Read Mapping mit Varianten

Kada Benadjemia, Sven Schrinner

TU Dortmund, LS11 Informatik, Projektgruppe 583,
Prof. Dr. Sven Rahmann, Prof. Dr. Johannes Fischer

03. April 2014

Betreuer: Dominik Kopczysnki, Henning Timm

Basierend auf:
Sebastian Wandelt, Johannes Starlinger, Marc Bux, Ulf Leser, RCSI: Scalable similarity search in thousand(s) of genomes, VLDB '13
Lin Huang, Victoria Popic, Serafim Batzoglou, Short read alignment with populations of genomes, Bioinformatics '13

Read Mapping - Überblick

Bereits gesehen Read Mapping mit Hilfe eines Referenzgenoms

Read Mapping - Überblick

Bereits gesehen Read Mapping mit Hilfe eines Referenzgenoms

Problemstellung:

- Gegeben: Große Menge von „reads“ (=Gensequenzen mit 30-200 Basenpaare) durch Sequenzierung gewonnen
- Gegeben: Ein vollständiges Genom (=Referenzgenom)
- Aufgabe: Alle reads im Referenzgenom wiederfinden

Read Mapping - Überblick

Bereits gesehen Read Mapping mit Hilfe eines Referenzgenoms

Problemstellung:

- Gegeben: Große Menge von „reads“ (=Gensequenzen mit 30-200 Basenpaare) durch Sequenzierung gewonnen
- Gegeben: Ein vollständiges Genom (=Referenzgenom)
- Aufgabe: Alle reads im Referenzgenom wiederfinden

In einfacher Form $\rightarrow$ String-Matching-Problem

Read Mapping - Überblick

Bereits gesehen Read Mapping mit Hilfe eines Referenzgenoms

Problemstellung:

- Gegeben: Große Menge von „reads“ (=Gensequenzen mit 30-200 Basenpaare) durch Sequenzierung gewonnen
- Gegeben: Ein vollständiges Genom (=Referenzgenom)
- Aufgabe: Alle reads im Referenzgenom wiederfinden

In einfacher Form $\rightarrow$ String-Matching-Problem

- Lösungsansätze schon gesehen (z.B. Burrows Wheeler Transform)

Probleme des einfachen Read Mappings

Bisher Vorhandene Reads in einem Referenzgenom gesucht

Probleme des einfachen Read Mappings

Bisher Vorhandene Reads in einem Referenzgenom gesucht

Problem Auswahl des Referenzgenoms

Probleme des einfachen Read Mappings

Bisher Vorhandene Reads in einem Referenzgenom gesucht

Problem Auswahl des Referenzgenoms

Es gibt nicht DAS Referenzgenom!

Probleme des einfachen Read Mappings

Bisher Vorhandene Reads in einem Referenzgenom gesucht

Problem Auswahl des Referenzgenoms

Es gibt nicht DAS Referenzgenom!

- Jedes Referenzgenom letztlich ein willkürliches Individuum

Probleme des einfachen Read Mappings

Bisher Vorhandene Reads in einem Referenzgenom gesucht

Problem Auswahl des Referenzgenoms

Es gibt nicht DAS Referenzgenom!

- Jedes Referenzgenom letztlich ein willkürliches Individuum
- Abweichung von Referenz muss keine „bösartige“ Mutation sein

Probleme des einfachen Read Mappings

Bisher Vorhandene Reads in einem Referenzgenom gesucht

Problem Auswahl des Referenzgenoms

Es gibt nicht DAS Referenzgenom!

- Jedes Referenzgenom letztlich ein willkürliches Individuum
- Abweichung von Referenz muss keine „bösartige“ Mutation sein
- Häufige Abweichungen (=Varianten) sollen als solche erkannt werden

Beispiel für Variante

Beispiel.

Referenzgenom: CGTAACTGGGCATG

Reads:
GTAACTGGG
TACCTGGGC
CGTACCTGT

Beispiel für Variante

Beispiel.

Referenzgenom: CGTAACTGGGCATG

Reads: GTAACTGGG
TACCTGGGC
CGTACCTGT

Ersetzung A→ C: Häufige Abweichung von Referenz → Variante

Beispiel für Variante

Beispiel.

Referenzgenom: CGTAACTGGGCATG

Reads: GTAACTGGG
TACCTGGGC
CGTACCTGT

Ersetzung A→ C: Häufige Abweichung von Referenz → Variante
Ersetzung G→ T: Relevante Abweichung von Referenz

Beispiel für Variante

Beispiel.

Referenzgenom: CGTAACTGGGCATG

Reads: GTAACTGGG
TACCTGGGC
CGTACCTGT

Ersetzung A→ C: Häufige Abweichung von Referenz → Variante
Ersetzung G→ T: Relevante Abweichung von Referenz

Wie können wir mit Varianten umgehen?

Beispiel für Variante

Beispiel.

Referenzgenom: CGTAACTGGGCATG

Reads: GTAACTGGG
TACCTGGGC
CGTACCTGT

Ersetzung A→ C: Häufige Abweichung von Referenz → Variante
Ersetzung G→ T: Relevante Abweichung von Referenz

Wie können wir mit Varianten umgehen?

Verwendung von mehreren Referenzgenomen!

Referenzgenommenge

Beispiel.

Referenzgenome:
CGTAACTGGGCATG
CGTAACTGGGCATG
CGTACCTGGGCATG

Read:
CGTACCTGT

Referenzgenommenge

Beispiel.

Referenzgenome:	CGTAACTGGGCATG
	CGTAACTGGGCATG
	CGTACCTGGGCATG
Read:	CGTACCTGT

Ersetzung A$\rightarrow$ C: In Referenzmenge vorhanden $\rightarrow$ bekannte Variante

Referenzgenommenge

Beispiel.

Referenzgenome:	CGTAACTGGGCATG
	CGTAACTGGGCATG
	CGTACCTGGGCATG
Read:	CGTACCTGT

Ersetzung A→ C: In Referenzmenge vorhanden → bekannte Variante

Ersetzung G→ T: Bisher unbekannte Variante

Referenzgenommenge

Beispiel.

Referenzgenome:	CGTAACTGGGCATG
	CGTAACTGGGCATG
	CGTACCTGGGCATG
Read:	CGTACCTGT

Ersetzung A→ C: In Referenzmenge vorhanden → bekannte Variante

Ersetzung G→ T: Bisher unbekannte Variante

Neues algorithmisches Problem:

Referenzgenommenge

Beispiel.

Referenzgenome:
CGTAACTGGGCATG
CGTAACTGGGCATG
CGTACCTGGGCATG

Read:
CGTACCTGT

Ersetzung A→ C: In Referenzmenge vorhanden → bekannte Variante
Ersetzung G→ T: Bisher unbekannte Variante

Neues algorithmisches Problem:

Suche kurze Reads in einer Menge von Referenzgenomen

Analyse des neuen Problems

Klar Einzige Änderung: Referenzmenge statt Individuum

Also Problem durch wiederholtes einfaches Read Mapping lösbar

Analyse des neuen Problems

Klar Einzige Änderung: Referenzmenge statt Individuum

Also Problem durch wiederholtes einfaches Read Mapping lösbar

Wo ist das Problem?

Analyse des neuen Problems

Klar Einzige Änderung: Referenzmenge statt Individuum

Also Problem durch wiederholtes einfaches Read Mapping lösbar

Wo ist das Problem?

- Aktueller Bestand von sequenzierten Genomen liegt bei über 1000 Genomen („1000 Genomes Project“)
- Read muss in jedem Genom gesucht werden
- Aufwand steigt linear mit Größe der Genommenge → schlechte Skalierung für große Datenbanken!

Analyse des neuen Problems

Klar Einzige Änderung: Referenzmenge statt Individuum

Also Problem durch wiederholtes einfaches Read Mapping lösbar

Wo ist das Problem?

- Aktueller Bestand von sequenzierten Genomen liegt bei über 1000 Genomen („1000 Genomes Project“)
- Read muss in jedem Genom gesucht werden
- Aufwand steigt linear mit Größe der Genommenge $\rightarrow$ schlechte Skalierung für große Datenbanken!

Beobachtung Unterschied des Genoms zweier Menschen 1%

Analyse des neuen Problems

Klar Einzige Änderung: Referenzmenge statt Individuum

Also Problem durch wiederholtes einfaches Read Mapping lösbar

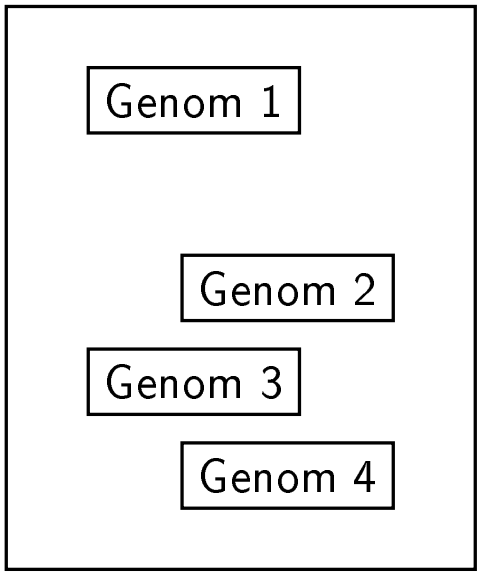
Wo ist das Problem?

- Aktueller Bestand von sequenzierten Genomen liegt bei über 1000 Genomen („1000 Genomes Project“)
- Read muss in jedem Genom gesucht werden
- Aufwand steigt linear mit Größe der Genommenge → schlechte Skalierung für große Datenbanken!

Beobachtung Unterschied des Genoms zweier Menschen 1%

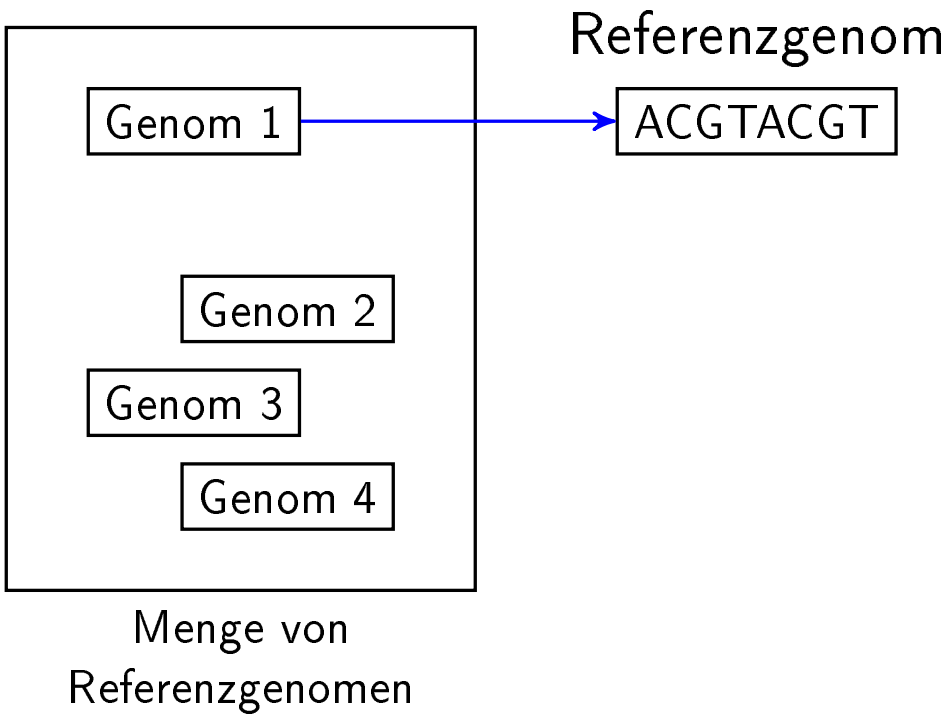
→ Viel redundante Suche bei naivem Ansatz!

