VIŠESTRUKO PORAVNANJE SEKVENCI

GENETSKI ALGORITAM

PMF MATEMATIKA

FRAN MIŠIĆ, ANDREJ SLAPNIČAR

2021.

OPĆENITO O PROBLEMU

- Proces poravnanja više bioloških sekvenci (proteini, DNA/RNA i sl.)
- Koristi se kako bi se usporedila sličnost sekvenci te napravila filogenetička analiza
- Optimalni algoritmi za pronalazak višestrukog poravnanja više od 3 sekvenci zahtijevaju izrazito puno resursa pa su skoro neupotrebljivi (npr. dinamičko programiranje)
- Za pronalazak dovoljno dobrih rješenja koriste se heurističke metode (npr. genetski algoritam, simulirano kaljenje)

MATEMATIČKI OPIS PROBLEMA

- Ulaz: niz od m sekvenci $S_i,\ i = 1, 2, \dots, m$

$$S := \begin{cases} S_1 = (S_{11}, S_{12}, \dots, S_{1n_1}) \\ S_2 = (S_{21}, S_{22}, \cdots, S_{2n_2}) \\ \quad \vdots \\ S_m = (S_{m1}, S_{m2}, \dots, S_{mn_m}) \end{cases}$$

- Višestruko poravnanje umetanjem praznina pretvara gornji niz S_i u drugi niz S_i', tako da su sve sekvence drugog niza jednake duljine $L \geq \max\{n_i,\ i = 1, 2, \dots, m\}$

- Izlaz: niz od m sekvenci $S_i',\ i = 1, 2, \dots, m$

$$S' := \begin{cases} S_1' = (S_{11}', S_{12}', \dots, S_{1L}') \\ S_2' = (S_{21}', S_{22}', \dots, S_{2L}') \\ \quad \vdots \\ S_m' = (S_{m1}', S_{m2}', \dots, S_{mL}') \end{cases}$$

GENETSKI ALGORITAM

- Heuristički algoritam inspiriran preživljavanjem najjačih
- Dio veće skupine evolucijskih algoritama
- Koristi operatore križanja i mutacije kako bi došao do rješenja
- Rješava razne optimizacijske probleme

PSEUDOKOD GENETSKOG ALGORITMA

1. Inicijalizacija početne populacije

Ponavljaj do terminalnog uvjeta:

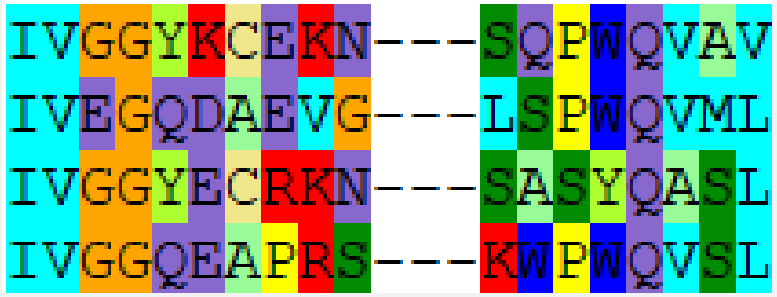
1. Izračunaj funkciju dobrote svih jedinki populacije
2. Napravi selekciju
3. Provedi operatore križanja i mutacije nad populacijom
4. Kreiraj populaciju nove generacije

IMPLEMENTACIJA ALGORITMA

Implementacija jedinki u populaciji

Sekvenca aminokiselina ⟼ binaran niz; 0 predstavlja slovo (aminokiselinu), 1 predstavlja prazninu

Primjer:




```
00000000001110000000
00000000001110000000
00000000001110000000
00000000001110000000
```

Funkcija dobrote (suma parova)

$$score(alignment) = \sum_{i=2}^{m} \sum_{j=1}^{i-1} W_{ij} * score(A_i, A_j)$$

$$score(A_i, A_j) = \sum_{l=1}^{L} p_l$$

$$p_l = \begin{cases} -10 & otvaranje\ praznine \\ -0.05 & nastavak\ praznine \\ + & vrijednost\ iz\ "scoring\ matrix" \end{cases}$$

„Scoring matrix”

- Matrica koja za svaki par od 20 aminokiselina sadrži numeričku vrijednost koja odražava koliko su dvije aminokiseline „slične” u poravnanju

- Postoje mnoge verzije, ali su daleko najkorištenije dvije: PAM250, Blosum62

PAM250 Matrix

	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	2	-2	0	0	-4	1	-1	-1	-1	-2	-1	0	1	0	-2	1	1	0	-6	-3
C	-2	12	-5	-5	-4	-3	-3	-2	-5	-6	-5	-4	-3	-5	-4	0	-2	-2	-8	0
D	0	-5	4	3	-6	1	1	-2	0	-4	-3	2	-1	2	-1	0	0	-2	-7	-4
E	0	-5	3	4	-5	0	1	-2	0	-3	-2	1	-1	2	-1	0	0	-2	-7	-4
F	-4	-4	-6	-5	9	-5	-2	1	-5	2	0	-4	-5	-5	-4	-3	-3	-1	0	7
G	1	-3	1	0	-5	5	-2	-3	-2	-4	-3	0	-1	-1	-3	1	0	-1	-7	-5
H	-1	-3	1	1	-2	-2	6	-2	0	-2	-2	2	0	3	2	-1	-1	-2	-3	0
I	-1	-2	-2	-2	1	-3	-2	5	-2	2	2	-2	-2	-2	-2	-1	0	4	-5	-1
K	-1	-5	0	0	-5	-2	0	-2	5	-3	0	1	-1	1	3	0	0	-2	-3	-4
L	-2	-6	-4	-3	2	-4	-2	2	-3	6	4	-3	-3	-2	-3	-3	-2	2	-2	-1
M	-1	-5	-3	-2	0	-3	-2	2	0	4	6	-2	-2	-1	0	-2	-1	2	-4	-2
N	0	-4	2	1	-4	0	2	-2	1	-3	-2	2	-1	1	0	1	0	-2	-4	-2
P	1	-3	-1	-1	-5	-1	0	-2	-1	-3	-2	-1	6	0	0	1	0	-1	-6	-5
Q	0	-5	2	2	-5	-1	3	-2	1	-2	-1	1	0	4	1	-1	-1	-2	-5	-4
R	-2	-4	-1	-1	-4	-3	2	-2	3	-3	0	0	0	1	6	0	-1	-2	2	-4
S	1	0	0	0	-3	1	-1	-1	0	-3	-2	1	1	-1	0	2	1	-1	-2	-3
T	1	-2	0	0	-3	0	-1	0	0	-2	-1	0	0	-1	-1	1	3	0	-5	-3
V	0	-2	-2	-2	-1	-1	-2	4	-2	2	2	-2	-1	-2	-2	-1	0	4	-6	-2
W	-6	-8	-7	-7	0	-7	-3	-5	-3	-2	-4	-4	-6	-5	2	-2	-5	-6	17	0
Y	-3	0	-4	-4	7	-5	0	-1	-4	-1	-2	-2	-5	-4	-4	-3	-3	-2	0	10

