

האוניברסיטה הפתוחה
המחלקה למתמטיקה ולמדעי המחשב

עבודה מסכמת בנושא
סקירת אלגוריתמים בהשראת הטבע לפתרון
בעיית ה-SCS

העבודה המסכמת מוגשת כחלק מהדרישות לקבלת תואר
"מוסמך למדעים" M.Sc. במדעי המחשב
החטיבה למדעי המחשב
האוניברסיטה הפתוחה

על ידי
רענן מנור
ת.ז. 023679202
דואר אלקטרוני: raananm@hotmail.com
כתובת: תל אביב, קהילת סלונים 14 דירה 35, פלאפון: 057-8147024.

העבודה הוכנה בהדרכתה של ד"ר מירי אביגל

ספטמבר
2012

תוכן עניינים

.....2	תוכן עניינים	
.....3	רשימת האיורים	
.....3	רשימת הטבלאות	
.....4	רשימת האלגוריתמים	
.....5	תקציר	1
.....6	מבוא	2
.....8	בעיית מציאת מחרוזת-על משותפת קצרה ביותר	3
.....10	סקירת אלגוריתמים לפתרון בעיית ה- SCS	4
.....10	אלגוריתמים מוקדמים	4.1
.....10	אלגוריתמים היוריסטיים בסיסיים	4.2
.....13	אלגוריתמים מעל-היוריסטיים (METAHEURISTICS)	4.3
.....14	אלגוריתמים גנטיים	4.4
.....16	אלגוריתם גנטי מבוסס היוריסטיקה	4.5
.....21	אופטימיזציה קן הנמלים (ACO – Ant Colony Optimization)	4.6
.....28	אלגוריתם ממטי עם תיקון מקומי חלקי	4.7
.....32	שילוב MA ו- Branch and Bound	4.8
.....38	אופטימיזצית אב טיפוס אבולוציונית - POEMS	4.9
.....51	כוורת דבורים מלאכותית	4.10
.....53	הצגת אלגוריתם למציאת SCS ע"י ABC – כוורת דבורים	5
.....54	מימוש אלגוריתם למציאת SCS ע"י ABC	5.1
.....66	יישום ותוצאות	6
.....66	אלגוריתמים היוריסטיים	6.1
.....69	אלגוריתמים גנטיים מבוססים היוריסטיקה	6.2
.....70	אלגוריתם ממטי עם חיפוש מקומי	6.3
.....72	אלגוריתם מבוסס התנהגות נמלים בקן	6.4
.....74	אלגוריתם מבוסס התנהגות דבורים בכוורת - ABC	6.5
.....77	סיכום	7
.....77	כיווני מחקר עתידיים	8
.....78	רשימת המקורות	9
.....79	Abstract	10
.....80	Contents	11

רשימת האיורים

.....42	איור 1 - הבדלים במציאת כיסוי בעזרת Align _{orig} ו- Align _{mod}
.....44	איור 2 - חישוב כיסוי C עבור החלק הקבוע

רשימת הטבלאות

.....19	טבלה 1 – אחוז שיפור מחרוזות אקראיות גודל משתנה
.....19	טבלה 2 – אורך מחרוזת-על ממחרוזות דומות
.....20	טבלה 3 – אורך מחרוזת-על ממחרוזות מיוחדות
.....25	טבלה 3 – פרמטרים להרצת אלגוריתם ACO
.....26	טבלה 5 – אורך ואחוז שיפור מחרוזת-על עבור מחרוזות אקראיות ודומות
.....27	טבלה 6 – אורך ואחוז שיפור מחרוזת-על עפ"י עומק הסתכלות קדימה
.....27	טבלה 7 – אורך מחרוזת-על עבור מקרים מיוחדים
.....30	טבלה 8 – פרמטרים להרצת MA
.....31	טבלה 9 – אורך מחרוזת-על הקצרה, ממוצע וסטיית תקן MA על מחרוזות אקראיות
.....31	טבלה 10 – אורך מחרוזת-על הקצרה, ממוצע וסטיית תקן MA על 158-SARS
.....35	טבלה 11 – הגדרת פרמטרים לאלגוריתם BS-MA
.....36	טבלה 12 – אורך מחרוזת-על קצרה ממוצעת ואחוז שיפור מחרוזות אקראיות אורך משתנה
.....36	טבלה 13 – אורך מחרוזת-על קצרה ממוצעת ואחוז שיפור 158-SARS
.....40	טבלה 14 - פעולות עדכון אפשריות ב- POEMS
.....47	טבלה 15 - פרמטרים לביצוע הרצות האלגוריתם האבולוציוני ב- POEMS
.....48	טבלה 16 – אורך מחרוזת-על קצרה, ממוצעת, סטיית תקן, וכמות חישובים על 158-SARS
.....48	טבלה 17 – אורך מחרוזת-על קצרה, ממוצעת, סטיית תקן, וכמות חישובים על 1269-SARS
.....49	טבלה 18 – אורך מחרוזת-על קצרה, ממוצעת, סטיית תקן, וזמן ריצה על 1269-SARS
.....49	GAP=20%, 20 הרצות
.....50	טבלה 19 – אורך מחרוזת-על קצר, ממוצע, סטיית תקן, וזמן חישוב על DNA_N1000_K500
.....50	טבלה 20 – אורך מחרוזת-על קצר, ממוצע, סטיית תקן, וזמן חישוב על Protein_N500_K1000
.....53	טבלה 21 – דוגמאות לחישוב כשירות באלגוריתם ABC
.....55	טבלה 22 – דוגמא למחיקת תו בעזרת AlignAndFix
.....68	טבלה 23 – השוואת תוצאות הרצת אלגוריתם היוריסטיים
.....70	טבלה 24 – תוצאות הרצת אלגוריתמים GA/H1 בהשוואה לאלגוריתמים היוריסטיים בסיסיים
.....71	טבלה 25 – תוצאות הרצת אלגוריתם MA בהשוואה לאלגוריתמים היוריסטיים
.....73	טבלה 26 – תוצאות הרצת אלגוריתמים ACO עם וללא הסתכלות קדימה/התחשבות באורך
.....74	טבלה 27 – תוצאות הרצת אלגוריתם ACO בהשוואה לאלגוריתמים היוריסטיים
.....75	טבלה 28 – פרמטרים והגדרות לאלגוריתם ABCI
.....76	טבלה 29 – תוצאות הרצת אלגוריתמים ABC בהשוואה לאלגוריתמים אחרים

רשימת האלגוריתמים

.....11.....	אלגוריתם 1 - Heuristic MM
.....12.....	אלגוריתם 2 - Heuristic H1
.....12.....	אלגוריתם 3 - Heuristic H2
.....13.....	אלגוריתם 4 - Heuristic H3
.....17.....	אלגוריתם 5 - Heuristic GA/H1
.....22.....	אלגוריתם 6 - Ant k Iteration
.....29.....	אלגוריתם 7 - Heuristic LS
.....33.....	אלגוריתם 8 - General Hybrid BS and MA
.....38.....	אלגוריתם 9 - POEMS בסיסי
.....41.....	אלגוריתם 10 - POEMS-f חישוב $\text{Align}_{\text{mod}}$
.....43.....	אלגוריתם 11 - POEMS-fw
.....46.....	אלגוריתם 12 - POEMS-3
.....51.....	אלגוריתם 13 - ABC General Algorithm
.....52.....	אלגוריתם 14 - ABC SCS Algorithm
.....55.....	אלגוריתם 15 - alignAndFix
.....57.....	אלגוריתם 16 - FitnessAndFix
.....58.....	אלגוריתם 17 - LocalChange
.....59.....	אלגוריתם 18 - LocalSearchInWindow
.....60.....	אלגוריתם 19 - CoverSequenceToLength
.....61.....	אלגוריתם 20 - InitializationPhase
.....62.....	אלגוריתם 21 - EmployedBeePhase
.....63.....	אלגוריתם 22 - OnlookersBeePhase
.....64.....	אלגוריתם 23 - ScoutBeePhase
.....65.....	אלגוריתם 24 - RM ABC SCS

1 תקציר

בעיית ה- Shortest Common Supersequence (להלן SCS) היא בעיה מתחום בעיות המחרוזות. הבעיה ידועה כבעיית אופטימיזציה קומבינטורית קשה לפתרון שיש לה מספר שימושים באפליקציות פרקטיות כמו זיהוי אב קדמון משותף לזנים עפ"י מחרוזת ה-DNA שלהם. בעבודה זו נסקרות מספר פתרונות מסוגים שונים: פתרונות המבוססים על הוריסטיקה, על תהליכים אבולוציוניים וגנטיים, ועל חיקוי מנגנונים ומערכות בטבע כמו התנהגות נמלים בקן ודבורים בכוורת. העבודה סוקרת עבודות ופרסומים קיימים ומציגה את הנתונים שפורסמו בעבודות אלו. במסגרת העבודה מומשו חלק מהאלגוריתמים ואומתו התוצאות המוצגות בפרסומים אלו אל מול המימוש בפועל. העבודה כוללת פיתוח של אלגוריתם חדש לפתרון בעיית ה- SCS המבוסס על התנהגות דבורים בכוורת. האלגוריתמים פותח על בסיס האלגוריתם הכללי לאופטימיזציה נומרית המבוסס על התנהגות הדבורים בכוורת תוך ביצוע ההתאמות הנדרשות לפתרון הבעיה. תוצאות הרצת אלגוריתם זה הושוו לתוצאות האלגוריתמים הקיימים ונמצא כי ביצועי אלגוריתמים זה אינם נופלים ולעיתים אפילו משפרים את הביצועים של אלגוריתמים אבולוציוניים אחרים.

2 מבוא

בעיית ה- Shortest Common Supersequence (להלן SCS) הינה בעיה מתחום בעיות המחרוזות. בעיות מתחום המחרוזות נחקרות לעומק בעולם מדעי המחשב. הסיבה לכך היא שניתן לייצג תהליכים או אובייקטים בעולם ובטבע בצורה מופשטת ע"י מחרוזות של תווים. דוגמא לכך היא המידע הגנטי הקיים ב- DNA של התאים. בכל תא מן החי נמצא DNA אשר מורכב מ- 4 סוגי מולקולות (nucleotides) – Adenine, Cytosine, Guanine ו- Thymine. בפועל ניתן לייצג DNA ע"י מחרוזת המורכבת מתווים מעל שפה בעלת 4 אותיות (A, C, G, T).

בצורה כללית ומופשטת ניתן לתאר בעיה מתחום בעיות המחרוזות. המטרה היא למצוא מחרוזת s המאפשרת ייצוג "טוב" של קבוצת מחרוזות אחרת (L). משמעות ייצוג "טוב" היא שניתן להגיע ממחרוזת s לכל אחת מהמחרוזות ב- L ע"י מחיקת תווים בלבד מתוך מחרוזת s . במקרה זה מחרוזת s תקרא ה- Common Supersequence (מחרוזת-על משותפת) של קבוצת המחרוזות L . הגדרה לא רשמית של מציאת ה- SCS היא מציאת המחרוזת s הקצרה ביותר המהווה Common Supersequence של קבוצת מחרוזות L [8].

נוכל לזהות אב קדמון בעזרת SCS. אם ניקח DNA של מספר זנים, נייצג את ה- DNA שלהם ע"י מחרוזות ונבצע מציאת SCS, נוכל לאתר את ה- DNA (או חלקו) של הזן שהיה האב הקדמון של קבוצת הזנים המקורית.

בעיית מציאת ה- SCS הינה NP-hard והיא נשארת כזו גם תחת הגבלות על כמות האותיות בשפה או על גודל המחרוזות. גם אם נגביל את אורך המחרוזות ל- 2 תווים או נגביל את המחרוזות להיות מעל שפה בעלת 2 אותיות, לא נוכל למצוא אלגוריתם פולינומיאלי לפתרון הבעיה.

אלגוריתמים בשיטת ה- Dynamic Programming מאפשרים להגיע לפתרון לבעיית ה- SCS אך הם מתקשים להגיע לתוצאות טובות במקרים בהם כמות המחרוזות ב- L גדולה. נדרש להפעיל אלגוריתמים לאופטימיזציה לצורך מציאת SCS קצר ככל האפשר. באלגוריתמים אלו נדרש להפעיל פונקציה היוריסטית לביצוע החלטות בשלבי האלגוריתם. כך מבצעים באלגוריתם חמדניים, למשל ה- MAJORITY MERGE (MM), בו בונים את מחרוזת הפתרון s עפ"י התו השכיח הנמצא בראש המחרוזות ב- L . הסבר כללי לבעיית ה- SCS והצגת אלגוריתם MM ושיפורים נוספים מתוארים ב- [1].

בעבודה זו נתמקד בפתרון בעיית ה- SCS ע"י שיטות ואופטימיזציות שקיבלו את השראתם מתהליכים המתרחשים בטבע (bio-inspired). לדוגמא, שיטות ואלגוריתמים אבולוציוניים (EA) קיבלו את השראתם מהתהליך האבולוציה בטבע. על בסיס זה פותח אלגוריתם ממטי (Memetic MA – Algorithm) בו ממשיכים את תהליך האבולוציה הטבעי ומוסיפים לו תיקון המדמה התנהגות

(חיקוי), כפי שמתואר במאמר שפורסם ב- 2005 [5]. מנגד קיימת אופטימיזציה איטרטיבית מבוססת אב-טיפוס (POEMS – Prototype Optimization with Evolved Improvement Steps) לפתרון בעיית ה- SCS. ב- POEMS מבצעים בחירת אב טיפוס, שיהיה הבסיס לתהליך אבולוציוני, לצורך יצור אב הטיפוס לשלב הבא. סדרה של 3 מאמרים פורסמו בנושא POEMS ופתרון בעיית ה- SCS בעזרתו [2-4]. ניתן לראות במאמרים את תהליך התפתחות הפתרון עד להגעה לתוצאות מרשימות. חיקוי התהליכים המתרחשים בכוורת דבורים והדרך שלהם להגיע למקורות מזון בצורה יעילה מהווה בסיס לתהליך אופטימיזציה נוסף למציאת פתרון לבעיית ה- SCS. לאחרונה פורסם מאמר המתאר פתרון לבעיית ה- SCS בצורה זו [6-7]. דבר דומה מבוצע ע"י הסתכלות על התנהגות קן הנמלים [8]. בשנת 1999 פורסם מאמר בנושא ישום זה.

3 בעיית מציאת מחרוזת-על משותפת קצרה ביותר

בעיית מציאת מחרוזת-על משותפת קצרה ביותר (Shortest Common Supersequence Problem - SCSP) היא בעיה קלאסית מתוך עולם עיבוד המחרוזות. ב-SCSP נדרש למצוא את המחרוזת הקצרה ביותר s , כך שכל מחרוזת s_i מתוך קבוצת מחרוזות L מוכלת ב- s , כלומר שניתן להגיע מ- s אל כל המחרוזות s_i ע"י השמטת תווים מ- s בלבד.

ה-SCSP הינה בעיית אופטימיזציה קומבינטורית. קיימות קבוצות מחרוזות שמאד קשה למצוא עבורם את מחרוזת-העל הקצרה ביותר, כלומר הפתרון האופטימאלי אינו ניתן לחישוב בזמן סביר. הצורך למצוא את מחרוזת-העל המשותפת עבור קבוצת מחרוזות קיים עבור שימושים מעשיים מתחום הביו-אינפורמטיקה. לדוגמא [10] ניקח קבוצת מחרוזות DNA של יצורים מזנים שונים. שאלה רלוונטית בנוגע לאבולוציה של זנים אלו היא האם קיים אב קדמון משותף לכל הזנים. אם ניקח בחשבון שהמוטציות היחידות במהלך האבולוציה של הזנים המדוברים היו מחיקת תווים בלבד, הרי שחישוב מחרוזת-העל המשותפת הקצרה ביותר של מחרוזות ה-DNA, תהיה ה-DNA של האב הקדמון.

נגדיר פורמלית מהי מחרוזת-על, ומחרוזת-על משותפת (ההגדרות הפורמליות נלקחו מ-[5]).

יהי אלפבית Σ . תהינה s ו- r מחרוזות מעל האלפבית Σ . ב- s ו- r רצף של 0 או יותר תווים מתוך

אלפבית Σ . מחרוזת ϵ מוגדרת כמחרוזת ריקה. תהינה a ו- b תווים המקיימים $a \neq b, a, b \in \Sigma$.

מחרוזת s היא מחרוזת-על של r (מסומן $s > r$) עפ"י ההגדרה הרקורסיבית הבאה:

$$s > r \triangleq \begin{cases} \text{true if } r = \epsilon \\ \text{false if } r \neq \epsilon \text{ and } s = \epsilon \\ s' > r' \text{ if } r = \alpha r' \text{ and } s = \alpha s' \\ s' > r \text{ if } r = \alpha r' \text{ and } s = \beta s' \end{cases}$$

ההגדרה של מחרוזת-על מבוססת על 4 תנאים:

1. במידה ו- r הינה מחרוזת ריקה, s היא מחרוזת-על של r . בודאי שכל מחרוזת s הינה

מחרוזת-על של מחרוזת ריקה (ניתן להגיע אליה ע"י מחיקת כל התווים ב- s).

2. במידה ו- r אינה מחרוזת ריקה, ו- s היא מחרוזת ריקה בודאי ש- s אינה מחרוזת-על של r .

זאת מכיוון שממחרוזת ריקה אפשר להגיע ע"י מחיקת תווים רק למחרוזת ריקה ובודאי שלא ניתן להגיע ל- r שאינה מחרוזת ריקה.

שני התנאים הראשונים מהווים את האינדיקציות המידיות והברורות להגדרת מחרוזת-העל (בדומה לתנאי עצירה באלגוריתם רקורסיבי).

שני התנאים הבאים מהווים את ההתקדמות בהגדרה הרקורסיבית במקרים בהם s ו- r אינם מחרוזות ריקות. בתנאים אלו מבצעים הקטנה של המחרוזת s (ואולי גם את r) בתו אחד וממשיכים לבדוק את התנאים על שארית המחרוזות.

3. במידה והמחרוזות r ו- s מתחילות באותו תו אזי נדרש לבדוק האם s ללא התו הפותח (s') הינה מחרוזת-על של r ללא התו הפותח (r'). אם עובדה זו נכונה אזי אפשר למחוק תווים ב- s' וכך להגיע ל- r' . ע"י מחיקת אותם תווים אפשר להגיע גם מ- s ל- r .

4. במידה והמחרוזות r ו- s מתחילות בתו שונה אזי נדרש לבדוק האם s ללא התו הפותח (s') הינה מחרוזת-על של r . אם עובדה זו נכונה אזי אפשר למחוק תווים ב- s' וכך להגיע ל- r . ע"י מחיקת אותם תווים ומחיקת התו הפותח ב- s אפשר להגיע מ- s ל- r .

בצורה יותר פשוטה, $s > r$ מרמזת על כך שניתן להגיע מ- s ל- r ע"י השמטת תווים בלבד. נגדיר את בעיית מציאת מחרוזת-על משותפת קצרה ביותר (SCSP):

עבור קבוצת מחרוזות L בעלת m מחרוזות $\{s_1, s_2, \dots, s_m\}$, $s_i \in \Sigma^*$, יש למצוא את המחרוזת הקצרה ביותר s המקיימת $s > s_i$ עבור כל $s_i \in L$.

ניתן להגדיר את SCSP בצורה של בעיית החלטה:

עבור קבוצת מחרוזות L בעלת m מחרוזות $\{s_1, s_2, \dots, s_m\}$, $s_i \in \Sigma^*$, ומספר חיובי שלם k , האם קיימת מחרוזת s באורך קטן או שווה ל- k המקיימת $s > s_i$ עבור כל $s_i \in L$.

בעיית ה-SCS הינה NP_HARD גם כאשר מטילים מגבלות על Σ או L , כמו למשל הגבלת כמות התווים באלפבית ל-2 ($|\Sigma|=2$) או הגבלת כמות התווים הכוללת במחרוזות לקבוע

$$(\max(s_i \in L | |s_i|) < \text{const}).$$

למרות זאת, אופי ה-NP הוא בעיקר במצבי קיצון (Worst Case) ולכן יתכן וטיפול ממוקד בבעיה יאפשר לזהות סיבוכיות אמיתית עבור המקרים היותר שכיחים. שיטה לחישוב סיבוכיות כזו נקראת parameterized complexity. שיטה זו ניגשת לבעיה ממספר כיווני מבט, הבנה של המבנה הפנימי של הבעיה ובידוד פרמטר (אחד או יותר) שע"י הגבלתו ניתן להגיע לפתרון פולינומיאלי לבעיה. במצב זה ניתן להגיע לסיבוכיות של $O(f(k)n^c)$, כאשר k הוא הפרמטר המבודד, f הינה פונקציה התלויה רק ב- k , c הוא קבוע שלא תלוי ב- n או k ו- n הוא גודל הבעיה. בעיות שניתן להפעיל עליהם parameterized complexity נקראות FPT – fixed parameter tractable.

מספר פרמטרים מועמדים להגדרה עבור SCSP. עבור בעיית ההחלטה ניתן להגדיר את גודל המחרוזת המקסימאלי – k . במצב זה אם כמות התווים בשפה הוא קבוע, אז הבעיה היא FPT מכיוון שיש $|\Sigma|^k$ מחרוזות, ואפשר פשוט לבדוק את כולם. למרות זאת, אין תועלת להגדרה זו מכיוון שבדרך כלל $k \geq \max |s_i|$ (ולכן בעצם הוא מושפע מגודל הקלט).

אם נגדיר את כמות המחרוזות m כפרמטר ונוסיף כפרמטר גם כמות התווים בשפה, בעיית SCS נשארת none-FPT כפי שהוצג ב-[6]. לנושא זה יש משמעות בשימושים הביולוגיים של הבעיה – 4

הנוקלואידים ב-DNA אינם הופכים את הבעיה להיות FPT – כלומר אין אלגוריתם שייתן בוודאות פתרון בזמן פולינומיאלי.

4 סקירת אלגוריתמים לפתרון בעיית ה-SCS

4.1 אלגוריתמים מוקדמים

קיימים אלגוריתמים למציאת פתרון אופטימאלי לבעיית ה-SCS המבוססים על תכנות דינאמי. אלגוריתמים אלו מתאימים לבעיות המכילות כמות קטנה של מחרוזות או מחרוזות באורך קצר. נפח הזיכרון הנדרש לתהליך החיפוש עבור בעיות המכילות כמות גדולה של מחרוזות (או אורך מחרוזות גדול) אינו סביר.

המצאות הבעיה ב- none-FPT מחייבת לנקוט בגישה היוריסטית לפתרון בעיית ה-SCS. תהליך פשוט למציאת מחרוזת-על משותפת כלשהי ניתן לבצע ע"י חישוב מחרוזת-על בצורה מדורגת, כל פעם רק על 2 מחרוזות בלבד. בצורה זו מקטינים את כמות המחרוזות ב- L עד להגעה למחרוזת אחת המהווה מחרוזת-על משותפת לכלל המחרוזות. האלגוריתם מבוצע כך: בוחרים 2 מחרוזות אקראיות מ- L , מבצעים חישוב s כמחרוזת-על שלהם, ומוסיפים את s לקבוצה L במקום 2 המחרוזות שנבחרו. כך יבוצע עד לקבלת מחרוזת בודדת ב- L . מחרוזת זו היא בודאי מחרוזת-על משותפת של כל המחרוזות המקוריות ב- L . קיימות היוריסטיקות המבוססות על תהליך זה. הפונקציה ההיוריסטית תגרום לבחירה של 2 המחרוזות הנכונות בכל שלב כך שמחרוזת-העל המשותפת שתישאר בסוף התהליך תהיה קצרה יותר. אלגוריתם בסיסי זה נקרא אלגוריתם H_0 .

4.2 אלגוריתמים היוריסטיים בסיסיים

שיטה היוריסטית נפוצה ומקובלת לפתרון בעיית ה-SCS ממומשת באלגוריתם ה-Majority Merge (MM) [5,7,10]. אלגוריתם זה משמש כבסיס למגוון אלגוריתמים מתקדמים נוספים. ה-MM הינו אלגוריתם חמדני שבונה את s כמחרוזת-על משותפת בצורה אינקרימנטלית ע"י הוספת התו הנפוץ ביותר שנמצא בראש רשימת המחרוזות ב- L , והורדת התו מראש המחרוזות ב- L לצורך המשך החיפוש.

אלגוריתם 1 - Heuristic MM

Heuristic MM ($L = \{s_1, \dots, s_m\}$)

```

1: let  $s \leftarrow \epsilon$ 
2: do
3:   for  $a \in \Sigma$  do let  $v(a) \leftarrow \sum_{s_i = as'_i} 1$ 
4:   let  $b \leftarrow \max\{v(a) | a \in \Sigma\}$ 
5:   for  $s_i \in L, s_i = bs'_i$  do let  $s_i \leftarrow s'_i$ 
6:   let  $s \leftarrow sb$ 
7: until  $\sum_{s_i \in L} |s_i| = 0$ 
8: return  $s$ 

```

ניתן לראות שבשורה 3 מחשבים עבור כל תו באלפבית את סכום הפעמים שהתו נמצא בראש מחרוזות ב- L . בוחרים את התו המופיע מקסימום פעמים בראש המחרוזות (שורה 4), ומבצעים את פעולות הנדרשות לצורך המשך התהליך – מחיקת התו הנבחר מראש כל המחרוזות (שורה 5) ושרשור התו בסוף מחרוזות הפתרון (שורה 6). ביצוע התהליך הנ"ל עד שכל המחרוזות ב- L ריקות (שורה 7).

