדר. כדי להגדיר טמפרטורה משתמשים במושגים המוכרים כמו "חם" ו"קר". לכן אפשר להגדיר t=[too-cold,cold,warm,hot,to-hot]. כפריסה של המשתנה טמפרטורה, וכל ערך נקרא מושג (term).

11.2.2.פונקציית החברות (membership function):

בפונקציית החברות נעשה שימוש בשלבים ה- fuzzification וה- defuzzification לצורך הפיכה ממשתנה בדיד למעורפל ולהיפך.

ניתן לראות בתרשים באיור 25 את פונקציית החברות של טמפרטורה, כלומר איזה ערך של מעלות נמצא באיזו משפחה. חשוב לשים לב שהעקרון הוא שערכי הטמפרטורה במעבר בין משפחות הם מעורפלים, כלומר טמפרטורה יכולה להיחשב גם too-cold וגם cold.



איור 25 - פונקציית החברות לטמפרטורה [11]

ישנם מספר סוגי גרפים שיכולים להגדיר פונקציית חברות, בדוגמא ניתן לראות טרפזים, אך ניתן לעשות שימוש גם במשולשים, גם ב- singletone וגם בצורות אחרות.

11.2.3.חוקי FL (Fuzzy Rules):

חוק נוצר בכדי להגדיר את מוצא המערכת אל מול הפרמטרים בכניסה. זהו חוק פשוט בפורמט של IF-THEN.

דוגמא לחוקים על מערכת מיזוג האוויר :

	Fuzzy Rules
1.	IF (temperature is *cold* OR *too-cold*) AND (target is *warm*) THEN command is *heat*
2.	IF (temperature is *hot* OR *too-hot*) AND (target is *warm*) THEN command is *cool*
3.	IF (temperature is *warm*) AND (target is *warm*) THEN command is *no-change*

ניתן גם לסכם את כלל החוקים לכדי טבלה/מטריצה שמסכמת את כלל החוקים במערכת כמו בדוגמא הבאה:

temperature/target	too-cold	cold	warm	hot	too-hot
too-cold	no-change	heat	heat	heat	heat
cold	cool	no-change	heat	heat	heat
warm	cool	cool	no-change	heat	heat
hot	cool	cool	cool	no-change	heat
too-hot	cool	cool	cool	cool	no-change

11.2.4.פעולות על משתני FL(Fuzzy Sets Operators):

כדי לקבוע את הערך של כל חוק במערכת ה- FLיש צורך בהגדרת חוקי פעולות, שכן לא מדובר במתמטיקה נומרית, אלא מעורפלת (fuzzy).

באם μA ו- μB הם פונקציית החברות עבור A ו-B. הפעולות בשימוש התדיר הם AND- OR והם מומרים לאופרטורים Max (עבור OR) ו – Min (עבור AND). כאשר NOT מקבל את הערך הבא:

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x)$$

לאחר שמחשבים את התוצאה מכל חוק התוצאות צריכות להתחבר כדי לקבל תוצאה סופית.

ישנן מספר דרכים לחבר את הערכים והם מפורטים בטבלה הבאה:

Operation	Formula
Maximum	$Max\{\mu_A(x), \mu_B(x)\}$
Bounded sum	$Min\{1, \mu_A(x) + \mu_B(x)\}$
Normalized sum	$\frac{\mu_A(x)+\mu_B(x)}{Max\{1, Max\{\mu_A(x'), \mu_B(x')\}\}}$

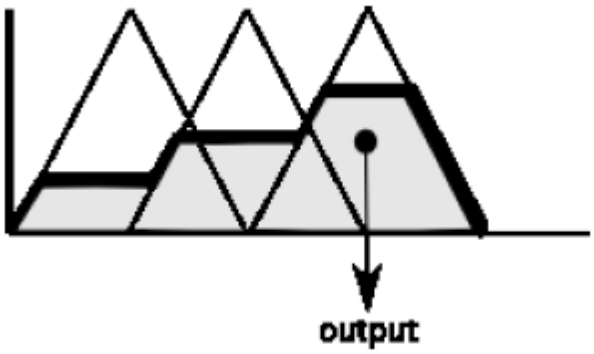
הדרך הנפוצה ביותר הינה סכימה על פי מתודת Maximum.

11.2.5.defuzzification

לאחר השלב הקודם יש ברשותנו מגוון רחב של ערכים מעורפלים אותם אנו רוצים להפוך לערכים בדידים לצורך הגדרת מוצא המערכת. זוהי מטרת שלב זה.

