SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

**SEMINAR**

# Faza konsenzusa u OLC paradigmi sastavljanja genoma – Sparc

*Nikola Bukovac, Vinko Kolobara*
Voditelj: *Mile Šikić*

Zagreb, siječanj 2018.

# SADRŽAJ


**1. Uvod** **1**

**2. Sparc algoritam** **2**

2.1. Opis algoritma . . . . . 2

2.2. Primjer . . . . . 3

**3. Naša implementacija algoritma** **5**

3.1. Korišteni formati podataka . . . . . 5

3.1.1. FASTA . . . . . 5

3.1.2. FASTQ . . . . . 5

3.1.3. SAM . . . . . 5

3.2. Korištene biblioteke . . . . . 6

3.2.1. GraphMap . . . . . 6

3.3. Struktura implementacije . . . . . 6

3.4. Instalacija i pokretanje algoritma . . . . . 7

**4. Analiza implementacije** **8**

4.1. Alati za analizu . . . . . 8

4.1.1. DnaDiff . . . . . 8

4.1.2. cgmemtime . . . . . 8

4.2. Testna konfiguracija . . . . . 9

4.3. Analiza kvalitete rješenja . . . . . 9

4.4. Analiza utroška memorije i vremena . . . . . 10

**5. Zaključak** **12**

**6. Literatura** **13**

**7. Sažetak** **14**

# 1. Uvod

DNK je važna sastavnica svakog živog bića s obzirom da sadrži svu biološku informaciju svake jedinke, što je jedan od razloga zašto ju znanstvenici pokušavaju što preciznije očitati. Današnji uređaji su dovoljno brzi i jeftini, ali problem predstavljaju kratka očitanja koja je moguće napraviti s takvim uređajima, te se stoga razvijaju i algoritmi koji će dobivena očitanja pokušati spojiti u jedan slijed.

Danas je jedna od najraširenijih metoda sekvenciranja genoma takozvana *shotgun* [6, Poglavlje 1.2.2] metoda sekvenciranja kod koje se DNK cjepka slučajnim načinom na male dijelove na različitim pozicijama i različitim duljinama. Takav način sekvenciranja dovodi do nepreciznosti samih očitanja DNK pa je taj proces potrebno provoditi nekoliko puta nad istim dijelovima za kvalitetnu rekonstrukciju DNK. Uređaji koji se danas pretežno koriste pripadaju drugoj generaciji uređaja za sekvenciranje, koji iako su jako precizni ostvaruju jako male dužine očitanja, veličine do nekoliko stotina parova nukleotida što značajno usporava sam proces očitavanja. Kako bi se doskočilo ovom problemu, razvijena je treća generacija uređaja koja može očitati od 5 tisuća do 120 tisuća parova nukleotida u jednom čitanju, ali veliki problem predstavlja jako velika pogreška u očitanju koja iznosi od 15% do 50%.

Probleme koji nastaju pri očitavanju genoma rješavamo s algoritmima sastavljanja genoma te tako spajamo kraća očitanja i popravljamo nastale greške kod očitanja. Moderni algoritmi koji se bave ovim problemom temelje se na grafovima, a najkorištenije su dvije metode: *Preklapanje-Razmještaj-Konsenzus* metoda temeljena na grafu preklapanja ili metoda temeljena na *k-mer/de Bruijn* grafovima [6].

Cilj ovog rada je upoznavanje, implementacija te analiza implementiranog algoritma faze konsenzusa u OLC paradigmi sastavljanja genoma, naziva Sparc. Postavljeni ciljevi nam određuju i samu strukturu rada pa je tako u drugom poglavlju, opisana ideja algoritma Sparc te prikazan grafički primjer koji prikazuje način na koji algoritam radi. Treće poglavlje opisuje kako smo ostvarili našu implementaciju algoritma te što smo sve koristili za nju. Četvrto poglavlje donosi našu analizu rješenja koje smo implementirali te način na koji smo proveli analizu.

# 2. Sparc algoritam

Algoritam Sparc je algoritam faze konsenzusa u Preklapanje-Razmještanje-Konzenzus (engl. *Overlap-Layout-Consensus, OLC*) paradigmi sastavljanja očitanja genoma. Temelj algoritma se zasniva na *de Bruijn/k-mer* grafu nad kojim se potom provodi ostatak algoritma [5].