אלגוריתם ה-MM לא מביא בחשבון שאורך המחרוזות ב- L יכול להיות שונה. במצב בו אורך המחרוזות שונה, סבירות גבוהה לכך שמחרוזות קצרות יסתיימו ראשונות. במצב זה יעילות הפונקציה ההיוריסטית נפגעת מכיוון שהרוב נלקח מתוך קבוצה יותר קטנה. זו הסיבה להעדיף תווים מהמחרוזות הארוכות על פני תווים מהמחרוזות הקצרות. קיים שיפור ל-MM [7] כך שבבחירת התו הבא להוספה יילקח בחשבון גודל המחרוזות שהתו נמצא בראשם. כל מחרוזת מקבלת משקל לפי אורכה והתווים עם המחרוזות היותר ארוכות מקבלות עדיפות בחישוב ה"רוב". פתרון כזה ניתן ליישום ע"י החלפת שורה 3 בשורה:

```

3:   for  $a \in \Sigma$  do let  $v(a) \leftarrow \sum_{s_i = as'_i} |s'_i|$ 

```

עבור כל תו מחשבים את סכום אורכי המחרוזות שהתו נמצא בראשם (חישוב האורך ללא התו עצמו). באלגוריתם H1 (WMM – Weight Majority Merge) [7] הממש שיפור זה נמצא גם טיפול במקרה בו קיימים מספר תווים בעלי אותו סכום אורכים. במקרה כזה הבחירה תהיה אקראית.

להלן אלגוריתם H1:

אלגוריתם H1 - 2

Algorithm H1 ($L=\{s_1\cdots,s_m\}$)

```
1: let  $s \leftarrow \epsilon$ 
2: do
3:   let  $\Sigma' \leftarrow \{a \in \Sigma | wsum(a) = \max\{wsum(b) | b \in \Sigma\}\}$ , where  $wsum(a) = \sum_{s_i = as'_i} |s'_i|$ 
4:   let  $b \leftarrow$  random letter from  $\Sigma'$ 
5:   for  $s_i \in L$ ,  $s_i = bs'_i$  do let  $s_i \leftarrow s'_i$ 
6:   let  $s \leftarrow sb$ 
7: until  $\sum_{s_i \in L} |s_i| = 0$ 
8: return  $s$ 
```

שיפור נוסף לאלגוריתם MM הבסיסי ניתן לביצוע ע"י הסתכלות קדימה על המשך התווים, ולא רק על התו הנמצא בראש המחרוזות. כלומר נבחר את התו הבא להכנסה למחרוזת הפתרון ע"י בחינת המצב שנוצר לאחר ההכנסה של התו. להלן אלגוריתם H2 המממש שיפור זה:

אלגוריתם H2 - 3

Algorithm H2 ($L=\{s_1\cdots,s_m\}$)

```
1: let  $s \leftarrow \epsilon$ 
2: do
3:   for  $a \in \Sigma$  do
4:      $P_a = \max\{wsum_a(b) | b \in \Sigma\}$ , where  $wsum_a(b)$  is the sum of the lengths of string that  $b$  in the front of  $L-a$ , and  $L-a$  is the set of string after removing all  $a$ 's from the front.
5:   let  $\Sigma' \leftarrow \{a \in \Sigma | wsum(a) + P_a = \max\{wsum(b) + P_b | b \in \Sigma\}\}$ 
6:   let  $\beta \leftarrow$  random letter from  $\Sigma'$ 
7:   for  $s_i \in L$ ,  $s_i = \beta s'_i$  do let  $s_i \leftarrow s'_i$ 
8:   let  $s \leftarrow s\beta$ 
9: until  $\sum_{s_i \in L} |s_i| = 0$ 
10: return  $s$ 
```

עבור כל תו באלפבית מבצעים סימולציה של המצב של רשימת המחרוזות L לאחר הוצאת התו, וחישוב המשקל המרבי שנקבל במצב זה. ההחלטה על התו הנבחר תהיה לפי סכום האורכים של התו הנמצא בראש המחרוזות, והמצב לאחר בחירת התו (שורה 5). במקרה של תיקו במצב זה, בחירה אקראית של התו מבין התווים בעלי התוצאה הטובה ביותר.

שיפור נוסף לאלגוריתם MM מבוצע ע"י הסתכלות קדימה עד לסיום שלב ה-MM, בחינת התוצאה המיטבית, ובחירת התו שמיצר מחרוזת-על קצרה ביותר. לכאורה, אין באלגוריתם זה שיפור מול

אלגוריתם MM הבסיסי. העניין העיקרי שאלגוריתם זה משפר הוא היכולת להתגבר על בחירה אקראית במצב בו קיימים מספר תווים אשר מגיעים לפתרון בעלי אורך מחרוזת-על זהה. במצב כזה, בחירה בתו בצורה אקראית לא תאפשר שיפור בתוצאות. באלגוריתם זה, בעת הגעה למצב של תיקו, ההחלטה היא לקחת דווקא את התו שמיצר מחרוזת ארוכה יותר (כאשר מנתחים בצורה מצומצמת את כל המחרוזות המתחילות בתו המועמד):

אלגוריתם H3 - 4 Heuristic

Algorithm H3 ($L = \{s_1 \dots, s_m\}$)

```

1: let  $s \leftarrow \epsilon$ 
2: do
3:   let  $\Sigma' \leftarrow \{a \in \Sigma | H1(L-a) = \min\{|H1(L-b)| b \in \Sigma\}\}$ 
4:   let  $\Sigma'' \leftarrow \{a \in \Sigma | H1(L_a) = \max\{|H1(L_b)| b \in \Sigma'\}\}$  where  $L_a$  is the set of string in  $L$  beginning with  $a$ 
5:   let  $\beta \leftarrow$  random letter from  $\Sigma''$ 
6:   for  $s_i \in L, s_i = \beta s'_i$  do let  $s_i \leftarrow s'_i$ 
7:   let  $s \leftarrow s\beta$ 
8: until  $\sum_{s_i \in L} |s_i| = 0$ 
9: return  $s$ 

```

עבור כל תו באלפבית מבצעים פתרון מלא H1 (WMM) על קבוצת המחרוזות לאחר הוצאת התו מראש המחרוזות ב- L . עפ"י התוצאה של הפעלת H1 בוחרים בקבוצת התווים המייצרים אורך מחרוזת-על משותפת קצרה ביותר. מתוך הקבוצה הנ"ל נדרש לבחור את התו להכנסה למחרוזת התוצאה (בהנחה שיש יותר מתו אחד כזה). עבור כל תו מקבוצת המועמדים נפעיל את H1 על קבוצה מצומצמת של מחרוזות – רק המחרוזות שמתחילות בתו הנבדק. נבחר את התו שמיצר את המחרוזת הארוכה ביותר כמחרוזת-על משותפת של קבוצת המחרוזות המצומצמת (שורה 4). הרעיון של לוגיקה זו הוא שכאשר מחרוזת-העל המשותפת של קבוצת מחרוזות זו ארוכה, יהיה קושי ליצר מהם מחרוזת-על משותפת קצרה ולכן רצוי להקטין אותם כמה שיותר ולבחור בתו המוביל אותם כבר עכשיו. במקרה של תיקו גם במצב זה תבוצע בחירה אקראית של התו מבין התווים בעלי התוצאה הטובה ביותר.

4.3 אלגוריתמים מעל-היוריסטיים (METAHEURISTICS)

בסעיפים הקודמים הוצגה בעיית ה-SCS והוסבר כי הבעיה היא מתחום בעיות ה-NP-HARD כלומר – אין אלגוריתם פולינומיאלי למציאת פתרון אופטימאלי לבעיה. הוצגו שיטות היוריסטיות בסיסיות לפתרון הבעיה. השיטות הבסיסיות פועלות עפ"י "חוקי אצבע" לצורך קבלת החלטה הגיונית בעת ההתקדמות לכיוון מציאת פתרון לבעיה. אלגוריתמים מעל-היוריסטים [12] משתמשים באלמנטים אקראיים בחיפוש אחר פתרון אופטימאלי ובאלמנטים של חיפוש מקומי. גם באלגוריתמים אלו אין התחייבות למציאת פתרון אופטימאלי. הדרך לפתרון היא ע"י התקדמות בשיטה של ניסוי וטעייה (trial and error) למציאת פתרונות נכונים ואופטימאליים לבעיה מורכבת

בזמן סביר. מורכבות הבעיה לא מאפשרת לסרוק את כל הפתרונות האפשריים לבעיה. הרעיון הוא להגיע לאלגוריתם יעיל ופרקטי שיעבוד עבור רוב המקרים (ובעיקר המקרים בעלי חשיבות גבוהה) ויאתר פתרונות איכותיים לבעיה.

קיימים 2 רכיבים עקרוניים בכל אלגוריתם מעל-היוריסטי: רכיב ממקד (intensification) ורכיב מגוון (diversification). הרכיב המגוון נועד ליצור פתרונות מגוונים, באזורים שונים. הרכיב המתמקד מחפש אחר פתרונות מקומיים טובים אבל באותו אזור, כלומר שיפורים מקומיים. שני רכיבים אלו פועלים בשילוב עם יכולת הבחירה בפתרון הטוב ביותר. הרכיב המתמקד מתקדם מקומית בחיפוש אחר פתרון אופטימאלי, ואילו הרכיב המגוון מונע מהאלגוריתם להתכנס למינימום מקומי. שילוב טוב בין 2 רכיבים אלו יאפשר הגעה לפתרון אופטימאלי גלובאלי.

אלגוריתמים מעל-היוריסטיים ניתנים לסיווג בצורות שונות. ניתן לחלקם לאלגוריתמים מבוססי אוכלוסייה ולאלגוריתמים מבוססי אב-טיפוס. באלגוריתמים מבוססי אב-טיפוס משתמשים בפתרון יחיד, ומנסים בצורות שונות לעדכן אותו כך שבסוף התהליך הוא יהפוך לפתרון אופטימאלי. באלגוריתמים מבוססי אוכלוסייה משתמשים בפתרונות רבים שבסוף התהליך, אחד מהם יהווה את הפתרון האופטימאלי לבעיה.

במהלך השנים למד האדם למצוא פתרון לבעיות בשיטה של ניסוי וטעייה. שיטת הניסוי וטעייה מחייבת החלטה על כיוון ההתקדמות ללא ידיעה מוקדמת האם הכיוון יניב תוצאות משופרות. בתנאי אי ודאות ניסה האדם לחקות תופעות והתנהגויות טבעיות על מנת לבחור בכיוון ההתקדמות שייתן תוצאות מיטביות. חיקוי התהליכים ההתפתחותיים בטבע התפתח לידי קבוצת אלגוריתמים – האלגוריתמים האבולוציוניים. תת קבוצה עיקרית של האלגוריתמים האבולוציוניים היה האלגוריתם הגנטי, המדמה את תהליך התפתחות המינים בטבע. קבוצה נוספת תחת משפחת האלגוריתם האבולוציוניים הוא האלגוריתם הממטי המשלב בתוכו התפתחות תוך לימוד חיקוי ושיפורים מקומיים. מכיוון אחר, מבוצע ניסיון לחקות התנהגויות של בעלי חיים שונים. לדוגמא, בדומה לצורה שהנמלים מצליחים למצוא את מקור האוכל הנמצא במרחק לא מבוטל מהקן, וכמו שהדבורים מצליחות למצוא מקורות צוף באזור מרוחק מהכוורת, ניתן ליצר אלגוריתמים למציאת פתרונות אופטימאליים.

4.4 אלגוריתמים גנטיים

אלגוריתמים גנטיים הם כנראה האלגוריתמים האבולוציוניים הפופולאריים ביותר. פתרונות מבוססים אלגוריתמים גנטיים מומשו למגוון בעיות אופטימיזציה מוכרות. אלגוריתמים גנטיים הם אלגוריתמים מבוססי אוכלוסייה שפותחו ע"י John Holland בשנות ה-60/70 של המאה ה-20. מדובר במודל (או בהפשטה) של האבולוציה הביולוגית, המבוסס על התיאוריה של צ'רלס דרווין – הבחירה הטבעית. השימוש בפעולות הבסיסיות: שחלוף (crossover), איחוד (recombination), מוטציה (mutation) ובחירה (selection) מרכיבים את השלבים השונים באלגוריתם הגנטי כאסטרטגיה לפתרון בעיות. קיימים שימושים רבים לאלגוריתמים גנטיים הן לפתרון בעיות תיאורטיות (למשל, בעיית הסוכן הנוסע) והן בנושאים פרקטיים (כלכלה, הנדסה וכו').

אלגוריתמים גנטיים מסוגלים להתמודד עם בעיות מורכבות בהצלחה גבוהה ביחס לאלגוריתמים סטנדרטיים. צאצאים או תני אוכלוסיות יכולים לחקור את המרחב במגוון כיוונים במקביל, ולכן ניתן לממש אלגוריתמים גנטיים בצורה מקבילית וניתן לנצל את יכולת המחשוב המתקדמות הקיימות כיום בעולם. באלגוריתמים גנטיים מבוצע שימוש במספר פרמטרים המשפיעים על התנהגות האוכלוסייה. פרמטרים אלו נועדו לכיול האלגוריתמים לצורך התמודדות עם הבעיה הנתונה. פרמטרים כמו גודל האוכלוסייה, אחוז ההסתברות לפעולה גנטית בסיסית והגדרת פונקציית הכשירות (fitness) הינם קריטיות להצלחת האלגוריתם. הגדרות לא נכונות או לא מתאימות לבעיה, ימנעו התכנסות של האלגוריתם או יגרמו לאלגוריתם למצוא פתרונות לא אופטימאליים. המהות של אלגוריתם גנטי הוא להגדיר אוסף של ביטים או מחרוזת של תווים לייצור הכרומוזום של הפרט. ביחד עם הכרומוזום מוגדרים הפעולות הגנטיות על הכרומוזום, ופונקציית הכשירות המגדירה את איכות הכרומוזום ביחס לבעיה. השלבים העקרוניים בתהליך האלגוריתם הגנטי:

1. הגדרת קידוד הכרומוזום, הגדרת פונקציית הכשירות ופרמטרי כיול (הסתברות לשחלוף/מוטציה)

2. יצר אוכלוסייה ראשונית

3. עבור T דורות בצע

a. בחר את ההורים של הדור הבא

b. בהסתברות P_c בצע שחלוף

c. בהסתברות P_m בצע מוטציה

4. החזר את התוצאה הטובה ביותר מתוך האוכלוסייה

פעולה השחלוף מבוצעת בד"כ בהסתברות גבוהה. פעולה זו מבוצעת ע"י החלפת קטעי כרומוזומים בין זוג כרומוזומים ליצירת זוג כרומוזומים חדש כפי שמופיע בדוגמא הבאה:

פרט 1

1	0	1	1	0	1	0	1
---	---	---	---	---	---	---	---

פרט 2

0	1	1	0	0	0	1	1
---	---	---	---	---	---	---	---

לאחר פעולת שחלוף בין פרט 1 לפרט 2 יוצרו 2 פרטים חדשים:

1	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

ר-1

0	1	1	0	0	1	0	1
---	---	---	---	---	---	---	---

פעולת מוטציה הינה שינוי אקראי של גן אקראי בכרומוזום. ההסתברות לפעולת מוטציה היא בד"כ נמוכה.

פרט 1

1	0	1	1	0	1	0	1
פרט 1 לאחר הפעלת מוטציה (בחירה אחראית באיבר החמישי ושינוי אקראי ל-1)							
1	0	1	1	1	1	0	1

פעולת הבחירה מבוצעת עפ"י פונקציית הכשירות וניתן להגדיר מספר שיטות לבחירה: בחירה פרופורציונאלית ביחס לכשירות, או בחירה עפ"י מדרוג. כאמור לעיל, חשיבות גבוהה לבחירת הפרמטרים לאלגוריתם.

בבעיות למציאת מינימום דרך אפשרית להגדרת פונקציית הכשירות היא להגדיר $F=A-f(x)$ כאשר A הוא קבוע גדול מספיק (או ש- $A=0$ יספיק אם אפשרי שפונקציית הכשירות תחזיר ערכים שליליים). הסתברות לשחלוף (P_d) מקבלת בד"כ ערך גבוה, בין 0.7 ל-1. לעומת זאת ההסתברות למוטציה (P_m) מקבלת ערך נמוך, בין 0.001 ל-0.05. ערך P_c נמוך יגרום לתהליך הגנטי להיות איטי ולא להתקדם מספיק לכיוון פתרון מאחר וכמות פעולות השחלוף בדור תהיה קטנה. ערך P_m גבוה יגרום לפתרונות להיות קופצניים גם כאשר הפתרון האופטימאלי מתקרב.

בבחירת הדור הבא חשוב שהפתרון הטוב ביותר יעבור בצורה אוטומטית לדור הבא. בד"כ בוחרים בסוג מסוים של אליטיזם, כלומר, בחירה בכשיר/אופטימאלי ביותר למעבר לדור הבא ללא שינוי כלל. הגדרת גודל אוכלוסייה קטן עלול לגרום לאוכלוסייה ללכת לכיוון אחד, שלא יהיה הכיוון שביא לפתרון אופטימאלי. לעומת זאת אוכלוסייה גדולה תגרום לכמות חישובים רבה יותר בכל דור ועיכוב בהתקדמות לכיוון פתרון.

מנגנון נוסף שמשתמשים בו הוא מודל האי. מודל זה מיצר מספר אתרים לאוכלוסייה שפועלים בנפרד, אבל משתפים מידע אחת למספר דורות. שיתוף כזה יכול להיות ע"י העתקת המצטיין בכל אוכלוסייה לכל האוכלוסיות האחרות. מנגנון האי מאפשר גיוון באזורי החיפוש, ואח"כ הפצת תוצאות החיפוש לצורך התמקדות בפתרונות טובים.

4.5 אלגוריתם גנטי מבוסס היוריסטיקה

קיים קושי בהפעלת אלגוריתם גנטי לפתרון בעיית ה-SCS מהסיבות הבאות:

1. ייצוג טבעי של אוכלוסיית המועמדים לפתרון יהיה בצורה של מחרוזת. מחרוזת הפתרון יכולה להיות באורך משתנה. קיים קושי להפעיל אלגוריתם גנטי על אוכלוסייה עם כרומוזום באורך משתנה.
2. שינוי קל בפתרון חוקי (מחרוזת-על משותפת) כמו למשל שינוי תו או מחיקת תווים, יגרום ברוב המקרים למחרוזת שאינה מחרוזת-על משותפת.
3. רוב המחרוזות האפשריות מעל השפה אינם מחרוזת-על משותפת.

4. האלגוריתם חייב לספק פתרון אופטימאלי או קרוב אליו להצדקת מאמץ הפיתוח. למרות קשיים אלו ניתן לממש אלגוריתם גנטי לפתרון בעיית ה-SCS כפי שהוצג ב- [7]. הבסיס לאלגוריתם הגנטי הוא אלגוריתם H1 והוא משולב עם תהליך אבולוציוני גנטי. התוצאה הינה אלגוריתם היוריסטי/גנטי המאפשר שיפור במציאת פתרונות אופטימאליים המכונה GA/H1. הרעיון באלגוריתם הוא לבסס את H1 על מספר פרמטרים רב (ככמות התווים בקבוצת המחרוזות), ולכיל את הפרמטרים הללו ע"י התהליך הגנטי.

לצורך ביצוע האלגוריתם H1 הורחב ע"י הוספת פרמטר לכל תו בכל מחרוזת ב- L , להלן W המכיל m מערכים $w_1, \dots, w_m$. כל מערך $w_i \in W$ הוא באורך $|s_i|$ והוא מכיל מספר ממשי. תהליך מציאת מחרוזת-על עודכן כך שיהיה שימוש בקבוצת הפרמטרים W בנוסף לשימוש בפרמטר הרגיל של אורך המחרוזת. כפי קיים ב- H1. בשורה 3 באלגוריתם H1 עודכנה פונקציה חישוב המשקלים מ- $wsum$ ל- $wsum2$. עבור כל תו באלפבית יחושב הסכום של המשקל המקורי (כלומר אורך המחרוזות שהוא נמצא בראשם) והמשקל החדש הכולל את סכום המשקלים המתאימים לתו זה (מתוך W) במחרוזות בהם הוא נמצא בראש.

להלן איטרציה אחת באלגוריתם המעודכן. נציין כי $w_i[1]$ הוא האיבר מספר 1 במערך w_i .

אלגוריתם 5 - Heuristic GA/H1

Iteration GA/H1 ($L = \{s_1 \dots, s_m\}$, $W = \{w_1 \dots, w_m\}$)

- 1: let $s \leftarrow \epsilon$
- 2: do
- 3: let $\Sigma' \leftarrow \{a \in \Sigma | wsum2(a) = \max\{wsum2(b) | b \in \Sigma\}\}$, where $wsum2(a) = \sum_{s_i = as'_i} (|s'_i| + w_i[1])$
- 4: let $b \leftarrow$ random letter from Σ'
- 5: for $s_i \in L$, $s_i = bs'_i$ do let $s_i \leftarrow s'_i$
- 6: let $s \leftarrow sb$
- 7: until $\sum_{s_i \in L} |s_i| = 0$
- 8: return s

תהליך קביעת הערכים ב- W יהיה באמצעות אלגוריתם גנטי. הכרומוזום לאלגוריתם הגנטי יהיה W כלומר אורכו כסכום אורכי המחרוזות ב- L . ניתן ליצר בעזרת הכרומוזום מחרוזת-על משותפת ע"י הפעלת GA/H1. הכשירות של כל כרומוזום מוגדרת עפ"י אורך מחרוזת-העל הנוצרת מהכרומוזום. את הכרומוזום נסדר פיזית בצורה שתשמור ברצף את המשקלים של האינדקס הראשון, לאחר מכן האינדקס השני וכך הלאה. דבר זה נובע מהקשר לתהליך בחירת התווים – בשלב הראשון בוחרים אחד התווים הנמצאים בראש המחרוזות, ולכן רצוי לשים אותם במקום קרוב בכרומוזום. הערכים שיבחרו יאותחלו לערכים אקראיים בין 0 ל- 1. יבוצע בכל דור בחירת הורים לדור הבא, פעולות שחלוף (crossover), מוטציות, וערבוב אוכלוסיות עפ"י הפרמטרים הבאים:

1. 10 תתי אוכלוסיות בגודל 150.
 2. סבירות למוטציה – 2% עבור גן (כלומר כל איבר בכל מערך w ישונה בהסתברות של 2%). מוטציה הינה שינוי של לפחות 0.05 בערך הגן. הערך החדש נקבע אקראית.
 3. סבירות לשחלוף – 100% (ביצוע תמיד).
 4. כל 10 דורות מעבר בין אוכלוסיות. כל אוכלוסיה בוחרת את הנציג הטוב ביותר ושולחת אותו לכל האוכלוסיות האחרות.
- האלגוריתם מבצע בכל דור בחירת צמדי הורים עפ"י מדרוג הכשירות שלהם ביחס לשאר האוכלוסייה, יצירת בן ע"י החלפת גנים ב-2 אזורים ויצירת בן בודד חדש, יצירת מוטציה בהסתברות המוגדרת, והכנסתו לאוכלוסיה. הנציג באוכלוסיה עם הכשירות הכי נמוכה נמחק. בכל 10 דורות מבצעים חילופי נציגים בין תתי האוכלוסיות: הטוב ביותר בכל אוכלוסיה מועתק לכל תתי האוכלוסיות האחרות. עצירה כאשר כל האוכלוסיות בעלי כשירות זהה.
- שיפור לאלגוריתם אשר מחזק את האלגוריתם הבסיסי, מבוצע ע"י הוספת ערך בסיסי קבוע לכל משקל לפני ביצוע חישוב. הוספה קבועה של ערך לכל משקל, מחזק את המשקל הכולל ככל שהתו נמצא בראש יותר מחרוזות. הוספת ערך קבוע גדול יגרום לכך שהאלגוריתם האבולוציוני ישפיע רק במצב של תיקון בלוגיקה הרגילה של ה-H1. ערכים בסיסים אפשריים:
- ללא ערך בסיסי
 - ערך בסיס $\frac{1}{||L||}$ כאשר $||L||$ הוא סכום אורכי המחרוזות ב-L.
 - ערך בסיס 1
- נמצא כי אין מספר קבוע טוב לכל המופעים של הבעיה. ניתן להוסיף ערך זה לכרומוזום כפרמטר שמקבל ערך בין 0 ל- $\frac{1}{||L||}$.
- בוצעה הרצות האלגוריתמים השונים (MM, H1, H2, H3 ו-GA/H1) על מספר סוגי קלטים: כמות תווים שונה באלפבית, כמות מחרוזות שונה, מחרוזות אקראיות, מחרוזות דומות ובנוסף מקרים מיוחדים בהם הפתרון הוא טריוויאלי אבל MM אינו מצליח למצוא פתרון אופטימאלי.
- כל תוצאות GA/H1 מבוססות על 4 הרצות. תוצאות הרצות האלגוריתמים ההיוריסטיים מבוססות על 150 הרצות.
- בעיית אלגוריתמים עם מחרוזות אקראיות בגודל קבוע יוצרו ע"י מחרוזות באורך 80 תווים עם אלפבית בגודל 2, 4 ו-16 תווים. כמות המחרוזות ב-L הם 4, 8 ו-16 מחרוזות. לכל קומבינציה יוצרו 5 בעיות אקראיות שונות.
- בכל המקרים אלגוריתם GA/H1 הגיע לתוצאות הטובות ביותר כאשר אחריו אלגוריתם H3. תוצאות עבור כמות קטנה של מחרוזות (קטן או שווה 8) וכמות תווים גדולה (16) מראה כי GA/H1 מצליח לשפר בצורה ניכרת את אלגוריתם H1 (שעליו הוא מבוסס), למרות שאלגוריתם H1 היה חלש יחסית

במקרים אלו. עבור כמות גדולה של מחרוזות (16) וכמות תווים קטנה (4) אלגוריתם H1 הגיע לתוצאות טובות, אבל גם עבור מקרים אלו אלגוריתם GA/H1 הצליח לשפר בין 4% ל-6%.