Idee des RCSI-Algorithmus

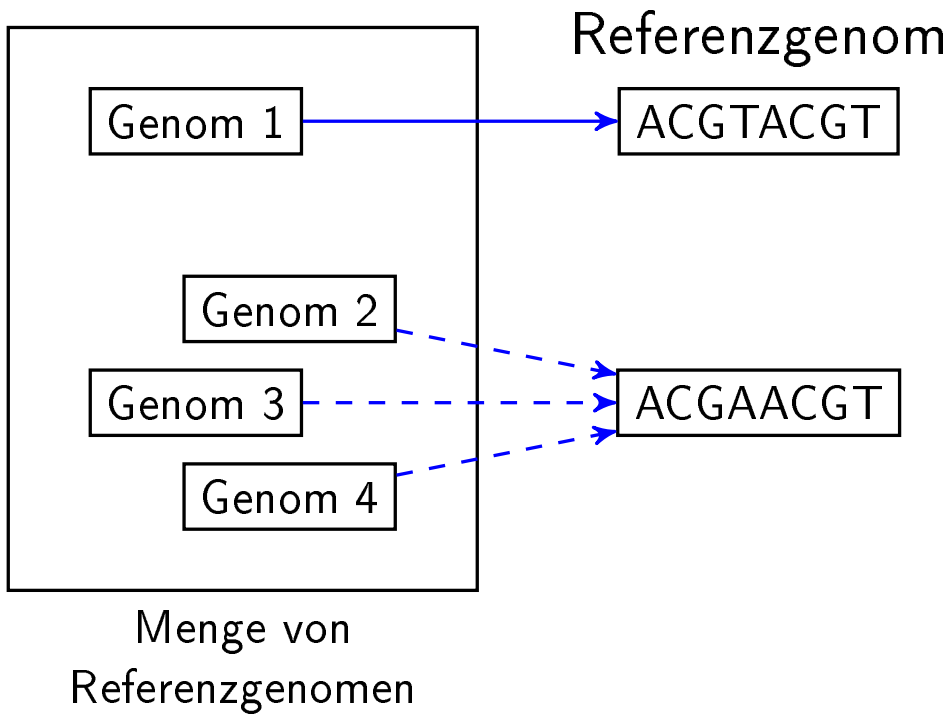


Menge von
Referenzgenomen

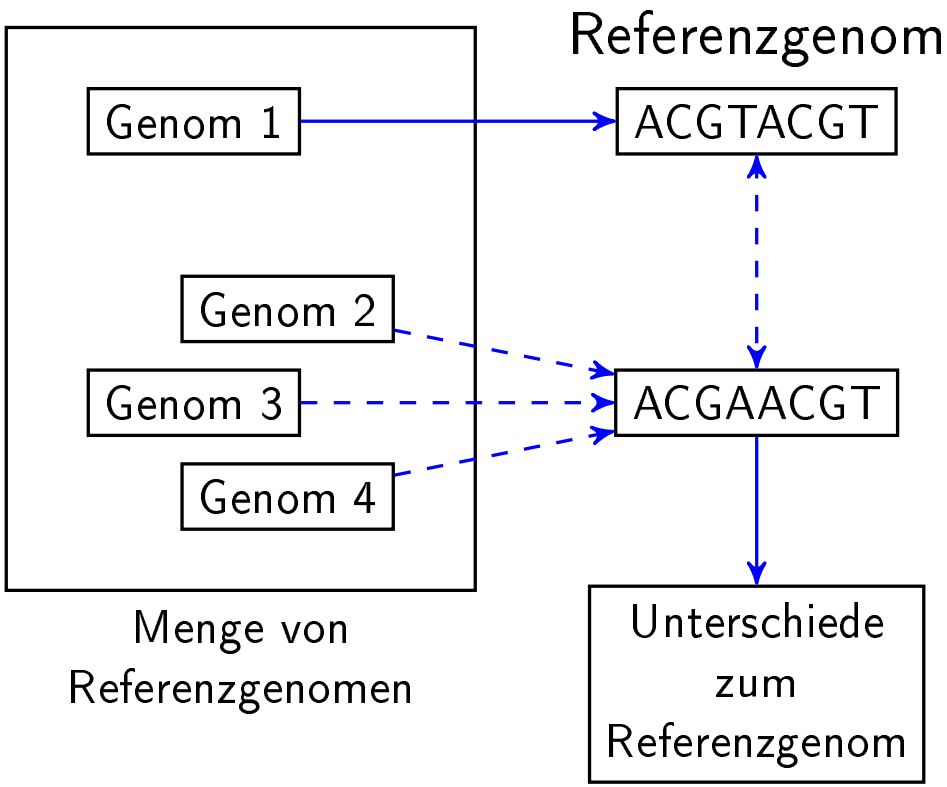
Idee des RCSI-Algorithmus



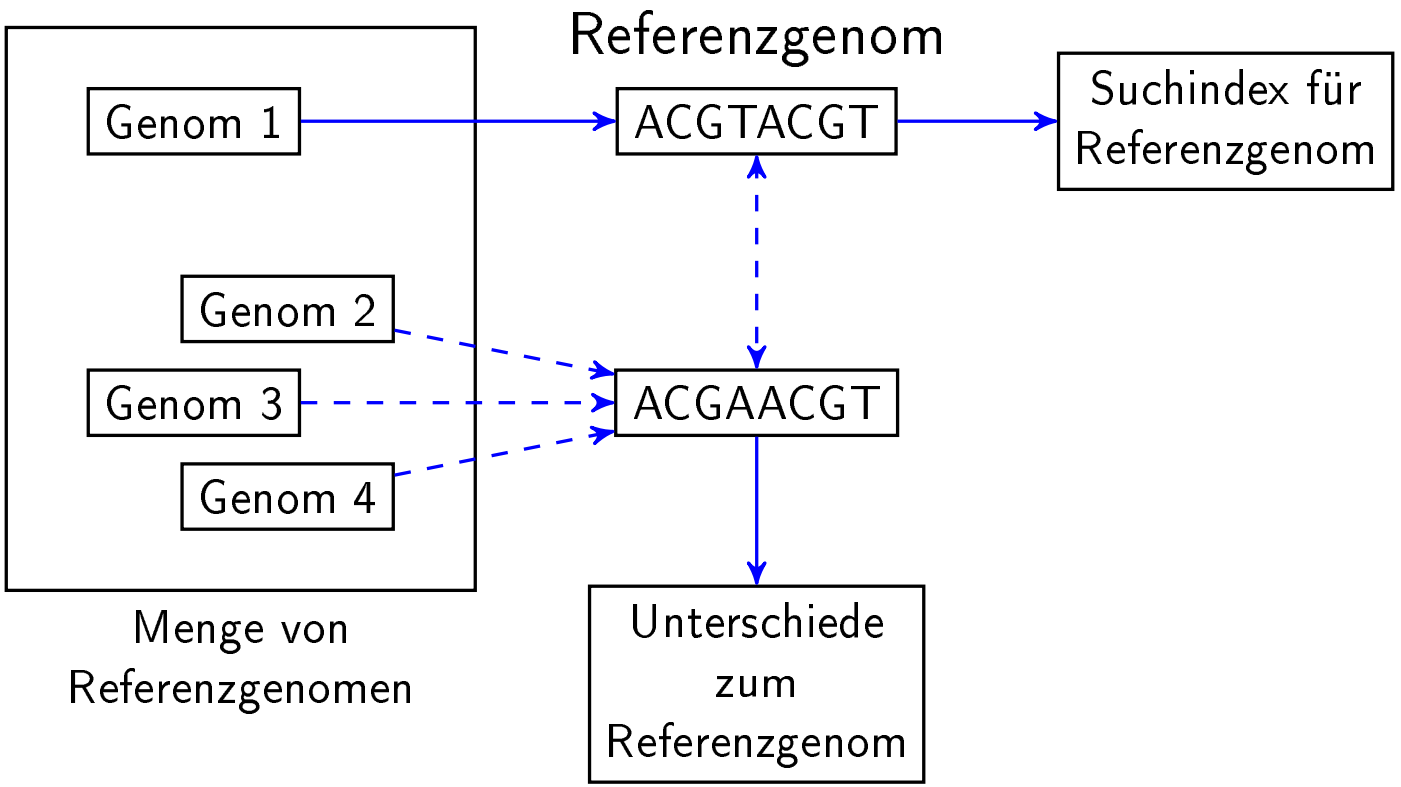
Idee des RCSI-Algorithmus



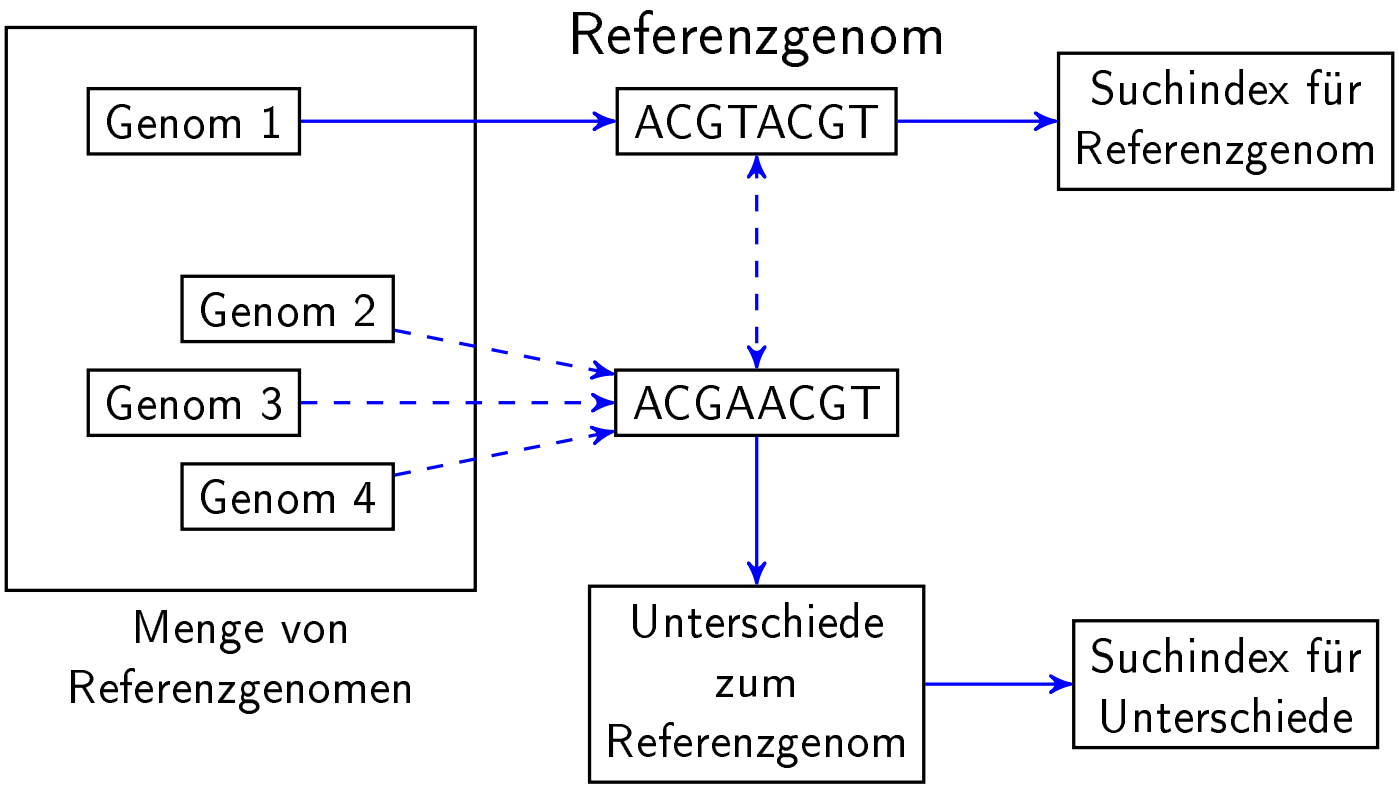
Idee des RCSI-Algorithmus



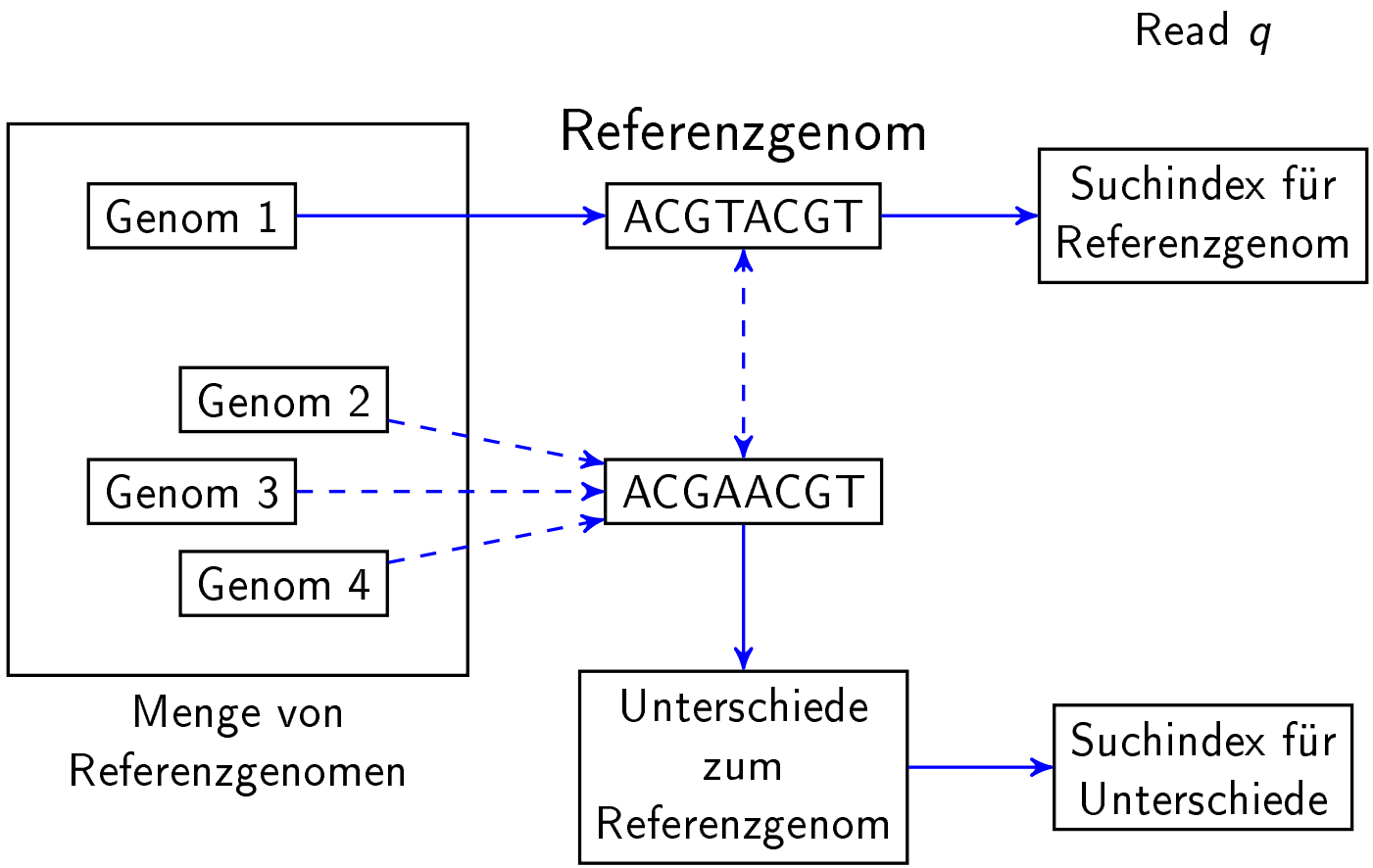
Idee des RCSI-Algorithmus



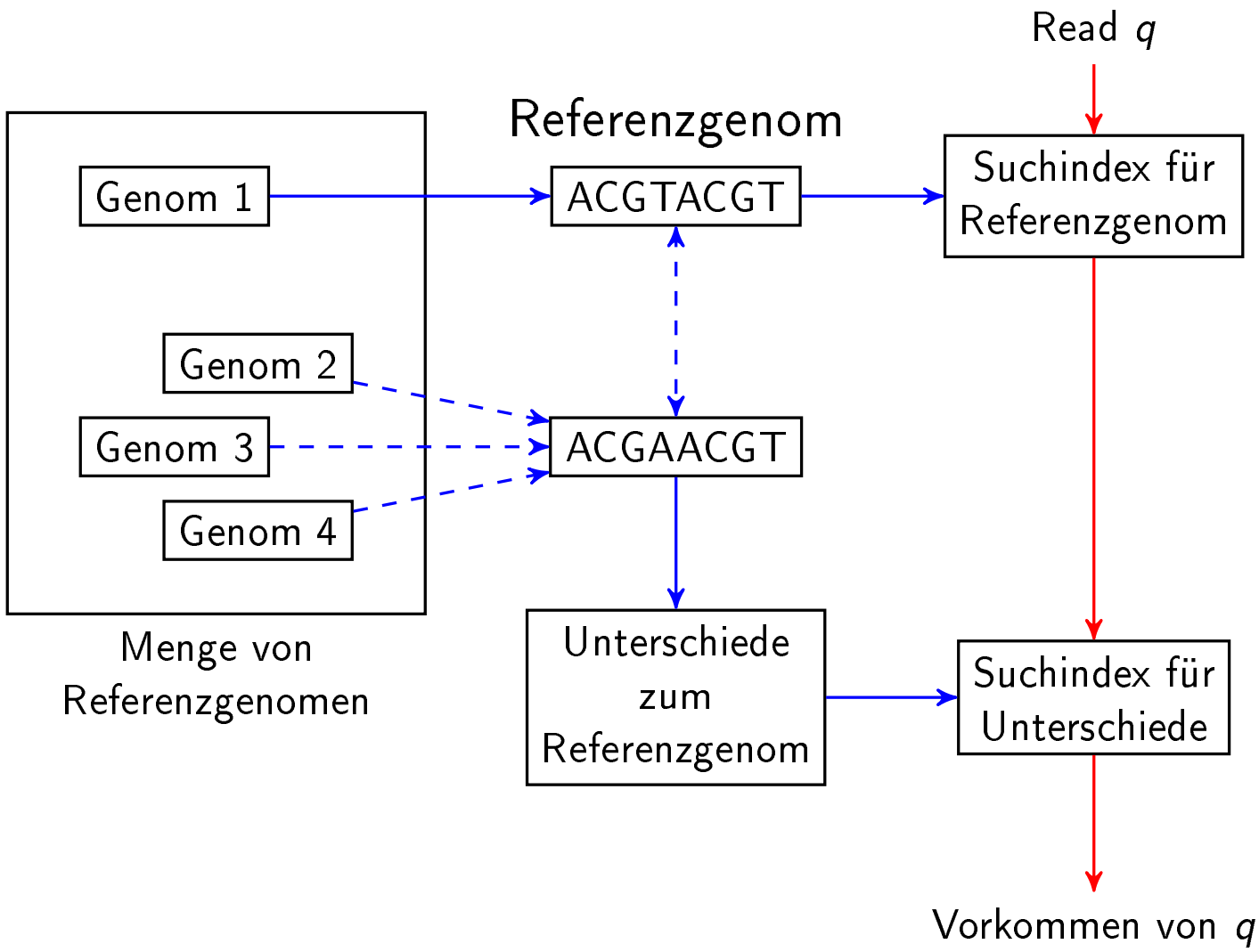
Idee des RCSI-Algorithmus



Idee des RCSI-Algorithmus



Idee des RCSI-Algorithmus



k-Ähnlichkeit

Betrachten Strings über Alphabet Σ $(= \{A, C, G, T\})$

k-Ähnlichkeit

Betrachten Strings über Alphabet Σ $(= \{A, C, G, T\})$

Notation

$\lvert s \rvert$	Länge eines Strings s
$s(i, n)$	Teilstring von s der Länge n ab Position i
$s(i)$	$= s(i, 1)$, wobei $s(0)$ erstes Zeichen
$s \circ t$	Konkatenation von s und t

k-Ähnlichkeit

Betrachten Strings über Alphabet Σ $(= \{A, C, G, T\})$

Notation

$\lvert s \rvert$	Länge eines Strings s
$s(i, n)$	Teilstring von s der Länge n ab Position i
$s(i)$	$= s(i, 1)$, wobei $s(0)$ erstes Zeichen
$s \circ t$	Konkatenation von s und t