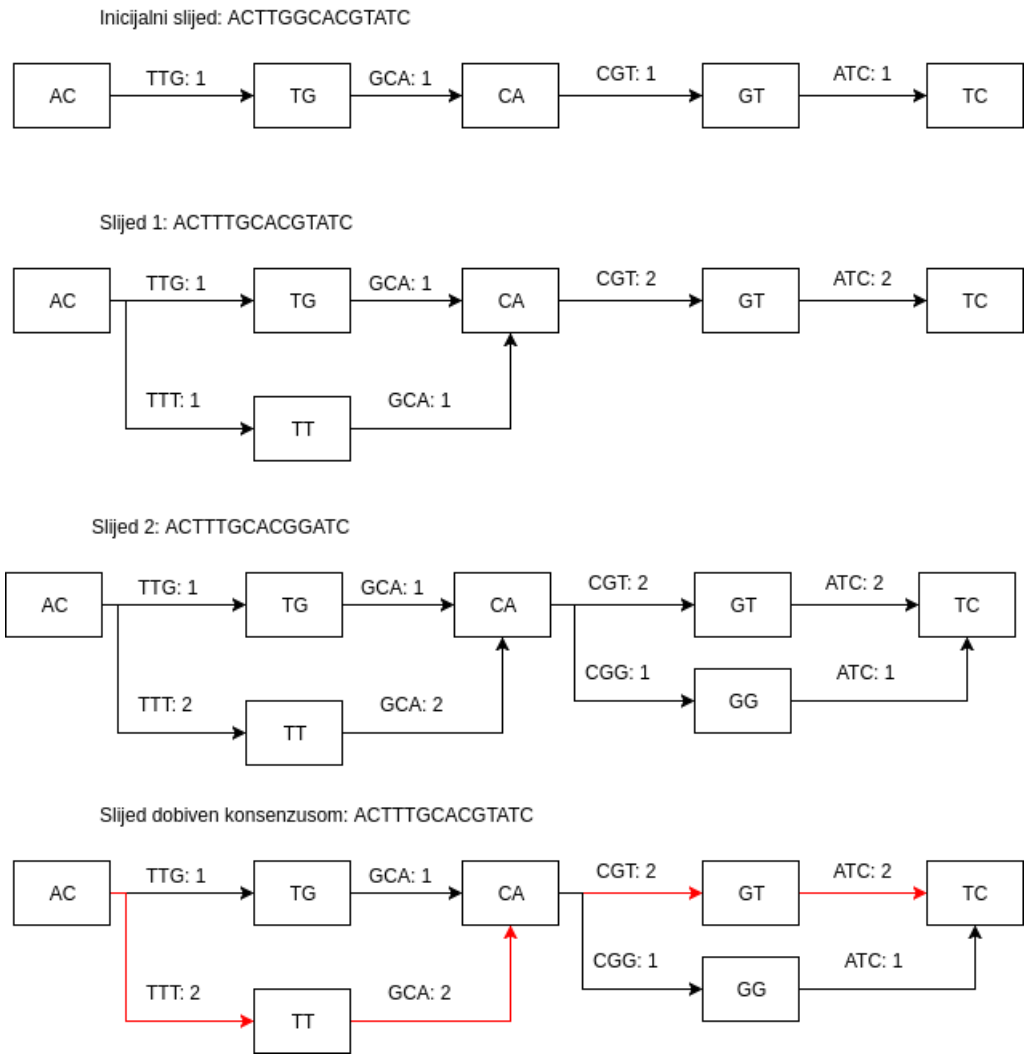
## 2.1. Opis algoritma

Prvi korak algoritma je konstrukcija k-mer grafa na temelju predanog ulaza koji sadrži izlaz iz faze Razmještaja, OLC paradigme. Ovisno o parametrima $k$ i $g$ kreira se inicijalni k-mer graf, gdje navedeni parametri određuju strukturu grafa, konkretno $k$ specificira koliko će nukleotida biti sadržano u pojedinom čvoru grafa, a $g$ specificira koliko će se nukleotida nalaziti na svakom bridu. Inicijalni graf je usmjeren sa samo jednim bridom iz svakog vrha osim završnog, koji ga nema. Razlika između ovog grafa i klasičnog de Bruijn grafa je u to tome što su isti k-meri na različitim pozicijama nezavisni jedni od drugih dok su kod de Bruijn grafa smješteni u jednom vrhu pa se ovaj graf smatra *sparse* grafom.