טבלה 1 – אחוז שיפור מחרוזות אקראיות גודל משתנה

$ \Sigma $	H1	H2	H3	GA/H1
2	4.7%	0.3%	7.0%	8.6%
4	7.3%	3.2%	12.2%	15.7%
16	11.3%	9.0%	20.5%	21.8%

עבור מחרוזות אקראיות בגודל משתנה נלקחו 4 מחרוזות בגודל 40 תווים ו-4 מחרוזות בגודל 80 תווים. הוגרלו אקראית 5 מופעים של בעיה זו. ניתן לראות תוצאות טובות עבור GA/H1. בעיית מחרוזות דומות יוצרו ע"י מחרוזות-על בגודל 100 מעל אלפבית בגודל 4 ויצירת מחרוזות ע"י השמטה אקראית של תווים ממחרוזת הפתרון. נוצרו 8 ו-16 מחרוזות באורך 90 ו-80.

טבלה 2 – אורך מחרוזת-על ממחרוזות דומות

$ \Sigma =4$	MM	H1	H2	H3	GA/H1
$n=90, m=8$	117	113	106	100	100
$n=90, m=16$	107	107	100	100	100
$n=80, m=8$	167	163	158	107	100
$n=80, m=16$	179	162	164	116	100

אלגוריתם GA/H1 מצא את פתרון אופטימאלי בכל המקרים. אלגוריתם H3 מצא פתרון אופטימאלי עבור מחרוזות בגודל 90 ופתרונות קרובים לאופטימאלי עבור בעיות על מחרוזות באורך 80. בעיות עם מחרוזות מיוחדות הוגדרו במיוחד להקשות על אלגוריתם ה-MM הבסיסי. אלגוריתם MM מגיע לתוצאות לא טובות למרות שהפתרון האופטימאלי הוא ברור.

- L1: 9 מחרוזות של a^{40} , 4 מחרוזות ba^{39} , 2 מחרוזות bba^{38} , ומחרוזת $bbba^{37}$. פתרון אופטימאלי באורך 43 ע"י לקיחת b לפני a . ה-MM ייקח את ה- a ואז b ואז שוב את ה- a של המחרוזות שמתחילות ב- b .
- L2: 9 מחרוזות של a^{40} , 4 מחרוזות $b^{13}a^{27}$, 2 מחרוזות $b^{26}a^{14}$, ומחרוזת $b^{39}a$. דומה לבעיה הראשונה, אבל כמות תווי ה- b הנדרשים להורדה לפני מעבר ל- a הוא גדול יותר.
- L3: 8 מחרוזות $a^{20}b^{20}$ ו-8 מחרוזות $b^{20}c^{20}$. נדרש לקחת קודם את ה- a ורק אח"כ את ה- b . ה-MM ימצא זאת רק אם יפתור את כל המקרים בהם קיבל תוצאת תיקו במשקלים לכיוון ה- a . הסיכוי לכך הוא אפסי.

טבלה 3 – אורך מחרוזת-על ממחרוזות מיוחדות

	אופטימאלי	MM	H1	H2	H3	GA/H1
L1	43	157	92	80	43	43
L2	79	121	91	121	79	79
L3	60	75	79	74	78	60

אלגוריתם GA/H1 מצא את פתרון אופטימאלי בכל המקרים. אלגוריתם H3 מצא פתרון אופטימאלי עבור L1 ו-L2. מלבד L3 אלגוריתמים היוריסטיים מגיעים לתוצאות טובות מתוצאות ה-MM המקורי (מלבד L2 עבור H2 בו התוצאה זהה). עבור L3 ה-MM מגיע לתוצאות טובות אך עדיין לא לפתרון האופטימאלי.

4.6 אופטימיזציה קן הנמלים (ACO – Ant Colony Optimization)

אלגוריתם ACO (Ant Colony Optimization) פותר בעיות אופטימיזציה ע"י היוריסטיקה ושיתוף פעולה בין נמלים מלאכותיות המחפשות אחר מזון (פתרונות). הנמלים משתפים פעולה בצורה לא ישירה ע"י שינוי דינאמי של ייצוג הבעיה כך שהנמלים הבאות ילכו לכיוון פתרונות טובים של הבעיה. גישה סטנדרטית לתיכון אלגוריתם ACO היא למצוא ייצוג בצורת עץ של הדרך לכיוון פתרון טוב של הבעיה. צומת בעץ היא בעצם נקודת החלטה של הנמלה ונתיב שלם בעץ הוא בעצם הדרך להגיע לפתרון של הבעיה. נמלים שמוצאות פתרון משאירות פרומון מלאכותי (pheromone – הורמון מעורר תגובה התנהגותית) לאורך הצמתים של הגרף. כמות הפרומון שהנמלה תשאיר יהיה לפי איכות המזון ובעצם איכות הפתרון. נמלה המתקדמת לכיוון פתרון תיקח את החלטתה בכל צומת בעץ עפ"י כמות הפרומון הנמצא בנקודת החלטה, ועפ"י הפונקציה ההיוריסטית שלה עצמה, באותה נקודת החלטה. נתאר את אלגוריתם ACO לפתרון בעיית SCS כפי שהוצג ב-[10].

הפונקציה ההיוריסטית הבסיסית לבעיית SCS עבור ה-ACO תהיה מסוג ה-MM.

להלן הבעיה מנקודת מבטה של הנמלה:

הנמלה מתחילה מפתרון הכולל מחרוזת ריקה. בשלב זה הנמלה נדרשת לבחור את התו הראשון להכנסה למחרוזת הפתרון. הנמלה תבחר תו מבין התווים הנמצאים בראש המחרוזות ב- L בצורה אקראית עם הסתברות יותר גבוהה לתווים שכמות המופעים שלהם בראש המחרוזות גבוהה. כלומר תו שנמצא בראש 3 מחרוזות של L יבחר בהסתברות של פי 3 מתו הנמצא בראש מחרוזת אחת. לאחר בחירה זו, התו הנבחר נמחק מראש המחרוזות ב- L . וכך הלאה עד להוצאת כל התווים מ- L ויצירת מחרוזת-על משותפת. עבור כל תו, בכל מחרוזות מ- L , הייתה נקודה אחת בתהליך שבו התו נמחק. נקודה זו היא נקודת התמיכה של תו זה בהחלטת הנמלה. מסלול הנמלה הוא בעצם סדר בחירת התווים במחרוזת הפתרון. הנמלה מחלקת את הפרומון שהיא אמורה לפזר על כל המסלול שהיא בחרה. כל תו מקבל כמות של פרומון כאשר כמות הפרומון תלויה בהשפעה של התו על הפתרון. השפעתו של תו שנבחר בשלב מוקדם גדולה יותר מהשפעה של תו שנבחר מאוחר בתהליך, כיון שמשפיע יותר מוקדם על המחרוזות ב- L . בהתאם לכך מחולק הפרומון – כמות גדולה לתווים שנקודת ההחלטה שלהם היא יותר מוקדמת.

נציג את התהליך השלם בצורה פורמאלית. סימון $s_i[j]$ מציין את התו ה- j במחרוזת i ב- L כאשר $i \in \{1, 2, \dots, n\}$. המערך τ_i יציין את כמות הפרומון של כל מחרוזות s_i ובהתאם $\tau_i[j]$ יציין את כמות הפרומון של התו $s_i[j]$.

באתחול האלגוריתם יקבע ערך פרומון 1 עבור כל האיברים ב- τ .

תהינה m נמלים. עבור כל נמלה $k \in \{1, 2, \dots, m\}$ מוחזק וקטור מצב v לצורך מעקב אחרי התקדמות הנמלה בתהליך החישוב.

$$v^{\rightarrow k} = (v_1^k, v_2^k, \dots, v_n^k)$$

הוקטור v^k מצביע על מיקום התו הנמצא בראש המחרוזת בתהליך ההתקדמות של הנמלה.

דוגמא :

$L = \{raananm, mraanan\}$ אם $v = (3,5)$ אזי "ra" של המחרוזת הראשונה ו-"mraa" של המחרוזת השנייה כבר נמצאים במחרוזת הפתרון וכעת 'a' ו-'ו' ממתינים בראש המחרוזות לכניסה לפתרון. בתחילת כל איטרציה $v \rightarrow k$ מאותחל ל-1 כלומר מיקום התו הבא להוספה הוא בראש כל המחרוזות. s^k הינה מחרוזת הפתרון של הנמלה k .

תהליך החיפוש של הנמלה מסתיים כאשר הוקטור מצביע על סוף כל המחרוזות. כלומר :

$$v \rightarrow k = v_{fin} = (|s_1| + 1, |s_2| + 1, \dots, |s_n| + 1)$$

רשימת המועמדים להכנסה למחרוזת הפתרון s^k הם כל התווים הנמצאים בראש מחרוזות ב- L .

$$C^k = \{a \in \Sigma \mid \exists i: a = s_i[v_i^k]\}$$

סכום כמויות הפרומון הנמצאים בחזית של כל תו ב- C^k מיוצג ע"י $\tau^k[a]$

$$\tau^k(a) = \sum_{i \in \tau^k: s_i[v_i^k] = a} \tau_i[v_i^k]$$

כאשר τ^k מציין את כל האינדקסים שמקיימים $v_i < |s_i| + 1$ (המחרוזות שעדיין לא "נוקו" לגמרי מתוים).

להלן אלגוריתם לפעולת נמלה k :

אלגוריתם 6 - Ant k Iteration

Ant-k Iteration

1: $v^k \leftarrow \{\text{all } 1\}$

2: $s \leftarrow \epsilon$

4: repeate

3: calc C^k (all symbols in head position in string)

5: $a \leftarrow \text{choose}(C^k)$

6: $s \leftarrow sa$

7: $v^k \leftarrow v_{new}^k$

8: until ($v^k = v_{fin}$)

בחירת $\text{choose}(C^k)$ מבוצעת ע"י בחירה אקראית יחסית עם גבול q_0 . משמעות בחירה כזו היא שבהסתברות q_0 בוחרים את המקסימום $\tau^k(a)$ ובהסתברות $1 - q_0$ בוחרים אקראית בהסתברות ל- a עפ"י הנוסחה הבאה :

$$p(a, v^k) = \frac{[\tau_k(a)]^\alpha}{\sum_{a' \in C^k} [\tau_k(a')]^\alpha}$$

כאשר α הוא פרמטר לשליטה בחיזוק ההסתברות לבחירה בחזקים.

חישוב v_{new}^k מבוצע ע"י קידום האינדקס המציין את תחילת המחרוזת לטיפול עבור כל המחרוזות שבהם התו הנבחר היה ראשון.

חישוב v_{new}^k מבוצע כך עבור כל i :

$$v_i^k = \begin{cases} v_i^k + 1 & \text{if } s_i[v_i^k] = a \\ v_i^k & \text{otherwise} \end{cases}$$

לאחר שלב זה בו נמצא פתרון עבור כל נמלה, כל נמלה משאירה, על פני נתיב ההתקדמות, את התרומה שלה מבחינת כמות הפרומון. תרומה זו מצוינת ע"י $\Delta_i^k[j]$. כמות הפרומון הכללית שמשאירה כל נמלה על כל הנתיב שלה:

$$\theta^k = f(\text{rank}(k)) \frac{1}{|s^k|}$$

כאשר $|s^k|$ מציין את אורך הפתרון ולכן ככל שהפתרון קצר כך כמות הפרומון של הנמלה גדל. הערך $f(\text{rank}(k))$ הוא פקטור המבוסס על אורך הפתרון של הנמלה ביחס לאורך הפתרונות של הנמלים האחרות.

חישוב חלוקת כלל הפרומון עבור התווים השונים מבוצע עפ"י מיקום התו במחרוזת הפתרון. כל תו במחרוזת הפתרון מקושר לאוסף התווים שהוא "מחק" במחרוזות המקוריות ב- L . נסמן ב- $M_k(l)$ את קבוצת כל התווים $s_i[j]$ שנמחקו בשלב הוספת התו ה- l ($s[l]$) למחרוזת הפתרון.

$$\Delta_i^k[j] = \frac{\theta^k}{|M^k|} 2 \frac{|s^k| - l + 1}{|s^k|^2 + |s^k|}$$

ככל שהתו נוסף בשלב מוקדם יותר (כלומר, ככל ש- l קטן) נדרש להשאיר יותר פרומון. כמות הפרומון בנקודה זו מתחלקת בצורה שווה לכל התווים שנוספו באותו שלב (ולכן החלוקה ב- $|M^k|$). הנוסחה נועדה לוודא שכל הפרומון שהוגדר לנמלה לחלק אכן מחולק. ניתן לראות שסכום כל הפרומון מתחלק על כל התווים:

$$\sum_{l=1}^{|s^k|} (|s^k| - l + 1) = \frac{|s^k|^2 + |s^k|}{2}$$

בסיום האיטרציה אוספים את הפרומון של כל הנמלים שפוזרו באותה איטרציה:

$$\Delta\tau_i[j] = \sum_{k=1}^m (\Delta\tau_i^k[j])$$

ומעדכנים את הנתיב עצמו עבור כל התווים:

$$\tau_i[j] = (1 - \rho)\tau_i[j] + \gamma\Delta\tau_i[j]$$

כאשר $\rho \in (0,1]$ ומאפשר להגדיר את קצב ההתנדפות של הפרומון מהאיטרציות הקודמות, ו- γ הוא פרמטר הגברה של כמות הפרומון של המחזור הנוכחי.

ניתן ליישם תוספות ושיפורים לאלגוריתם ACO לפתרון בעיית SCS. שיפור הפונקציה היוריסטית לבחירת התו הבא (MM) כך שילקח בחשבון אורך המחרוזת בצורה דומה לשיפור שהוצג לעיל באלגוריתם H1 לאלגוריתם MM. לצורך כך נעדכן את החישוב של בחירת התו.

$$\tau^k(a) = \sum_{i \in \tau^k: s_i[v_i^k] = a} \tau_i[v_i^k] (|s_i| - v_i^k + 1)$$

הביטוי $|s_i| - v_i^k + 1$ מייצג את אורך המחרוזת שנשארה עוד לטיפול באיטרציה. שיפור זה הוא שיפור ההיוריסטיקה של אלגוריתם MM הבסיסי. שיפור נוסף אפשרי ע"י היוריסטיקה של הסתכלות קדימה. כלומר בחינה של המצב שנעבור אליו לאחר בחירה בתו מסוים בדומה לשיפור שבוצע באלגוריתם H2. נגדיר את הפונקציה η לחישוב כמות הפרומון המקסימאלית במצב מסוים אם נבחר בתו a להכנסה למחרוזת הפתרון.

$$\eta(v^k, a) = \max_{a' \in \tilde{C}^k} \tilde{\tau}^k(a')$$

כאשר הביטויים $\tilde{C}$ ו- $\tilde{\tau}$ מציינים את קבוצת המועמדים והפרומון במצב החדש לאחר שהנמלה בחרה ב- a כתו הבא להכנסה למחרוזת הפתרון. כעת חישוב ההסתברות לבחירה האקראית תלוי בפרומון הנוכחי של תו a וגם בכמות הפרומון הכוללת שתהיה במועמדים החדשים שיהיה לאחר בחירת a .

$$p(a, v^k) = \frac{[\tau_k(a)]^\alpha [\eta(v^k, a)]^\beta}{\sum_{a' \in C^k} ([\tau_k(a')]^\alpha [\eta(v^k, a')]^\beta)}$$

פרמטר β שולט בהשפעה של כמות הפרומון של המצב בהסתכלות קדימה. ניתן גם לבצע הסתכלות קדימה עמוקה יותר.

ניתן להגיע לשיפור ביצועים ע"י גישת מודל האיים. הרעיון הוא לאפשר למספר מושבות של נמלים לפעול בנפרד על אותה בעיה, למשך מספר מחזורים, ואז לאפשר להם לחלוק מידע על הפתרון הטוב ביותר. כלומר מושבת נמלים מוכנה לקבל את הנמלה (הפתרון) הטובה ביותר כחברה במושבה. קבלת הפתרון הטוב ביותר תגרום לעדכון הפרומון בהתאם לפתרון זה ותשפיע על המשך התקדמות המושבה לקראת פתרון. ניתן גם להחליט שחלק מהמושבות יפעלו בכיוון ההפוך, כלומר ימצאו פתרון על מחרוזות הפוכות. שיתוף פעולה כזה יכול לגוון את המושבות וליצר פתרונות אופטימאליים יותר. במידה ומושבות עובדות על מחרוזות הפוכות יש להפוך את הפתרון לפני הכנסתו למושבה שפועלת בכיוון המקורי.

לצורך בחינת ביצועי אלגוריתם ACO בוצעה השוואה מול GA/H1 ו-H1. בדומה לבחינת GA/H1 הוגדרו מספר סוגי בעיות לבחינה:

1. מחרוזות אקראיות באורכים שווים ומשתנים.
2. מחרוזות דומות
3. מקרים מיוחדים

מחרוזות אקראיות מאופיינות בכל שאין קשר הגיוני בין תווים בתוך מחרוזת ואין קשר הגיוני בין מחרוזות. מחרוזות דומות חשובות בחקר DNA של זנים דומים. עבור מקרים מיוחדים נלקחו כאלו שה- MM פועל גרוע עבור אחד ואופטימאלי עבור השני.

מחרוזות אקראיות 1 (TR1): 16 מחרוזות באורך 160 גודל אלפבית 16.

מחרוזות אקראיות 2 (TR2): 4 מחרוזות באורך 80 ו-4 באורך 40, גודל אלפבית 16.

מחרוזות דומות (TS) 16 מחרוזות אקראיות באורך 80 שנבחרו כתתי מחרוזות של מחרוזת אקראית בגודל 100, גודל אלפבית 16.

מקרים מיוחדים 1 (TW1): 9 מחרוזות a^{40} , 4 מחרוזות $b^{13}a^{27}$, 2 מחרוזות $b^{26}a^{14}$, מחרוזת אחת $b^{39}a$.

מקרים מיוחדים 2 (TW2): 8 מחרוזות $a^{20}b^{20}$, 8 מחרוזות $b^{20}c^{20}$.

הפרמטרים להרצת ACO מפורטים בטבלה 4.

טבלה 4 – פרמטרים להרצת אלגוריתם ACO

פרמטר	ערך	הערות
α	9	
β	9	
γ	1000	
q_0	0.9	
מושבות	4	2 קדימה ו-2 אחורה
כמות נמלים כל מושבה	16	
$f(1), f(2), f(3), f(4)$	3	
$f(5), f(6)$	2	
$f(7), f(8)$	1	
$f(9), \dots, f(16)$	0	כלומר רק 8 נמלים הטובות ביותר יפזרו פרומון בנתיב.
אליטיזם $f(0)$	5	הפתרון הטוב ביותר יפזר עוד פרומון על הנתיב שביצע
החלפה בין מושבות	כל 8 מחזורים	הטוב מכל המושבות נשלח לכל המושבות
כמות הרצות	25	5 הרצות לכל סוג בעיה
איטרציות	150	חוץ מ-GA אשר עוצר שכולם זהים

טבלה 5 – אורך ואחוז שיפור מחרוזת-על עבור מחרוזות אקראיות ודומות

	LMM-H1	LMM-H2	LMM-L2	GA	ACO	ACO-L	ACO-L2
	Length MM	With lookahead	With lookahead Depth 2	GA/H1	With lookahead	ACO With lookahead	ACO With lookahead Depth 2
TR1	985.2	907.8	872.8	965.5	940.6	875.0	787.8
	0%	7.86%	11.41%	2.0%	4.53%	11.19%	20.04%
TR2	296.8	274.8	260.8	260.2	261.2	245.8	240.1
	0%	7.41%	12.13%	12.33%	11.99%	17.18%	19.10%
TS	166.4	152.0	129.8	100.0	102.6	102.7	100.5
	0%	8.65%	22.0%	39.9%	38.34%	38.28%	39.60%

ביצועים דומים של GA ו- ACO ללא הסתכלות קדימה כאשר בבעיות TR1 ביצועים מעט יותר טובים ל- ACO. זמן הריצה של ACO קצר בצורה משמעותית – דקות ספורות מול מספר שעות עד סיום GA (למרות שבוצע על מחשבים שונים ההבדל ניכר). למרות ביצועים טובים יותר של GA ו- ACO מה- MM, השיפור הקטן ביותר היה על מחרוזות אקראיות באורך שווה. כעקרון, ביצועי MM טובים על בחירות אקראיות מכיוון שהאלגוריתם מבצע שיפור מקומי הגיוני שממשיך להיות הגיוני גם לאחר בחירת התו הבאה. שיפור יותר משמעותי מול מחרוזות בגודל משתנה. תוספת הסתכלות קדימה משפרת בצורה משמעותית את ביצוע ACO ו- MM על בעיית מחרוזות אקראיות על חשבון זמן ריצה: ללא הסתכלות קדימה סיום ריצה בתוך שניות. בהסתכלות קדימה של צעד אחד זמן ריצה עולה לסדר גודל של עשרות דקות והסתכלות קדימה של 2 צעדים זמן הריצה עולה לשעות. גם קצב ההתקדמות איכות הפתרון משתפר משמעותית – לאחר סבב אחד הכולל הסתכלות קדימה מגיעים לתוצאה טובה כמו ב- 150 סבבים ללא הסתכלות קדימה. כאשר קיימת מגבלת זמן נראה שעדיף לבצע MM-L2 על מחרוזות אקראיות, אבל את ACO ללא הסתכלות קדימה עבור בעיות עם מחרוזות דומות.

טבלה 6 – אורך ואחוז שיפור מחרוזת-על עפ"י עומק הסתכלות קדימה

	0	1	2	3	4	5	6
TR1	985.2	907.8	872.8	849.0	840.4	829.0	816.8
	0%	7.86%	11.41%	13.82%	14.7%	15.85%	17.09%
TS	166.4	152.0	129.8	117.4	128.2	118.0	118.8
	0%	8.65%	22.0%	29.45%	22.96%	29.09%	28.61%

טבלה 7 – אורך מחרוזת-על עבור מקרים מיוחדים

	optimum	LMM	GA	ACO
TW1	79	91	79	79
TW2	60	79	60	60

עבור מקרים מיוחדים, אלגוריתם MM לא מוצא את הפתרון האופטימאלי לעומת ACO ו-GA שמצליחים למצוא אותו.

השימוש בשילוב של אלגוריתמים שמפעילים מושבות שפועלות קדימה ואחורה במקביל אינו משפר את הביצועים על מחרוזות אקראיות אבל על מחרוזות דומות מהווה שיפור ניכר (103~ מול 112~). השפעה נוספת לבחינה באלגוריתם ACO הוא הפרמטר q_0 לשיטת בחירת הפתרון הטוב ביותר באיטרציה של הנמלה (בהסתברות q_0 לוקחים את הפתרון הטוב ביותר, פחות מזה בוחרים עפ"י היחס בין איכות הפתרונות). מציאת פתרון על מחרוזות אקראיות משתפר ככל שמגדילים את q_0 . הגדרת q_0 מגבירה את הנטייה של האלגוריתם לכיוון בחירה חמדנית ולכן משפר את הביצועים עבור מחרוזות אקראיות.

4.7 אלגוריתם ממטי עם תיקון מקומי חלקי

ב- Memetic Algorithm (MA) לפתרון בעיית ה- SCS [5] ממשיכים את כיוון התפתחות האלגוריתמים האבולוציוניים. הרעיון של אלגוריתם ממטי הוא לבצע תהליך אבולוציוני אבל להכניס לו תיקונים משפרים בין ה"דורות". ב- MA משתמשים ב- 2 צורות של אסטרטגיות שיפורים מקומיים: מצד אחד מנגנון תיקון לשחזור תקינות המחרוזות כך שתהיה בכל שלב מחרוזת-על משותפת. מצד שני, ביצוע מנגנון הקטנת הפתרון ע"י ניסיון של מחיקת תווים ותיקון חוזר של מחרוזות הפתרון.