נעשה שימוש בפונקציית החברות של משתני המוצא כדי לחשב שלב זה.

למעשה אנו מקבלים בסוף השלב הקודם ערכים שנראים בפונקציית החברות לפי הגרף באיור 26:



איור 26 – קביעת ערך בדיד על פי פונקציית חברות [11]

כלומר קיבלנו ערך לא בדיד שמושפע מחישוב החוקים בסעיף הקודם. המטרה היא להפוך את המוצא לערך בדיד. לכך ישנם מספר שיטות כאשר הנפוצה ביותר היא center of gravitiy שהיא למעשה מחשבת ומוצאת את הערך שהוא מרכז שיווי המשקל של האנטגרל מתחת לגרף.

סיכום של כלל השיטות מופיעות בטבלה הבאה:

Operation	Formula
Center of Gravity	$U = \frac{\int_{min}^{max} u\,\mu(u)\,du}{\int_{min}^{max} \mu(u)\,du}$
Center of Gravity for Singletons	$\frac{\sum_{i=1}^{p} [u_i\,\mu_i]}{\sum_{i=1}^{p} [\mu_i]}$
Left Most Maximum	$U = inf(u'),\ \mu(u') = sup(\mu(u))$
Right Most Maximum	$U = sup(u'),\ \mu(u') = sup(\mu(u))$

כאשר הערכים של הטבלה מפורטים כאן:

Variable	Meaning
U	result of defuzzification
u	output variable
p	number of singletons
μ	membership function after accumulation
i	index
min	lower limit for defuzzification
max	upper limit for defuzzification
sup	largest value
inf	smallest value

12.מקורות

[1]Preface to Glen Myers, Art of structure Testing. New York: John Wiley, 1979

[2]Bill Hetzel, The Complete guide to software testing, John Wiley & sons , Inc.

[3] Darrell W.: A Genetic Algorithm Tutorial, Statistics and Computing, Vol 4 Nov 2002, 65-85

[4] Holland J.,:Adaption in Natural and Artifical systems. University of Michigan Press.

[5] Klir G. J.., and Yuan B.: Fuzzy Sets and Fuzzy Logic: Theory and Applications,Prentice-Hall inc., 1995.

[6] Last M., and Eyal , S.: A Fuzzy-Based Lifetime Extension of Genetc Algorithms, Fuzzy Sets and Systems, Vol. 149, Issue 1, January 2005, 131-147

[7] Last M.,Eyal, S. Kandel,A :Effective Black-Box Testing with Genetic Algorithms,Lecture Notes in Computer Science,2006,vol 3875. 2006, 134-148

[8] Arabas J.,Michalewicz Z.,Milawka J.: GAVaPS – a Genetic Algorithm with varying population size

[9] K. Deb, S. Agrawal, Understanding interactions among genetic algorithm parameters, in: W. Banzhaf, C. Reeves (Eds.),Foundations of Genetic Algorithm 5, Morgan Kaufmann, San Francisco, CA, 1998, pp. 265–286.

[10] Fuzzy control programming. Technical report, International Electrotechnical commission, 1977.

[11] J. Mendel. Fuzzy Logic Systems for engineering: a tutorial. Proceeding of the IEEE, 83(3):345-377, Mar 1995.

[12] S,Miles.M,Harman,M,Luck: Evolutionary Testing of Autonomous Software Agents. AAMAS 2009, 8th international conference on autonomous and multiagent system, may 2009

[13]Srivastava. P.R, Tai-hoon.K: Application of genetic Algorithms in software testing. International Journal of software engineering and its applications Vol.3 No.4 october 2009

[14] Buhler.O, Wegener.J: Evolutionary functional testing. Computer and Operations Research. 35(10):3144-3160. 2008

[15] Wegener.J,Baresel. A, and Sthamer.H : suitability of Evolutionary Algorithms for Evolutionary testing.26th Annual international computer software and applications conference, Oxford.England, August 26-29, 2002.

[16] Korel.B: automated software date generation. IEEE transaction on software engineering, 16(8) august 1990

[17] Jones.B.F, Sthamer.H, and Eyres.D.E: automatic structural testing using genetic algorithms. Software Engineering Journal. Pages 299-306. September 1996.

[18] McMinn.P: Search-based software test data generations: A Survey. The department of computer science, university of Sheffield