Sljedeći korak algoritma je poravnanje dodatnih slijedova čiji se postupak provodi ovisno o tome odgovara li k-mer u novom slijedu, k-meru u originalnom slijedu i njegovom bridu gdje se onda samo poveća težina brida za definiranu vrijednost, ili ukoliko ne odgovara dodaje se novi brid u graf i kreira se dodatni k-mer i samim time kreira novi put u grafu. Ovaj postupak je jako sličan kreiranju de Bruijn grafa, ali zbog razlikovanja istih k-mera ovisno o njihovoj poziciji, postoji razlika u postupku. Ovaj korak se ponavlja za sve slijedove koje smo dobili sekvenciranjem.

Završni korak Sparc algoritma je traženje puta u grafu koji ima najveću težinu, što je zahvaljujući činjenici da je konstruirani graf usmjeren i acikličan moguće napraviti s BFS ili DFS algoritmom kojim računamo težinu svakog vrha u grafu. Određivanje konsenzusnog slijeda se provodi tako da krenemo od najtežeg vrha i vraćamo se po najvećim težinama natrag sve do početnog vrha.

## 2.2. Primjer




**Slika 2.1:** Postupak izgradnje grafa s k=2, g=3

Slika 2.1 prikazuje cjelokupni postupak algoritma Sparc. Inicijalni slijed služi za kreiranje inicijalnog lanca (engl. *backbone*). Nakon kreiranja *backbone*-a grafa, sljedeći slijed poravnavamo tako da krećemo od početka slijeda i vidimo da je prvi k-mer AC jednak k-meru konstruiranom u grafu, ali je prijelaz na sljedeći k-mer TTT različit od onoga koji se nalazi u grafu te je stoga potrebno konstruirati novi k-mer TT te novi brid iz k-mera AC prema novom k-meru TT, a taj brid je TTT. Sljedećih $g$ nukleotida je GCA koji trebaju završiti u k-meru CA koji već postoji u konstruiranom grafu te je potrebno kreirati brid CGA od k-mera TT prema k-meru CA. Sljedećih $g$ nukleotida je CGT, budući da taj brid postoji u konstruiranom grafu potrebno je samo povećati težinu postojećeg brida, isto vrijedi i za sljedećih $g$ nukleotida ATC. Postupak je jednak

i za slijed 2. Za određivanje najtežeg puta pratimo bridove s najvećim težinama, a u ovdje kontruiranom grafu to je put od k-mera AC bridom TTT potom bridom GCA, CGT i ATC te je stoga rekonstruirani slijed ACTTTGCACGTATC.

# 3. Naša implementacija algoritma

Za ostvarivanje naše implementacije algoritma faze konzensuza Sparc, koristili smo programski jezik C++ uz korištenje dodatnih biblioteka i alata, koji će biti navedeni u nastavku. Implementacija koristi specifične formate podataka koji su također opisani u nastavku.

## 3.1. Korišteni formati podataka

Svi podaci o slijedovima koje koristimo u našem algoritmu nalaze se u predodređenim formatima podataka kako bi se algoritam mogao što jednostavnije koristiti u opće svrhe.

### 3.1.1. FASTA

Naš algoritam ovaj format koristi kako bi napravio početni lanac (engl. *backbone*), za ulaz te na izlaz stavlja rekonstruirani slijed također u FASTA formatu. Ulaz u naš algoritam je u biti izlaz iz faze razmještaja OLC paradigme [6, Poglavlje 1.3.1].

### 3.1.2. FASTQ

Sličan FASTA formatu, ali osim samog slijeda sadrži i oznaku kvalitete svakog očitanja. Ovaj format podataka sadrži sekvencirane slijedove koji će poslužiti za rekonstrukciju originalnog slijeda [6, Poglavlje 1.3.2] . Iako naša implementacija ne koristi direktno ovaj format podataka, on se koristi kao ulaz za alat GraphMap koji služi za generiranje SAM formata podataka.

### 3.1.3. SAM

Ovaj format podataka sadrži grupirane informacije o svim očitanjima iz FASTQ formata podataka [1]. Podaci iz ovog formata podataka služe za rekonstrukciju genoma.