אחד הקשיים שניצבים בפני אלגוריתם אבולוציוני בניסיון לפתור את בעיית ה- SCS הוא לשמור על חוקיות המחרוזות, כלומר מחרוזות אקראית מעל השפה בכל אורך, אינה בהכרח מחרוזת-על משותפת של כל המחרוזות ב- L . בד"כ ניתן לפתור קושי זה ע"י אחת מ- 3 טכניקות:

1. אפשר המצאות פתרון לא חוקי אך הענש פתרון זה בהתאם (יוצג בהמשך ב- POEMS).

2. בצע תיקון לפתרונות לא אפשריים כך שיהפכו להיות אפשריים

3. אל תאפשר ליצור פתרונות שאינם אפשריים (כפי שבוצע באלגוריתם GA/H1).

עבור אלגוריתם MA נבחרה טכניקה מספר 2 כפי שהוצג ב- [1].

האלגוריתם כולל אוכלוסיה של פתרונות אפשריים בצורת מחרוזות, כאשר לפני הערכת איכות הפתרון מבצעים פונקציות תיקון ρ על מחרוזות הפתרון s וקבוצת המחרוזות L :

$$\rho(s, L) = \begin{cases} s & \text{if } \forall i: s_i = \epsilon \\ \rho(s', L) & \text{if } \exists i: s_i \neq \epsilon \text{ and } \nexists i: s_i = \alpha s'_i \text{ and } s = \alpha s' \\ \alpha \rho(s', L|_\alpha) & \text{if } \exists i: s_i = \alpha s'_i \text{ and } s = \alpha s' \\ MM(L) & \text{if } \exists i: s_i \neq \epsilon \text{ and } s = \epsilon \end{cases}$$

כאשר $L|_\alpha$ מוגדר ע"י הוצאת α מראש כל המחרוזות הנמצאות ב- L .

עבור L המכיל מחרוזות ריקות אין צורך לתקן את s . אם קיימות מחרוזות ב- L ואף תו אשר נמצא בתחילת המחרוזות לא שווה לתו שנמצא בתחילת s אזי ניתן למחוק את התו מ- s מכיוון שהוא מיותר (בעצם לא מבוצע כאן תיקון אלא שיפור ע"י קיצור s). לעומת זאת אם קיימים ב- L מחרוזות שהתו בתחילתם שווה לתו בתחילת s אזי מוחקים את התו מכל המחרוזות שהוא נמצא בתחילתם וממשיכים לתו הבא ב- s (ביצוע פונקצית תיקון ρ על המחרוזות s ללא התו הראשון וקבוצת המחרוזות L המעודכנת).

כאשר מגיעים למצב בו קיימות עדיין מחרוזות לא ריקות ב- L (לפחות אחת) ו- s ריקה מפעילים אלגוריתם MM על המחרוזות שנשארו.

קעת חישוב פונקצית ה- fitness על מנת לקבל פתרון קצר ככל שניתן:

$$fitness(s, L) = \begin{cases} 0 & \text{if } \forall i: s_i = \epsilon \\ 1 + fitness(s', L|_\alpha) & \text{if } \exists i: s_i \neq \epsilon \text{ and } s = \alpha s' \end{cases}$$

מחשבים את $|s|$ תוך חיסור תווים שלא מכסים אף תו ב- L . בפועל, לאחר הפעלת ρ בעצם פונקצית הכשירות תחזיר בדיוק את $|s|$, מאחר וכל התווים ש- s לא מכסה ירדו. כפי שצוין לעיל, בשלב זה מבוצע שיפור מקומי. לצורך כך נגדיר פונקציה המשנה במעט את הפתרון, לכיוון של מחרוזת יותר קטנה (כדרך לשפר את הפתרון). הפונקציה שהוגדרה היא $DELETE_k$.

$$DELETE_k(s, L) = \begin{cases} \rho(s', L) & \text{if } k = 1 \text{ and } s = as' \\ aDELETE_{k-1}(s', L|_a) & \text{if } k > 1 \text{ and } s = as' \end{cases}$$

פונקציה זו מוחקת את התו ה- k ממחרוזת הפתרון, ומבצעת פונקצית תיקון שהוצגה ρ . הפונקציה $DELETE$ יכולה לקבל ערך k בין 1 ל- $|s|$ והיא פועלת ומעדכנת את s ו- L . הפונקציה מוגדרת בצורה רקורסיבית כך שכאשר $k=1$ (תנאי העצירה) מוחקים את התו הראשון ואז מבצעים את פעולת התיקון ρ על המחרוזת ללא התו הראשון. במקרה של $k>1$ משאירים את התו הראשון ומפעילים את $DELETE_{k-1}$ על המחרוזת ללא התו הראשון תוך הוצאת כל המופעים של התו מראשית כל המחזורות ב- $L|_a$.

לאחר המחיקה וביצוע התיקון יתכן וקוצרה מחרוזת הפתרון ב- 1 ועדיין היא פתרון חוקי.

להלן תיאור Heuristic LS (חיפוש מקומי של פתרונות עבור s ו- L)

אלגוריתם 7 - Heuristic LS

Heuristic LS ($s \in \Sigma^*$, $L = \{s_1, \dots, s_m\}$)

```

1: let  $k \leftarrow 0$ 
2: while  $k < |s|$  do
3:   let  $r \leftarrow DELETE_k(s, L)$ 
4:   if  $fitness(r) < fitness(s)$  then
5:     let  $s \leftarrow r$ ,  $k \leftarrow 0$ 
7:   else
8:     let  $k \leftarrow k+1$ 
10: end while
11: return  $s$ 
```

ביצוע חיפוש מקומי מלא יהיה לנסות ולמחוק כל אחד מהתווים ב- s . בכל פעם שנמצא קיצור (כלומר שיפור בכשירות הפתרון) נתחיל את החיפוש מהתו הראשון.

העלות של החיפוש המקומי נמדד בכמות הפעמים שמבצעים את $DELETE_k$.

אלגוריתם MA הורץ עם הפרמטרים המפורטים בטבלה 8.

טבלה 8 – פרמטרים להרצת MA

אוכלוסיה	$popsiz$	100
הסתברות ל- CROSS	P_x	0.9
הסתברות למוטציה	P_m	0.01
מספר אטרציות	$Maxeval$	100000
בחירה	$Selection$	Tournament
שחלוף	$Cross$	Uniform
מוטציה	$Mutation$	Random substitution
הסתברות ל- LS (MA)	$MA^{x\%}$ כאשר x הוא ההסתברות באחוזים להפעלת LS באיטרציה.	ביצוע עבור האופציות הבאות : $\{0, 0.01, 0.1, 0.5, 1\}$ $x=0$ משמעותו תיקון רגיל ללא השיפור המקומי (הפעלת $DELETE_k$)

הבדיקות בוצעו מול 2 סטים של נתונים :

1. רנדומאלי. 4 מחרוזות בגודל 40, ו- 4 מחרוזות בגודל 80 מעל שפה עם 2, 4, 8, 16 תווים.
 2. מקרה מציאותי. סדרת DNA של SARS בגודל 158 נבחרה כמחרוזת-על. 10 מחרוזות נוצר ממחרוזת ה-DNA ע"י השמטה של רצפים בהסתברויות שונות – 10%, 15%, 20%.
- אלגוריתם הייחוס הנבחר להשוואה הוא Alphabet Leftmost – AL אשר דומה בביצועיו ל-MM. הרצת אלגוריתם AL על כל הפרמוטציות האפשריות, מוגבל ל-100000 עבור השפה עם 16 תווים. עבור הרצת MA נלקח ממוצע של 30 הרצות.

טבלה 9 – אורך מחרוזת-על הקצרה, ממוצע וסטיית תקן MA על מחרוזות אקראיות

\Sigma	AL (MM)	MA ^{0%}	MA ^{1%}	MA ^{10%}	MA ^{50%}	MA ^{100%}
2	121.4	111.2	110.4	111.6	111.4	111.2
	123.4±2.0	112.6±0.8	112.8±1.0	113.1±0.8	113.2±0.8	113.1±0.8
4	183.0	151.6	149.4	149.4	150.0	149.2
	191.2±4.7	155.2±1.9	152.7±1.7	153.5±1.5	153.3±1.4	153.3±1.6
8	252.2	205.4	201.8	202.4	204.0	203.0
	276.8±6.4	213.5±4.0	207.3±2.2	207.9±2.2	208.2±2.0	208.2±2.1
16	320.6	267.0	266.2	266.6	268.4	267.2
	352.9±7.4	281.8±5.9	274.2±3.0	274.7±3.0	275.0±2.8	275.0±3.2

ניתן לראות שביצועי MA טובים יותר מביצועי MM (מציאת המחרוזת הקצרה ביותר).
 ככל שגדל כמות התווים יש השפעה לסוג ה- MA. ביצועים מעט טובים יותר ל- $P > 0$.

טבלה 10 – אורך מחרוזת-על הקצרה, ממוצע וסטיית תקן MA על 158-SARS

Gap%	AL	MA ^{0%}	MA ^{1%}	MA ^{10%}	MA ^{50%}	MA ^{100%}
10%	307	158	158	158	158	158
	315.2±6.8	158.0±0.0	158.0±0.0	158.0±0.0	158.0±0.0	158.0±0.0
15%	293	158	158	158	158	158
	304.3±8.8	158.0±0.0	159.0±2.8	159.8±3.7	159.8±3.0	159.1±2.1
20%	274	165	159	163	159	161
	288.3±8.6	180.8±15.7	177.0±9.3	179.4±9.2	178.1±9.9	176.5±9.0

ביצועי MA טובים יותר מביצועי AL אבל MM (לא מוצג בטבלה) הפעם מספק תוצאות דומות ל-
 MA^{0%} עבור בעיות עם רצפים ברצף של 10%-15%. עבור מחרוזות עם פער של 20% ביצועי MA
 עדיפים על MM. ברצפים אלו (20%) MA שיפור מסוים כאשר $p > 0$.

המטרה המרכזית של אלגוריתמי MA הוא לבדוק האם שיפורים מקומיים בין השלבים
 האבולוציוניים ישפרו את ביצועי האלגוריתם, כלומר יצליחו למצוא פתרון קצר יותר. האלגוריתם
 האבולוציוני הכולל תיקון מבצע שיפור משמעותי לאלגוריתם ההיוריסטי הכללי, אבל ביצוע תיקונים
 מקומיים אינו מהווה שיפור משמעותי ואינו שווה את המאמץ וזמן החישוב.

4.8 שילוב MA ו- Branch and Bound

אלגוריתם Branch and Bound (BnB) ואלגוריתמים ממטיים הינם אלגוריתמים מסוג שונה במהות. למרות זאת ניתן לשלב את האלגוריתמים בצורה שתיצור חיפוש פתרון עם אופי היוריסטי – Beam Search.

אלגוריתם BnB הינו אלגוריתם ידוע לפתרון בעיות אופטימיזציה קומבינטורית. הרעיון הוא לבצע סריקה חכמה של מרחב הפתרונות ע"י יצירת מספר מחיצות ומציאת הגבולות העליונים והתחתונים של כל מחיצה. בפועל הסריקה של מרחב הפתרונות מתבצעת כמו סריקת עץ. הדרך היעילה ביותר היא BEST FIRST, כלומר סריקת המחיצה עם הפתרון הכי מבטיח. הבעיה בדרך זו היא כמות הזיכרון הנדרשת בייחוד בבעיות עם קלטים גדולים. גישה אחרת היא DEPTH FIRST, כלומר הגעה עד לפתרון ואז המשך חיפוש אחר פתרונות יותר טובים. גישה זו לא מחייבת זיכרון רב, אבל משך החיפוש גדול יותר. ניתן לשקול לבצע שילוב בין שתי הגישות (BEST FIRST WITH DIVING).

לעומת זאת אלגוריתמים אבולוציוניים מבצעים אופטימיזציות מבוססות אוכלוסיות אשר מתקדמות בצורה הסתברותית אל כיוון הפתרון. אלגוריתמים ממטיים מנסים לשפר את היכולות של האלגוריתמים האבולוציוניים בצורה של שיפור ההתקדמות ההסתברותית אל הבעיה. אחת הדרכים לשיפור היא ביצוע חיפוש מקומיים דבר שמגדיל את ההסתברות של האיטרציות להתקדם לכיוון פתרון טוב יותר.

ביצוע שילוב של אלגוריתמים שונים ניתן לביצוע ע"י 2 שיטות עקרוניות: שיטה שיתופית (האלגוריתמים פועלים באותה רמה בניסיון לפתור את הבעיה) ושיטה כופה (אלגוריתם אחד שולט ומפעיל במידת הצורך יכולות באלגוריתם הסביל). בשילוב הנוכחי בוחרים בשיטה השיתופית ע"י הפעלת האלגוריתמים לחילופין. בצורה זו מנגנון ה-BnB יכול להשתמש בתוצאות של האלגוריתם MA להגדרה מדויקת של הגבולות לשלבים הבאים. בנוסף ניתן להשתמש בתוצאות ה-MA להוספת התחלה של פתרונות טובים למנגנון ה-BnB.

כפי שצוין קודם, מנגנון ה-BnB סורק את העץ לרוחב למציאת פתרון מיטבי (עם עלות שמירת הזיכרון) או לעומק (עם עלות של זמן ריצה) או בשיטה משולבת. בשיטה המשולבת כל צומת ברמה מסוימת נבחנת פעם אחת, ובוחרים עפ"י פונקציה היוריסטית k צמתים לבחינה בהמשך. האסטרטגיה הזו נקראת סריקת עלומה: מסתכלים על מספר אפשרויות התקדמות, כמו הסתכלות על כל הפתרונות שנמצאים בתוך עלומת הפנס.

להלן האלגוריתם הכללי לביצוע התהליך המשולב.

אלגוריתם 8 - General Hybrid BS and MA

- 1: for l_0 levels do run BS
- 2: do select best *popsizes* nodes from problem queue
- 3: initialize MA population with selected nodes
- 4: run MA
- 5: if MA solution better than BS solution then
- 6: BS solution $\leftarrow$ MA solution
- 7: for l levels do run BS
- 8: until timeout or tree-exhausted
- 9: return BS solution

האלגוריתם מתחיל בהרצת שלב BS רגיל. לאחר מכן מריץ MA ו- BS לחילופין, כאשר ה- MA מתחיל מההתחלה של הפתרונות הכי טובים שה- BS מצא עד כה. יש לזכור שה- BS בצמתים השונים של העץ לא מצא פתרונות מלאים אלא התחלה של פתרון לבעיה. ה- MA מקבל מפתרונות אלו כיוון התקדמות ראשוני. לאחר החלטה על סיום שלב ה- MA, התוצאה הטובה של ה- MA מושוות לתוצאה של ה- BS ומוחלפת אם נמצאה טובה יותר. בשלב זה ממשיכים בעוד l שלבים של תהליך BS. הפסקת כל האלגוריתם לאחר סריקת כל העץ או החלטה שרירותית של מגבלת זמן הרצה. את האלגוריתם הכללי ניתן לממש על מגוון בעיות שאין להם פתרון פולינומיאלי. עבור כל סוג בעיה נדרש לשנות את הפרמטרים כך שהאלגוריתם יצליח למצוא פתרון אופטימאלי בזמן סביר.

להלן הגדרות והסברים לשלב ה- BnB למימוש ספציפי לפתרון בעיית SCS.

עבור מחרוזות r ו- s נגדיר צמד (r^e, r^f) כך שהחלק r^e הוא החלק הארוך ביותר מתוך התחילית של r ש- s מכיל. r^f הוא השארית שאינה מוכלת ב- s . נגדיר פונקציה $\gg$ לצורך חישוב הצמדים (r^e, r^f) עבור מחרוזות כלשהי ביחס למחרוזת הפתרון s .

$$s \gg \epsilon \triangleq (\epsilon, \epsilon)$$

$$\epsilon \gg r \triangleq (\epsilon, r), \text{ if } r \neq \epsilon$$

$$as \gg ar \triangleq (ar^e, r^f), \text{ where } s \gg r = (r^e, r^f)$$

$$as \gg br \triangleq s \gg br, \text{ if } a \neq b$$

בצורה אינטואיטיבית הפונקציה מאפשרת להגדיר את החלק במחרוזת שכבר "מטופל" ע"י התחלה של מחרוזת-על. בכל שלב, עפ"י מחרוזת-העל המוצעת (s^t) מחשבים את קבוצות המחרוזות מתוך L שאינה "מכוסה" עדיין ע"י מחרוזת הפתרון. חלק זה מוגדר R .

$$R = \{r_i | (s_i^e, r_i) = s^t \gg s_i, s_i \in L\}$$

האלגוריתם מתחיל מפתרון ריק ($s^t = \epsilon$). כעת מחלקים ל- $|\Sigma|$ בעיות משנה שכל אחד מזוהה ע"י התו הראשון שאפשר להוסיף למחרוזת הפתרון. צמתים שמכילים תו שלא קיים בראש אף אחת מהמחרוזות ב- L נזרקים מידית. כעת נספור את כמות התווים הנמצאת בראש כל אחת מהמחרוזות ב- R עבור כל תו מהאלפבית. הסכום המקסימאלי של המצאות התו במחרוזת מוגדר כ- $M(a, R)$. ברור שמחרוזת-על חייבת להכיל את הכמות הזו של התווים (עפ"י הגדרת מחרוזת-על נדרש להגיע אל המחרוזת רק ע"י מחיקת תווים). לכן הגבול התחתון הוא סכום כל הסכומים המקסימאליים פלוס אורך הפתרון עד כה. הגבול העליון הוא פשוט שרשרת כל המחרוזות ב- R אל מחרוזת הפתרון. על מנת להקטין את כמות הצמתים ובעיות המשנה שצריכים לטפל בהם נדרש להפעיל מנגנון בחירה חכם. בכל צומת משתמשים בהיוריסטיקה מסוג MM עם משקלים לבחירת קבוצת הצמתים הנבחרת (H1).

כמו באלגוריתם MA, גם כאן שלב ה-MA המבוצע צריך להתמודד עם העובדה שמחרוזת כלשהי אינה בהכרח מחרוזת-על משותפת ולכן מבוצע תהליך תיקון להגעה לפתרון חוקי כמחרוזת-על משותפת. תהליך התיקון מבוצע ע"י הפעולות הבאות:

1. סריקת מחרוזת הפתרון משמאל לימין והורדת תווים לא מועילים (כלומר שלא מכסים אף תו במחרוזות הבעיה).
2. במידה ועדיין לא נוצרה מחרוזת-על משותפת מפעילים MM (עם משקלים – H1) על קבוצות המחרוזות שעדיין לא "כוסתה" (R) להשלמת יצירת פתרון חוקי.
3. בהסתברות מוגדרת מבוצעת פעולת שיפור מקומית ע"י פעולת מחיקת תווים (LS) כפי שהוצג באלגוריתם MA.

טבלה 11 – הגדרת פרמטרים לאלגוריתם BS-MA

אוכלוסייה	P_{size}	100
הסתברות ל- CROSS	P_x	0.9
הסתברות למוטציה	P_m	0.01
מספר אטרציות	$Maxeval$	100000
בחירה	$Selection$	Tournament
שחלוף	$Cross$	Uniform
מוטציה	$Mutation$	Bit Flip
כמות צמתים כוללת ל- BnB	K	10000
שלבים ב- BnB	L	10
הסתברות ל- LS (MA)	ביצוע עבור האופציות הבאות: $\{0, 0.01, 0.1, 0.5, 1\}$ $x=0$ משמעותו תיקון רגיל ללא השיפור המקומי כאשר x הוא ההסתברות באחוזים להפעלת LS באיטרציה.	

חישוב איכות הפתרון החלקי בכל שלב BnB מבוצע ע"י:

$$quality(s^t, L) \triangleq \sum_{s_i \in L} \{ |s_i^e| \mid (s_i^e, r_i) = s^t \gg s_i \}$$

כלומר סכום האורכים של המחרוזות שכבר מכוסות ע"י המחרוזות המוצעת בצומת התהליך BnB. איכות גבוהה מציינת כיסוי יותר תווים מהמחרוזות השונות ב- L . בהנחה שאלגוריתם BnB מיצר ברמה מסוימת פתרונות זהים באורכם, אלגוריתם BnB ייקח את הצמתים שמספקים את האיכות הטובה ביותר. לפני הכנסה לשלב ה- MA הפתרונות הטובים ביותר (שוב, מבחינת איכות quality) יתוקנו ליצירת מחרוזת-על משותפת. שני סוגי קלטים נבחרו לבחינה.

1. מחרוזות אקראיות באורך משתנה. 8 מחרוזות. 4 באורך 40 תווים, 4 באורך 80 תווים. 5 קבוצות בעיה עם 2, 4, 8, 16 ו- 24 תווים באלפבית.
2. מקרה מציאותי. סדרת DNA של SARS בגודל 158 נבחרה כמחרוזת-על. 10 מחרוזות נוצר ממחרוזת ה-DNA ע"י השמטה של רצפים בהסתברויות שונות – 10%, 15%, 20%.

אלגוריתם הייחוס הנבחר להשוואה הוא Majority Merge – MM. הרצת אלגוריתמים MM / על כל הפרמוטציות האפשריות, מוגבל ל-100000 עבור השפה עם 16 תווים. עבור הרצת MA נלקח ממוצע של 25 הרצות.

בכל הרצה מצוינת התוצאה הטובה ביותר, הממוצע עם סטיות התקן, ואחוז השיפור בתוצאה הממוצעת מאלגוריתם הייחוס.

טבלה 12 – אורך מחרוזת-על קצרה ממוצעת ואחוז שיפור מחרוזות אקראיות אורך משתנה

ש	MM	WMM	AL	MA	MA-BS	BS
2	112.0 112.0±0.1 0.0%	114.8 114.8±0.0 -2.5%	121.4 123.4±2.0 -10.2%	111.2 112.6±0.8 -0.4%	110.6 110.7±0.0 1.2%	110.8 110.8±2.2 1.1%
4	152.6 153.4±0.7 0.0%	157.8 157.8±0.0 -2.8%	183.0 191.2±4.7 -24.6%	151.6 155.2±1.9 -0.5%	145.6 146.4±0.5 4.6%	149.4 149.4±3.6 2.6%
8	212.4 213.8±0.9 0.0%	208.2 208.2±0.0 2.6%	252.2 276.8±6.4 -29.5%	205.4 213.5±4.0 3.9%	191.6 192.6±1.4 9.9%	198.2 198.9±1.9 7.3%
16	283.8 286.1±2.0 0.0%	272.8 273.4±0.5 4.4%	320.6 352.9±7.4 -23.3%	267.0 281.8±5.9 5.5%	242.8 244.0±1.0 14.7%	253.0 253.0±5.3 11.6%
24	330.2 333.9±2.3 0.0%	324.0 325.2±0.7 2.6%	363.8 390.8±6.4 -17.0%	301.8 305.6±3.4 8.5%	280.2 281.2±0.8 15.8%	287.8 287.8±1.7 13.8%

טבלה 13 – אורך מחרוזת-על קצרה ממוצעת ואחוז שיפור SARS-158

gap%	MM	WMM	AL	MA	MA-BS	BS
10%	158.0 158.0±0.0	158.0 158.0±0.0	307.0 315.2±6.8	158.0 158.0±0.0	158.0 158.0±0.0	158.0
15%	160.0 160.0±0.0	231.0 231.0±0.0	293.0 304.3±8.8	158.0 159.0±2.8	158.0 158.0±0.0	158.0
20%	228.0 229.6±1.8	266.0 266.0±0.0	274.0 288.3±8.6	159.0 177.0±9.3	158.0 158.0±0.0	158.0

עבור מחרוזות אקראיות האלגוריתם המשולב טוב מהאחרים בצורה ברורה.
אלגוריתמים MM ו-WMM מוצאים את הפתרון האופטימאלי עבור בעיות SARS במרווחים נמוכים (10%). אלגוריתמים BS ו-BS-MA מוצאים את הפתרון האופטימאלי, אולם MA מחמיץ את הפתרון עבור מרווחים גדולים (20%).

לצורך ניתוח דינאמי וניתוח רגישות של המודל המשולב BS-MA בוצעה בחינת המנגנון הפנימי של האלגוריתם המשולב. בבחינה מדוקדקת של צעדי האלגוריתמים נראה כי הפתרונות המשופרים נמצאו ע"י שלב ה-MA. ב-37% מההפעלות של שלב ה-MA נמצא פתרון משופר ואילו בשלבי ה-BS רק ב-10%. לעומת זאת, ה-BS יצר מועמדים טובים לפתרונות אופטימאליים שעזרו ל-MA להתפתח לפתרונות טובים יותר מאשר הרצת MA רגילה. עבור אלפבית עם 24 תווים, ה-MA הרגיל נדרש ל-30815 דורות ואילו ה-MA המשולב נדרש ל-18055 דורות.