Definition. Seien s, t zwei Strings. s heißt *k-ähnlich* zu t, $s \sim_k t$, falls s durch höchstens k Modifikationen zu t transformiert werden kann. Modifikationen: Einfügen, Löschen oder Ersetzen eines Zeichens in s.

k-Ähnlichkeit

Betrachten Strings über Alphabet Σ $(= \{A, C, G, T\})$

Notation

$\lvert s \rvert$	Länge eines Strings s
$s(i, n)$	Teilstring von s der Länge n ab Position i
$s(i)$	$= s(i, 1)$, wobei $s(0)$ erstes Zeichen
$s \circ t$	Konkatenation von s und t

Definition. Seien s, t zwei Strings. s heißt *k-ähnlich* zu t, $s \sim_k t$, falls s durch höchstens k Modifikationen zu t transformiert werden kann. Modifikationen: Einfügen, Löschen oder Ersetzen eines Zeichens in s.

Beispiel $\underline{G}TC\underline{C}AGTA \simeq_3 CTCAGT\underline{T}A$

k-approximative Suche

Definition. Seien s, q zwei Strings. Die Menge der *k-approximativen Vorkommen* von q in s ist definiert als:

$$\text{search}(s)_q^k = \{(i, s(i,j) | s(i,j) \sim_k q\}$$

k-approximative Suche

Definition. Seien s, q zwei Strings. Die Menge der *k-approximativen Vorkommen* von q in s ist definiert als:

$$\text{search}(s)_q^k = \{(i, s(i,j) | s(i,j) \sim_k q\}$$

Für eine Menge $S = \{s_1, \ldots, s_n\}$ von Strings definieren wir

k-approximative Suche

Definition. Seien s, q zwei Strings. Die Menge der *k-approximativen Vorkommen* von q in s ist definiert als:

$$\text{search}(s)_q^k = \{(i, s(i,j) | s(i,j) \sim_k q\}$$

Für eine Menge $S = \{s_1, \ldots, s_n\}$ von Strings definieren wir

$$\text{DBsearch}(s)_q^k = \left\{ \left(l, \text{search}(s_l)_q^k \right) | s_l \in S \right\}$$

Beispiel für k-approximative Suche

Beispiel. Seien $S = \{s_1, s_2, s_3\}$ mit $s_1 = ACACTG, s_2 = GGCTA, s_3 = ACTGA$:

$$\text{DBsearch}(S)^1_{CGA} = \{(1, \{(1, CA)\}), (2, \{(2, CTA)\}), (3, \{(1, CTGA), (2, TGA), (3, GA)\})\}$$

Referenzkompression

Definition. Ein *Referenzeintrag* ist ein Tripel $RE = (I, L, F)$ mit I ein Startindex, L die Länge eines Teilstrings und *Fehler* ein Zeichen. Es gilt $|RE| = L + 1$.

Referenzkompression

Definition. Ein *Referenzeintrag* ist ein Tripel $RE = (I, L, F)$ mit I ein Startindex, L die Länge eines Teilstrings und *Fehler* ein Zeichen. Es gilt $|RE| = L + 1$.

Definition. Seien s, ref zwei Strings. Eine *Referenzkompression* von s bezüglich ref, compress(s, ref), ist eine Liste von Referenzeinträgen

Referenzkompression

Definition. Ein *Referenzeintrag* ist ein Tripel $RE = (I, L, F)$ mit I ein Startindex, L die Länge eines Teilstrings und *Fehler* ein Zeichen. Es gilt $|RE| = L + 1$.

Definition. Seien s, ref zwei Strings. Eine *Referenzkompression* von s bezüglich ref, compress(s, ref), ist eine Liste von Referenzeinträgen

$$[(I_1, L_1, F_1), \ldots, (I_n, L_n, F_n)]$$

Referenzkompression

Definition. Ein *Referenzeintrag* ist ein Tripel $RE = (I, L, F)$ mit I ein Startindex, L die Länge eines Teilstrings und *Fehler* ein Zeichen. Es gilt $|RE| = L + 1$.

Definition. Seien s, ref zwei Strings. Eine *Referenzkompression* von s bezüglich ref, compress(s, ref), ist eine Liste von Referenzeinträgen

$$[(I_1, L_1, F_1), \ldots, (I_n, L_n, F_n)]$$

sodass

$$s = (ref(I_1, L_1) \circ F_1) \circ \cdots \circ (ref(I_n, L_n) \circ F_n)$$

Referenzkompression (Beispiel)

Beispiel.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
G	A	C	G	A	T	C	G	A	C	G	A	C	G	G	A	C	A	A	A	C	A

G	A	C	G	A	T	C	C	A	C	G	A	C	G	A	C	A	A	A	C	A	T

Referenzkompression (Beispiel)

Beispiel.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
G	A	C	G	A	T	C	G	A	C	G	A	C	G	G	A	C	A	A	A	C	A

G	A	C	G	A	T	C	C	A	C	G	A	C	G	A	C	A	A	A	C	A	T

Referenzkompression (Beispiel)

Beispiel.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
G	A	C	G	A	T	C	G	A	C	G	A	C	G	G	A	C	A	A	A	C	A

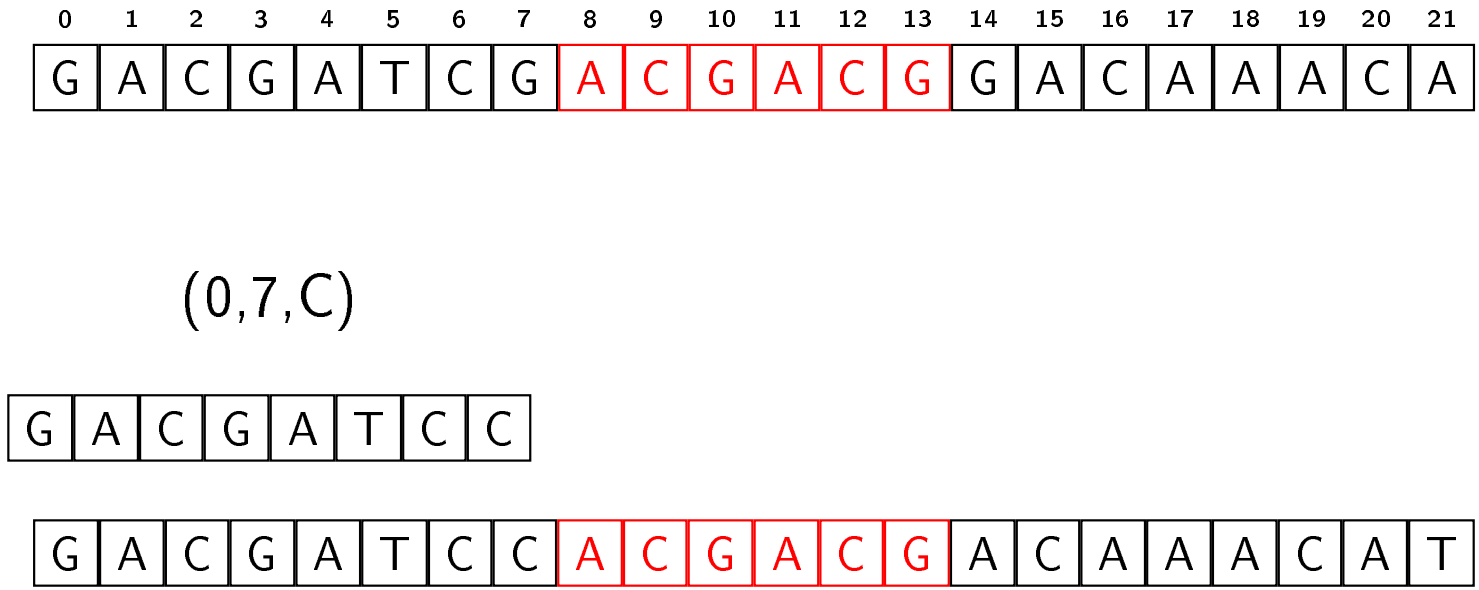
(0,7,C)

G	A	C	G	A	T	C	C

G	A	C	G	A	T	C	C	A	C	G	A	C	G	A	C	A	A	A	C	A	T

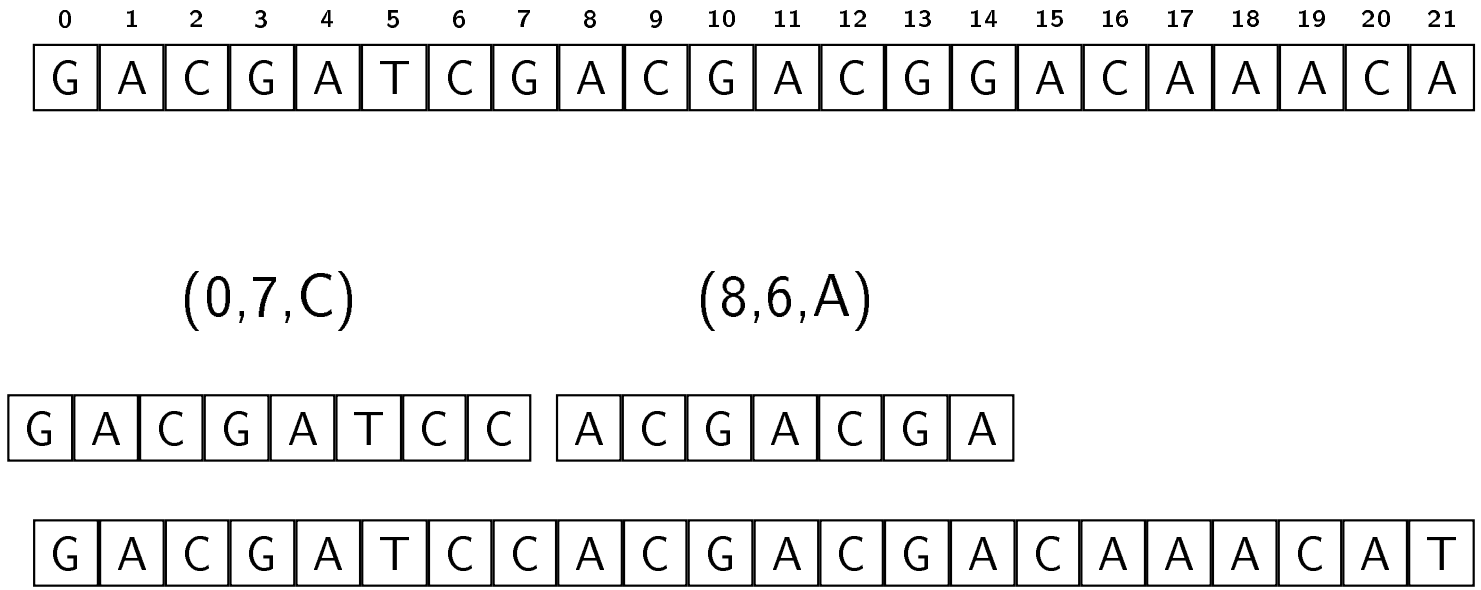
Referenzkompression (Beispiel)

Beispiel.



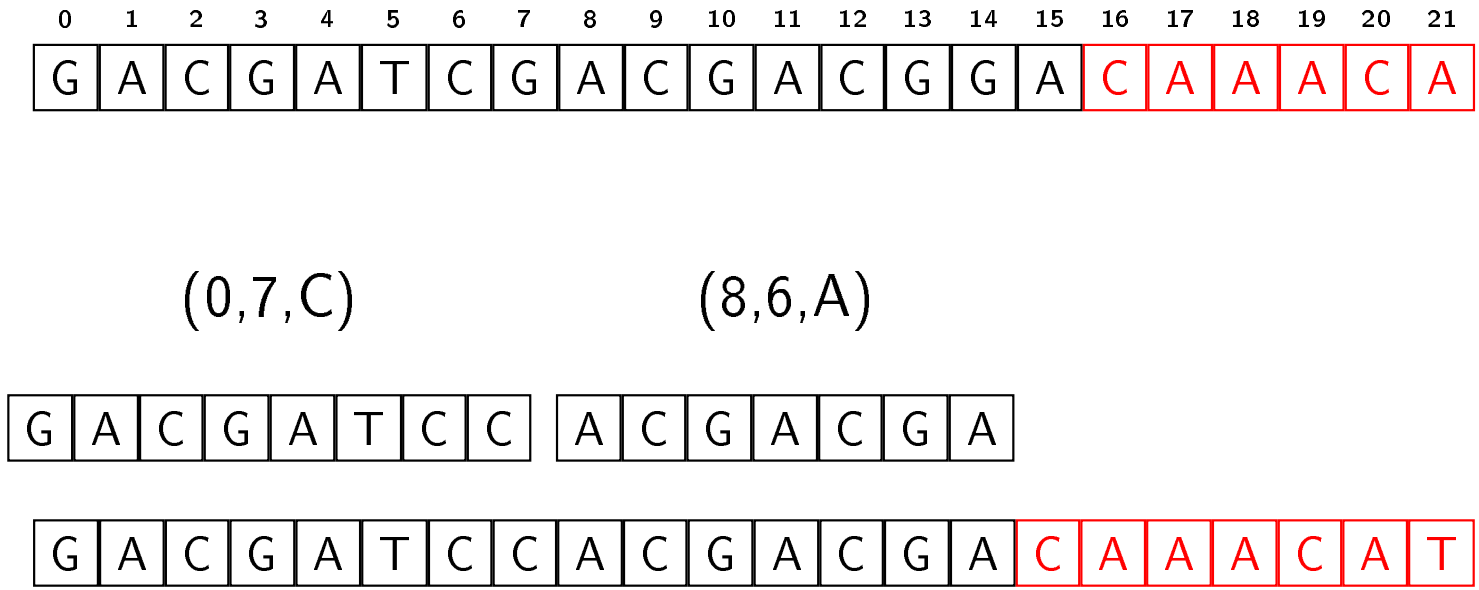
Referenzkompression (Beispiel)

Beispiel.