Blosum62 Matrix

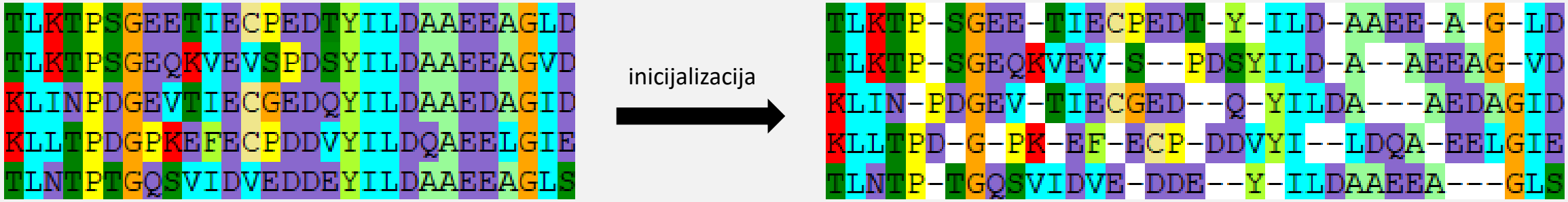
	A	C	D	E	F	G	H	I	K	L	M	N	P	Q	R	S	T	V	W	Y
A	4	0	-2	-1	-2	0	-2	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-2	-3	-2
C	0	9	-3	-4	-2	-3	-3	-1	-3	-1	-1	-3	-3	-3	-3	-1	-1	-1	-2	-2
D	-2	-3	6	2	-3	-1	-1	-3	-1	-4	-3	1	-1	0	-2	0	1	-3	-4	-3
E	-1	-4	2	5	-3	-2	0	-3	1	-3	-2	0	-1	2	0	0	0	-3	-3	-2
F	-2	-2	-3	-3	6	-3	-1	0	-3	0	0	-3	-4	-3	-3	-2	-2	-1	1	3
G	0	-3	-1	-2	-3	6	-2	-4	-2	-4	-3	-2	-2	-2	-2	0	1	0	-2	-3
H	-2	-3	1	0	-1	-2	8	-3	-1	-3	-2	1	-2	0	0	-1	0	-2	-2	2
I	-1	-1	-3	-3	0	-4	-3	4	-3	2	1	-3	-3	-3	-3	-2	-2	1	-3	-1
K	-1	-3	-1	1	-3	-2	-1	-3	5	-2	-1	0	-1	1	2	0	0	-3	-3	-2
L	-1	-1	-4	-3	0	-4	-3	2	-2	4	2	-3	-3	-2	-2	-2	-2	3	-2	-1
M	-1	-1	-3	-2	0	-3	-2	1	-1	2	5	-2	-2	0	-1	-1	-1	-2	-1	-1
N	-2	-3	1	0	-3	0	-1	-3	0	-3	-2	6	-2	0	0	1	0	-3	-4	-2
P	-1	-3	-1	-1	-4	-2	-2	-3	-1	-3	-2	-1	7	-1	-2	-1	1	-2	-4	-3
Q	-1	-3	0	2	-3	-2	0	-3	1	-2	0	0	-1	5	1	0	0	-2	-2	-1
R	-1	-3	-2	0	-3	-2	0	-3	2	-2	-1	0	-2	1	5	-1	-1	-3	-3	-2
S	1	-1	0	0	-2	0	-1	-2	0	-2	-1	1	-1	0	-1	4	1	-2	-3	-2
T	-1	-1	1	0	-2	1	0	-2	0	-2	-1	0	1	0	-1	1	4	-2	-3	-2
V	0	-1	-3	-2	-1	-3	-3	3	-2	1	1	-3	-2	-2	-3	-2	-2	4	-3	-1
W	-3	-2	-4	-3	1	-2	-2	-3	-3	-2	-1	-4	-4	-2	-3	-3	-3	-3	11	2
Y	-2	-2	-3	-2	3	-3	2	-1	-2	-1	-1	-2	-3	-1	-2	-2	-2	-1	2	7

1. INICIJALIZACIJA – nasumična

Parametar *gap coefficient* koristi se za određivanje maksimalne duljine sekvenci

$$L = gap\ coefficient\ * max\{n_i, i = 1, 2, \dots, m\}$$

Za svaku sekvencu u ulazu, na nasumična mjesta ubacuje praznine dok na dođe do granice L