מספר פרמטרים ניתן לכייל בהפעלת האלגוריתם BS-MA. הפרמטר l (כמות סריקות ב-BnB) נועד לאיזון בין השלבים השונים של האלגוריתם. הפרמטר k (כמות צמתים ב-BnB) להגדרת רוחב הסריקה של ה-BS. ניתוח רגישות בוצע על פרמטרים אלו.

הפרמטר k קשור ישירות לכמות ההשקעה הנדרשת בביצוע חישובי הגבולות עבור שלב ה-BS. הגדלת הפרמטר ישפר את התוצאות המחושבות ע"י ה-BS אבל ישפיע על זמן הסריקה בעץ במיוחד כאשר חישוב הגבולות הוא כבד. במקרה של בעיית ה-SCS חישוב הגבולות הוא פשוט (סריקת כל המחרוזות) ולכן ניתן להעלות את k בלי לשלם הרבה בזמן החישוב. העלאה של k מעל ל-10000 גורמת לירידה בביצועים מכיוון שזה מחזק את שלב ה-BS בצורה שמבטלת את היתרונות של שלב ה-MA.

הפרמטר l קשור לגבולות של תהליך ה-BnB ועד כמה הם קרובים/צפופים. כשהגבולות צפופים, ה-BnB מבצע התקדמות מהירה יחסית (מבטל הרבה פתרונות מחוץ לגבולות), ואז מומלץ לשמור ערכים נמוכים ל- l . במקרה של בעיית ה-SCS הגבולות אינם קרובים. נראה כי פתרונות טובים מתגלים לאחר "צלילה" מסוימת בעץ. לצורך כך בוצע חישוב דינאמי של l . החישוב הדינאמי כלל הרצה מקדימה של H1 לבעיה, ולקיחת 0.7 כפול אורך הפתרון שנוצר בהרצת H1.

4.9 אופטימיזציה אב טיפוס אבולוציונית - POEMS

אלגוריתמים היוריסטיים חמדניים (MM) עשויים להיות מושפעים לרעה מתכונת ה"חמדנות". אלגוריתמים אבולוציוניים וממטים נועדו לעקופת בעיה זו ע"י הרחבת אזורי החיפוש. לאחרונה הוצעה אפליקציה לאופטימיזציה אבולוציונית בעזרת אב טיפוס (Prototype Optimization with Evolved Improvement Steps – POEMS) [2]. ה- POEMS הוא אלגוריתם לאופטימיזציה שמחזיק פתרון מועמד הנקרא "אב טיפוס". מפתרון זה מפעילים תהליך אבולוציוני בניסיון לשפר אותו. בכל איטרציה מריצים אלגוריתם אבולוציוני המאתר את השיפור המיטבי האפשרי כאשר מתחילים מתוך המועמד (אב הטיפוס). ב- [3] הוצע מימוש אלגוריתם לפתרון בעיית ה- SCS בעזרת POEMS.

POEMS שונה מאלגוריתמים חיפוש מקומי רגיל בכמה נושאים:

- באיטרציה בודדת מחפשים בסביבה רחבה ע"י ביצוע מספר פעולות בסיסיות על הפתרון בניגוד לביצוע שינוי מקומי בודד.
 - חיפוש שיפור ע"י אלגוריתם אבולוציוני אשר יעיל כאשר גודל הקלט הוא גדול.
- ה- POEMS מוגן יותר מאלגוריתמים אחרים מהסכנה להיכנס למינימום/מקסימום מקומי. הגנה זו נובעת בזכות חיפוש פתרון בסביבה יותר גדולה, כלומר מציאת שיפור ע"י ביצוע רצף של מספר שינויים במחרוזת הפתרון. הרעיון המרכזי באלגוריתם POEMS הבסיסי הוא למצוא בכל שלב את העדכון הטוב ביותר שניתן לבצע לפתרון הבסיסי (אב הטיפוס).

אלגוריתם 9 - POEMS בסיסי

POEMS (L)

Input: $L - m$ strings $s_1, \dots, s_m$

1: $i \leftarrow 0$

2: generate ($s^{(i)}(L)$)

3: repeat

4: $i \leftarrow i+1$

5: $BestUpdateSequence \leftarrow \text{run_EA}(s^{(i-1)}, L)$

6: $Cand \leftarrow \text{apply}(BestUpdateSequence, s^{(i-1)})$

7: if $Cand$ not worse_than $Prototype^{(i-1)}$

8: $s^{(i)} \leftarrow Cand$

9: else

10: $s^{(i)} \leftarrow Prototype^{(i-1)}$

11: until Poems Termination Condition

12: return ($s^{(i)}$)

הפעלת generate (שורה 2) יוצרת אב טיפוס ראשוני (s) . אב טיפוס הינו מחרוזת בגודל משתנה מעל השפה Σ . כמובן ישנה חשיבות לנקודת ההתחלה של האלגוריתם. נקודת התחלה אפשרית הינה המחרוזת הארוכה ביותר ב- L . כך מקבלים את כל התווים של מחרוזת זו (כבר מחרוזת-על של מחרוזות זו) ונדרש להוסיף את התווים של המחרוזות האחרות במקומות המתאימים. רצף פעולות עדכון מיוצג ע"י רצף של פעולות בסיסיות באורך קבוע. בשורה 5 אנו מנסים למצוא את רצף פעולות העדכון הטוב ביותר על בסיס אב הטיפוס של סוף האיטרציה הקודמת (BestUpdateSequence).

לצורך מציאת העדכון הטוב ביותר מבוצע תהליך אבולוציוני. אם לאחר סיום תהליך זה הושג שיפור באיכות הפתרון (או לפחות לא הורע הפתרון), תוצאת הפעלת רצף פעולות העדכון על אב הטיפוס נלקח להיות אב הטיפוס של השלב הבא. אם הורע הפתרון, אב הטיפוס לשלב הבא נשאר ללא שינוי. כל תו מוגדר ע"י צמד $(value, id)$. כאשר $value$ יכול להיות כל תו ב- Σ , ו- id הוא מזהה חד-חד-ערכי המצביע על תו ממחרוזת הפתרון. לא כל מחרוזת היא בהכרח מחרוזת-על משותפת ולכן חישוב הכשירות חייב להביא זאת לידי ביטוי. פונקצית הכשירות מחושבת כך:

$$F(s) = C(s) + S(s)$$

כאשר $C(s)$ מייצג את גודל הכיסוי של s על רשימת המחרוזות ב- L , כלומר עד כמה s קרובה להיות מחרוזת-על משותפת, ו- $S(s)$ מייצג עד כמה המחרוזת s קצרה. חישוב $C(s)$:

$$C(s) = \sum_{s_i \in L} \text{Align}_{\text{orig}}(s, s_i)$$

ערך $\text{Align}_{\text{orig}}(s, s_i)$ מחושב עפ"י אלגוריתם Needleman-Wunsch [11] ללא עונשים על פערים. ערכי $\text{Align}_{\text{orig}}(s, s_i)$ יכולים להיות מ-0 ועד אורך s_i . חישוב $S(s)$:

$$S(s) = \frac{(\text{sum}L - |s|)}{\text{sum}L}$$

$$\text{sum}L = \sum_{s_i \in L} |s_i|$$

ערכים אפשריים ל- $S(s)$ הם מספרים בין 0 ל-1. ככל שמחרוזת הפתרון s קצרה כך ערך הכשירות $S(s)$ גדול יותר. עפ"י הנוסחה המלאה של $F(s)$ חשיבות גבוהה יותר למציאת פתרון קביל, ורק אחר כך מציאת פתרון קצר.

האלגוריתם האבולוציוני ב- POEMS משפר "כרומוזום" בגודל MaxGenes, שכל גן מייצג פעולה בסיסית הכוללת את סוג הפעולה ופרמטר נלווה. בטבלה 14 מפורטות פעולות בסיסיות אפשריות.

טבלה 14 - פעולות עדכון אפשריות ב- POEMS

פעולה	פרמטרים	משמעות	הערות
NOP	אין	לא לבצע שינוי	בעזרת פעולה זה ניתן ליצר רצף פעולות באורך קצר מהאורך המוגדר בכרומוזום.
MoveLetter	$id1, id2$	הזזת התו ב- $id1$ למיקום הצמוד מיד אחרי $id2$	
InsertLetter	l, id	הכנסת תו חדש l מיד אחרי id	
DeleteLetter	Id	מחיקת התו ב- id	
ChangeLetter	l, id	משנה את התו ב- id לערך l	

בכל איטרציה של ה- POEMS האלגוריתם בוחר אקראית אוכלוסיה שבה רצפים של פקודות. כל הפקודות בעלי אותה הסתברות לבחירה. במהלך השלב האבולוציוני הפעולות המבוצעות משתנות ע"י ביצוע שחלוף (Crossover) ומוטציה (Mutation). לצורך שחלוף משתמשים בהחלפת פעולות בין פריטים באוכלוסיה, ולצורך ביצוע מוטציה מבצעים שינוי בפעולה עצמה או שינוי אחד הפרמטרים שלה.

ביצוע שחלוף בהסתברות p_{cross} אחרת מבוצעת מוטציה על כל פעולה בהסתברות P_{mutate} . בנוסף מבוצע שימוש ב"נישות" (Niching). ברור שככל שמתקדמים בתהליך והפתרון הבסיסי (אב הטיפוס) הופך להיות טוב יותר, לקיחת רצף פעולות אקראיות יקלקל את איכות התוצאה. טבעי שרצף אקראי ארוך, יהרוס יותר מרצף אקראי קצר. מצד שני, קיים רצון להתפרש ולחפש פתרונות בסביבה יותר רחבה. לכן צריך למנוע מהאלגוריתם האבולוציוני לנטות לפתרונות עם רצף פעולות קצר. לצורך כך הוצע אסטרטגית החלפת נישות ובחירת טורניר בנישות.

אוכלוסיה בגודל N מחולקת ל- $MaxGenes$ נישות $\{niche_1, niche_2, \dots, niche_{MaxGene}\}$ בגודל שווה $(N/MaxGenes)$. כל נישא יכולה להכיל רצפים באורך גדול או שווה ל- i , כאשר i הוא מספר הנישה. בחירת טורניר בנישות מבצע בשלב ראשון בחירה אקראית של נישא i , ואז הרצת טורניר בין t רצפים בתוך הנישה. כלומר רק רצפים באורך i או יותר יכולים להיבחר כהורים לדור הבא. אסטרטגית החלפת נישות דואגת שכל רצף נבחר להכנסה לאוכלוסיה, יכול להיכנס רק אם נמצא לו מחליף ראוי להוצאה. מחליף ראוי להוצאה הוא כזה שהכשירות שלו גרועה יותר מהכשירות של המועמד החדש

(תנאי טריביאלי), אבל גם כמות הפעולות שלו לא קטנה מכמות הפעולות של המועמד להכנסה (על מנת לעמוד בחוקי הנישואים).

אלגוריתם POEMS זה פועל בצורה טובה על בעיות מסדר גודל קטן. עבור בעיות מסדר גודל גדול זמן החישוב גדל בצורה מהירה. הסיבה העיקרית לכך היא תהליך חישוב $align_{orig}$. תהליך חישוב $align_{orig}$ הוא ב- $O(MN)$ כאשר M ו- N הם גדלי המחרוזות לחישוב s ו- s_i , ומבוצע m פעמים (כמות המחרוזות ב- L) בכל חישוב F . זמן החישוב היה ארוך מאד עבור מחרוזות ארוכות. לצורך שיפור זמני ריצת האלגוריתם הוצא אלגוריתם $POEMS-f$, שמטרתו שיפור זמני חישוב $F(s)$ [3]. לצורך שיפור זמני החישוב הארוכים בוצע שינוי בשגרת חישוב $align$. החלפת השגרה בתהליך הבא ($align_{mod}$):

אלגוריתם 10 - $POEMS-f$ חישוב $Align_{mod}$

$align_{mod}(s \in \Sigma^*, s_i \in L)$

```

1: let  $k \leftarrow 1$  (index for position in  $s$ )
2: let  $l \leftarrow 1$  (index for position in  $s_i$ )
3: let  $result \leftarrow 0$ 
4: while  $k \leq |s|$  and  $l \leq |s_i|$  do
5:   if  $s[k] = s_i[l]$  then
6:     let  $result \leftarrow result + 1$ 
7:     let  $l \leftarrow l + 1$ 
8:   let  $k \leftarrow k + 1$ 
9: end while
10: return  $result$ 

```

שגרה זו מחשבת עפ"י קלט: s המועמדת למחרוזת-על ו- s_i כנציגה תורנית של אחת המחרוזות מתוך L . התוצאה של החישוב היא כמות התווים ב- s_i המוכלת ב- s .

מחזיקים 2 אינדקסים לסריקת המחרוזות: k עבור s , ו- l עבור s_i ומקדמים את החישוב באחד בכל פעם שהתו הבא ב- s זהה לתו הראשון ב- s_i (הסימון $s[k]$ משמעותו התו ה- k ב- s). רק במידה ומתאים, ממשיכים לקדם את האינדקס של s_i .

סיבוכיות שיטה זו היא $O(|s|)$ (רק $|s|$ השוואות). אבל החישוב לא נותן מדד מדויק על האיכות של הכיסוי ש- s מכסה את s_i .

באיור 1 נציג את ההבדלים בחישוב שתי שגרות חישוב ה-align עבור דוגמא קיצונית. נבדוק עבור המחרוזות $s_i = \text{"mraanan"}$, $s = \text{"raananm"}$.

$$\text{align}_{\text{mod}}(s, s_i) = 1$$

$i =$	r	a	a	n	a	N	m
$s_i =$						m	R a a N a N
						1	

$$\text{Align}_{\text{orig}}(s, s_i) = 6$$

$s =$		r	a	a	n	a	n	M
$s_i =$	m	r	a	a	n	a	n	
		1	2	3	4	5	6	

איור 1 - הבדלים במציאת כיסוי בעזרת $\text{Align}_{\text{orig}}$ ו- $\text{Align}_{\text{mod}}$

ניתן לראות שב- $\text{Align}_{\text{mod}}$ קיימת אי התאמה בתו הראשון. התאמה ראשונה מתרחשת רק שמגיעים לתו האחרון של מחרוזת הפתרון. לעומת זאת, בחישוב $\text{Align}_{\text{orig}}$, ע"י הזזה של תו אחד מוצאים יכולים כיסוי כמעט מלאה.

בביצוע ניתוח התנהגות אלגוריתם POEMS לפתרון בעיית SCS נראה כי יעילות השיפור של רצף הפעולות הייתה גבוהה יותר כאשר פעולות השיפור פעלו בתוך אזור מוגבל של מחרוזת הפתרון. על מנת לנצל תכונה זו, הוגדר מצב עבודה לאופטימיזציה בעזרת חלון-נע (POEMS-fw) [4]. החלון נועד להגדיר את אזור הפעולה המותר לשינוי על הפתרון המוצא. האזור שמותר לשנות בעת התהליך האיטרטיבי הוגבל לחלון של גודל W והוא מוגדר ונבחר מחדש בכל מחזור. במהלך מחזור החישוב עצמו – לא מבוצע שינוי במיקום החלון.

אלגוריתם 11 - POEMS-fw

POEMS-fw (L, W)

Input: $L - m$ strings $s_1, \dots, s_m$

W – size of search window

```

1:  $i \leftarrow 0$ 
2:  $s^{(i)} \leftarrow \epsilon$ 
3: repeat
4:    $i \leftarrow i+1$ 
5:   if  $C(s^{(i-1)}) < \sum_{s_i \in L} |s_i|$ 
6:      $l = \max(0, |s^{(i-1)}| - W)$ 
7:   else
8:      $l = \text{random number between } 0 \text{ and } |s^{(i-1)}| - W + 1$ 
9:    $BestUpdateSequence \leftarrow \text{run\_EA}(s^{(i-1)}, L, l, W)$ 
10:   $Cand \leftarrow \text{apply}(BestUpdateSequence, s^{(i-1)})$ 
11:  if  $Cand$  not worse_than  $s^{(i-1)}$ 
12:     $s^{(i)} \leftarrow Cand$ 
13:  else
14:     $s^{(i)} \leftarrow s^{(i-1)}$ 
15: until Poems Termination Condition
16: return ( $s^{(i)}$ )

```

על מנת להתגבר על מגבלות פונקציה החישוב $align_{mod}$ שהוצע ב- POEMS עודכן התהליך כך שהוא יתחיל ממחרוזת ריקה (לקיחת אחת המחרוזות יכולה לגרום להיתקעות סביב פתרון לא אופטימאלי). מיקום החלון נקבע לפי מצבו של אב הטיפוס. אפשר לחלק את התהליך ל- 2 חלקים עקרוניים: חלק ראשון בו עדיין אב הטיפוס אינו מחרוזת-על ונדרש להוסיף או לשנות תווים על מנת ליצר ממנו פתרון חוקי. החלק שני בו אב הטיפוס כבר מחרוזת-על ומנסים להגיע לפתרון אופטימאלי ע"י הקטנת אורכו. בחלק הראשון, ורק באיטרציות הראשונות בהם אב הטיפוס קטן מגודל החלון ובודאי אינו מהווה מחרוזת-על משותפת ($|s| \leq W$), כל s יכולה להיות מעודכנת באיטרציה. בהמשך התהליך, כאשר s גדולה מגודל החלון ($|s| > W$) אבל אינה מהווה מחרוזת-על משותפת ($C(s) = \sum_{s_i \in L} |s_i|$) רק W תווים אחרונים של s יכולים להיות מעודכנים במחזור. בחלק השני, לאחר ש- s מהווה מחרוזת-על משותפת ($|s| > W$) וגם $C(s) = \sum_{s_i \in L} |s_i|$ מבוצעת בחירה אקראית של אזור רציף בגודל W . בשלב זה מנסים להקטין את אורך הפתרון וכל אזור יכול להוות שיפור. הפעלת האלגוריתם האבולוציוני (run_EA) מבוצעת עם הגבלת עדכון עפ"י נתוני החלון – מיקום התחלת החלון (l) וגודלו (W).

גם באלגוריתם מבוצע חישוב פונקציה הכשירות ע"י $Align_{mod}$.

ב- $Align_{mod}$ חישוב $C(s)$ מבוצע על ידי מעבר על כל s וחיפוש ההתאמות ל- s_i . בפועל רק חלון של W תווים יכולים להתעדכן במחרוזת הפתרון. ניתן לבצע שיפור נוסף ביעילות של חישוב $Align_{mod}$. הרעיון של שיפור זה הוא לבצע חישוב של $C(s)$ רק על החלק שיכול להיות מעודכן מהשינויים שמבוצעים באיטרציה (חלון בגודל W).

בשיפור זה (POEMS-2), בשלב האבולוציוני בתהליך, מעדכנים חלון בגודל W המתחיל באינדקס l במחרוזת הפתרון s . ברור שלא יבוצע עדכון לתווים $s[1..l-1]$ (הסימון $s[1..l-1]$ משמעותו $l-1$ התווים הראשונים במחרוזת s). כלומר ניתן לחשב את החלק הקבוע של $C(s)$ עבור כל אחד מהמחרוזות ב- L פעם אחת לפני תחילת האיטרציה. במהלך שינוי של s יבוצע החישוב עבור s ו- s_i אבל רק על החלק $s[l..|s|]$ (החלק המשתנה של s) מול חלקי המחרוזות $s_i [t_i+1..|s_i|]$ (חלקים שלא "כוסו" במחרוזות ב- L על ידי $s[1..l-1]$).

דוגמא ($W=5$)

$l=6$

$s=$	a	a	c	t	a	g	C	a	T	a		
$s_1=$	a	t	a	c	g	s	M	r	A	a	N	a
$s_2=$	a	a	b	y	e	d	M	r	A	a	N	a
$s_3=$	a	r	g	e	w	f	M	r	A	a	N	a

איור 2 - חישוב כיסוי C עבור החלק הקבוע

במצב זה 5 התווים הראשונים של s לא יכולים להתעדכן במהלך האיטרציה. לכן מסמנים את החלק בכל מחרוזות של L שמכוסה כבר על ידי החלק הראשון של המחרוזות (שלא מתעדכן באיטרציה) והוא יהיה הבסיס של המשך החישוב עבור הכיסוי של כל s .

מנקודת תחילת החלון נדרש לחשב כיסוי עד סוף המחרוזות מכיוון ששינוי בתוך החלון עלול להשפיע גם על הכיסוי של תווים מחוץ (אחרי) החלון.

ב- POEMS-2 אין שיפור ביכולת להגיע לפתרונות טובים יותר אלא רק שיפור בזמן הריצה כיון שחישובי C מבוצעים על חלקי מחרוזות יותר קצרים.

הבעיה באלגוריתם POEMS-2 היא שהתהליך האבולוציוני וחישובי ה- $F(s)$ גורמים לפתרונות לנטות לכיוון הוספת תווים, לא בהכרח בצורה יעילה. מאחר והוספת תו משפרת את הכיסוי מקדמת במספר שלם את פונקציית הכשירות, בודאי שזה עדיף למרות ה"תשלום" של ניקוד קטן מ- 1 בפונקציית הכשירות $S(s)$.

כלומר בסוף השלב הראשון של החישוב, נקבל אמנם מחרוזת-על משותפת אבל ארוכה מהנדרש, וכעת קיים קושי להגיע לתוצאות מיטביות כיון שכל הורדת תו עלולה לגרום לביטול כשירותה של מחרוזת זו כמחרוזת-על משותפת.

ב- POEMS-3 [4] שולב מנגנון נוסף – Epochs (עידנים/תקופות) לטיפול בבעיה זו. ע"י מנגנון זה ניתן לשפר את החיפוש של השלב הראשון כך שנקבל מחרוזת קצרה וקרובה לפתרון המיטבי. עידן (epoch) מיצג מספר מסוים של מחזורי POEMS. במהלך אותם מחזורי POEMS מגבילים את החלק ב- s שפעולות התיקון יכולים לפעול בו. למעשה באותו עידן נקודת ההתחלה של החלק המשתנה נשארת במקומה, וגודל החלק יכול לגדול עד לגודל חלון השינוי – W . שיטת העידנים מבוצעת רק על החלק הראשון של התהליך, בו s עדיין אינה מחרוזת-על משותפת. בחלק השני התהליך זהה לתהליך שמבוצע ב- POEMS-2.

אלגוריתם 12 - POEMS-3

POEMS-3 ($L, W, EPOCH$)

$L - m$ string $s_1, \dots, s_m$

W – size of fragment allowed to be changed

$EPOCH$ – number of iteration without changing fragment

1: $i \leftarrow 0$

2: $l \leftarrow 1$

3: $s \leftarrow s_1[1..5]$ (5 first symbols of s_1)

4: repeat

5: $i \leftarrow i+1$

6: if $C(s) < \sum_{s_i \in L} |S_i|$ then (still not a common Supersequence)

7: if ($i \bmod EPOCH == 0$)

8: let $l \leftarrow s.length$

9: else

10: $l = \text{random number between } 0 \text{ and } |s^{(i-1)}| - W + 1$

11: $BestUpdateSequence \leftarrow \text{run_EA}(s^{(i-1)}, L, l, W)$

12: $Cand \leftarrow \text{apply}(BestUpdateSequence, s^{(i-1)})$

13: if $Cand$ not worse_than $s^{(i-1)}$

14: $s^{(i)} \leftarrow Cand$

15: else

16: $s^{(i)} \leftarrow s^{(i-1)}$

17: until POEMS termination condition

18: return s

התהליך הינו דומה מאוד ל- POEMS-2. ב- POEMS-3 מאתחלים את s לקבל את 5 התווים הראשונים של s_1 . בהמשך מעדכנים את תחילת החלון רק אחת ל- $EPOCH$ מחזורי חישוב ומתחילים בכל שלב מהתו האחרון של אב-הטיפוס וזאת בניגוד ל- POEMS-2 בו תמיד החלון אפשרי בכל W התווים האחרונים של אב הטיפוס.

לצורך בדיקת ביצועי האלגוריתמים מבוססי POEMS הוגדרו הפרמטרים הבאים לשלבים האבולוציוניים:

טבלה 15 - פרמטרים לביצוע הרצות האבולוציוני ב- POEMS

100	<i>PopsSize</i>	אוכלוסיה
20	<i>nichSize</i>	גודל נישות
5	<i>MaxGenes</i>	גודל גן (רצף פעולות) מקסימאלי
0.75	P_x	הסתברות ל- CROSS
$(1 - P_x) 0.25$	P_m	הסתברות למוטציה
1000	<i>Maxeval</i>	מספר אטרציות
Tournament	<i>Selection</i>	בחירה
uniform	<i>Cross</i>	שחלוף
Random substitution	<i>Mutation</i>	מוטציה
50	W	גודל חלון
20		מספר הרצות
100	<i>Epoch</i>	עידנים

ההשוואה בין MM, POEMS, POEMS-f, POEMS-fw בוצעה על 2 מחרוזות מתוך DNA אמיתי של SARS. נלקחו מחרוזות בגודל של 158 ו- 1269. כל מחרוזת כזו נלקחה כמחרוזת-על ומכל אחת יוצרו 10 מחרוזות ע"י הורדת תווים בסבירות של $gap\%$. נתוני ה- gap שנבחרו היו 10%, 15% ו- 20%.