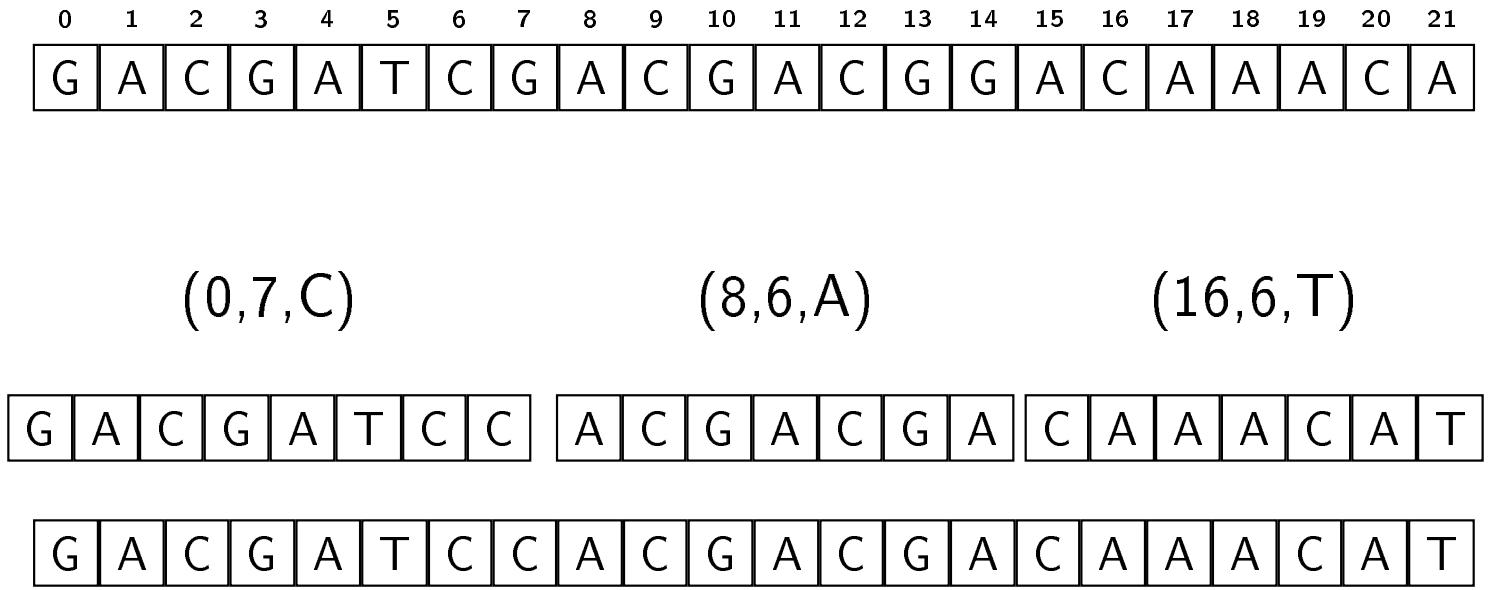
Referenzkompression (Beispiel)

Beispiel.



Referenzkompression (Beispiel)

Beispiel.



Referenzkompression (Beispiel)

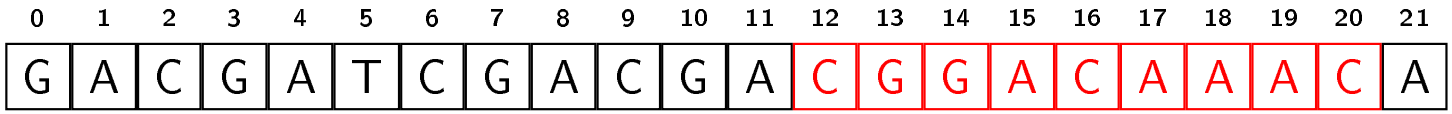
Beispiel.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
G	A	C	G	A	T	C	G	A	C	G	A	C	G	G	A	C	A	A	A	C	A

C	G	G	A	C	A	A	A	C	T	G	A	C	G	T	T	C	G	A	C	G

Referenzkompression (Beispiel)

Beispiel.



Referenzkompression (Beispiel)

Beispiel.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
G	A	C	G	A	T	C	G	A	C	G	A	C	G	G	A	C	A	A	A	C	A

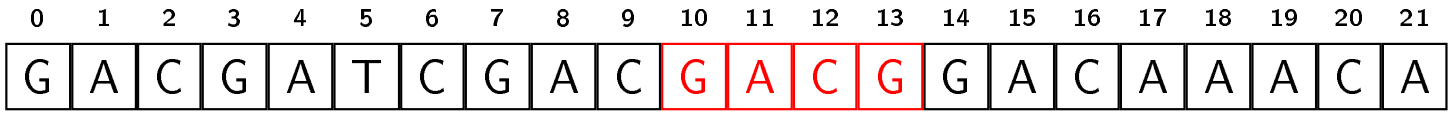
(12,9,T)

C	G	G	A	C	A	A	A	C	T

C	G	G	A	C	A	A	A	C	T	G	A	C	G	T	T	C	G	A	C	G

Referenzkompression (Beispiel)

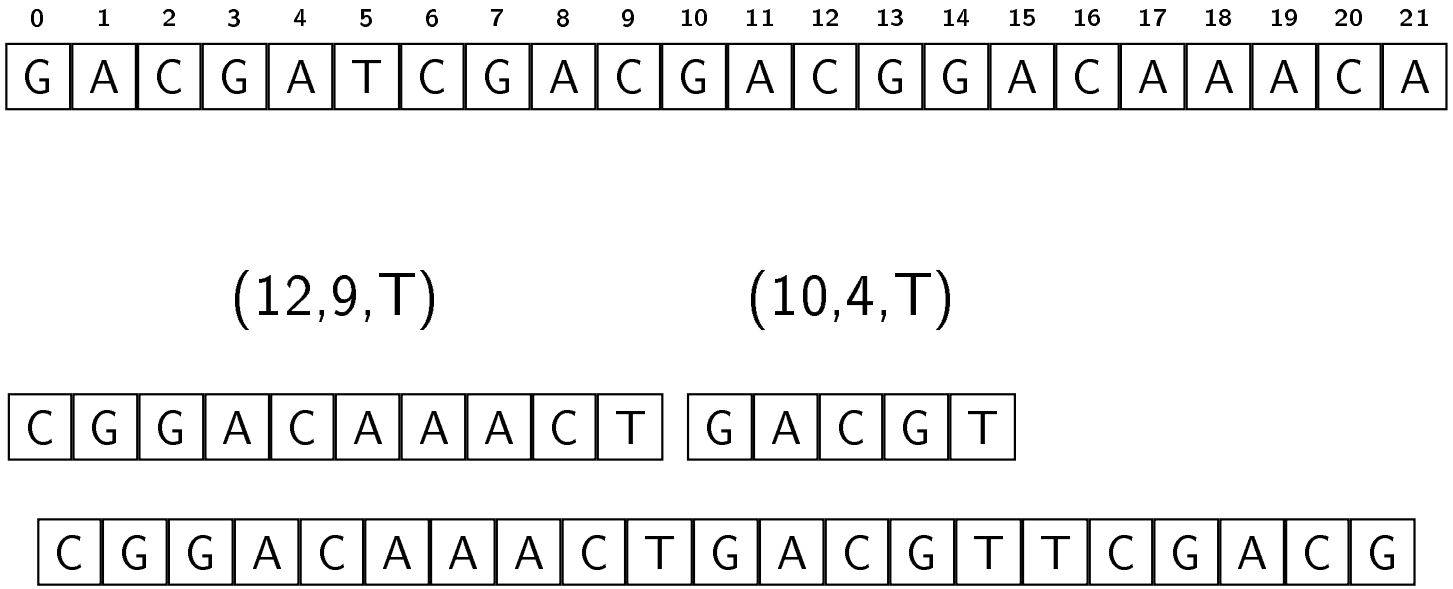
Beispiel.



(12,9,T)

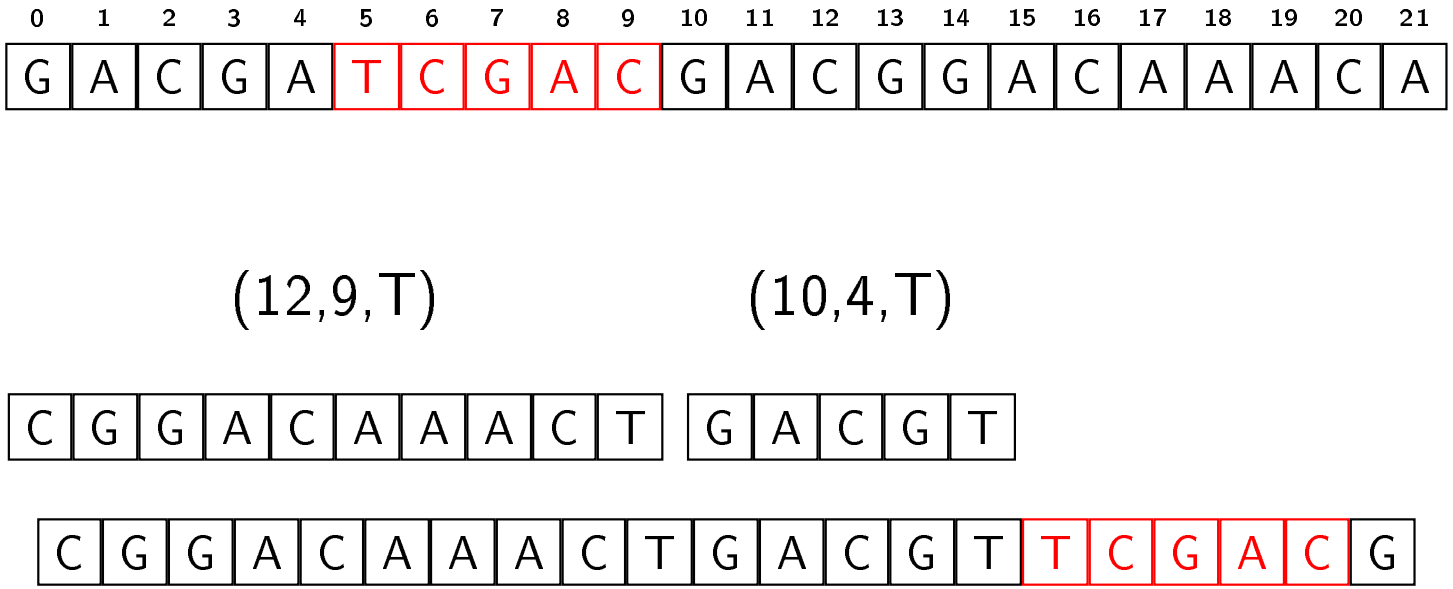
Referenzkompression (Beispiel)

Beispiel.



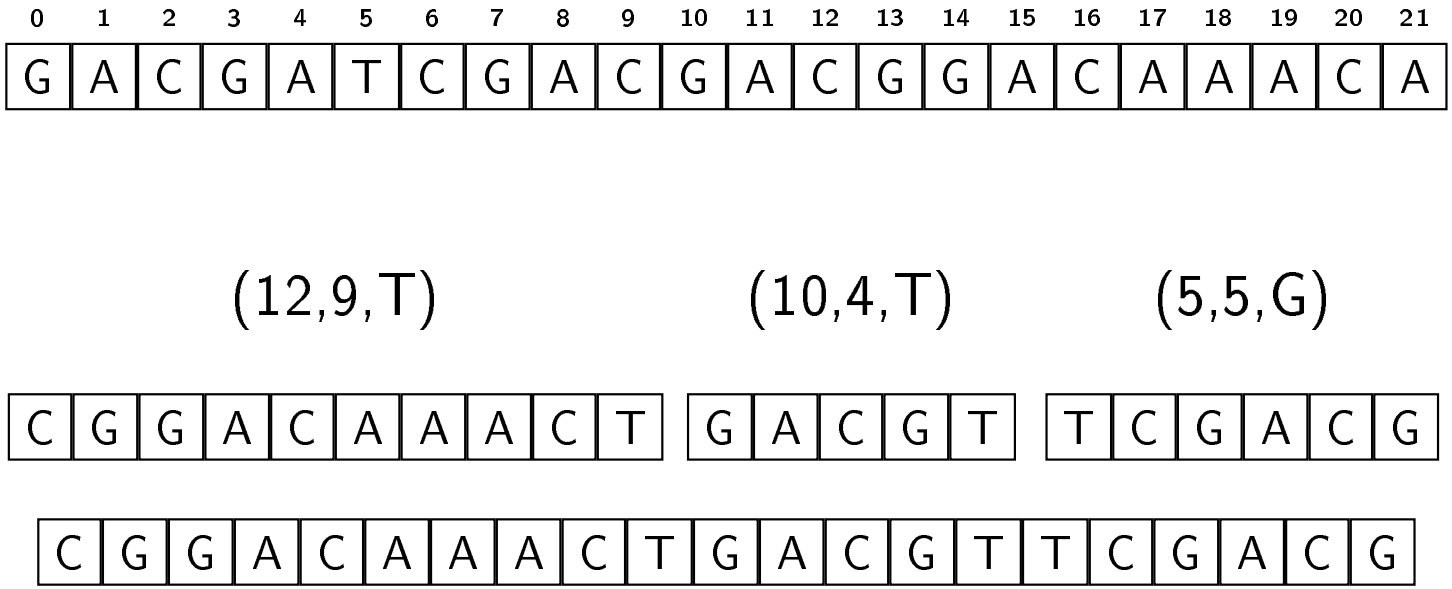
Referenzkompression (Beispiel)

Beispiel.



Referenzkompression (Beispiel)

Beispiel.