Podaci iz SAM formata podataka koje koristimo u našoj implementaciji su:

- FLAG - skup zastavica koje ovisno o vrijednosti utječu na naš algoritam. Vrijednost zastavice 4 označava da je trenutno mapiranje nepostojeće te stoga preskačemo mapiranje, također uzimamo u obzir i vrijednost zastavice 16 koja označava da je trenutni slijed u SEQ dijelu inverzno komplementiran. Ostale vrijednosti zastavica zanemarujemo.
- POS - pozicija na osnovnom slijedu na kojoj počinje trenutno mapiranje, pozicija je indeksirana od indeksa 1
- CIGAR - popis operacija koje su napravljene nad očitanjem kako bi se dobilo mapiranje
- SEQ - originalno očitan slijed, prije nego što su obavljene CIGAR operacije
- QUAL - ASCII string u *Phred* bazi, definira kvalitetu mapiranja

## 3.2. Korištene biblioteke

Osim standardnih biblioteka programskog jezika C++, kao što su na primjer biblioteke za I/O operacije te STL(*Standard Template Library*) biblioteke koja sadrži implementacije složenijih struktura podataka, koristili smo i biblioteku/alat GraphMap.

### 3.2.1. GraphMap

GraphMap [4] biblioteka pruža implementaciju mapiranja poravnanja očitanih slijedova u odnosu na početni slijed. Ovu biblioteku smo koristili kako bi dobili poravnanja slijedova koja onda koristimo kod Sparc algoritma za konstruiranje grafa. Mapiranja poravnanja se dobiju tako što se GraphMap alatu predaju slijed na kojem želimo raditi poravnanje u FASTA formatu te datoteku s očitanjima slijedova u FASTQ formatu. Korištenjem opcije *align* dobiju se mapiranja poravnanja u SAM formatu datoteke.

GraphMap alat nismo koristili direktno u našoj implementaciji, ali smo koristili nastala mapiranja kako bi napravili što efikasniji algoritam.

## 3.3. Struktura implementacije

Implementacija je raspodijeljena na nekoliko cjelina: glavni program, definicije formata podataka i njihove parsere te dio u kojemu je definirana struktura k-mer grafa kao i implementacija samog Sparc algoritma.

Datoteka *main.cpp* predstavlja glavni program u kojemu se instanciraju potrebne klase za provedbu algoritma kao i svi potrebni parseri koje koristimo.

Definicije svih formata datoteka se nalaze u direktoriju *format* gdje je za svaki format podataka napravljena *header* datoteka i njena implementacija.

Implementacija samoga algoritma se nalazi u direktoriju *algorithm* gdje je također napravljena *header* datoteka i njena implementacija. Sparc se nalazi u razredu *KMerGraph* koji predstavlja implementaciju k-mer grafa, dok strukture *Vertex* i *Edge* predstavljaju reprezentacije vrha te brida grafa. Kod instanciranja razreda *KMerGraph* predaju se parametri $k$ i $g$ kojima se definira struktura grafa, a sama izgradnja *backbone-a* se provodi s metodom *initialGraph* kojoj se predaje *layout* dobiven iz faze Razmještaja. Nakon izgradnje početnog grafa prolazimo po datoteci SAM formata koja sadrži podatke o obavljenim mapiranjima poravnanja na osnovni slijed. Pomoću dobivenih mapiranja gdje su nam najvažnije stavke originalni slijed *SEQ* prije *CIGAR* operacija, same *CIGAR* operacije, početna pozicija u poravnanju te kvaliteta mapiranja, radimo dodavanje na početni graf bilo povećavanjem težina postojećih bridova ili dodavanjem novih bridova i k-mera. Težine bridova se povećavaju za vrijednost koja se dobije izračunom prosjeka iz vrijednosti *QUAL* i $g$, čime radimo prosječnu vrijednost *QUAL* po broju nukleotida na bridu. Nakon čitanja cijele SAM datoteke pokreće se traženje najtežeg puta u grafu i određivanje konsenzusnog slijeda koji je definiran najtežim putem u grafu.

## 3.4. Instalacija i pokretanje algoritma

Cjelokupna implementacija je javno dostupna na poslužitelju GitHub na poveznici `https://github.com/vkolobara/bioinf`. Za instalaciju je potrebno ili klonirati postojeći repozitorij ili skinuti *.zip* arhivu te pokrenuti skriptu *install.sh* kako bi se povukle sve potrebne biblioteke i testni podaci te preveli izvorni kodovi u izvršne datoteke.