טבלה 16 – אורך מחרוזת-על קצרה, ממוצעת, סטיית תקן, וכמות חישובים על 158-SARS

Gap%	MM	MA-BS*	POEMS	POEMS-f	POEMS-fw
10%	158 158.0±0.0	158 158.0±0.0	158 158.0±0.0 13424 (#evals)	158 158.0±0.0 50932	158 158.0±0.0 103455
15%	160 160.0±0.0	158 158.0±0.0	158 158.0±0.0 18294	158 158.1±0.44 76085	158 158.0±0.0 99960
20%	228 229.6±1.8	158 158.0±0.0	158 158.0±0.0 20198	158 165.75±9.7 96018	158 158.0±0.0 125107

ניתן לראות שאלגוריתם POEMS משפר משמעותית את התוצאות מול MM ומשתווה לביצועים של BS-MA. ביצוע שינוי לפונקציה הכיסוי C (אלגוריתם POEMS-f) גורמת להגדלת כמות ההשוואות מכיוון שהיא מטעה את האלגוריתם בשלבים הראשון, ומבצע תיקון לכך בהמשך בביצוע האיטרציה האבולוציונית. ביצוע התיקון בחלון מצומצם (אלגוריתם POEMS-fw) מגדיל גם הוא את כמות ההשוואות.

טבלה 17 – אורך מחרוזת-על קצרה, ממוצעת, סטיית תקן, וכמות חישובים על 1269-SARS

Gap%	MM	MA-BS	POEMS	POEMS-f	POEMS-fw
10%	1970 2039.9±32.9	1269 1269.0±0.0	1269 1269.9±1.0 124334(#evals)	אין נתונים	1269 1269.0±0.0 1140264
15%	2151 2236.4±30.4	1269 1269.0±0.0	1269 1270.7±1.6 142094	2404 2404.0±0.0 1986476	1269 1269.4±0.98 981050
20%	2163 2180.2±13.9	1269 1269.0±0.0	1278 1281.4±3.4 149286	1727 1911.3±190.6 1935289	1269 1269.8±1.5 1202520

סיפור משמעותי בביצועי POEMS (מבחינת מציאת המחרוזת הקצרה) מול MM אבל זמן הריצה הוא 16 שעות בגלל מורכבות החישוב של פונקציה הכיסוי C. אלגוריתם (POEMS-fw) הגיע לפתרון

אופטימאלי (1269) גם עבור $GAP=20\%$ (ב- 12 מתוך 20 הרצות) ובזמן ריצה של 10 דקות (שיפור משמעותי בהשוואה ל- 16 שעות).

טבלה 18 – אורך מחרוזת-על קצרה, ממוצעת, סטיית תקן, וזמן ריצה על SARS-1269

$GAP=20\%$, 20 הרצות

		BS	POEMS-fw	POEMS-2	POEMS-3
ש ל ב 1	Best		1326	1326	1268
	Average		1352 ± 18.7	1352 ± 18.7	1268.6 ± 0.9
	Time (sec)		118	51	89
ש ל ב 2	Best	1269	1268*	1268	1268
	Average	1269 ± 0.0	1268 ± 0	1268 ± 0	1268 ± 0
	Time (sec)	600	223	113	95

* באותה הרצה, אחד התווים לא נבחר בכל תת המחרוזות להרצה.

כל האלגוריתמים הצליחו למצוא את מחרוזת-העל המשותפת האופטימאלית. האצת החישוב בעת ביצוע POEMS-2 מכיוון שמפחיתים את כמות החישובים. אלגוריתם POEMS-3 אפילו משפר את הזמן הכללי מאחר שהשלב הראשון מיצר תוצאה הקרובה יותר לפתרון האופטימאלי (השלב הראשון שלו לוקח יותר זמן אבל השלב השני מאד מהיר – בממוצע 6 שניות). ב- 18 מתוך 30 הרצות הפתרון האופטימאלי נמצא כבר בסוף השלב הראשון. השוואת זמני הריצה ל- MA-BS אינה מדויקת מכיוון שמדובר בקלט שונה.

טבלה 19 – אורך מחרוזת-על קצר, ממוצע, סטיית תקן, וזמן חישוב על DNA_N1000_K500

		DR*	POEMS-2	POEMS-3
ש ל ב 1	Best	NA	2831	2712
	Average		2923±56.4	2774±36.9
	Time (sec)	NA	355	1772
ש ל ב 2	Best	NA	2713	2678
	Average	2769.1±32.7	2757±27.9	2718±25.2
	Time (sec)	NA	24050	4344

Deposit and reduction algorithm – DR*

DNA_N1000_K500 הינו DNA (4 תווים באפלבית) של NCBI וירוס. כולל 10 סטים של נתונים. 500 מחרוזות באורך 1000 תווים.

טבלה 20 – אורך מחרוזת-על קצר, ממוצע, סטיית תקן, וזמן חישוב על Protein_N500_K1000

		DR	POEMS-2	POEMS-3
ש ל ב 1	Best	NA	5904	5382
	Average		5981±44.1	5446±37.55
	Time (sec)	NA	863	4262
ש ל ב 2	Best	NA	5437	5308
	Average	5734±43.9	5487±21.9	5362±28.9
	Time (sec)	NA	100990	17141

Protein_N500_K1000 הוא protein (20 תווים) של NCBI וירוס. כולל 10 סטים של נתונים. 500 מחרוזות באורך 1000 תווים.

ביצועי POEMS-3 טובים מהשאר בשני סוגי הקלטים גם במציאת פתרון קצר יותר וגם מבחינת זמן הריצה.

4.10 כוורת דבורים מלאכותית

אלגוריתם כוורת דבורים מלאכותית (ABC – Artificial Bee Colony) הינו אלגוריתם מבוסס נחיל (swarm) שהוצג ע"י Karaboga בשנת 2005 לאופטימיזציה של בעיות נומריות [8]. ההשראה לאלגוריתם באה מההתנהגות של דבורי דבש בכוורת. מודל ההתנהגות של הדבורים כולל 3 רכיבים: דבורים מועסקות (employed bees), דבורים לא מועסקות סורקות (unemployed foraging bees) ומקורות מזון. תפקיד הדבורים הוא לחפש ולאתר מקורות מזון איכותיים. העיקרון הבסיסי הוא שיש משוב חיובי על מקורות מזון טובים ומשוב שלילי על מקורות מזון חלשים. ב-ABC קיימים 3 סוגי דבורים:

1. דבורים מועסקות – employed (forager) bees. הדבורים הללו מחוברות למקור מזון (פתרון).
 2. דבורים צופות – onlookers (observer) bees. דבורים שמסתכלות על ריקוד הדבורים המועסקות בכוורת.
 3. דבורים גששיות – scouts. דבורים המחפשות מקורות מזון בצורה אקראית.
- בתחילת הפעלת האלגוריתם כל מקורות המזון מתגלים ע"י הדבורים הגששיות (בצורה אקראית). מקורות המזון הללו מנוצלים ע"י דבורים אלו לצורך הגדלת מקורות המזון, עד שלא ניתן להשתמש בהם יותר ואז הדבורים הללו הופכות לדבורים גששיות. הדבורים הצופות מביטות על הדבורים המועסקות ולומדות איפה מקורות המזון הטובים ביותר. עפ"י מידע זה הדבורים מחפשות מזון נוסף בסביבת מקורות מזון קיימים. כמות המזון בכל אתר מדמה את איכות הפתרון שבפועל מחושב ע"י פונקציות הכשירות.

אלגוריתם 13 - ABC General Algorithm

ABC algorithm

1: initialization phase

2: repeat

3: employed bees phase

4: onlookers bees phase

5: scout bees phase

6: memories the best solution so far

7: until (cycle=maximum cycle number or maximum cpu time)

נפרט את השלבים השונים באלגוריתם. בשלב האתחול (שורה 1) מאתחלים את כל הדבורים באוכלוסייה לפתרונות אקראיים. בשלב הדבורים המועסקות (שורה 3), דבורים מועסקות מחפשות מקורות מזון נוספים בסביבתם. לאחר מציאת מקור מזון זה הן מעריכות את טיבו ומחליטות האם ללכת אליו. ביצוע חיפוש כזה יבוצע לדוגמא ע"י בחירה אקראית באחד המשתנים, ושינוי ערכו בצורה אקראית. מבצעים חישוב כשירות ובחרים את מקור המזון הטוב מבין המקורי והמעודכן.

בשלב הדבורים הצופות (שורה 4), הדבורים המועסקות משתפות את המידע שיש להם לגבי מקורות המזון שמצאו עד כה. הדבורים הצופות בוחרות את מקורות המזון שלהם עפ"י מידע זה. הבחירה מבוצעת כך שמקורות מזון טובים ייבחרו בהסתברות גבוהה יותר ממקורות מזון גרועים. ניתן להפעיל את שיטת גלגל הרולטה (שטח הבחירה בפתרון עם כשירות גבוהה יהיה גדול מהשטח של פתרון עם כשירות נמוך). אפשר לחשב את ההסתברות של בחירת דבורה m כך :

$$P_m = \frac{fitness_m}{\sum_{n \in B} fitness_n}$$

כאשר $fitness_m$ הוא ערך הכשירות של הדבורה m , ו- B הינה קבוצת כל הדבורים. לאחר בחירת פתרון, הדבורה מבצעת חיפוש אחר מזון בסביבת הפתרון, ובחירת הפתרון החדש במידה וערך הכשירות טוב מהערך הקודם. בשלב הדבורים הגששיות מבוצעת בדיקה וסינון של דבורים מועסקות אשר לא מצליחות למצוא מקורות מזון טובים, במשך מספר מחזורי חיפוש. דבורים אלו הופכות להיות דבורים גששיות והן מחפשות פתרונות אקראיים אחרים. מימוש ABC לפתרון בעיית ה- SCS (ABC-SCS) הוצג בצורה חלקית ב- [9]. להלן תאור הפתרון כפי שהוצג.

על מנת לשפר את יכולת האלגוריתם לעמוד במגבלה על פתרון אפשרי ל- SCS כמקור מזון כלשהו, מבוצעת פעולה מקדימה על L . מבוצע מעבר על המחזורות ב- L וסימון תדירות הופעת כל תו ב- Σ בכל קבוצת המחרוזות L . בעזרת אוסף כמות המופעים הזו יבחרו הדבורים הגששיות את הפתרון האקראי. התווים המופיעים הרבה פעמים יבחרו בהסתברות גבוהה יותר מתווים המופיעים בהסתברות נמוכה. כמות התווים שתבחר תהיה אקראית.

אלגוריתם 14 - ABC SCS Algorithm

ABC-SCS algorithm (L)

- 1: $AL \leftarrow$ calculate alphabet frequency in L
- 2: randomly generate new solution for worker bee using AL
- 3: cycle $\leftarrow$ 0
- 4: repeat
- 5: cycle $\leftarrow$ cycle+1
- 6: calculate fitness for workers bee F_m
- 7: for each worker bee
- 8: if fitness < avg(F_m) generate new solution for worker bee using AL
- 9: memories the best solution (highest fitness and shortest length)
- 10: until (cycle=maximum cycle number)

חישוב הכשירות של כל פתרון אפשרי מתואר בצורה כללית ע"י בדיקה של כמה תווים מופיעים באותו סדר במחרוזת הפתרון וברשימת המחרוזות ב- L. לא מוסבר במאמר איך מבוצע חישוב של כמות התווים המופיעים באותו סדר. דוגמאות כפי שתוארו במאמר:

טבלה 21 – דוגמאות לחישוב כשירות באלגוריתם ABC

פתרון אפשרי	מחרוזת S (מתוך L)	Fitness עבור מחרוזת S	התווים המתאימים
BABCcAACAAABb	ABCABAA	5	ABCABAA
cAAAAACCBC	ABCABAA	3	ABCABAA
BCCAABABCABA	ABCABAA	7	לא ברור איך מגיעים ל- 7

לא ברור מתוך המאמר מה מבוצע בשלב החיפוש באזור הפתרון: איך מבוצע השינוי המקומי (שלב הדבורים הצופים). במאמר לא מוצגות תוצאות עבור בעיות במימדים גדולים ולכן לא אציג השוואה לאלגוריתם זה.

5 הצגת אלגוריתם למציאת SCS ע"י ABC – כוורת דבורים

בסעיפים הקודמים הוצגו מספר אלגוריתמים לפתרון בעיית ה- SCS תוך התמקדות באלגוריתמים המבוססים על תהליכים בטבע. האלגוריתם המבוסס על כוורת דבורים לפתרון בעיית ה- SCS שהוצג בסעיף הקודם לא נותן פתרון מספק לבעיה מכיוון שלא מציג את תהליך החיפוש עצמו של הדבורים. כפי שאפשר לראות באלגוריתם 14 בשורות 5-9 מתוארת האיטרציה של המנגנון: אין בו תהליך שינוי של הפתרון, אלא רק מחיקת הדבורים שהכשירות שלהם היא מתחת לממוצע הכשירויות של כלל הדבורים.

בהמשך המאמר מוצגות דוגמאות לפתרון בעיות מסדר גודל קטן (תווים בודדים). לפיכך החלטתי לבצע מימוש אלגוריתם מבוסס כוורת דבורים לפתרון בעיית ה- SCS.

החלטה עקרונית שנדרשתי לה היא שיטת הייצוג של הבעיה. במהלך האיטרציות של התהליך האבולוציוני מבוצעים שינויים בפתרון המוצא על מנת לנסות להתקדם לכיוון פתרון אופטימאלי לבעיה. שינויים אלו, אפילו שיהיה קטנים, עלולים לגרום למחרוזת להפוך לפתרון לא חוקי, כלומר המחרוזת לא תהיה מחרוזת-על.

כפי שהוצג לעיל, קיימות 3 גישות להתגברות על מכשול זה:

1. שמירת הפתרון תמיד כמחרוזת-על משותפת (לדוגמא ב- GE/H1 בסעיף 4.5)
2. לאחר ביצוע השינויים (בתהליך האבולוציוני) הפעלת פונקציית תיקון (לדוגמא ב- MA בסעיף 4.7)
3. הגדרת פונקציית כשירות נכונה כך שפתרונות נכונים יקבלו ערך כשירות גבוה יותר. (לדוגמא ב- POEMS בסעיף 4.9)

מאחר ואת הגישות הראשונה והשנייה מימשתי באלגוריתמים קודמים, באלגוריתם ABC שמימשתי החלטתי על הגישה השלישית – לאפשר פתרונות לא חוקיים ולהתקדם לכיוון פתרון חוקי ע"י פונקצית הכשירות. העיקרון הבסיסי ב- POEMS הוא הגדרת פתרון אב טיפוס וביצוע תהליך גנטי על קבוצות של פעולות על מנת למצוא שיפור לאב טיפוס. הרעיון העקרוני של כוורת הדבורים הוא לנצל את כמות הדבורים הקיימת על מנת להרחיב את אזור החיפוש, בניגוד לאב טיפוס בודד. מאידך, על מנת לפשט את תהליך החיפוש עצמו, במקום חיפוש ע"י תהליך גנטי, חיפוש ע"י סריקה אקראית.

5.1 מימוש אלגוריתם למציאת SCS ע"י ABC

במימוש שביצעתי לפתרון בעיית ה- SCS מקורות המזון הם מחרוזות שמהוות הצעה לפתרון בעיית ה- SCS אבל אינם בהכרח מחרוזות-על משותפת. איכות מקור המזון נקבע עפ"י פונקצית כשירות. כאמור, פונקצית הכשירות חיונית לאלגוריתם לצורך התכנסות לפתרון חוקי ואופטימאלי לבעיה. הפונקציה צריכה לייצג את איכות הפתרון ב- 2 היבטים:

1. מידת הכיסוי (בדומה לתהליך החישוב שהוצג בסעיף 4.7 באלגוריתם $Align_{mod}$, 10).
 2. אורך הפתרון, כך שהתהליך יתקדם לכיוון של פתרון קצר יותר.
- מטרות כוורת הדבורים (ובפרט כל אחת מהדבורים) להתקדם ולשפר את איכות הפתרונות ע"י צעדים לכיוון פתרונות שבהם פונקצית הכשירות גדלה.
- באלגוריתם זה לכל דבורה מועסקת מותר לחפש מזון בטבעת חיפוש מזון. טבעת זו מוגדרת עפ"י המרחק המקסימאלי שמותר לדבורה להתרחק וגודל הטבעת. המרחק מתורגם בפועל לכמות תווים במחרוזת הפתרון של בעיית ה- SCS.
- לצורך הצגת האלגוריתם אגדיר מספר תהליכי משנה.

alignAndFix

שגרה זו דומה ל- $Align_{mod}$ שהוצג בסעיף 4.7 באלגוריתם 10 ומטרתה לחשב את הכיסוי של מחרוזות הפתרון של מול השפה. כלומר, האם המחרוזות היא מחרוזת-על משותפת ואם לא, אינדיקציה כמותית לגודל הכיסוי שלה (כלומר כמות התווים שהיא מכסה). שיפור שהוספתי לשגרה הוא היכולת לשנות את המחרוזות הנבדקת במידה וקיים תו שאינו מוסיף לערך של תוצאת ה- align.

בטבלה 22 אציג דוגמא לכך. בחישוב כיסוי של 3 מחרוזות אל מול פתרון מוצא. נבדוק עבור $s="mraabnanm"$ את תוצאות החישוב עבור $L=\{"raanan", "mraanan", "raananm"\}$.

טבלה 22 – דוגמא למחיקת תו בעזרת AlignAndFix

I	1	2	3	4	5	6	7	8	9	
S	m	R	a	a	b	n	a	n	M	
כמות התווים שמתאימים ב- L	1	3	3	3	0	3	3	3	1	20

I	1	2	3	4	5	6	7	8		
S_{new}	m	R	a	a	n	a	n	m		
כמות התווים שמתאימים ב- L	1	3	3	3	3	3	3	1		20

ניתן לראות שהתו b (התו החמישי) אינו משפר את פונקציית הכיסוי והפעלת הפונקציה עם המחרוזת $s_{fixed} = \text{"mraananm"}$ תחזיר תוצאה זהה אבל עם מחרוזת קצרה.
לצורך ביצוע המחיקה של התו נשתמש בשגרה $DELETE_k$ שהוצגה בסעיף 4.7.

אלגוריתם 15 - alignAndFix

$AlignAndFix(s \in \Sigma^*, L)$

```

1: let  $k \leftarrow 1$  //index for position in s
2: let  $result \leftarrow 0$ 
3: while  $k \leq |s|$  do
4:   if  $s[k] \neq S_i[1]$  for all  $S_i \in L$  then //the current char is not needed for cover
5:      $DELETE_k(s, L)$ 
6:   else
7:     let  $k \leftarrow k+1$ 
8:     for all  $S_i \in L$ 
9:       if  $|S_i| > 0$  and  $s[k] = S_i[1]$ 
10:        let  $result \leftarrow result+1$  //increase cover result
11:        let  $S_i \leftarrow S_i'$ , for  $S_i = aS_i'$  //  $S_i$  without  $S_i[1]$ 
12: end while
13: return  $result, s$ 

```

שגרה זו מבצעת חישוב ערך הכשירות של מחרוזת הפתרון. ראשית נדרש לחשב את גודל הכיסוי ע"י alignAndFix. קיימים 2 מצבים למחרוזת: מצב בו למחרוזת כיסוי מלא (כלומר היא מהווה פתרון חוקי לבעיית ה-SCS) ומצב בו הכיסוי חלקי.

במצב בו הכיסוי חלקי, לצורך חישוב הכשירות נדרש לשלב בין כמות התווים שהמחרוזת מכסה לבין אורך המחרוזת. נבצע את שקלול האורך ביחס לאורך המותר המקסימאלי למחרוזת זו. נניח כי ההפרש בין אורך המחרוזת לאורך המקסימאלי המותר למחרוזת הוא x . בודאי שניתן להגיע לכיסוי יותר גדול ע"י הוספת תו כלשהו הנמצא בראש אחת המחרוזות שעדיין לא כוסו ע"י מחרוזת הפתרון. אנו מניחים שבד"כ ניתן להוסיף יותר מתו אחד שנמצא בראש מספר מחרוזות. כמות התווים הממוצעת שאפשר להוסיף לכיסוי ע"י כל תו היא מקדם הקיצור $FitnessShortBonus$. מקדם זה יגדיל את תוצאת חישוב פונקציית הכיסוי במידה והמחרוזת קצרה מהגודל המותר לה להיות באותו שלב בתהליך.

כלומר, חישוב פונקציית הכשירות למחרוזות שאינן מהוות פתרון חוקי הוא:

$$Fitness(s) = alignAndFix(s) + (Distance - |s|) * FitnessShortBonus$$

עפ"י פונקציית הכשירות הזו, ניתן להגיע לערך כשירות הגדול מסה"כ הכיסוי האפשרי. אנו נאפשר זאת על מנת לאפשר לאלגוריתם להתקדם לכיוון פתרון בשלבים האחרונים לפני מציאת מחרוזת-על משותפת. במידה ולא נאפשר זאת אזי לא יהיה מוטיבציה למנגנון להתקדם ולמצוא כיסוי יותר גבוה (אם ערך כשירות עם מחרוזת בכיסוי נוכחי הוא שווה ערך לכשירות מחרוזת עם כיסוי גבוה יותר). ערך הכשירות המקסימאלי למחרוזת שאינה מהווה עדיין מחרוזת מתקבל כאשר חסר תו בודד:

$$\sum_{s_i \in L} |s_i| + W * FitnessShortBonus - 1$$

כאשר W הינו גודל טבעת החיפוש המותרת.

במידה והכיסוי מלא, נדרש לתת ערך כשירות גבוה יותר למחרוזות יותר קצרות. יש לשים לב שערך הכשירות של מחרוזת ארוכה הוא קטן יחסית למחרוזת קצרה אבל אינו קטן מערך כשירות של מחרוזת שאינה פותרת את בעיה ה-SCS. נבצע את חישוב הכשירות של מחרוזת s עבור השפה L :

$$Fitness(s) = 2 \sum_{s_i \in L} |s_i| - length(s) + W * FitnessShortBonus$$

כאשר $\sum_{s_i \in L} |s_i|$ הוא סכום כל המחרוזות הקיימות בשפה והוא מהווה את הפתרון הגרוע ביותר שניתן להגיע אליו. גם במקרה שהפתרון הוא פשוט שרשור כל המחרוזות הקיימות בשפה, ערך הכשירות גבוהה מכל מחרוזת התוצאה יותר טובה מכל מחרוזת שאינה מהווה מחרוזת-על משותפת.

אלגוריתם 16 - FitnessAndFix

$\text{FitnessAndFix}(s \in \Sigma^*, L, W)$

```

1:  $s, \text{cover} \leftarrow \text{AlignAndFix}(s, L)$ 
2: if  $\text{cover} = \sum_{s_i \in L} |s_i|$ 
3:    $\text{result} = \text{cover} * 2 - |s| + W * \text{FitnessShortBonus}$ 
4: else
5:    $\text{result} \leftarrow \text{cover} + (W - |s|) * \text{FitnessShortBonus}$ 
12: return  $\text{result}, s$ 

```

predictedResult

בשגרה זו מחשבים תוצאה חזויה למחרוזות הפתרון. בשלב הדבורים הצופות, הדבורים בוחרות את מקורות המזון בצורה יחסית לאיכות שלהם. הבעיה היא שהאורך הפתרונות בכוורת שונה. אם נבחר דבורים עפ"י ערך הכשירות בלבד, כנראה שנבחר דבורים עם פתרונות ארוכים כיון שהכיסוי שלהם כנראה יותר גדול מפתרונות קצרים ולכן בסבירות גבוהה ערך הכשירות שלהם גבוה יותר. דבר זה לא יאפשר לפתרונות קצרים להיבחר על מנת לנסות ולשפר את איכות המזון שלהם. הסיכוי שיתפתח פתרון ממחרוזת קצרה הוא נמוך.

לצורך בחירה יחסית נכונה בין פתרונות קצרים וארוכים נחשב את האורך הצפוי של הפתרון החוקי (פתרון שמהווה מחרוזת-על משותפת) שיתקבל מכל מחרוזת. ההנחה היא שהדבורה תמשיך למצוא מזון באותו קצב שהיא מצאה מזון עד כה. כלומר המשך הגדלת ערך הכשירות באותו קצב (כיסוי ביחס לאורך המחרוזת). להלן פונקציית החישוב האורך החזוי:

$$\text{PredictedResult}(s) = \frac{\sum_{s_i \in L} |s_i|}{\text{AlignAndFix}(s)} * |s|$$

עבור מחרוזת שמהווה פתרון חוקי האורך הנוכחי הינו האורך החזוי.

Rank

לצורך הערכת איכות הדבורה מבצעים חישוב דרוג הדבורים המועסקות. החישוב מבוצע בתחילת שלב הדבורים הצופות ובעצם מבצע מיון הדבורים עפ"י ערך הכשירות שלהן מהגבוה לנמוך.