Berechnung der Referenzeinträge

Algorithmus Referenzkompressionsalgorithmus

Eingabe: Referenzstring ref und zu komprimierender String s
Ausgabe: compress(s, ref)

compress(s, ref) $\leftarrow$ []
while $|s| \neq 0$ **do**
    $pre \leftarrow$ maximales Präfix von s mit $(i, pre) \in \text{search}(ref)^0_{pre}$
    if $s \neq pre$ **then**
        compress(s, ref)+=[$(i, |pre|, s(|pre|))$]
        $s \leftarrow s(|pre| + 1, |s| - 1)$
    else
        compress(s, ref)+=[$(i, |pre| - 1, s(|pre| - 1))$]
        $s \leftarrow \epsilon$ /* leeres Wort*/

Speicherung von Varianten

Referenzeinträge:

- Genom verlustfrei in Referenzeinträge zerlegbar

Speicherung von Varianten

Referenzeinträge:

- Genom verlustfrei in Referenzeinträge zerlegbar
- Benötigter Speicher $\sim$ Anzahl Referenzeinträge

Speicherung von Varianten

Referenzeinträge:

- Genom verlustfrei in Referenzeinträge zerlegbar
- Benötigter Speicher $\sim$ Anzahl Referenzeinträge
- Anzahl Referenzeinträge beschränkt durch Unterschiede zur Referenz
  $\rightarrow$ gute Eigenschaft für unseren Anwendungsfall

Speicherung von Varianten

Referenzeinträge:

- Genom verlustfrei in Referenzeinträge zerlegbar
- Benötigter Speicher $\sim$ Anzahl Referenzeinträge
- Anzahl Referenzeinträge beschränkt durch Unterschiede zur Referenz
  $\rightarrow$ gute Eigenschaft für unseren Anwendungsfall

Größenvergleich (1092 Genome):

Speicherung von Varianten

Referenzeinträge:

- Genom verlustfrei in Referenzeinträge zerlegbar
- Benötigter Speicher $\sim$ Anzahl Referenzeinträge
- Anzahl Referenzeinträge beschränkt durch Unterschiede zur Referenz
  $\rightarrow$ gute Eigenschaft für unseren Anwendungsfall

Größenvergleich (1092 Genome):

3 TB Rohdaten

Speicherung von Varianten

Referenzeinträge:

- Genom verlustfrei in Referenzeinträge zerlegbar
- Benötigter Speicher $\sim$ Anzahl Referenzeinträge
- Anzahl Referenzeinträge beschränkt durch Unterschiede zur Referenz
  $\rightarrow$ gute Eigenschaft für unseren Anwendungsfall

Größenvergleich (1092 Genome):

3 TB Rohdaten vs.

Speicherung von Varianten

Referenzeinträge:

- Genom verlustfrei in Referenzeinträge zerlegbar
- Benötigter Speicher $\sim$ Anzahl Referenzeinträge
- Anzahl Referenzeinträge beschränkt durch Unterschiede zur Referenz
  $\rightarrow$ gute Eigenschaft für unseren Anwendungsfall

Größenvergleich (1092 Genome):

3 TB Rohdaten vs.

- 10 GB für Referenzgenom (inkl. Suchindex)
- 105 GB für Varianten (inkl. Suchindex)

Speicherung von Varianten

Referenzeinträge:

- Genom verlustfrei in Referenzeinträge zerlegbar
- Benötigter Speicher $\sim$ Anzahl Referenzeinträge
- Anzahl Referenzeinträge beschränkt durch Unterschiede zur Referenz
  $\rightarrow$ gute Eigenschaft für unseren Anwendungsfall

Größenvergleich (1092 Genome):

3 TB Rohdaten vs.

- 10 GB für Referenzgenom (inkl. Suchindex)
- 105 GB für Varianten (inkl. Suchindex)

Wie können wir die komprimierten Daten effizient durchsuchen?

Fallunterscheidung

Beobachtung. Vorkommen von Reads in einer Referenzkompression können auf zwei Weisen auftreten:

Fallunterscheidung

Beobachtung. Vorkommen von Reads in einer Referenzkompression können auf zwei Weisen auftreten:

1. Der Read liegt vollständig innerhalb eines REs

Fallunterscheidung

Beobachtung. Vorkommen von Reads in einer Referenzkompression können auf zwei Weisen auftreten:

1. Der Read liegt vollständig innerhalb eines REs
2. Der Read überdeckt mindestens ein Fehlerzeichen eines REs

Fallunterscheidung

Beobachtung. Vorkommen von Reads in einer Referenzkompression können auf zwei Weisen auftreten:

1. Der Read liegt vollständig innerhalb eines REs
2. Der Read überdeckt mindestens ein Fehlerzeichen eines REs

Suchalgorithmus arbeitet in zwei Phasen:

Fallunterscheidung

Beobachtung. Vorkommen von Reads in einer Referenzkompression können auf zwei Weisen auftreten:

1. Der Read liegt vollständig innerhalb eines REs
2. Der Read überdeckt mindestens ein Fehlerzeichen eines REs

Suchalgorithmus arbeitet in zwei Phasen:

1. Suche nur innerhalb REs

Fallunterscheidung

Beobachtung. Vorkommen von Reads in einer Referenzkompression können auf zwei Weisen auftreten:

1. Der Read liegt vollständig innerhalb eines REs
2. Der Read überdeckt mindestens ein Fehlerzeichen eines REs

Suchalgorithmus arbeitet in zwei Phasen:

1. Suche nur innerhalb REs
2. Durchsuche Umgebung der Fehlerzeichen

Suche innerhalb der Referenzeinträge (1/3)

Klar Einfachste Möglichkeit: Durchsuche Referenzgenom

Suche innerhalb der Referenzeinträge (1/3)

Klar Einfachste Möglichkeit: Durchsuche Referenzgenom

denn Alle REs per Definition im Referenzgenom enthalten!

Suche innerhalb der Referenzeinträge (1/3)

Klar Einfachste Möglichkeit: Durchsuche Referenzgenom

denn Alle REs per Definition im Referenzgenom enthalten!

Zwei Aufgaben:

Suche innerhalb der Referenzeinträge (1/3)

Klar Einfachste Möglichkeit: Durchsuche Referenzgenom

denn Alle REs per Definition im Referenzgenom enthalten!

Zwei Aufgaben:

1. Suche in Referenzgenom

Suche innerhalb der Referenzeinträge (1/3)

Klar Einfachste Möglichkeit: Durchsuche Referenzgenom

denn Alle REs per Definition im Referenzgenom enthalten!

Zwei Aufgaben:

1. Suche in Referenzgenom
2. Identifikation: Vorkommen in Referenz $\rightarrow$ Position in anderen Genomen

Suche innerhalb der Referenzeinträge (1/3)

Klar Einfachste Möglichkeit: Durchsuche Referenzgenom

denn Alle REs per Definition im Referenzgenom enthalten!

Zwei Aufgaben:

1. Suche in Referenzgenom
2. Identifikation: Vorkommen in Referenz $\rightarrow$ Position in anderen Genomen

Zu 2.)

Suche innerhalb der Referenzeinträge (1/3)

Klar Einfachste Möglichkeit: Durchsuche Referenzgenom

denn Alle REs per Definition im Referenzgenom enthalten!

Zwei Aufgaben:

1. Suche in Referenzgenom
2. Identifikation: Vorkommen in Referenz $\rightarrow$ Position in anderen Genomen

Zu 2.)

- Für Treffer in Referenzgenomen müssen alle überdeckenden REs bestimmt werden (z.B. durch vorige Sortierung nach Start/Ende)

Suche innerhalb der Referenzeinträge (1/3)

Klar Einfachste Möglichkeit: Durchsuche Referenzgenom

denn Alle REs per Definition im Referenzgenom enthalten!

Zwei Aufgaben:

1. Suche in Referenzgenom
2. Identifikation: Vorkommen in Referenz $\rightarrow$ Position in anderen Genomen

Zu 2.)

- Für Treffer in Referenzgenomen müssen alle überdeckenden REs bestimmt werden (z.B. durch vorige Sortierung nach Start/Ende)
- Autoren verwendeten Hashtabellen, die REs auf Startposition in Genomen abbildet

Suche innerhalb der Referenzeinträge (2/3)

Zu 1.)

Suche k-approximative Vorkommen eines Reads q in Referenzgenom *ref*

Suche innerhalb der Referenzeinträge (2/3)

Zu 1.)

Suche k-approximative Vorkommen eines Reads q in Referenzgenom *ref*

Grundidee des „seed-and-extend“:

Suche innerhalb der Referenzeinträge (2/3)

Zu 1.)

Suche k-approximative Vorkommen eines Reads q in Referenzgenom *ref*

Grundidee des „seed-and-extend“:

- Zerlege q gleichmäßig in $k+1$ Teilstrings (=Blöcke)

Suche innerhalb der Referenzeinträge (2/3)

Zu 1.)

Suche k-approximative Vorkommen eines Reads q in Referenzgenom *ref*

Grundidee des „seed-and-extend“:

- Zerlege q gleichmäßig in $k+1$ Teilstrings (=Blöcke)
- Jedes k-approximative Vorkommen enthält einen fehlerfreien Block

Suche innerhalb der Referenzeinträge (2/3)

Zu 1.)

Suche k-approximative Vorkommen eines Reads q in Referenzgenom *ref*

Grundidee des „seed-and-extend“:

- Zerlege q gleichmäßig in $k+1$ Teilstrings (=Blöcke)
- Jedes k-approximative Vorkommen enthält einen fehlerfreien Block
- Suche nach exakten Vorkommen der Blöcke in *ref*

Suche innerhalb der Referenzeinträge (2/3)

Zu 1.)

Suche k-approximative Vorkommen eines Reads q in Referenzgenom *ref*

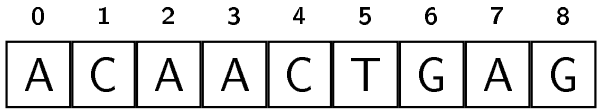
Grundidee des „seed-and-extend“:

- Zerlege q gleichmäßig in $k+1$ Teilstrings (=Blöcke)
- Jedes k-approximative Vorkommen enthält einen fehlerfreien Block
- Suche nach exakten Vorkommen der Blöcke in *ref*
- Erweitere alle Treffer auf ganzen String q

Suche innerhalb der Referenzeinträge (3/3)

Beispiel.

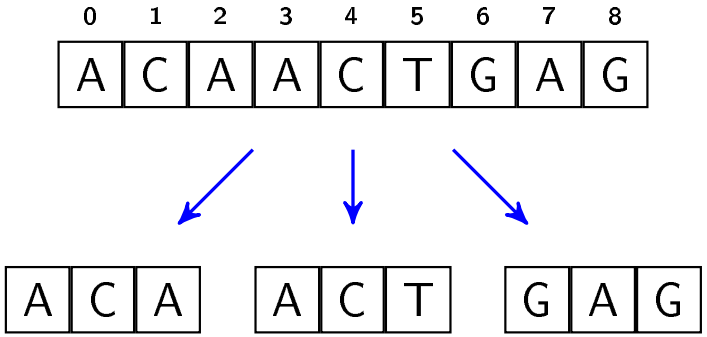
$k = 2, q = ACAACTGAG$



Suche innerhalb der Referenzeinträge (3/3)

Beispiel.

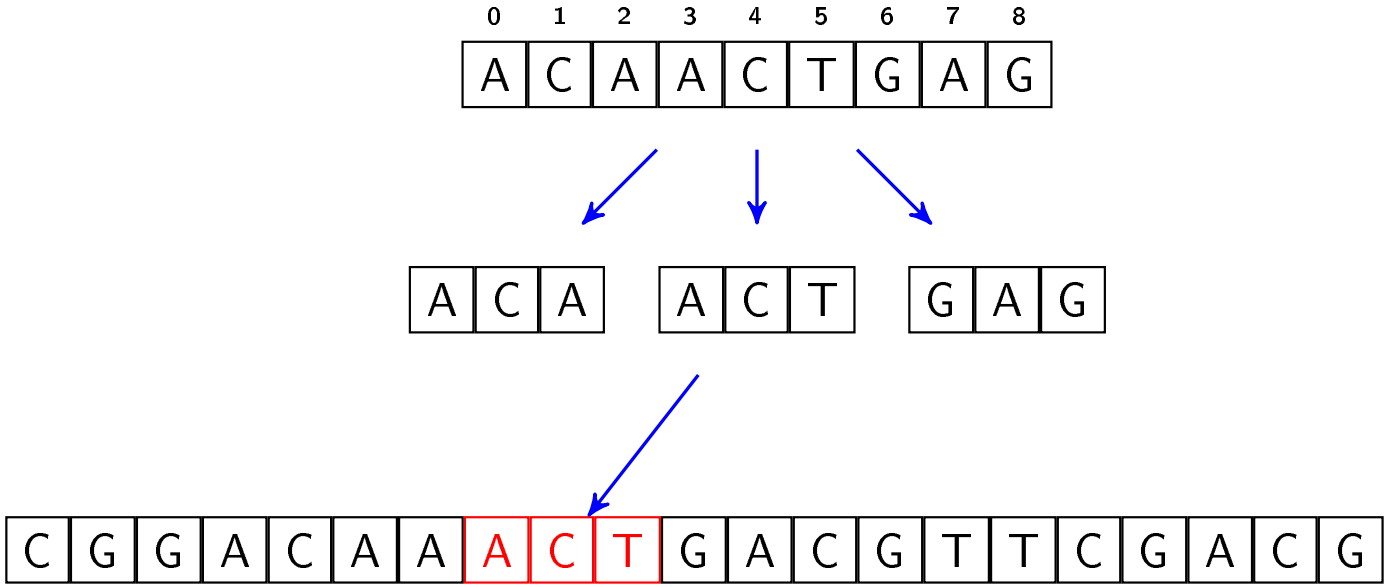
$k = 2, q = ACAACTGAG$



Suche innerhalb der Referenzeinträge (3/3)

Beispiel.

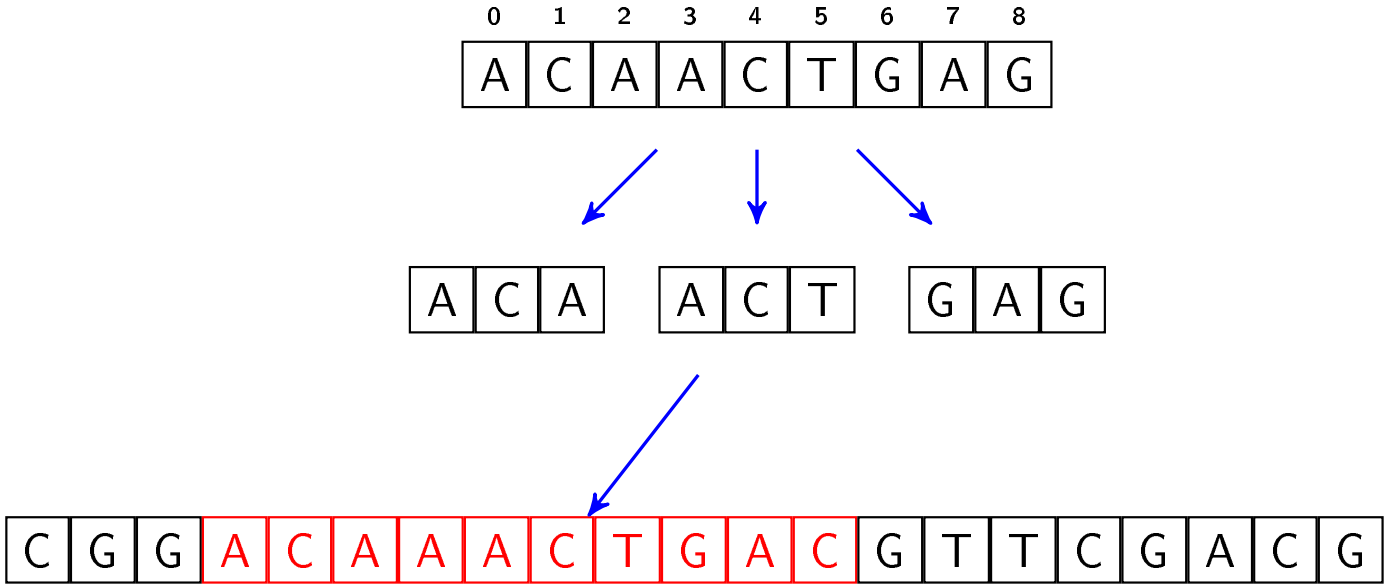
$k = 2, q = ACAACTGAG$



Suche innerhalb der Referenzeinträge (3/3)

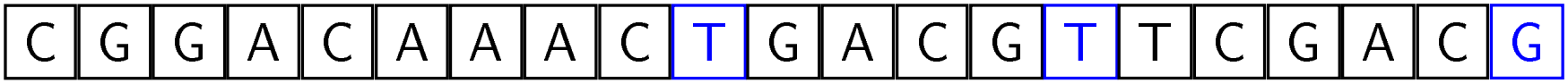
Beispiel.