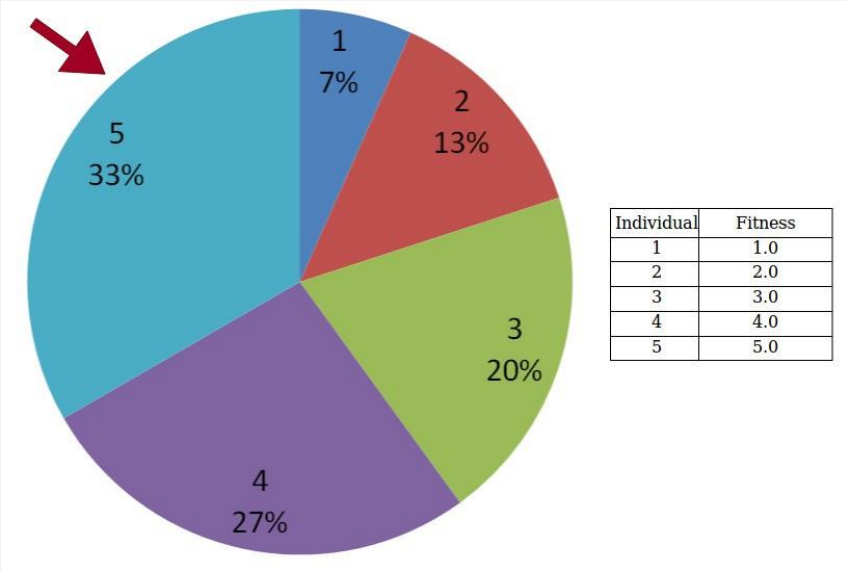


2. SELEKCIJA

Koristimo funkciju dobrote kako bi odredili koje jedinke sudjeluju u križanjima

Ideja: jedinke s boljom vrijednosti funkcije dobrote (fitness) imaju veću vjerojatnost sudjelovanja u križanju

Princip „kotača” vjerojatnosti



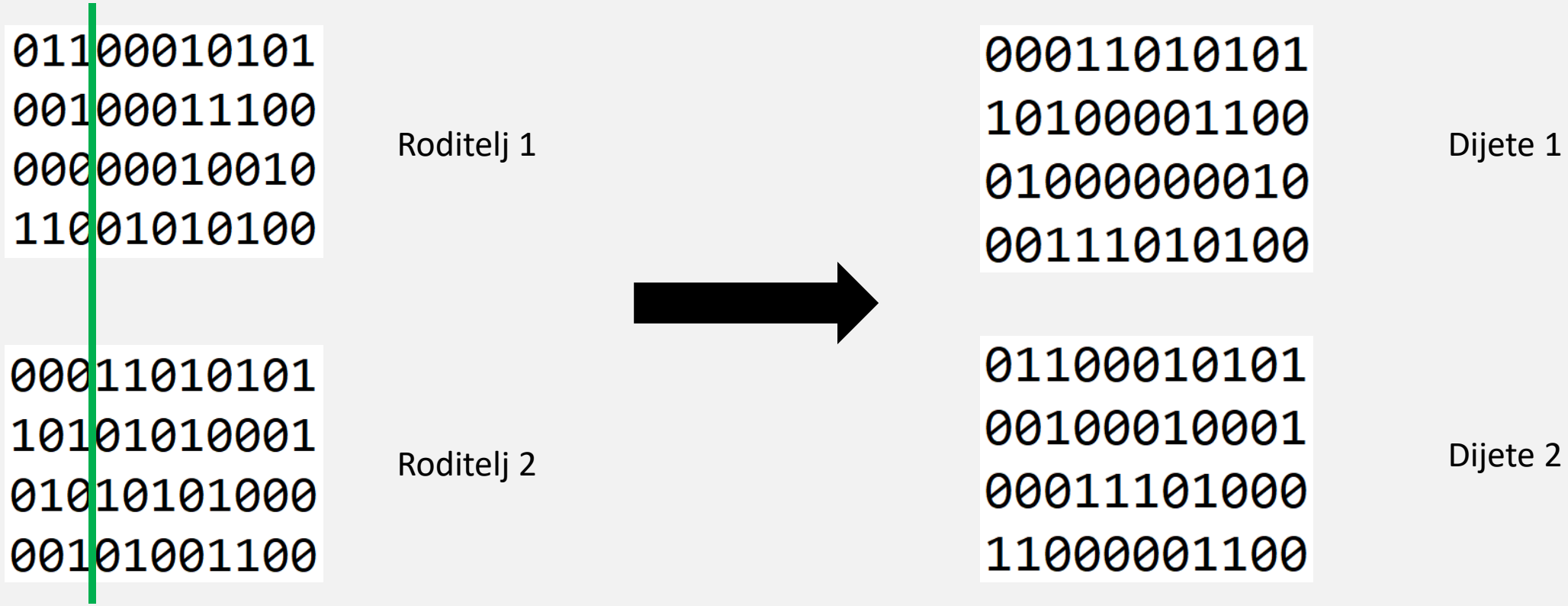
Individual	Fitness
1	1.0
2	2.0
3	3.0
4	4.0
5	5.0

```
int selection(int select_index, alignment_list* alignments) {

    int i, j, k, num, sum;
    int sum_fitness = 0;
    for (i = 0; i < alignments->size; i++)
    {
        sum_fitness += alignments->list[i].normalized_score;
    }
    for (k = 0; k < 5; k++)
    {
        num = (rand() % (sum_fitness + 1));
        sum = 0;
        for (i = 0; i < alignments->size; i++)
        {
            if (alignments->list[i].normalized_score > 0)
                sum += alignments->list[i].normalized_score;

            if (num <= sum)
            {
                if (i != select_index)
                    return i;
                else
                {
                    j = i;
                    break;
                }
            }
        }
    }
    return (j + 1) % alignments->size;
}
```