Pokretanje algoritma se provodi pokretanjem skripte *run.sh* kojoj je moguće predati dodatne parametre koji specificiraju parametre $k$ i $g$ te broj iteracija i naziv testne datoteke koju želimo pokrenuti.

# 4. Analiza implementacije

Za utvrđivanje kvalitete implementacije, radimo usporedbu slijeda koji smo koristili kao ulaz u algoritam te konsenzusnog slijeda kojeg smo dobili kao izlaz iz algoritma s referentnim slijedom. Usporedbu radimo pomoću DnaDiff [2] alata koji nam pruža informaciju koliki je postotak podudarnosti našeg i referentnog slijeda. Postotak podudarnosti slijedova nam je najbitniji podatak prilikom analize, ali jednako tako su nam važni i podaci o utrošku memorijskog prostora te vremenu izvođenja algoritma.

## 4.1. Alati za analizu

Kako bi analiza podataka bila što preciznija koristimo gotove alate, kojima provjeravamo prije navedene čimbenike koje pratimo.

### 4.1.1. DnaDiff

DnaDiff [2] je jedan od alata iz programskog paketa otvorenog koda MUMmer, koji osim alata DnaDiff sadrži i druge alate koji se koriste na području bioinformatike. DnaDiff radi analizu između dva genetska slijeda i utvrđuje njihovu sličnost. Analiza provedena alatom daje detaljne informacije o sličnostima slijedova, ali kao faktor kvalitete implementacije smo koristili podatke iz datoteke *out.report* i poglavlja o sličnosti poravnanja, *Alignments*. Navedeno poglavlje ima analizu za 1-1 i M-M poravnanja koja su prilikom naših testiranja u većini slučajeva za polje AvgIdentity bila identične ili gotovo identične vrijednosti pa smo stoga kao faktor kvalitete rješenja odlučili koristiti vrijednost pod 1-1 AvgIdentity.

### 4.1.2. cgmemtime

Alat cgmemtime [3] služi za analizu vremena i potrošene memorije. Za memoriju se ispisuje najveća zabilježena vrijednost za vrijeme izvođenja algoritma te tako imamo informaciju koliko je minimalno memorije potrebno rezervirati za izvođenje programa.

## 4.2. Testna konfiguracija

Osnovni podaci o računalnoj konfiguraciji koja je korištena za analizu izvođenja ostvarenog rješenja navedena je u nastavku:

- Operacijski sustav - Arch Linux x86 64
- Procesor - Intel Core i3-6100 @ 3.70GHz
- RAM - 16 GiB DDR4 @ 2133MHz

## 4.3. Analiza kvalitete rješenja

| | k | g | vrijednost |
|---|---|---|---|
| 1 | 1 | 1 | 88.93 |
| 2 | 1 | 2 | 90.19 |
| 3 | 1 | 3 | 89.32 |
| 4 | 1 | 4 | 90.67 |
| 5 | 1 | 5 | 89.74 |
| 6 | 2 | 3 | 88.86 |
| 7 | 2 | 4 | 90.64 |
| 8 | 2 | 5 | 89.70 |
| 9 | 2 | 6 | 90.19 |
| 10 | 3 | 4 | 90.94 |
| 11 | 3 | 5 | 89.78 |
| 12 | 3 | 6 | 89.92 |
| 13 | 4 | 5 | 89.85 |
| 14 | 3 | 2 | 90.19 |
| 15 | 4 | 2 | 90.11 |

**Tablica 4.1:** Testiranje algoritma za različite vrijednosti parametara k i g na testnom skupu lambda za jednu iteraciju

Tablica 4.1 prikazuje kretanje vrijednosti podudarnosti slijeda dobivenog algoritmom i referentnog slijeda. Moguće je primijetiti da se najbolja rješenja pronalaze za vrijednosti parametara $k = 3$ i $g = 4$, te su stoga te vrijednosti korištene za daljnju analizu na drugim testnim skupovima.

| | vrijednost |
|---|---|
| 1. | 90.95 |
| 2. | 91.47 |
| 3. | 91.67 |
| 4. | 91.78 |
| 5. | 91.85 |

**Tablica 4.2:** Promjena vrijednosti kvalitete kroz više iteracija za k=3 i g=4 na testnom skupu lambda.

Tablica 4.2 prikazuje kako se uz korištenje više iteracija algoritma, na način da se izlaz iz iteracije $i$ koristi kao ulaz u iteraciji $i+1$, moguće poboljšati kvalitetu poklapanja između referentnog slijeda i našeg konsenzusnog slijeda. Također je vidljivo da više iteracije donose sve manja poboljšanja u odnosu na prethodne.

| | lambda | ecoli |
|---|---|---|
| layout | 86.16 | 88.57 |
| naš | 91.85 | 95.87 |
| original | 96.83 | 99.01 |

**Tablica 4.3:** Usporedba naše implementacije sa slijedom iz faze razmještanja (layout) i referentnim radom (Sparc) za k=3 i g=4.