$k = 2, q = ACAACTGAG$

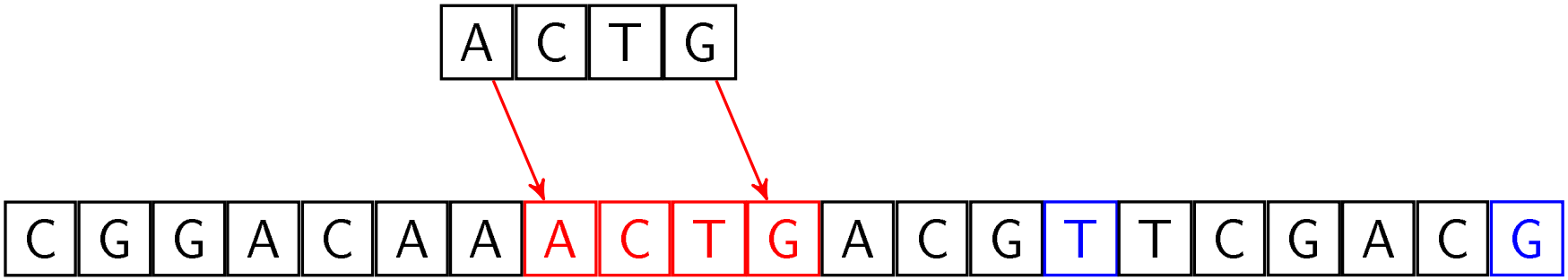


Überlappende Suche (1/2)

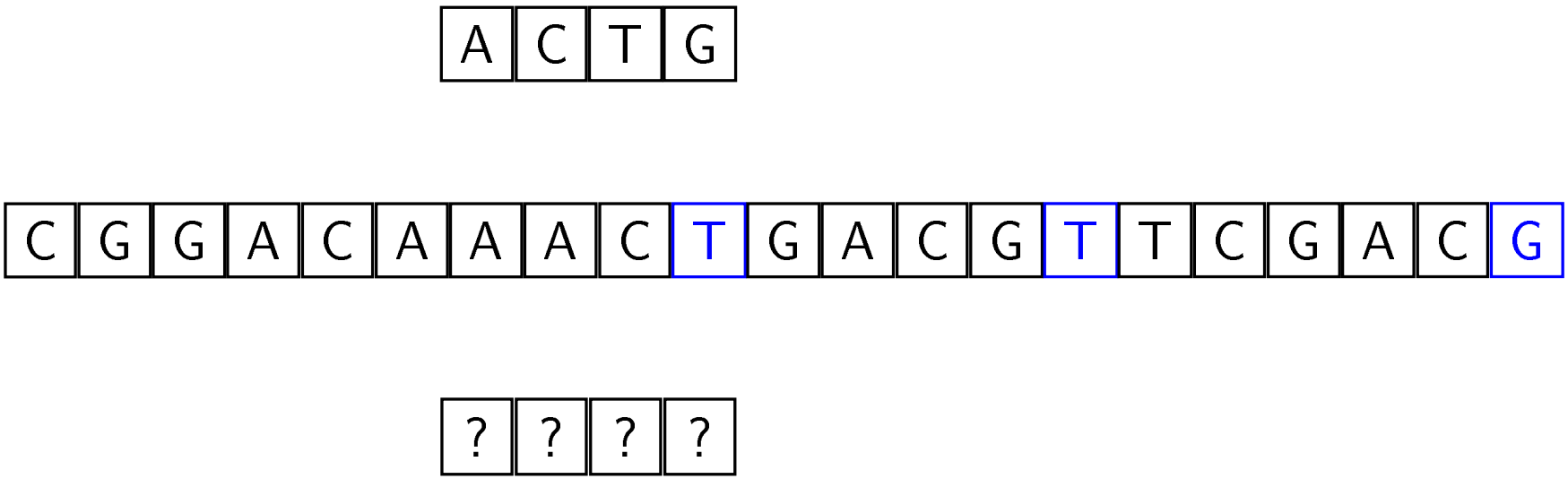
Überlappende Suche (1/2)



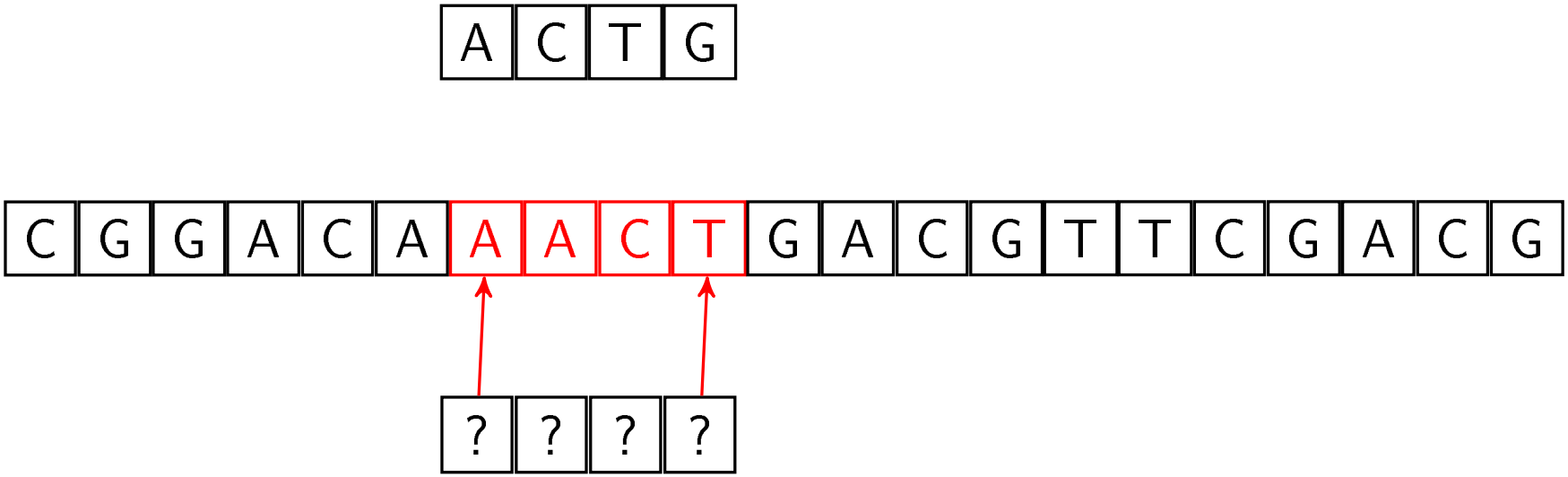
Überlappende Suche (1/2)



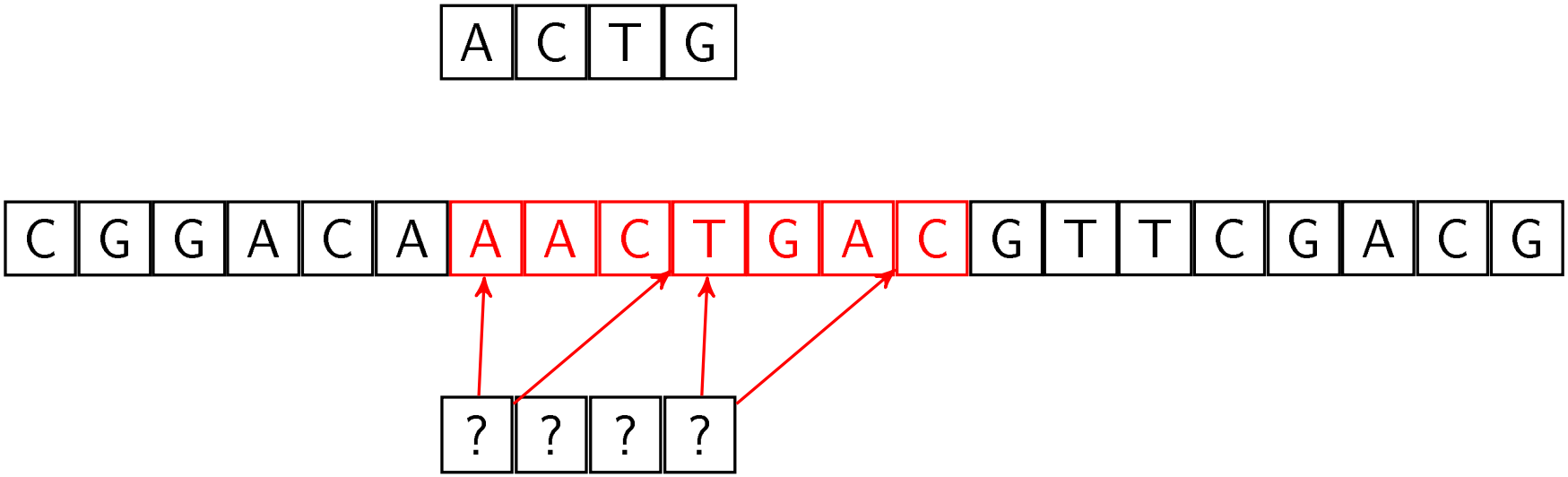
Überlappende Suche (1/2)



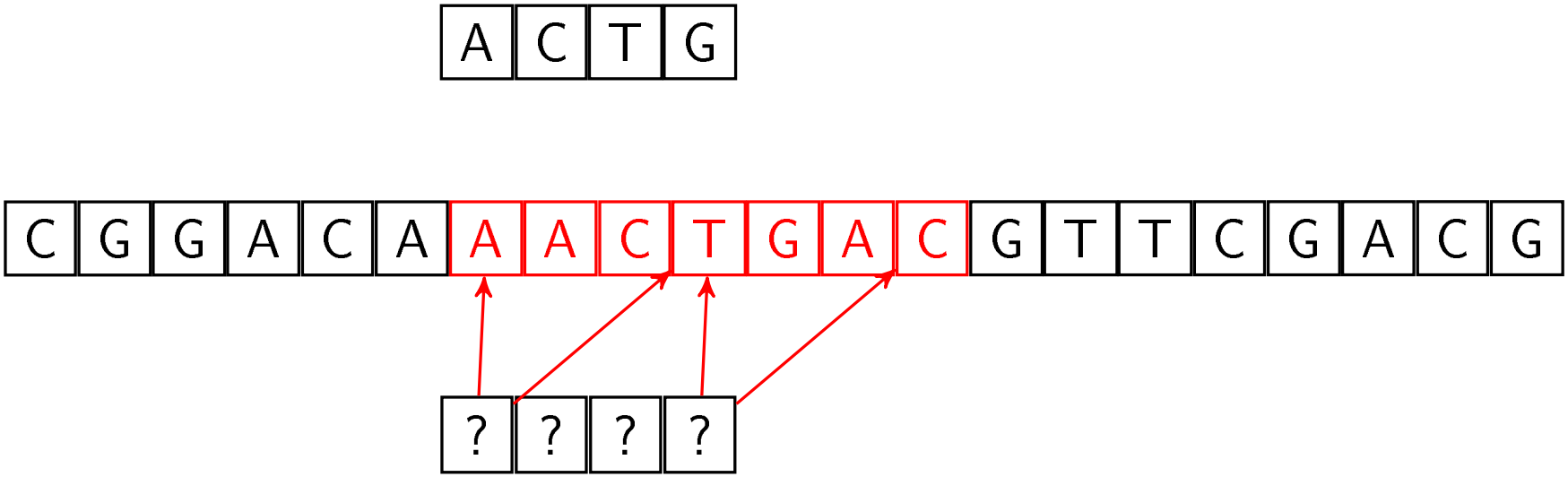
Überlappende Suche (1/2)



Überlappende Suche (1/2)

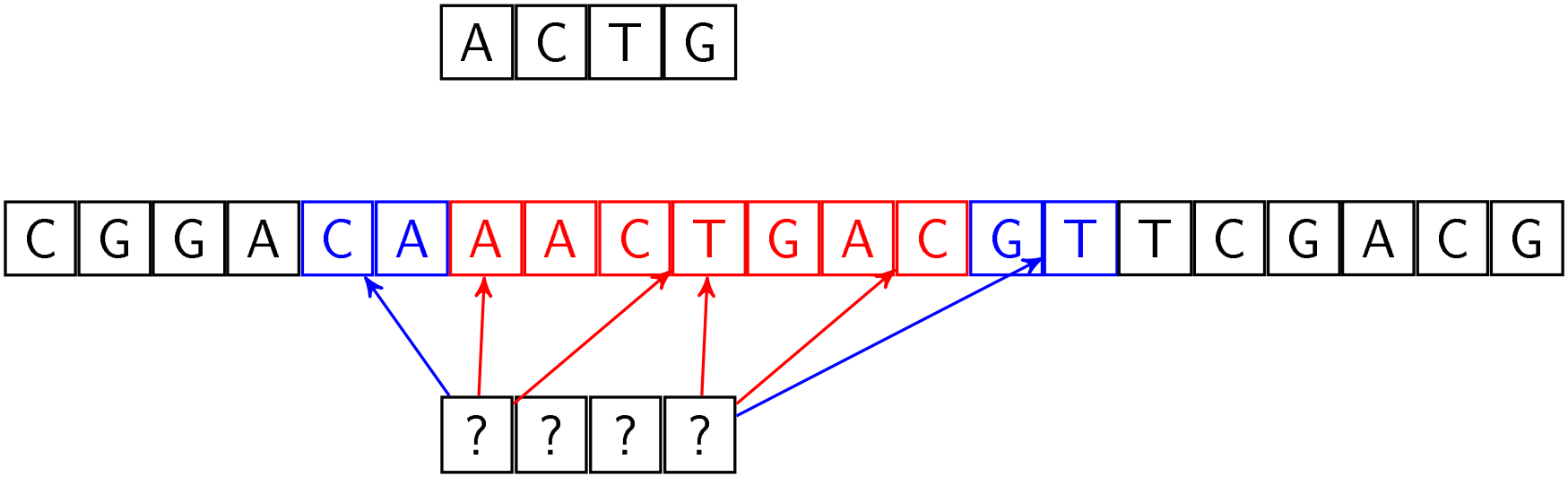


Überlappende Suche (1/2)



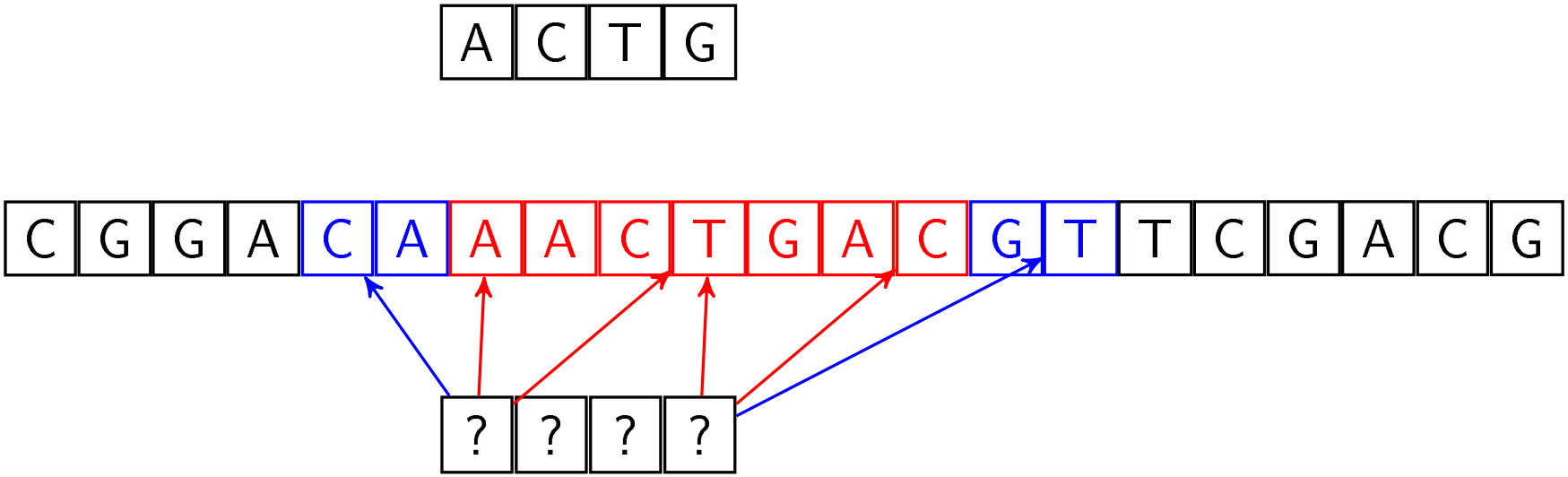
Zusätzlich $k = 2$ Fehler erlaubt

Überlappende Suche (1/2)



Zusätzlich $k = 2$ Fehler erlaubt

Überlappende Suche (1/2)



Zusätzlich $k = 2$ Fehler erlaubt

Also Alle $|q| + k - 1$ Zeichen links und rechts relevant für Suche!