CombinedRank

דרוג דבורים המשלב את ערך הכשירות של הדבורים ונתון החיזוי (PredictedResult) של כל דבורה. מבוצע תהליך מיון של כל הדבורים עפ"י תוצאת הכשירות שלהם (Rank) ומיון כל דבורה עפ"י תוצאת החיזוי שלהם (מהארוך לקצר). סכום המיקומים הינו הציון החדש המשולב והוא זה שמשפיע על הסתברות בחירת הדבורים הצופות.

LocalChange

פעולת חיפוש מזון בסיסית של דבורה מועסקת. ביצוע החיפוש ע"י ביצוע שינויים במחרוזת הנתונה. שינויים יכולים להיות :

1. שינוי תו
2. הוספת תו בסוף המחרוזת
3. מחיקת תו

כמות השינויים נבחרת בצורה אקראית ע"י בחירת ערך בין 0 לחצי מגודל החלון. ניתן לגרום לדבורה להיות נוטה לביצוע שינויים ע"י הגדלת כמות השינויים ע"י פרמטר $ChangeCount$. כלומר כמות

$$\text{השינויים שיבחרו יהיו בין } ChangeCount \text{ ל- } ChangeCount + \frac{W}{2}.$$

כל אחד מהשינויים מבוצע ראשית ע"י בחירת מיקום התו לביצוע השינוי. השינויים מבוצעים בתוך טבעת חיפוש כאשר נקודת הסיום יכולה להיות רחוקה מגודל המחרוזת. במידה ונבחר תו במיקום הגדול מאורך המחרוזת, מבוצעת פעולת הוספת תו לסוף המחרוזת. בחירת התו להוספה מבוצעת בצורה אקראית מתוך האלפבית.

כאשר מיקום התו אינו גדול מאורך המחרוזת מבוצעת פעולת שינוי או מחיקת תו. ביצוע הפעולה ע"י בחירת ערך השינוי לתו הנבחר. בוחרים אקראית מספר בין 1 לגודל האלפבית. מספר זה הוא ערך השינוי - $randomChange$. הערך של התו לאחר השינוי הוא :

$$NewChar = (CurrChar + randomChange) \bmod (ABsize + 1)$$

כלומר לאחר ביצוע השינוי, התו יהיה תו שונה מהתו הנוכחי והוא יכול להיות תו הנמצא מחוץ לשפה המוכרת (שווה ל- $ABsize$ שאינו תו חוקי בשדה). בחירת תו מחוץ לשפה יגרום בפועל למחיקת התו לאחר פעולת $AlignAndFix$. ניתן להגדיל את ההסתברות למחיקת תו ע"י הגדרת אלפבית וירטואלי גדול יותר וכך יתכן ויבחרו יותר תווים שאח"כ ימחקו ע"י פעולת $AlignAndFix$. הגדלת ההסתברות לביצוע פעולת מחיקה ע"י פרמטר $deleteFactor$.

$$NewChar = [CurrChar + random(1..ABsize - 1 + DeleteFactor)] \bmod (ABsize + 1 + DeleteFactor)$$

אלגוריתם 17 - LocalChange

LocalChange ($s \in \Sigma^*$, $L = \{s_1 \dots, s_m\}$, $StartPos$, $EndPos$)

- 1: let $s[i] \leftarrow$ not valid char for i if $EndPos \geq i > |s|$ // set all window positions in non valid char
- 2: let $k \leftarrow$ random change count
- 3: for k times do
- 4: let $r \leftarrow$ random item between $StartPos$ and $EndPos$
- 5: let $s[r] \leftarrow s[r]$ with a random change/delete/add
- 6: end for
- 7: return s

LocalSearchInWindow

בשגרה זו מבצעים חיפוש ע"י ניסיונות מחיקה של תווים לצורך שיפור ערך הכשירות של הפתרון. תהליך LocalSearch מבוצע באלגוריתם Heuristic LS (אלגוריתם 2 בסעיף 4.2). תהליך זה מבצע פעולת מחיקת תו במיקומים שונים במחרוזת וביצוע תיקון כך שהמחרוזת תהפוך חזרה למחרוזת-על משותפת. במידה והתיקון יוצר מחרוזת-על משותפת קצרה יותר, לוקחים את המחרוזת החדשה כפתרון. כך מבצעים בצורה מחזורית על כל התווים במחרוזת הפתרון. באלגוריתם LocalSearchInWindow נבצע תהליך דומה. נבצע חיפוש מקומי על מחרוזת שאינה בהכרח פתרון חוקי. החיפוש ינסה לגלות מחרוזת שפונקציית הכשירות שלה טובה יותר מפונקציית הכשירות של המחרוזת המקורית. את החיפוש נבצע ע"י חלון שהוגדר לשגרה (נקודת התחלה ונקודת סיום). זמן הריצה תלוי בגודל החלון ולא באורך המחרוזת (בהתעלמות מחישוב ה- fitness שבדאי תלוי בגודל המחרוזת אבל ממילא מתרחש בכל ניסיון שיפור).

אלגוריתם 18 - LocalSearchInWindow

LocalSearchInWindow ($s \in \Sigma^*$, $L = \{s_1 \dots s_m\}$, $StartPos, EndPos$)

```
1: let  $k \leftarrow StartPos$ 
2: while  $k < EndPos$  do //must be less than  $|s|$ 
3:   let  $r \leftarrow DELETE_k(s, L)$ 
4:   if  $fitness(r) < fitness(s)$  then
5:     let  $s \leftarrow r$ 
7:   else
8:     let  $k \leftarrow k+1$ 
10: end while
11: return s
```

CoverSequenceToLength

שגרה זו מבצעת הגדלת מחרוזת לגודל המותר. תהליך זה מבוצע לפני הגדלת טווח החיפוש של הדבורה (יוצג בהמשך בשלב הדבורים הצופות). התהליך מייצר מחרוזת חדשה, ע"י הוספת תווים בלבד עד האורך הרצוי. הוספת התווים מבוצעת עפ"י עקרון היוריסטיקה H1 (כלומר Majority Merge עם משקל לאורך. אלגוריתם H1 הוצג בסעיף 4.2 אלגוריתם 2).

אלגוריתם 19 - CoverSequenceToLength

CoverSequenceToLength ($s \in \Sigma^*$, $L = \{s_1 \dots s_m\}$, w)

```

1: let  $k \leftarrow 1$ 
2: while  $k < |s|$  do
3:   for  $s_i \in L$ ,  $s_i = s[k]s'_i$  do let  $s_i \leftarrow s'_i$  //remove all chars from L that currently covered by s
4:   let  $k \leftarrow k+1$ 
5: end while
6: while  $|s| < w$  do
7:   let  $\Sigma' \leftarrow \{a \in \Sigma | wsum(a) = \max\{wsum(b) | b \in \Sigma\}\}$ , where  $wsum(a) = \sum_{s_i = as'_i} |s'_i|$ 
8:   let  $b \leftarrow$  random letter from  $\Sigma'$ 
9:   for  $s_i \in L$ ,  $s_i = bs'_i$  do let  $s_i \leftarrow s'_i$ 
10:  let  $s \leftarrow sb$ 
11: end while
12: return s

```

calcFailAllowed

חישוב כמות הזמן (מחזורי חישוב) שמאפשרים לדבורה להישאר בחיפוש אחרי מזון באותה נקודה ללא הצלחה. עבור מקורות מזון מבטיחים, מאפשרים לדבורה להישאר יותר מחזורים בחיפוש אחר שיפור מקור המזון (שיפור הפתרון). לעומתם מקורות מזון גרועים, שעד כה לא הצליחו להניב תוצאות טובות, נאפשר לדבורה להישאר זמן קצר ללא התקדמות, לפני שניתן לה הנחיות נוספות. מספר המחזורים מחושב עפ"י מדרוג הדבורה לפי פונקציית הכשירות (rank) ביחס לדבורים אחרות. פתרונות בעלי ערך כשירות גבוה יקבלו יותר זמן מפתרונות בעלי ערך כשירות נמוך. לזמן ההמתנה הממוצע (אשר נקבע כפרמטר חיצוני - *scoutFailCount*) מוסיפים את ההיסט המחושב עפ"י מיקום הדבורה ביחס לדבורים האחרות מוכפל ב- *offsetFactor* שגם הוא מוגדר כפרמטר חיצוני.

$$FailCount_k = \frac{Rank_k - \frac{numOfBees}{2}}{numOfBees} * offsetFactor + ScoutFailCount$$

InitializationPhase

בתחילת הפעלת האלגוריתם כל דבורה מקבלת באופן אקראי את התחילית (prefix) של אחת ממחרוזות הבעיה. לכל דבורה מוגדר אורך מקסימאלי ראשוני, ומונה הכישלונות מאופס.

אלגוריתם 20 - InitializationPhase

InitializationPhase ($L=\{s_1\cdots,s_m\},w,startLen$)

- 1: for every bee k
- 2: init bee $_k$ to be first $StartLen$ of random $Si\in L$
- 3: $distance_k \leftarrow W$
- 4: $failCount_k \leftarrow 0$

EmployedBeePhase

תהליך זה מבצע מעבר על כל הדבורים המועסקות. עבור כל דבורה קובעים את טבעת החיפוש המתאימה לה. עבור דבורה שמצאה כבר פתרון חוקי בוחרים בצורה אקראית טבעת חיפוש במקום כלשהו לאורך המחרוזת. טבעת חיפוש היא בגודל $WindowSize$ (מוגדר לכל האלגוריתם). דבורה שעדיין לא מצאה פתרון חוקי, טבעת החיפוש היא במיקום הרחוק ביותר המותר לדבורה (כלומר אורך מחרוזת מקסימאלי המוגדר לדבורה).

במידה ומדובר במקור מזון חדש (מחרוזת חדשה, כלומר מקור המזון של הדבורה עודכן מהאיטרציה הקודמת), הדבורה מבצעת תהליך $LocalSearchInWindow$, כלומר ניסיון מחיקה מסיבי. במידה ולא מדובר בפתרון חדש (מחרוזת זהה למחרוזת מאיטרציה הקודמת) אין צורך לבצע תהליך מחיקה $LocalSearchInWindow$ (כיוון שבוצע בפעם הראשונה של המחרוזת) והדבורה מבצעת פעולת $LocalChange$.

עבור כל דבורה, הפתרון החדש יילקח אם הוא משפר את ערך הכשירות. במידה ונמצא שיפור מאפסים את מונה הכישלונות, ומאידך אם לא נמצא שיפור, מקדמים את מונה הכישלונות ב-1.

אלגוריתם 21 - EmployedBeePhase

EmployedBeePhase ($L=\{s_1, \dots, s_m\}, w$)

```

1: for every Employed bee  $k$ 
2:   if  $\text{cover}(s_k) > \sum_{s_i \in L} |s_i|$ 
3:      $\text{startPos} \leftarrow \text{rand}(0, \text{distance}_k - W)$ 
4:   else
5:      $\text{startPos} \leftarrow \text{distance}_k - W$ 
6:    $\text{endPos} \leftarrow \text{startPos} + W$ 
7:   if  $\text{failCount}_k = 0$ 
8:      $\text{newa} \leftarrow \text{LocalSearchInWindow}(s, L, \text{startPos}, \text{endPos})$ 
9:   else
10:     $\text{news} \leftarrow \text{LocalChange}(s, L, \text{startPos}, \text{endPos})$ 
11:    $\text{newFitness} \leftarrow \text{FitnessAndFix}(s, L)$ 
12:   if  $\text{newFitness} > \text{Fitness}_k$ 
13:      $\text{Fitness}_k \leftarrow \text{newFitness}$ 
14:      $s \leftarrow \text{news}$ 
15:      $\text{Failcount}_k \leftarrow 0$ 
16:   else
17:      $\text{Failcount}_k \leftarrow \text{Failcount}_k + 1$ 
18: end for

```

OnlookersBeePhase

בשלב הדבורים הצופות, בוחרים מקורות מזון שנמצאו ע"י הדבורים המועסקות, ומבצעים חיפוש מקומי נוסף לשיפור מקורות מזון אלו. בחירת הדבורים מתבצעת בצורה אקראית עם עדיפות למקורות מזון טובים. בחירה כזו ממקדת את החיפוש במקורות מזון טובים שמגדילים את הסיכוי לשפר את מקורות המזון ולמצוא שיפורים טובים לפתרונות קיימים. לשם כך נבצע *CombinedRanked* של כל דבורה, כך שמיקומה ביחס לאחיותיה הדבורים עפ"י פונקצית הכשירות, ועפ"י חישוב החיזוי, יקבע את ההסתברות שלה להיבחר.

בחירת $\text{choose}(C^k)$ מבוצעת כמו שהוצג עבור אלגוריתם ממושבת נמלים (סעיף 4.6 אלגוריתם 6).

מבוצעת בחירה אקראית יחסית עם גבול q_0 .

על כל פתרון שנבחר בשלב זה מבוצע תהליך דומה לניסיון השיפור בשלב הדבורים המועסקות:

אם הפתרון כבר מחרוזת-על משותפת, בוחרים טבעת אקראית, אם לא הטבעת בקצה הטווח.

בהסתברות P (פרמטר חיזוני) מפעילים את פונקצית השינוי *LocalChange* בהסתברות $1-P$ מפעילים

את פונקצית השינוי *LocalSearchInWindow*.

עבור כל דבורה צופה, במידה ונמצא שיפור בערך הכשירות, מעדכנים את הפתרון של הדבורה המועסקת ומאפסים את מונה הכישלונות של הדבורה. במידה ופתרון לא נבחר בשלב זה, הוא עובר ללא שינוי לאיטרציה הבאה.

אלגוריתם 22 - OnlookersBeePhase

OnlookersBeePhase ($L=\{s_1 \dots, s_m\}, w$)

```

1: for every bee  $l$  (out of number of onlookers bee)
2:    $k \leftarrow \text{choose}(C^k)$ 
3:   if  $\text{cover}(s_k) > \sum_{s_i \in L} |s_i|$ 
4:      $\text{startPos} \leftarrow \text{rand}(0.. \text{distance}[k]-W)$ 
5:   else
6:      $\text{startPos} \leftarrow \text{distance}[k]-W$ 
7:    $\text{endPos} \leftarrow \text{startPos}+W$ 
8:   in  $P_0$  propability
9:      $\text{newa} \leftarrow \text{LocalSearchInWindow}(s_k, L, \text{startPos}, \text{endPos})$ 
10:  else
11:     $\text{news} \leftarrow \text{LocalChange}(s_k, L, \text{startPos}, \text{endPos})$ 
12:   $\text{newFitness} \leftarrow \text{FitnessAndFix}(s, L)$ 
13:  if  $\text{newFitness} > \text{Fitness}_k$ 
14:     $\text{Fitness}_k \leftarrow \text{newFitness}$ 
15:     $s \leftarrow \text{news}$ 
16:     $\text{Failcount}_k \leftarrow 0$ 
17: end for
18: for every bee  $k$ 
19:  if  $\text{Fitness}_k$  with no change from last round
20:     $\text{Failcount}_k \leftarrow \text{Failcount}_k + 1$ 
21: end for

```

ScoutBeePhase

מטרת השלב הזה הוא להסתכל על מצב הפתרונות הקיימים עד כה בכוורת, ולתת הוראות שינוי מעשיות לדבורים שלא מצליחות למצוא שיפור במקורות המזון שלהם.

קיימות 2 הוראות שינוי פעולה אפשריות:

1. להגדיל טווח החיפוש

2. להתחיל בחיפוש ממחרוזת אקראית.

עבור כל דבורה, בהתאם לאיכות המזון שלה, מחשבים את הזמן שמאפשרים לדבורה זו לחפש מזון. חישוב ע"י מבוצע ע"י calcFailAllowed .

ברגע ההגעה לזמן המוגדר ללא מציאת שיפור מבוצעת פעולת חיפוש אחרונה (מעין הזדמנות אחרונה): עבור דבורה עם פתרון שמהווה כבר מחרוזת-על משותפת מבוצע תהליך Heuristic LS. במידה וגם זה לא מצא פתרון עם פונקצית כשירות טובה יותר, מתחילים עם מקור מזון חדש, באורך התחלתי, כאשר התווים מאותחלים לערכים אקראיים, ומרחק מקסימאלי קרוב כמו שהוגדר בשלב ה-InitializationPhase.

עבור דבורה שהפתרון עדיין לא מהווה מחרוזת-על משותפת, מבוצע חיפוש אחרון במרחק המותר. החיפוש מבוצע ע"י $\text{coverSequenceToLength}$, כלומר ניסיון שיפור ע"י השלמת המחרוזת לאורך המקסימאלי המותר לדבורה, ע"י פונקציה היוריסטית מבוססת על אלגוריתם H1. אם גם זה לא עזר, מבוצעת הגדלת טווח החיפוש של הדבורה.

אלגוריתם 23 - ScoutBeePhase

ScoutBeePhase ($L=\{s_1, \dots, s_m\}, w$)

```

1: for every bee  $k$ 
2:    $allowedFail \leftarrow \text{calcFailAllowed}(k)$ 
3:   if  $failCount_k = allowedFail$ 
4:     if  $\text{cover}(s_k) = \sum_{s_i \in L} |s_i|$ 
5:        $news \leftarrow \text{LS}(s, L)$  //full local search, for last chance change
6:     else
7:        $news \leftarrow \text{LocalSearchInWindow}(s, L, StartPos, EndPos)$ 
8:     if news is better than current
9:        $failCount_k \leftarrow 0$ 
10:       $S_k \leftarrow news$ 
11:    else if  $failCount_k > allowedFail$ 
12:      if  $\text{cover}(s_k) = \sum_{s_i \in L} |s_i|$ 
13:        init beek to be first StartLen of random  $S_i \in L$ 
14:         $distance_k \leftarrow W$ 
15:         $failCount_k \leftarrow 0$ 
16:      else
17:         $distance_k \leftarrow distance_k + stepSize$ 
18: end for
```

התהליך הכללי כולל פעולות אתחול הכוורת והרצה מחזורית של תהליך דבורים מועסקות, דבורים צופות ודבורים גששיות. בכל איטרציה לוקחים את התוצאה הטובה ביותר שנמצאה עד כה. ממשיכים עפ"י כמות האיטרציות שהוגדרה כפרמטר חיצוני.

אלגוריתם 24 - RM ABC SCS

- 1: InitializationPhase
- 2: repeat
- 3: EmployedBeePhase
- 4: OnlookersBeePhase
- 5: ScoutBeePhase
- 6: memory the best solution so far
- 7: until (cycle=maximum cycle number)

6 יישום ותוצאות

במסגרת העבודה יושמו מרבית האלגוריתמים שנסקרו בעבודה עפ"י ההגדרות מהפרסומים הקיימים:

1. אלגוריתמים היוריסטיים
 2. אלגוריתם גנטיים מבוססים היוריסטיקה
 3. אלגוריתם ממטי (חיפוש מקומי)
 4. אלגוריתם מבוסס התנהגות נמלים בקן
- בנוסף מומש אלגוריתם מבוסס התנהגות דבורים בכוורת עפ"י הפיתוח העצמי.

היישומים בוצעו במערכת הפעלה Window7 בסביבת הפיתוח VS2008 ושפת פיתוח C++. השוואת האלגוריתמים השונים בוצעו ע"י מספר סוגי בעיות כפי שהוצגו בפרקים לעיל:

1. בעיות אקראיות בגודל אחד ובגודל שונה.
2. בעיות של מחרוזות דומות, מבוססים על מבנה DNA מוכר (SARS) שממנו מורידים תווים בהסתברות של 0.1 או 0.2.
3. בעיות של מחרוזות דומות המתקבלות ממחרוזת אקראית בגודל 100 תווים ממנה מיצרים 10 מחרוזות ע"י מחיקה אקראית של 10 או 20 תווים.
4. בעיות מיוחדות. רצף ייחודי של תווים המקשה על מציאת הפתרון האופטימאלי. הוצגו בסעיף 4.5.

6.1 אלגוריתמים היוריסטיים

במימוש האלגוריתם H1 התגלתה בעיה שגרמה למימוש למצוא פתרון לא מיטבי. להלן השורה הבעייתית:

$$3: \quad \text{let } \Sigma \leftarrow \{a \in \Sigma \mid \text{wsum}(a) = \max\{\text{wsum}(b) \mid b \in \Sigma\}\}, \text{ where } \text{wsum}(a) = \sum_{s_i = a} |s'_i|$$

בשורה זה מבוצע חישוב של התו הבא להכנסה למחרוזת הפתרון. עבור כל תו מבצעים חישוב של סכום אורכי המחרוזות, שהתו נמצא בראשם (אורך המחרוזות נלקח ללא התו הראשון). לקראת סוף התהליך אפשר להגיע למצב בו כל המחרוזות ריקות למעט 3 מחרוזות בהם קיים תו בודד. לדוגמא ניתן להגיע למצב הבא:

$L = \{ "", "c", "", "", "c", "c" \}$

$\text{alphabet} = \{a, b, c\}$

קצת האלגוריתם סוכם את wsum עבור כל התווים ב-alphabet.

ברור ש- $\text{Wsum}(b) = \text{Wsum}(a) = 0$ מכיוון שהתווים a ו-b אינם נמצאים בראש אף אחד מהמחרוזות. הבעיה שגם $\text{Wsum}(c) = 0$ מכיוון שאורך המחרוזות ללא התו c הוא גם 0. במצב זה כל התווים

מקבלים ציון זהה שמהווה את הציון הגבוה ביותר. לכן האלגוריתם יבחר אקראית מבין כל התווים למרות שברור שנדרש לבחור בתו c.

שיפור אפשרי לאלגוריתם הוא ע"י הוספת ערך בסיסי ($BaseValue > 0$) לכל תו שנמצא בראש מחרוזת (מלבד האורך של המחרוזת עצמה).

$$3: \quad \text{let } \Sigma' \leftarrow \{a \in \Sigma \mid wsum(a) = \max\{wsum(b) \mid b \in \Sigma\}\}, \text{ where } wsum(a) = \sum_{s_i = a} |s'_i| + BaseValue$$

בטבלה 23 מפורטים אורכי הפתרון של מחרוזת העל המשותפת שהתקבלו בהרצת המימוש של האלגוריתמים ההיוריסטיים. בעיות מיוחדות L1, L2, ו-L3 מתוארות בפירוט בסעיף 4.5. אלגוריתמים H0 מבצע מציאת מחרוזת-על משותפת על 2 מחרוזות והחזרת התוצאה לקבוצת המחרוזות, אלגוריתמים MM בוחר בכל שלב את התו שמופיעים ברוב המחרוזות, אלגוריתם H1 הינו אלגוריתם MM עם משקל לאורך המחרוזת, אלגוריתם H2 הוא אלגוריתם H1 עם הסתכלות קדימה (בוחרים תו שייתן את התוצאה הטובה ביותר לאחר עוד n צעדי בחירה), אלגוריתם H3 בוחר את התו המתאים ביותר במקרה של תוצאה זהה עפ"י סימלוח הרצה של אלגוריתם H1 על קבוצה חלקית של מחרוזות. האלגוריתמים מפורטים בסעיף 4.2.

טבלה 23 – השוואת תוצאות הרצת אלגוריתם היוריסטיים

H3	H2	H1	MM	H0	
43	77	92	157	157	בעיה מיוחדת L1 (פתרון מיטבי 43).
79	91	93	121	121	בעיה מיוחדת L2 (פתרון מיטבי 79).
88	77	79	60	60	בעיה מיוחדת L3 (פתרון מיטבי 60).
118	112	113	113	143	4 מחרוזות אקראיות באורך 80 מעל אלפבית באורך 2
146	155	163	164	207	4 מחרוזות אקראיות באורך 80 מעל אלפבית באורך 4
276	258	272	281	288	8 מחרוזות אקראיות באורך 80 מעל אלפבית באורך 16
364	362	398	420	503	16 מחרוזות אקראיות באורך 80 מעל אלפבית באורך 16
119	115	114	116	153	4 מחרוזות אקראיות באורך 80 ו- 4 מחרוזות באורך 40 מעל אלפבית באורך 2
151	151	163	170	220	4 מחרוזות אקראיות באורך 80 ו- 4 מחרוזות באורך 40 מעל אלפבית באורך 4
280	306	306	341	373	4 מחרוזות אקראיות באורך 80 ו- 4 מחרוזות באורך 40 מעל אלפבית באורך 16
100	104	151	188	273	8 מחרוזות דומות באורך 90, 4 תווים, שנוצרו ממחרוזות באורך 100
156	106	167	202	258	8 מחרוזות דומות באורך 80, 4 תווים, שנוצרו ממחרוזות באורך 100
158	159	159	162	523	10 מחרוזות שנוצרות מ-DNA SARS באורך 158 ע"י מחיקת 10% מהתווים
195	222	221	220	464	10 מחרוזות שנוצרות מ-DNA SARS באורך 158 ע"י מחיקת 20% מהתווים

עבור בעיות מיוחדות H3 מספק את התוצאות הטובות ביותר, אבל גם אלגוריתם זה לא מוצא את הפתרון האופטימאלי עבור L3. עבור מחרוזות אקראיות באורך שונה אלגוריתמים המתחשבים באורך המחרוזות (H1 למשל) מגיעים לביצועים טובים מההיוריסטיקות ללא התחשבות בנתון זה. עבור בעיות דומות ניכר שיפור בביצועי אלגוריתמים המבצעים חיפוש יותר מעמיק – H2, H3. מימוש אלגוריתם H2 (ביצוע היוריסטיקה המבצעת הסתכלות קדימה) בוצע בצורה רקורסיבית כך שניתן לבצע מספר רב של צעדי הסתכלות קדימה. כלומר בחירה בתו שייתן התוצאה הטובה ביותר לאחר תהליך של n בחירות של תווים. הרצת אלגוריתם H2 עם מספר צעדים של הסתכלות קדימה שיפרה לעיתים את הביצועים אבל לא בצורה חד משמעית. כל הפתרונות ההיוריסטיים לא נתנו תוצאות טובות (לא אופטימאליות) לבעיה המורכבת של ה-SARS 158 עם GAP 20% (השורה האחרונה בטבלה).