3. OPERATORI KRIŽANJA

Operator #1

- Klasičan „one point crossover”, križanje s jednom točkom presjeka



Operator #2

- Verzija „one point crossover-a” s horizontalnom točkom presjeka

01100010101
00100011100
00000010010
11001010100

Roditelj 1

00011010101
10101010001
01010101000
00101001100

Roditelj 2

01100010101
00100011100
00000010010
00101001100

Dijete 1

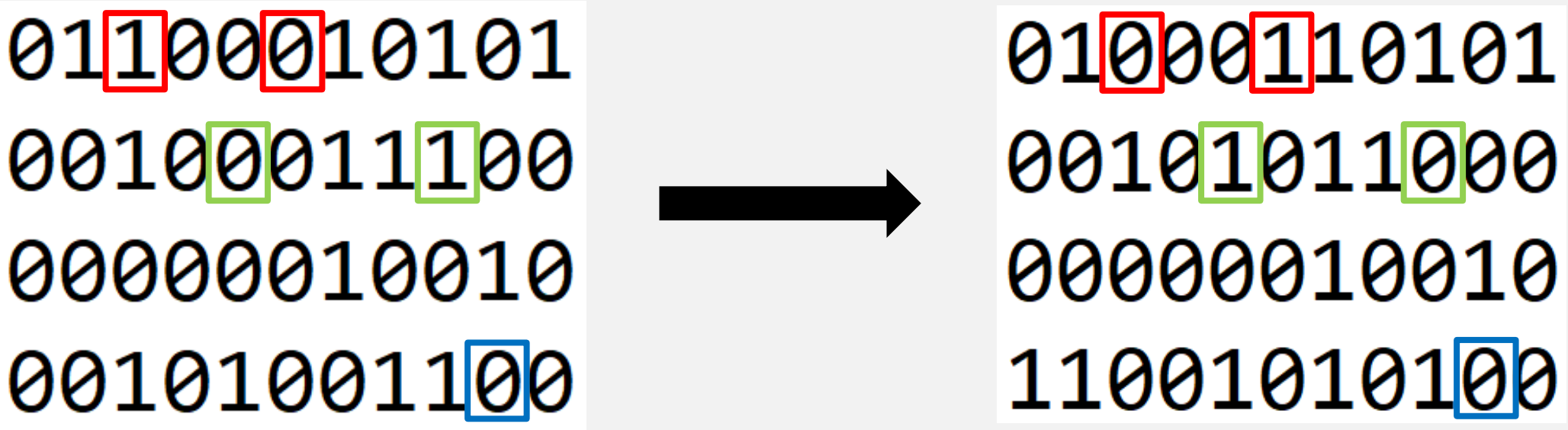
00011010101
10101010001
01010101000
11001010100

Dijete 2

4. OPERATORI MUTACIJE

Operator #1

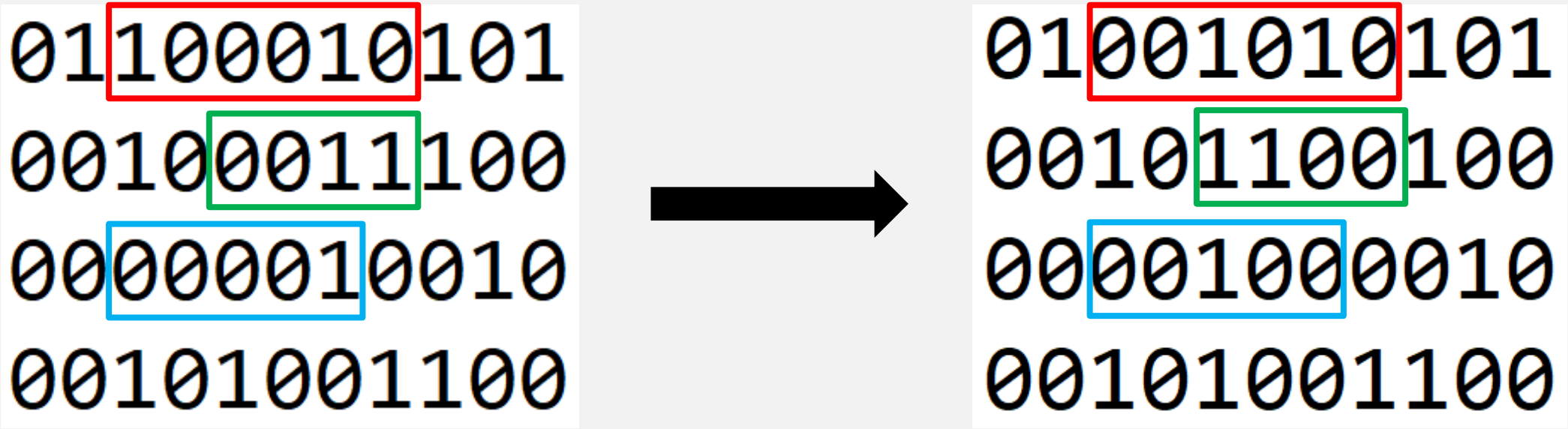
- Zamjena dva bita u svakoj sekvenci poravnanja s vjerojatnosti određenom parametrom: *MUTATION1_PARAMETER*



```c
void mutation1(alignment* A, constants_type* CONSTANTS)
{
    int i, j1, j2, temp;
    double rnum;
    for (i = 0; i < A->row; i++)
    {
        rnum = rand() / RAND_MAX;
        if (rnum < CONSTANTS->MUTATION1_PARAMETER)
        {
            j1 = rand() % A->column;
            j2 = rand() % A->column;
            temp = A->matrix[i][j1];
            A->matrix[i][j1] = A->matrix[i][j2];
            A->matrix[i][j2] = temp;
        }
    }
}
```