Überlappende Suche (2/2)

Definition. Seien RC eine Referenzkompression für einen String s bezüglich ref und $RE_i \in RC$ ein Referenzeintrag. Wir definieren den *Überlappungsbereich* von RE_i als:

Überlappende Suche (2/2)

Definition. Seien RC eine Referenzkompression für einen String s bezüglich ref und $RE_i \in RC$ ein Referenzeintrag. Wir definieren den *Überlappungsbereich* von RE_i als:

$$\text{ovl}_{RC}^{ref}(RE_i) := s(pos(F_i) - Q_{max} - k_{max} + 1, \\ 2(Q_{max} + k_{max}) - 1)$$

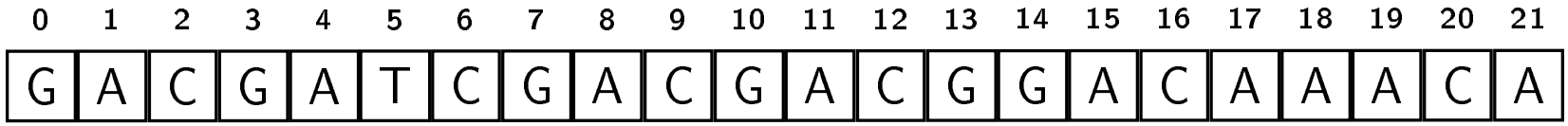
Überlappende Suche (2/2)

Definition. Seien RC eine Referenzkompression für einen String s bezüglich ref und $RE_i \in RC$ ein Referenzeintrag. Wir definieren den *Überlappungsbereich* von RE_i als:

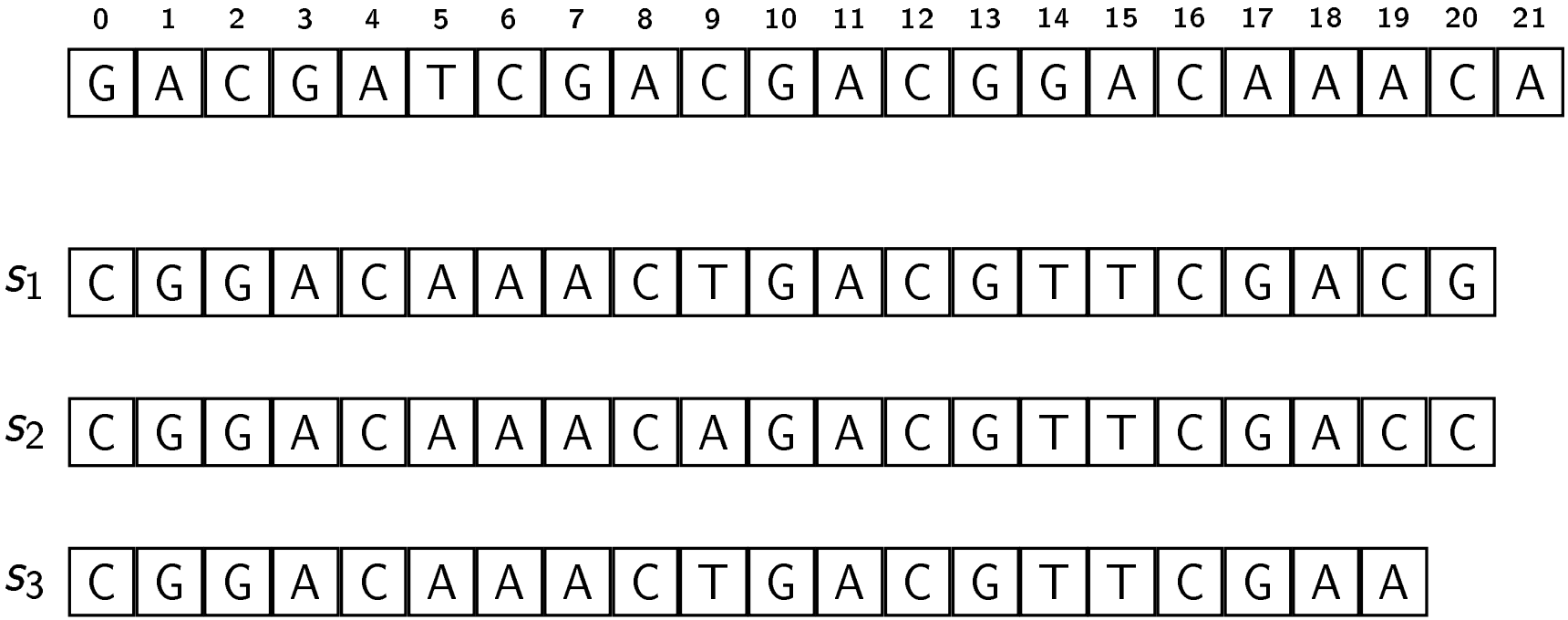
$$\text{ovl}_{RC}^{ref}(RE_i) := s(pos(F_i) - Q_{max} - k_{max} + 1, \\ 2(Q_{max} + k_{max}) - 1)$$

mit Q_{max} := maximale Readlänge
und k_{max} := maximale Fehlertoleranz
und $pos(F_i)$:= Position des i-ten Fehlerzeichens in s

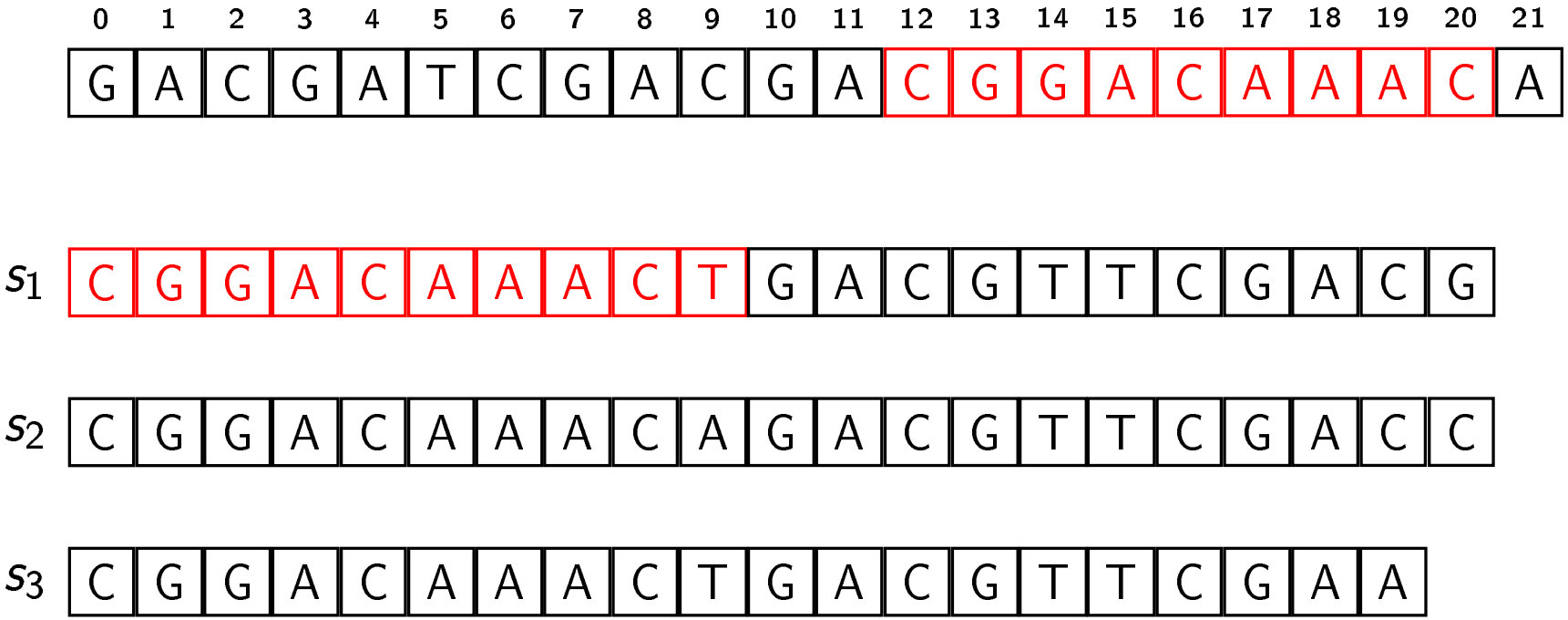
Gesamtalgorithmus (Beispiel aus [Wandelt et al., 2013])



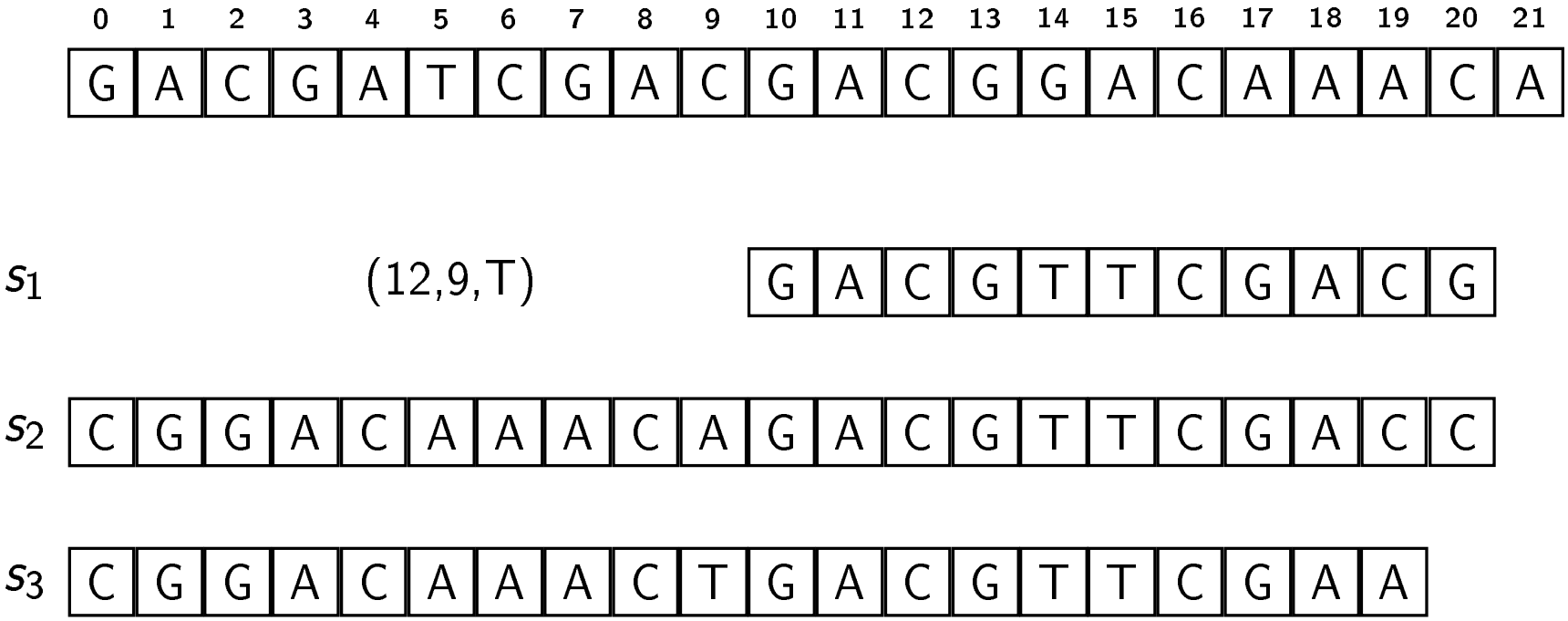
Gesamtalgorithmus (Beispiel aus [Wandelt et al., 2013])