6.2 אלגוריתמים גנטיים מבוססים היוריסטיקה

על בסיס השיטה היוריסטית מימשתי אלגוריתם גנטי. לצורך כך השתמשתי בכלי הנקרא OPEN BEAGLE [13] בגרסה 3.0.2.

על מנת להשתמש בכלי נדרש להגדיר מחלקת קוד המתארת את עולם הבעיה. נדרש לספק את פונקציית הכשירות, את ייצוג הגנום לצורך ביצוע התהליך הגנטי ופרמטרים נוספים. הפרמטרים הנוספים מוגדרים בקובץ XML ומשפיעים על ביצוע התהליך הגנטי.

את האלגוריתם GA/H1 מימשתי בעזרת כלים אלו. את שאר האלגוריתמים מימשתי על בסיס אותה פלטפורמה אבל ללא שימוש בכלים לביצוע התהליכים האבולוציוניים. להרצה נבחרו הפרמטרים הבאים :

1. 5 תתי אוכלוסיות בגודל 150 (במקום 10 עפ"י ההמלצה המקורית)
2. הסתברות לביצוע מוטציה – 5%
3. הסתברות לביצוע שינוי בגן במוטציה – 20% עבור גן (כלומר כל איבר ישונה בהסתברות של 20%).
4. מוטציה בגן תבוצע ע"י שינוי של לפחות 0.1 במשקל של הגן.
5. סבירות לשחלוף – 95%.
6. ביצוע העתקה לדור הבא – 1%.
7. כל 10 דורות מעבר בין אוכלוסיות.
8. ערך BaseValue – 0.01.

טבלה 24 – תוצאות הרצת אלגוריתמים GA/H1 בהשוואה לאלגוריתמים היוריסטיים בסיסיים

GA/H1	H2	H1	
43	77	92	בעיה מיוחדת L1 (פתרון מיטבי 43).
80	91	93	בעיה מיוחדת L2 (פתרון מיטבי 79).
60	77	79	בעיה מיוחדת L3 (פתרון מיטבי 60).
108	112	114	4 מחרוזות אקראיות באורך 80 מעל אלפבית באורך 2
149	155	163	4 מחרוזות אקראיות באורך 80 מעל אלפבית באורך 4
246	258	272	8 מחרוזות אקראיות באורך 80 מעל אלפבית באורך 16
392	362	398	16 מחרוזות אקראיות באורך 80 מעל אלפבית באורך 16
108	115	115	4 מחרוזות אקראיות באורך 80 ו- 4 מחרוזות באורך 40 מעל אלפבית באורך 2
153	151	163	4 מחרוזות אקראיות באורך 80 ו- 4 מחרוזות באורך 40 מעל אלפבית באורך 4
278	306	306	4 מחרוזות אקראיות באורך 80 ו- 4 מחרוזות באורך 40 מעל אלפבית באורך 16
102	104	151	8 מחרוזות דומות באורך 90, 4 תווים, שנוצרו ממחרוזות באורך 100
99	106	167	8 מחרוזות דומות באורך 80, 4 תווים, שנוצרו ממחרוזות באורך 100
175	222	221	10 מחרוזות שנוצרות מ-DNA SARS באורך 158 ע"י מחיקת 20% מהתווים
158	159	159	10 מחרוזות שנוצרות מ-DNA SARS באורך 158 ע"י מחיקת 10% מהתווים

אלגוריתם GE/H1 מוצא את הפתרון האופטימאלי עבור בעיות מיוחדות. האלגוריתם משפר בצורה ברורה את הביצועים עבור בעיות עם מחרוזות דומות על כל סוגי הבעיות הדומות מכיוון שמגיע לתוצאות עם אורך קצר אם כי לא תמיד האופטימאלי. עבור בעיות אקראיות, ברוב ההרצות האלגוריתם מצא פתרונות טובים מהאלגוריתמים ההיוריסטיים.

6.3 אלגוריתם ממטי עם חיפוש מקומי

מומש אלגוריתם MA עם חיפוש מקומי שהוצג בסעיף 4.7. פונקצית העזר DeleteK לקחה זמן חישוב רב ולכן הרצת האלגוריתם עם הסתברות של 100% ל-LS ($MA^{x\%}$) גרמה לאיטיות רבה של התהליך האבולוציוני.

לפיכך הוצגו התוצאות של הרצת האלגוריתם עם פרמטר של 1% ו-0.0%. על מנת לשפר את זמן הריצה ביצעתי שינוי נוסף באלגוריתם (אלגוריתם 7 - Heuristic LS). בשורה 4 בודקים אם פעולת המחיקה

שיפרה את הפתרון. אם כן, בשורה 5 לוקחים את הפתרון כמחרוזת-על משותפת ומאפסים את k על מנת להתחיל תהליך ניסיון מחיקה מתחילת המחרוזות.

4: if $fitness(r) < fitness(s)$ then

5: let $s \leftarrow r$, $k \leftarrow 0$

עפ"י ההרצות שביצעתי, לא ראיתי מקרים בהם נמצא שיפור נוסף בקטע ההרצה הראשוני (עד נקודת השיפור הקודמת). לכן ביטלתי את איפוס הפרמטר k .
כלומר שורה 5 תראה במימוש שלי כך:

5: let $s \leftarrow r$

שיפור זה גרם לתהליך החיפוש המקומי להיות קצר יותר, ואפשר לתהליך האבולוציוני להתקדם לכיוון פתרון משופר.

את המימוש של התהליך האבולוציוני מימשתי בעצמי (ללא הכלים והמנגנון של OPEN BEAGLE).
האלגוריתם הורץ עם הפרמטרים הבאים:

1. אוכלוסיה – 100
2. הסתברות ל- CROSS – 0.9
3. הסתברות לביצוע מוטציה – 0.2
4. הסתברות לביצוע מוטציה באיבר – 0.2
5. מספר איטרציות – 100
6. הסתברות ל- LS – 0.01

טבלה 25 – תוצאות הרצת אלגוריתם MA בהשוואה לאלגוריתמים היוריסטים

MA	H2	H1	
43	77	92	בעיה מיוחדת L1 (פתרון מיטבי 43).
79	91	93	בעיה מיוחדת L2 (פתרון מיטבי 79).
82	77	79	בעיה מיוחדת L3 (פתרון מיטבי 60).
180	175	188	8 מחרוזות אקראיות באורך 80 מעל אלפבית באורך 4
349	357	411	8 מחרוזות אקראיות באורך 80 מעל אלפבית באורך 16
199	194	213	16 מחרוזות אקראיות באורך 80 מעל אלפבית באורך 4
480	453	619	16 מחרוזות אקראיות באורך 80 מעל אלפבית באורך 16
111	115	115	4 מחרוזות אקראיות באורך 80 ו- 4 מחרוזות באורך 40 מעל אלפבית באורך 2
148	151	163	4 מחרוזות אקראיות באורך 80 ו- 4 מחרוזות באורך 40 מעל אלפבית באורך 4
264	306	306	4 מחרוזות אקראיות באורך 80 ו- 4 מחרוזות באורך 40

			מעל אלפבית באורך 16
100	104	151	8 מחרוזות דומות באורך 90, 4 תווים, שנוצרו ממחרוזת באורך 100
143	153	160	8 מחרוזות דומות באורך 100, 4 תווים, שנוצרו ממחרוזת באורך 120
160	222	221	10 מחרוזות שנוצרות מ-DNA SARS באורך 158 ע"י מחיקת 20% מהתווים
158	159	159	10 מחרוזות שנוצרות מ-DNA SARS באורך 158 ע"י מחיקת 10% מהתווים

אלגוריתם MA מוצא את הפתרון האופטימאלי עבור בעיות מיוחדות. האלגוריתם משפר בצורה ברורה את הביצועים עבור בעיות דומות על כל סוגי הבעיות הדומות. עבור בעיות אקראיות, אין שיפור מובהק בביצועי האלגוריתם.

6.4 אלגוריתם מבוסס התנהגות נמלים בקן

מימשי אלגוריתם המבוסס אופטימיזציה קן הנמלים שהוצג בסעיף 4.6 (ACO).

במימוש אלגוריתם ACO מצאתי חשיבות רבה להגדרת כמות הפרומון המפוזרת ע"י כלל הנמלים. הגדרה לא נכונה של כמויות אלו לא אפשרה לאלגוריתם לנצל את האינפורמציה על מנת להתקדם במציאת פתרון תוך שיתוף מידע בין הנמלים. כמות קטנה מידי גרמה לכך שההשפעה הכוללת של הפרומון הייתה זניחה והאלגוריתם התנהג כאלגוריתם היוריסטי רגיל. ביצועים משופרים הושגו כאשר שולבו מספר מושבות של נמלים, משני כיווני החיפוש (משני צידי המחרוזות) והוחלפה אינפורמציה לגבי הפתרון הטוב ביותר שהשיגה כל מושבה. הפתרון שהופץ גרם לשינוי כמויות הפרומון שפוזרו במסלול לכיוון הפתרון המשופר וכך קצב ההגעה לפתרונות משופרים וקרובים לפתרון האופטימאלי היה גבוה. האלגוריתם הורץ עם הפרמטרים הבאים:

1. מקדם $\alpha, \beta - 1$. שימוש בפרמטר המקורי (9) לא שיפר את ההתקדמות (העלאה בחזקה של ערך קטן מ-1 מקטינה את הערך מאד. לא הייתה התייחסות במאמר לנושא זה)
2. הסתברות לבחירה בטוב ביותר $q_0 - 0.9$
3. מושבות ונמלים – 4 מושבות, בכל אחת 16 נמלים. 2 מושבות לכל כיוון. הפצת הטוב ביותר כל 8 מחזורים.
4. פרמטר ההגבר של כמות הפרומון החדשה $1000 - \gamma$
5. מספר איטרציות – 150

6. פונקצית חיזוק פרומון לפי מיקום נלקחה זהה למוצג במאמר:

פרמטר	ערך
$f(1)$	8
$f(2), f(3), f(4)$	3
$f(5), f(6)$	2
$f(7), f(8)$	1
$f(9), \dots, f(16)$	0

לצורך בחינת האלגוריתם עם פרמטרים שונים בוצעו הרצות עם וללא הסתכלות קדימה ועם התחשבות באורך המחרוזת או ללא התחשבות.

טבלה 26 – תוצאות הרצת אלגוריתמים ACO עם וללא הסתכלות קדימה/התחשבות באורך

ACO^4	ACO^3	ACO^2	ACO^1	H2	H1	
102	114	112	150	148	169	8 מחרוזות דומות באורך 80, 4 תווים, שנוצרו ממחרוזות באורך 100
173	180	179	177	171	183	8 מחרוזות אקראיות באורך 80 מעל אלפבית באורך 4

¹ ללא הסתכלות קדימה ללא התחשבות באורך המחרוזות

² ללא הסתכלות קדימה עם התחשבות באורך המחרוזות

³ עם הסתכלות קדימה ללא התחשבות באורך המחרוזות

⁴ עם הסתכלות קדימה עם התחשבות באורך המחרוזות

תוצאות הרצת אלגוריתמים היוריסטים בהשוואה ל-ACO לאחר ההחלטה להמשיך עם הסתכלות קדימה והתחשבות באורך המחרוזות.

טבלה 27 – תוצאות הרצת אלגוריתם ACO בהשוואה לאלגוריתמים היוריסטים

ACO ⁴	H3	H2	H1	
43	43	76	92	בעיה מיוחדת L1 (פתרון מיטבי 43).
79	79	91	91	בעיה מיוחדת L2 (פתרון מיטבי 79).
61	79	80	80	בעיה מיוחדת L3 (פתרון מיטבי 60).
170	179	171	178	8 מחרוזות אקראיות באורך 80 מעל אלפבית באורך 4
248	334	264	295	4 מחרוזות אקראיות באורך 80 ו- 4 מחרוזות אקראיות בגודל 40 מעל אלפבית באורך 16
159	161	166	170	5 מחרוזות אקראיות באורך 80 ו- 5 מחרוזות אקראיות בגודל 40 מעל אלפבית באורך 4
173	180	171	183	8 מחרוזות אקראיות באורך 80 מעל אלפבית באורך 4
100	105	125	154	8 מחרוזות דומות באורך 80, 4 תווים, שנוצרו ממחרוזות באורך 100
100	157	165	199	8 מחרוזות דומות באורך 80, 16 תווים, שנוצרו ממחרוזות באורך 100
161	195	221	220	10 מחרוזות שנוצרות מ-DNA SARS באורך 158 ע"י מחיקת 20% מהתווים
158	158	159	162	10 מחרוזות שנוצרות מ-DNA SARS באורך 158 ע"י מחיקת 10% מהתווים

⁴ עם הסתכלות קדימה עם התחשבות באורך המחרוזות

אלגוריתם ACO מוצא את הפתרון האופטימאלי עבור בעיות מיוחדות. האלגוריתם משפר בצורה ברורה את הביצועים עבור בעיות דומות על כל סוגיים. וגם עבור בעיות אקראיות.

6.5 אלגוריתם מבוסס התנהגות דבורים בכוורת - ABC

שיטת העבודה למציאת הפרמטרים האופטימאליים לפתרון הבעיה היה שונה משאר האלגוריתמים שהוצגתי עד כה. בשיטה זו, בדומה לשיטת ה-POEMS, פתרונות הביניים אינם מהווים פתרון חוקי לבעיה והאלגוריתם מתחלק בפועל לשני שלבים: עד מציאת פתרון חוקי לבעיה, ולאחר מציאת פתרון חוקי לבעיה. נדרש לקבוע פרמטרים כך שכבר בסוף השלב הראשון מגיעים לפתרון קצר ככל שניתן.

טבלה 28 – פרמטרים והגדרות לאלגוריתם ABC

600	<i>Generation</i>	דורות
100	<i>Popsize</i>	גודל אוכלוסיית דבורים מועסקות
400	<i>LookerPopsize</i>	גודל אוכלוסיית דבורים צופות
14	<i>Scoutfailcount</i>	מספר מחזורים ממוצע לפני שינוי
25	<i>Radiusstepsize</i>	גודל צעד הגדלת מרחק חיפוש
20	<i>Initsize</i>	גודל מחרוזת התחלתי
50	<i>Windowsize</i>	גודל טבעת חיפוש
2.5	<i>Fitnessshortbonus</i>	בונוס ניקוד לפונקצית כשירות עבור כל תו מתחת לגודל מחרוזת מקסימאלי
0.9	q_0	הסתברות לבחירת איבר דבורה הטובה ביותר
2	<i>deleteFactor</i>	פרמטר הגדלת ההסתברות למחיקת תו
0.8	p	הסתברות לבחירת תיקון מקומי למול ביצוע LocalSearch
0 (ללא הגדלת כמות השינויים בפעולה דבורה אחת)	<i>ChangeCount</i>	פרמטר להגדלת כמות השינויים המקסימאלי בחיפוש המקומי של הדבורה

טבלה 29 – תוצאות הרצת אלגוריתמים ABC בהשוואה לאלגוריתמים אחרים

ABC	MA	ACO	H3	H2	H1	
43	43	43	43	77	92	בעיה מיוחדת L1 (פתרון מיטבי 43).
79	79	79	79	91	93	בעיה מיוחדת L2 (פתרון מיטבי 79).
60	82	61	79	77	79	בעיה מיוחדת L3 (פתרון מיטבי 60).
220	208	195	273	206	231	4 מחרוזות אקראיות באורך 80 ו- 4 מחרוזות אקראיות בגודל 40 מעל אלפבית באורך 8
159	169	159	161	166	170	4 מחרוזות אקראיות באורך 80 ו- 4 מחרוזות אקראיות בגודל 40 מעל אלפבית באורך 4
249	262	248	334	245	288	8 מחרוזות אקראיות באורך 80 מעל אלפבית באורך 8
158	195	159	274	215	285	10 מחרוזות שנוצרות מ-DNA SARS באורך 158 ע"י מחיקת 20% מהתווים
160	178	160	270	224	272	10 מחרוזות שנוצרות מ-DNA SARS באורך 158 ע"י מחיקת 20% מהתווים
158	158	158	235	159	161	10 מחרוזות שנוצרות מ-DNA SARS באורך 158 ע"י מחיקת 10% מהתווים

אלגוריתם ABC מוצא את הפתרון האופטימאלי עבור בעיות מיוחדות. האלגוריתם משפר בצורה ברורה את הביצועים עבור בעיות עם מחרוזות דומות על כל סוגיהן. אלגוריתם ABC מספק תוצאות סבירות עבור בעיות אקראיות ודומה בתוצאות לאלגוריתמים אחרים.

7 סיכום

אלגוריתמים היוריסטיים לא נותנים תוצאות מספקות לפתרון בעיית ה-SCS עבור כלל הבעיות. הפעלת אלגוריתמים אבולוציוניים משפרת בצורה משמעותית את היכולת להגיע לתוצאות משופרות ולעיתים אופטימליות לבעיה. ניתן לבסס אלגוריתם מבוסס התנהגות כוורת דבורים לפתרון בעיית ה-SCS אשר יספק תוצאות טובות ותחרותיות אל מול אלגוריתמים אבולוציוניים אחרים קיימים. בפתרון בעיות עם מחרוזות דומות, SARS158 כאשר 10% שונות (GAP), האלגוריתם פתר בצורה טובה. בעיית SARS158 כאשר 20% שונות הייתה הבעיה הקשה ביותר לפתרון. ברוב המקרים האלגוריתמים ההיוריסטיים לא הצליחו להגיע לפתרון קרוב לפתרון האופטימאלי המוכר. האלגוריתמים אבולוציוניים שמימשי (ABC, MA, GE/H1, ACO) הגיעו קרוב לפתרון זה, כאשר תוצאות טובות באחוזים גבוהים לאלגוריתם המבוסס על כוורת דבורים. ככלל, זמן הריצה של אלגוריתמים אבולוציוניים גדול מזמן הריצה של האלגוריתמים ההיוריסטיים. מומלץ להריץ את האלגוריתם ההיוריסטי אשר זמן הריצה שלו יחסית קצר וביצועיו סבירים עבור בעיות אקראיות, ואח"כ להריץ את האלגוריתמים האבולוציוניים לצורך מציאת פתרון משופר במידה והבעיה היא מיוחדת או מבוססת על מחרוזות דומות.

8 כיווני מחקר עתידיים

לצורך בחינת האלגוריתמים השונים יש להמשיך ולבחון ביצועים על בעיות גדולות בסדר גודל (למשל SARS1269). ניתן להמשיך ולבחון שינויים הפרמטרים השונים המשפיעים על התנהגות האלגוריתם – כמו למשל הגדלת כמות הדבורים המועסקות בשלבים השונים. לדעתי ניתן לשפר את שלב החיפוש המקומי של הדבורים. יכול להיות ששילוב בין אלגוריתם POEMS לחיפוש המקומי של הדבורים ייתן יכולת חיפוש משופרת. כלומר במקום תהליך חיפוש אקראי לדבורה, נבצע חיפוש ע"י הפעלת איטרציה גנטית על קבוצת שינויים. מעניין יהיה לבחון עד כמה התהליך הגנטי על סט פעולות הינו טוב יותר מבחירת פעולות שינוי בצורה אקראית. יש לשקול הוספת אלמנטים של זיכרון והעברת מידע בין הדבורים – כמו פיזור פרומון. לדוגמא, ניתן לתת לחלון חיפוש מוצלח כמות פרומון גדולה (עפ"י גודל הכיסוי שלו) על מנת שהרוב הדבורים יתחילו את החיפוש מאותו חלון מוצלח.

1. Andreas , W. 2003. *The Shortest Common Supersequence Problem*.
DOI= <http://www.update.uu.se/~shikaree/Westling/>
2. Kubalik, J., Faigl, J., Iterative Prototype Optimization with Evolved Improvement Steps. In *Genetic Programming, Proceedings of 9th European Conference, EuroGP 2006*. Heidelberg: Springer, 2006, p. 154-165. ISBN 3-540-33143-3 .
3. Kubalik, J.: Efficient stochastic local search algorithm for solving the shortest common supersequence problem. In *Proceedings of the 12th Genetic and Evolutionary Computation Conference*, New York: ACM, 2010, ISBN 978-1-4503-0073-5, pp. 249–256, 2010.
4. Evolutionary-Based Iterative Local Search Algorithm for the Shortest Common Supersequence Problem
GECCO'11, July 12–16, 2011, Dublin, Ireland. Copyright 2011 ACM 978-1-4503-0557-0/11/07
5. Carlos Cotta: Memetic Algorithms with Partial Lamarckism for the Shortest Common Supersequence Problem
6. Pietrzak K.: On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and System Sciences* 67 (2003) 757-771
7. Branke, J., Middendorf, M., Schneider, F.: Improved heuristics and a genetic algorithm for finding short supersequences. *OR-Spektrum* 20 (1998) 39-45
8. Dervis, K. 2010. Artificial bee colony algorithm. *Scholarpedia*. 5(3):6915. DOI=http://www.scholarpedia.org/article/Artificial_bee_colony_algorithm
9. Mustafa M. Noaman, Ameera S. Jaradat:
Solving Shortest Common Supersequence Problem Using Artificial Bee Colony Algorithm
Copyright ©2011 IJJ: The Research Bulletin of Jordan ACM - ISWSA; ISSN: 2078-7952 (print); 2078-7960 (online)
10. Rene Michel, Martin Middendorf:
An ACO Algorithm for the Shortest Common Supersequence Problem
New ideas in optimization, McGraw-Hill Ltd., UK Maidenhead, UK, England
©1999 ISBN:0-07-709506-5
11. Needleman, Saul B.; and Wunsch, Christian D.:
A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* **48** (3): 443–53. (1970)
12. Yang, Xin-She: *Nature-Inspired Metaheuristic Algorithms*, 2nd Edition. *Luniver press*. (2010)
13. Christian Gagne and Marc Parizeau: Open BEAGLE: A New Versatile C++ Framework for Evolutionary Computations. *Late Breaking Papers*, (GECCO-2002)
14. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* **48** (3): 443–53. (1970)

10 Abstract

The Shortest Common Supersequence (SCS) is a hard combinatorial optimization problem with numerous practical applications like biological sequence analysis. In this paper I'll review several approaches for solving the SCS problem: Improved heuristics, Genetic Algorithms, evolutionary algorithms and other bio-inspired optimization techniques.

Several algorithms were implemented and the actual results are presented. The paper describes a new algorithm for solving the SCS problem based on the Artificial Bee Colony. The algorithm is based on the bio-inspired swarm-based optimization that was motivated by the intelligent behavior of the honey bees. The results of this algorithm were compared to the results of the other proposed algorithms and it was demonstrated that the results are promising.

11 Contents

Table of Content	2
List of Figures	3
List of Tables	3
List of Algorithms	4
1 Abstract	5
2 Introtuction.....	6
3 Finding the Shortest Common Supersequence	8
4 Algorithms for finding SCS	10
4.1 Early Work.....	10
4.2 Basic Heuristics algorithm.....	10
4.3 Meta-Heuristics Algorithms.....	13
4.4 Genetic Algorithms.....	14
4.5 Genetic-Heuristics Algorithm.....	16
4.6 Ant Colony Optimization – ACO	21
4.7 Memetic with Local Improvement Algorithm	288
4.8 Branch and Bound and MA combined Algorithm	32
4.9 POEMS – Prototype Based.....	388
4.10 Artificial Bee Colony Optimization.....	51
5 Finding the SCS using ABC	533
5.1 Algorithms Description.....	544
6 Implementation and Results.....	666
6.1 Heuristics Algorithms	666
6.2 Genetic Algorithms	699
6.3 Memetic with Local Improvement Algorithm	70
6.4 Ant Colony Optimization	72
6.5 Artificial Bee Colony	74
7 Summary	77
8 Future Work	77
9 References	78
10 Abstract	79
11 Contents	80

The Open University of Israel
Department of Mathematics and Computer Science

Bio-Inspired Algorithms for the SCS problem

Final Paper submitted as fulfillment of the requirements
towards an M.Sc. degree in Computer Science
The Open University of Israel
Computer Science Division

By
Raanan Manor
(ID: 023679202)
Email: raananm@hotmail.com
Address: Tel Aviv, 14 Kehilat Saloniki app#35, Tel.: 057-8147024

Prepared under the supervision of Dr. Miray Avigal

September 2012