Operator #2

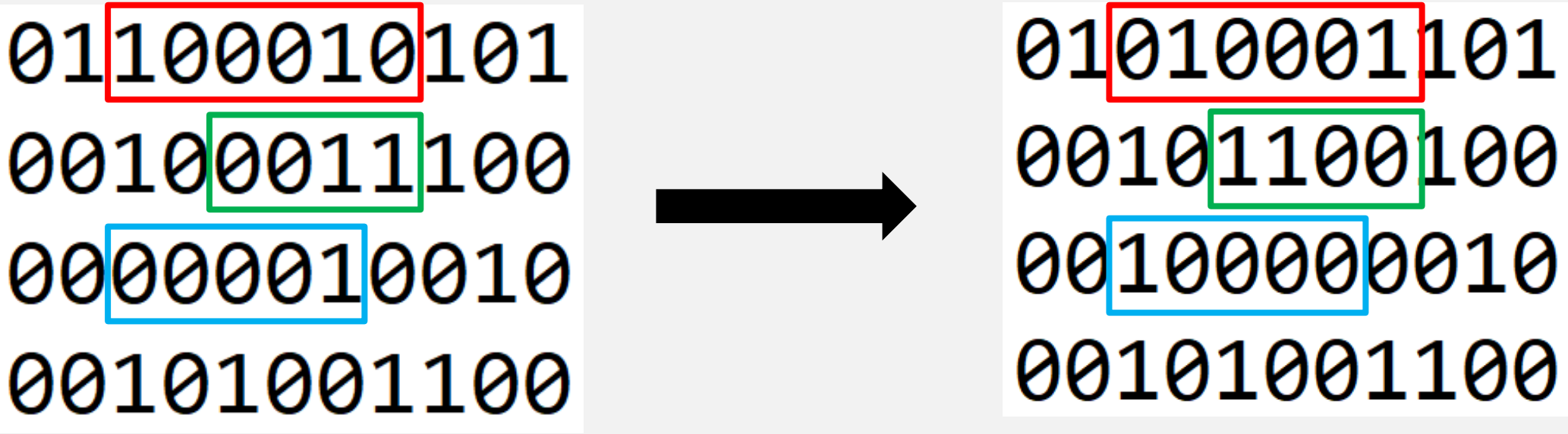
- Ciklički pomak bitova u lijevu stranu za nasumičan broj mjesta između dva stupca s vjerojatnosti određenom parametrom: *MUTATION2_PARAMETER*



```c
void mutation2(alignment* A, constants_type* CONSTANTS)
{
    int i, j, j1, j2, temp;
    double rnum;
    for (i = 0; i < A->row; i++)
    {
        rnum = rand() / RAND_MAX;
        if (rnum < CONSTANTS->MUTATION2_PARAMETER)
        {
            j1 = rand() % A->column;
            j2 = rand() % A->column;
            if (j2 < j1)
            {
                temp = j2;
                j2 = j1;
                j1 = j2;
            }
            temp = A->matrix[i][j1];
            for (j = j1; j < j2; j++)
                A->matrix[i][j] = A->matrix[i][j + 1];
            A->matrix[i][j2] = temp;
        }
    }
}
```

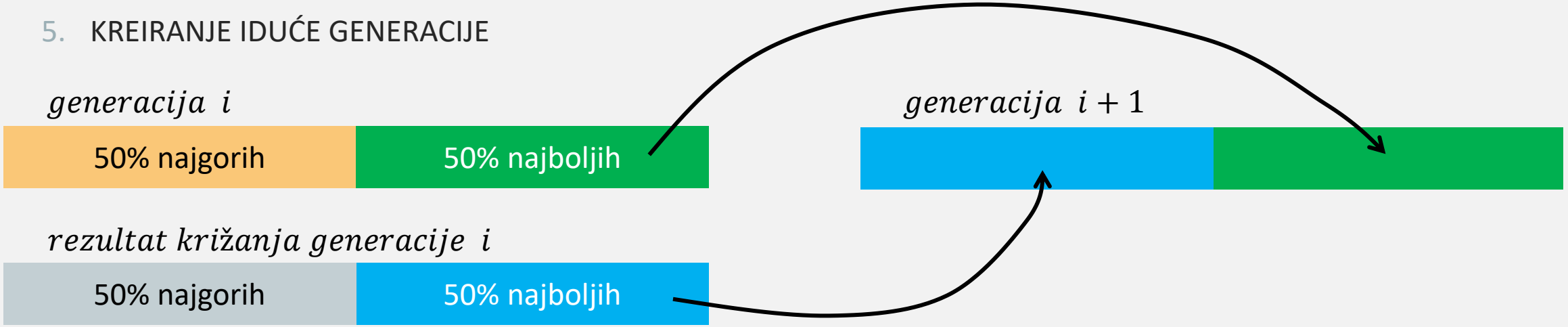
Operator #3

- Invertira poredak bitova između dva stupca s vjerojatnosti određenom parametrom: *MUTATION3_PARAMETER*