Tablica 4.3 prikazuje usporedbu naše implementacije i implementacije referentnog rada iz koje je vidljivo da implementacija iz referentnog rada dobiva bolja rješenja od našeg algoritma, iako oba rješenja uspijevaju poboljšati slijed iz faze Razmještaj. Tijekom testiranja našeg algoritma primijetili smo da najbolje rezultate dobivamo upravo za ovakve postavke parametara $k$ i $g$.

Referentni rad navodi da su optimalni parametri za taj algoritam $k$ u rasponu od 1 do 2 te $g$ u rasponu od 1 do 3.

## 4.4. Analiza utroška memorije i vremena

Tablica 4.4 prikazuje potrošnju memorije i vrijeme izvođenja naše implementacije algoritma i iz nje je moguće vidjeti da su zadovoljena zadana ograničenja za oba skupa podataka.

| | vrijeme[s] | memorija[MiB] |
|---|---|---|
| lambda | 0.17 | 20 |
| ecoli | 26.8 | 1600 |

**Tablica 4.4:** Analiza utroška vremena i memorije za provedbu algoritma za k=3 i g=4.

# 5. Zaključak

Paradigma Premještaj-Razmještaj-Konsenzus pruža jedan način rješavanja problema sastavljanja genoma poslije njihovog sekvenciranja i radom na ovom projektu imali smo priliku implementirati jedan od algoritama koji se bave ovim problemom. Implementirani algoritam spada u fazu konsenzusa spomenute paradigme i naziva se Sparc, algoritam je temeljen na k-mer grafovima te zbog toga spada u modernije načine rješavanja sastavljanja genoma.

Tijekom implementacije smo radili i analizu kvalitete rješenja na dostupnim testnim skupovima i ovisno o rezultatima smo prilagođavali implementaciju. Nakon što smo pronašli zadovoljavajuće parametre radili smo usporedbu s referentnim algoritmom i utvrdili da iako oba algoritma poboljšavaju početni slijed, referentni algoritam to radi bolje.

# 6. Literatura


[1] James Bonfield, John Marshall, Yossi Farjoun, Jay Carey, Tim Fennell, i Nils Homer. Sam format specification. `https://samtools.github.io/hts-specs/SAMv1.pdf`, 2017.

[2] Steffan Kurtz, Adam Philippy, Art Delcher, Steven Salzberg, i Corina Antonescu. Mummer. `http://mummer.sourceforge.net`, 2017.

[3] Georg Sauthoff. cgmemtime. `https://github.com/gsauthof/cgmemtime`, 2017.

[4] Ivan Sović, Mile Šikić, i Niranjam Nagarajan. graphmap. `https://github.com/isovic/graphmap`, 2017.

[5] Chengxi Ye i Zhanshan (Sam) Ma. Sparc: a sparsity-based consensus algorithm for long erroneous sequencing reads. *PeerJ*, 4:e2016, 2016. ISSN 2167-8359. doi: 10.7717/peerj.2016. URL `https://peerj.com/articles/2016`.

[6] Mile Šikić i Mirjana Domazet-Lošo. *Bioinformatika*. Fakultet Elektrotehnike i Računarstva, Sveučilište u Zagrebu, 2013.

# 7. Sažetak

Algoritam Sparc je jedna od implementacija algoritma faze konsenzusa u Premještaj-Razmještaj-Konsenzus paradigmi sastavljanja genoma. Navedeni algoritam je implementiran u ovom projektu te je dodatno provedena analiza kvalitete rješenja s gotovim alatima, te je također napravljena usporedna analiza s originalnom implementacijom koja je napravljena u referentnom radu. Naša implementacija uspijeva propraviti početni slijed, ali ne u takvoj mjeri kao što to uspijeva originalna implementacija.