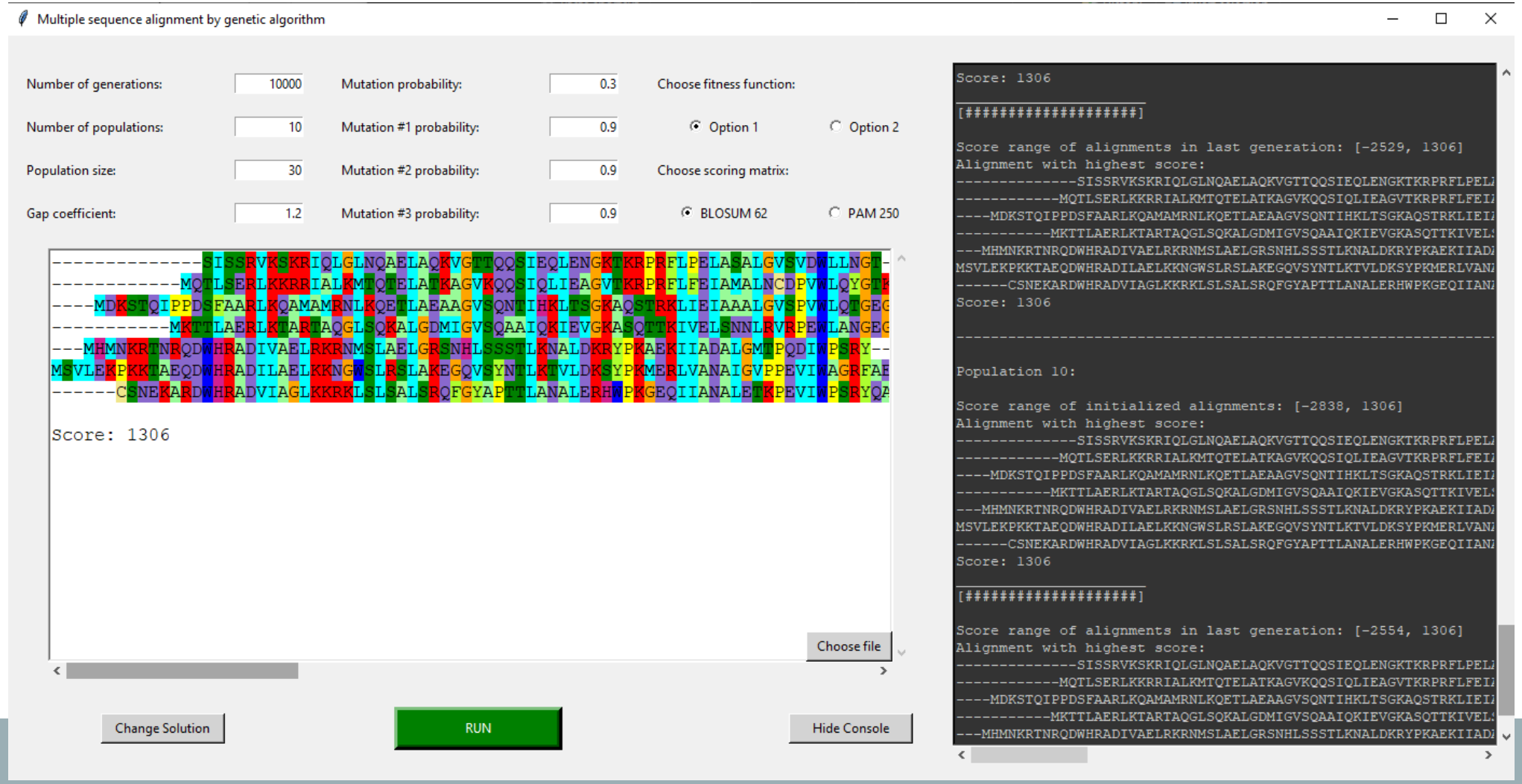


```c
void mutation3(alignment* A, constants_type* CONSTANTS)
{
    int i, j, j1, j2, temp;
    double rnum;
    for (i = 0; i < A->row; i++)
    {
        rnum = rand() / RAND_MAX;
        if (rnum < CONSTANTS->MUTATION3_PARAMETER)
        {
            j1 = rand() % A->column;
            j2 = rand() % A->column;
            if (j2 < j1)
            {
                temp = j2;
                j2 = j1;
                j1 = j2;
            }
            for (j = j1; j < j1 + (j2 - j1 + 1) / 2; j++)
            {
                temp = A->matrix[i][j];
                A->matrix[i][j] = A->matrix[i][j2 - j + j1];
                A->matrix[i][j2 - j + j1] = temp;
            }
        }
    }
}
```

5. KREIRANJE IDUĆE GENERACIJE



- Veličina populacije je fiksna, određena je parametrom *POPULATION_SIZE*
- 50% jedinki populacije s najvećom vrijednosti funkcije dobrote prelaze u iduću generaciju
- Preostalih 50% generacije se kreira tako da se selekcijom odabiru roditelji na koje se primjenjuju operatori križanja, tako da nastane *POPULATION_SIZE* jedinki, od kojih se 50% najboljih kopira u iduću generaciju



GUI – GRAFIČKO SUČELJE

TESTNI PRIMJERI

- Genetski algoritam je implementiran u jeziku C
- Grafičko sučelje je implementirano u jeziku Python

- Korišteno računalo ima sljedeće specifikacije:

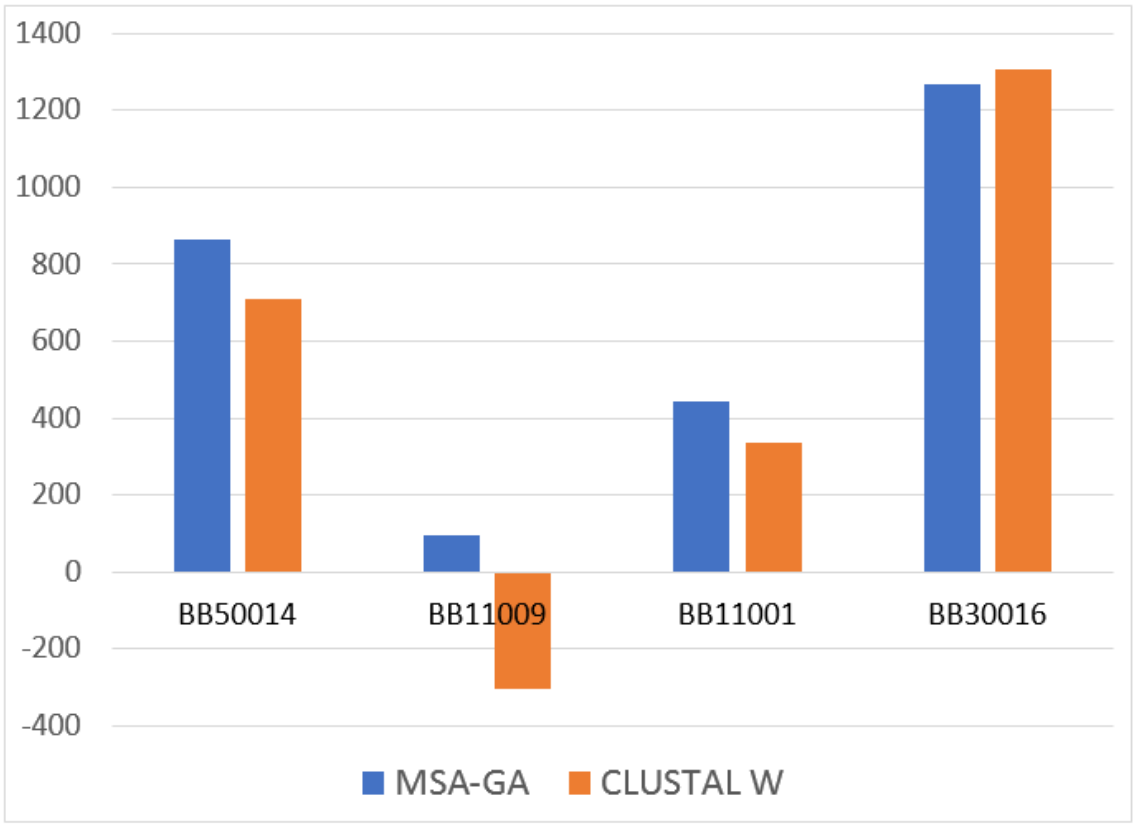
Procesor: Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz

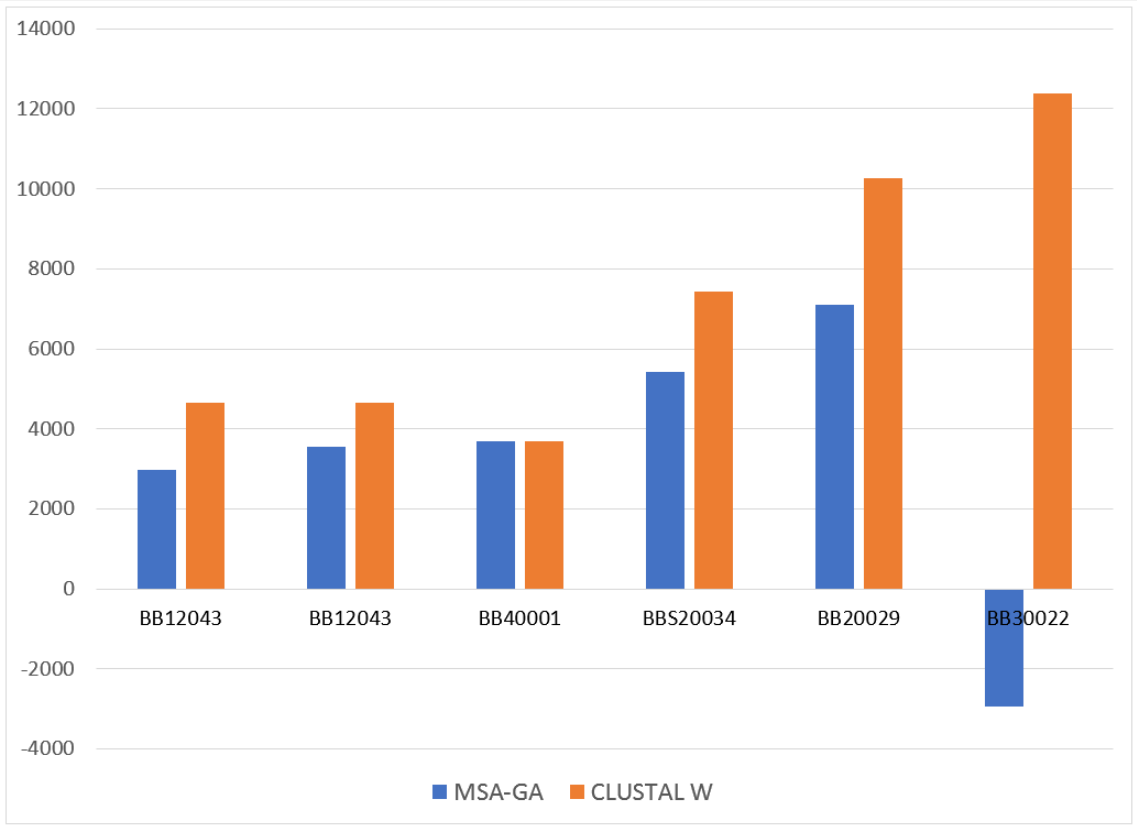
RAM: 16,0 GB

Operacijski sustav: 64-bit Windows

- Rezultati su uspoređeni s rezultatima popularnog softverskog paketa za poravnanja *CLUSTAL W*
- Poravnanje se boduje funkcijom dobrote - *fitness*

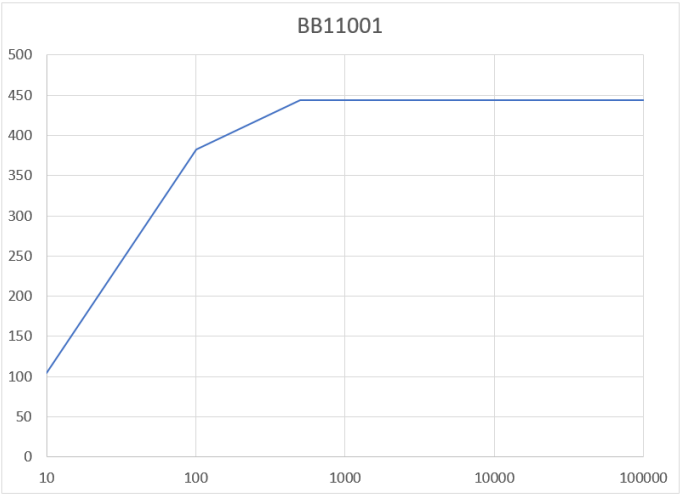
NAME	NUM_SEQ	NUM_GEN	NUM_POP	SIZE_POP	GAP_COEFF	M. PROB.	M. #1 PROB.	M. #2 PROB.	M. #3 PROB.	FITNESS	MATRIX	MSA-GA	CLUSTALW	TIME
BB50014	3	10,000	10	30	1,2	0,3	0,9	0,9	0,9	1	Blosum	865	709	9,4 min
BB11009	4	10,000	10	30	1,0001	0,3	0,9	0,9	0,9	1	Blosum	95	-305	3,5 min
BB11001	4	100,000	10	20	1,05	0,3	0,9	0,9	0,9	1	Blosum	444	335	5,6 min
BB12043	5	10,000	10	20	1,05	0,3	0,3	0,3	0,3	1	Blosum	2961	4660	3,8 min
BB12043	5	100,000	10	20	1,05	0,6	0,6	0,6	0,6	1	Blosum	3544	4660	23,8 min
BB40001	5	10,000	10	30	1,1	0,3	0,9	0,9	0,9	1	Blosum	3682	3677	3 min
BBS20034	6	10,000	10	30	1,1	0,3	0,9	0,9	0,9	1	Blosum	5411	7440	5,3 min
BB30016	7	10,000	10	30	1,2	0,3	0,9	0,9	0,9	1	Blosum	1268	1306	5,6 min
BB20029	9	10,000	10	30	1,2	0,3	0,3	0,3	0,3	1	Blosum	7106	10259	6 min
BB30022	12	10,000	10	30	1,2	0,3	0,9	0,9	0,9	1	Blosum	-2954	12387	6,3 min

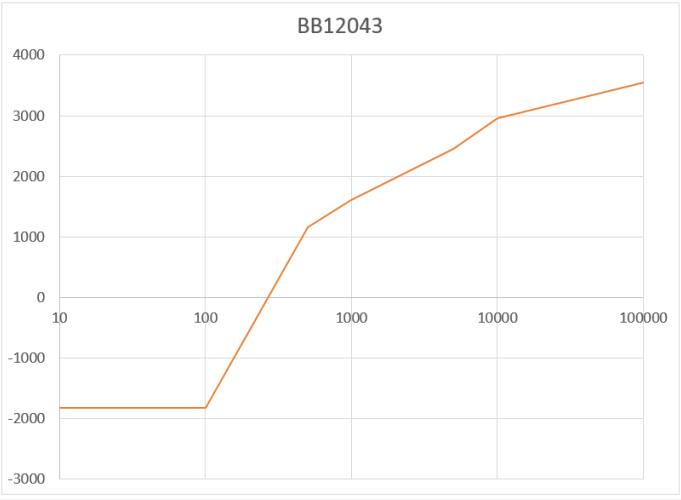

1400
1200
1000
800
600
400
200
0
-200
-400
BB50014
BB11009
BB11001
BB30016
MSA-GA
CLUSTAL W

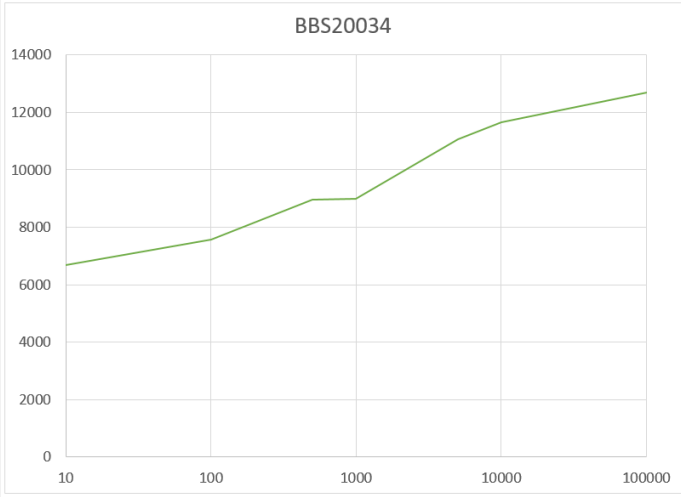

14000
12000
10000
8000
6000
4000
2000
0
-2000
-4000
BB12043
BB12043
BB40001
BBS20034
BB20029
BB30022
MSA-GA
CLUSTAL W

UTJECAJ BROJA GENERACIJA NA REZULTAT

- Proveli smo algoritam na sljedećim primjerima s različitim vrijednostima parametra GENERATION_NUMBER (broj generacija) te usporedili rezultate
- Testni primjeri:







OPTIMIZACIJA POČETNOG RJEŠENJA

- Genetski algoritmi se mogu koristiti i za optimizaciju već postojećeg rješenja
- Pronalaze rješenja koja imaju veću vrijednost funkcije dobrote
- Sljedeća tablica pokazuje primjere na kojima smo rješenje dobiveno od paketa CLUSTAL W dodatno optimizirali po našoj funkciji dobrote

NAME	NUM_SEQ	NUM_GEN	POP_SIZE	CLUSTAL W	MSA-GA
BB50014	3	10,000	30	709	1056
BB11001	4	100,000	50	335	457
BB11009	4	10,000	50	-305	61
BB12043	5	10,000	20	4660	4817
BB40001	5	20,000	30	3677	3789

LITERATURA

[1] Cédric Notredame, Desmond G. Higgins, *SAGA: Sequence Alignment by Genetic Algorithm, Nucleic Acids Research*, Volume 24, Issue 8, 1 April 1996, Pages 1515–1524

[2] Philip Heller, *MSG: A Gap-Oriented Genetic Algorithm for Multiple Sequence Alignment*, San Jose State University, 2009

[3] Maria Chatzou, Cedrik Magis, Jia-Ming Chang, Carsten Kemena, Giovanni Bussotti, Ionas Erb, Cedric Notredame, *Multiple sequence alignment modeling: methods and applications, Briefings in Bioinformatics*, Volume 17, Issue 6, November 2016, Pages 1009–1023

[4] Sean Luke, *Essentials of Metaheuristics,* Department of Computer Science George Mason University, Second Edition, Online Version 2.3, February 2016