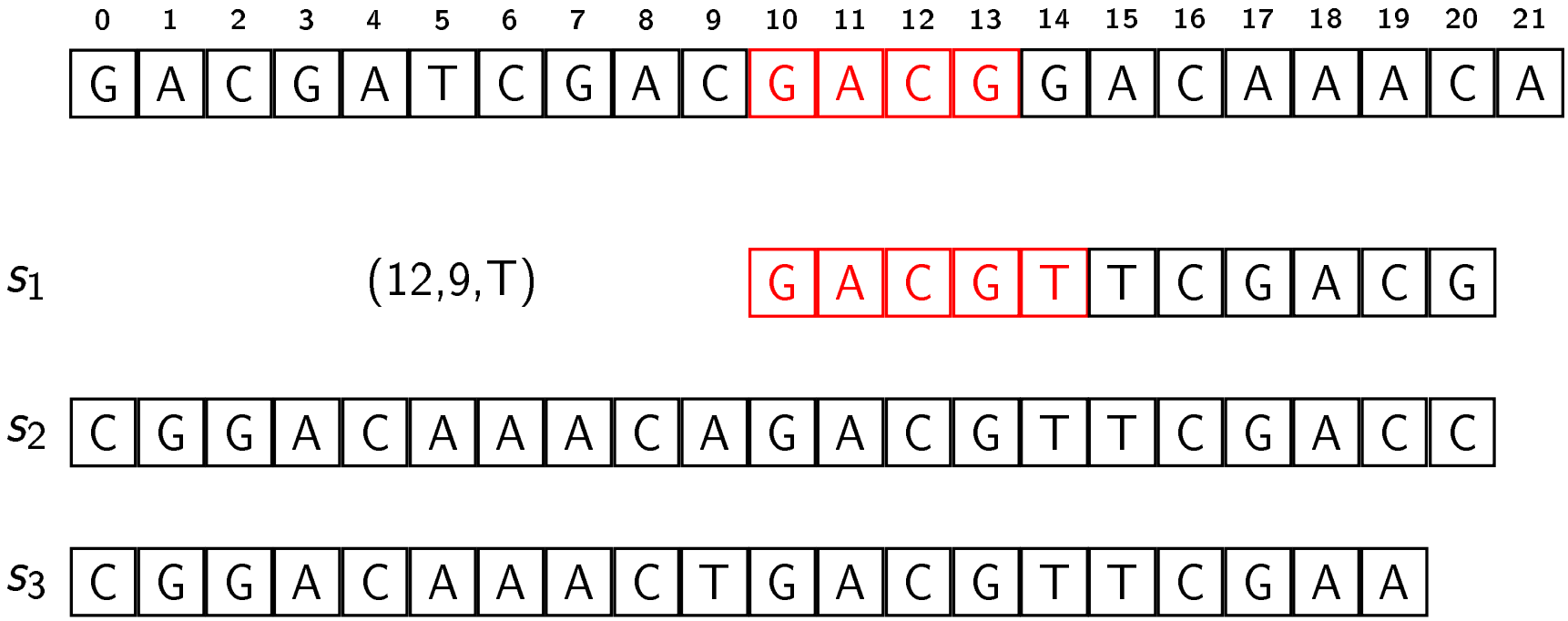
Gesamtalgorithmus (Beispiel aus [Wandelt et al., 2013])



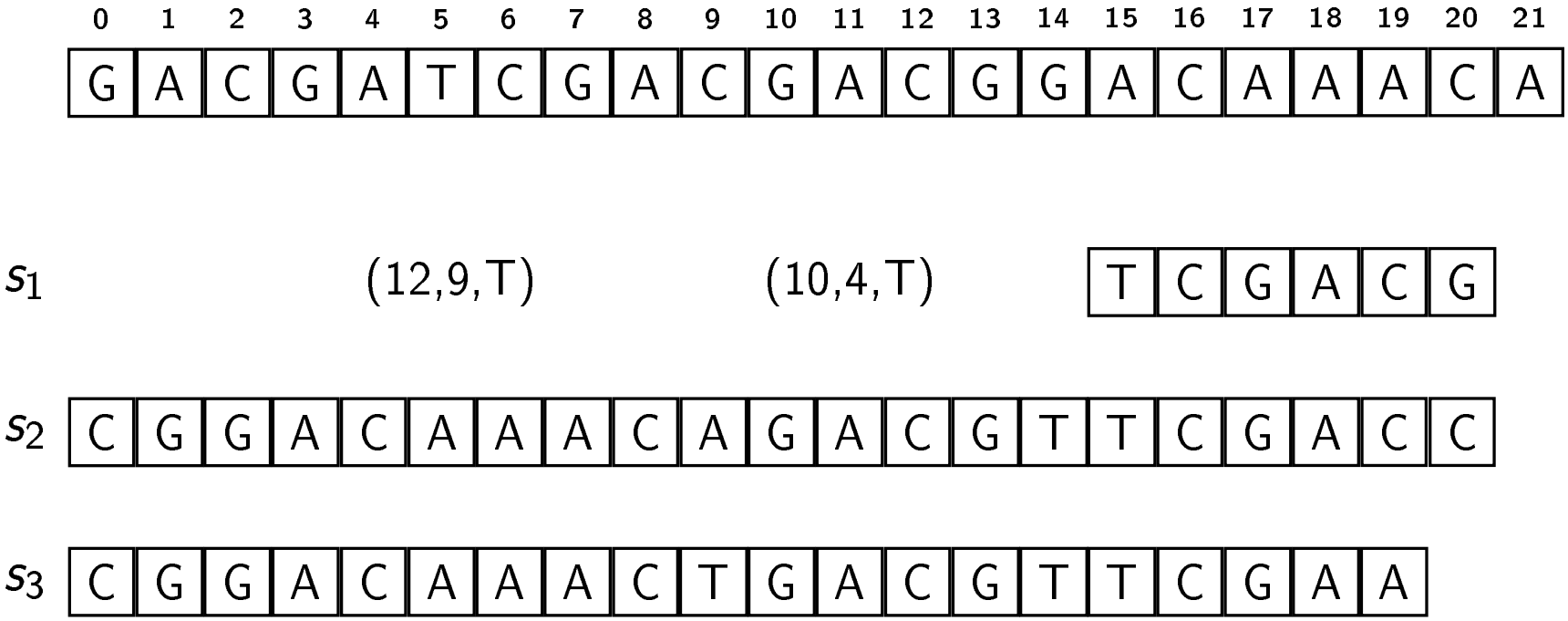
Gesamtalgorithmus (Beispiel aus [Wandelt et al., 2013])



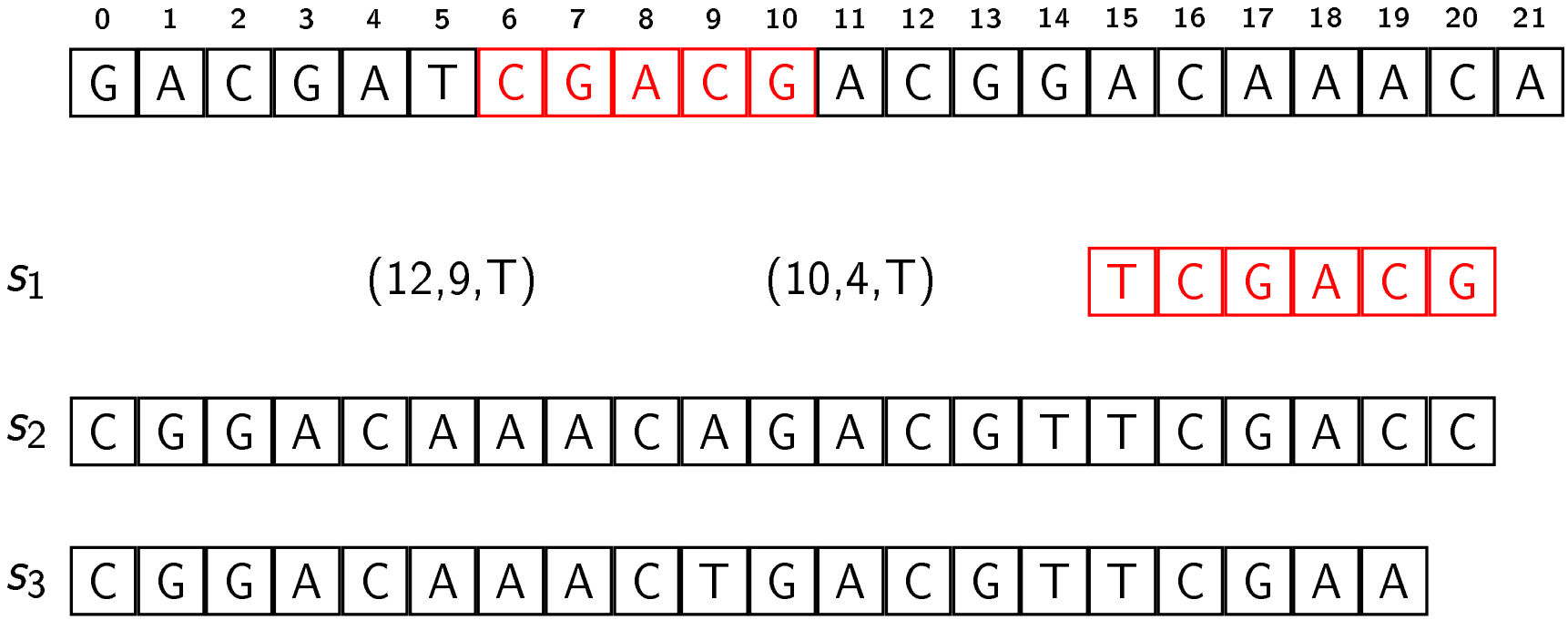
Gesamtalgorithmus (Beispiel aus [Wandelt et al., 2013])



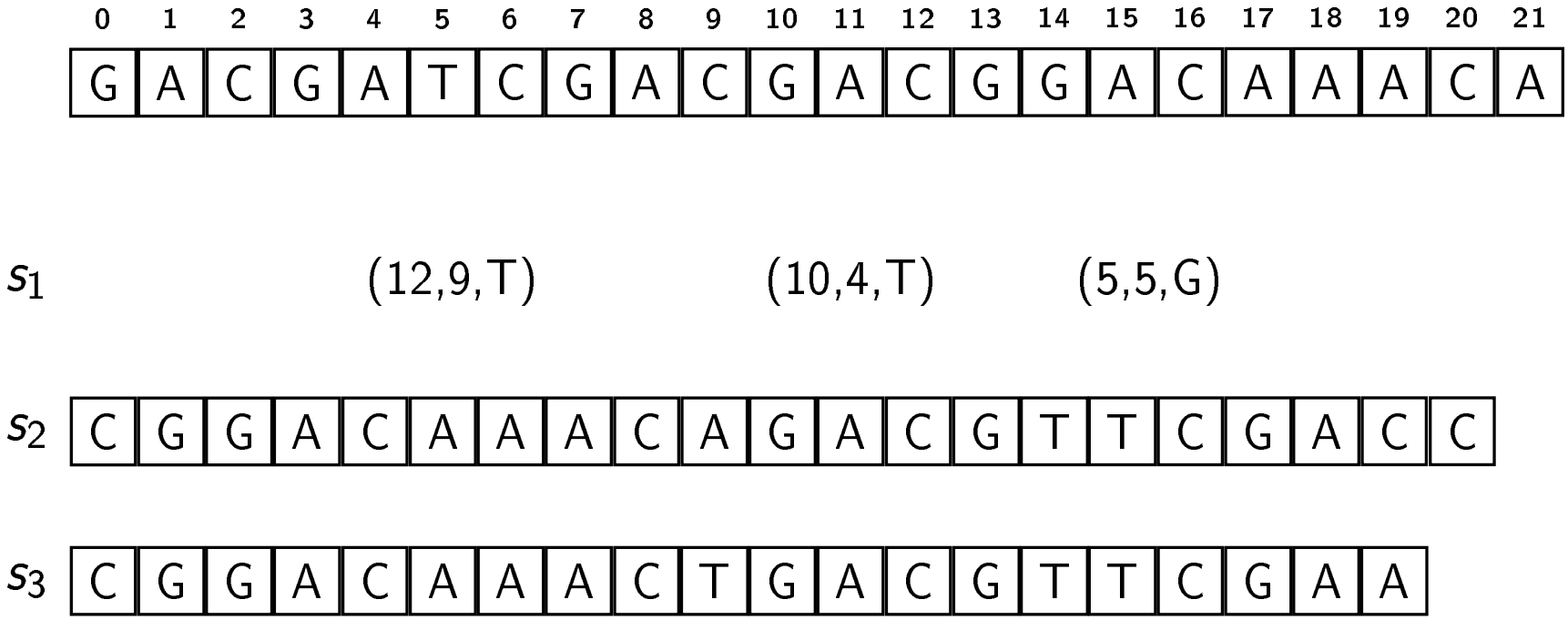
Gesamtalgorithmus (Beispiel aus [Wandelt et al., 2013])



Gesamtalgorithmus (Beispiel aus [Wandelt et al., 2013])



Gesamtalgorithmus (Beispiel aus [Wandelt et al., 2013])



Gesamtalgorithmus (Beispiel aus [Wandelt et al., 2013])

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
G	A	C	G	A	T	C	G	A	C	G	A	C	G	G	A	C	A	A	A	C	A

s_1	(12,9,T)	(10,4,T)	(5,5,G)
s_2	(12,9,A)	(10,4,T)	(5,5,C)
s_3	(12,9,T)	(10,4,T)	(5,4,A)

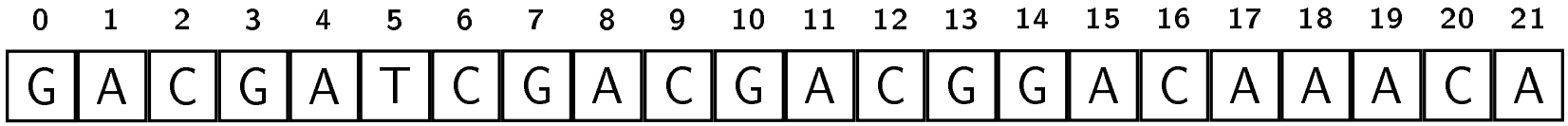
Gesamtalgorithmus (Beispiel aus [Wandelt et al., 2013])

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
G	A	C	G	A	T	C	G	A	C	G	A	C	G	G	A	C	A	A	A	C	A

s_1	(12,9,T)	(10,4,T)	(5,5,G)
s_2	(12,9,A)	(10,4,T)	(5,5,C)
s_3	(12,9,T)	(10,4,T)	(5,4,A)

$$\text{search}(ref)_{AA}^{0} = \{(17, AA), (18, AA)\}$$

Gesamtalgorithmus (Beispiel aus [Wandelt et al., 2013])



s_1	(12,9,T)	(10,4,T)	(5,5,G)
s_2	(12,9,A)	(10,4,T)	(5,5,C)
s_3	(12,9,T)	(10,4,T)	(5,4,A)

$$\text{search}(ref)^0_{AA} = \{(17, AA), (18, AA)\}$$

$$\text{DBsearch}(\{s_1, s_2, s_3\})^0_{AA} = \{(1, \{(5, AA)$$

Gesamtalgorithmus (Beispiel aus [Wandelt et al., 2013])

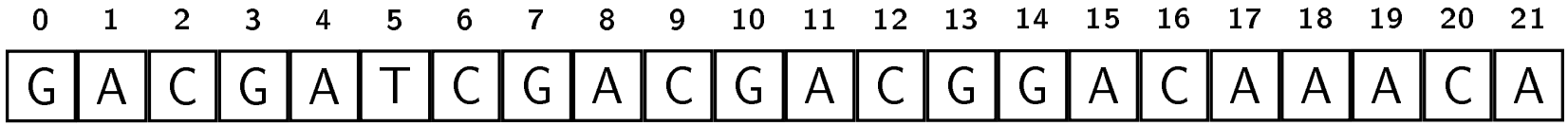
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
G	A	C	G	A	T	C	G	A	C	G	A	C	G	G	A	C	A	A	A	C	A

s_1	(12,9,T)	(10,4,T)	(5,5,G)
s_2	(12,9,A)	(10,4,T)	(5,5,C)
s_3	(12,9,T)	(10,4,T)	(5,4,A)

$$\text{search}(ref)^0_{AA} = \{(17, AA), (18, AA)\}$$

$$\text{DBsearch}(\{s_1, s_2, s_3\})^0_{AA} = \{(1, \{(5, AA)\ , (6, AA)\})$$

Gesamtalgorithmus (Beispiel aus [Wandelt et al., 2013])

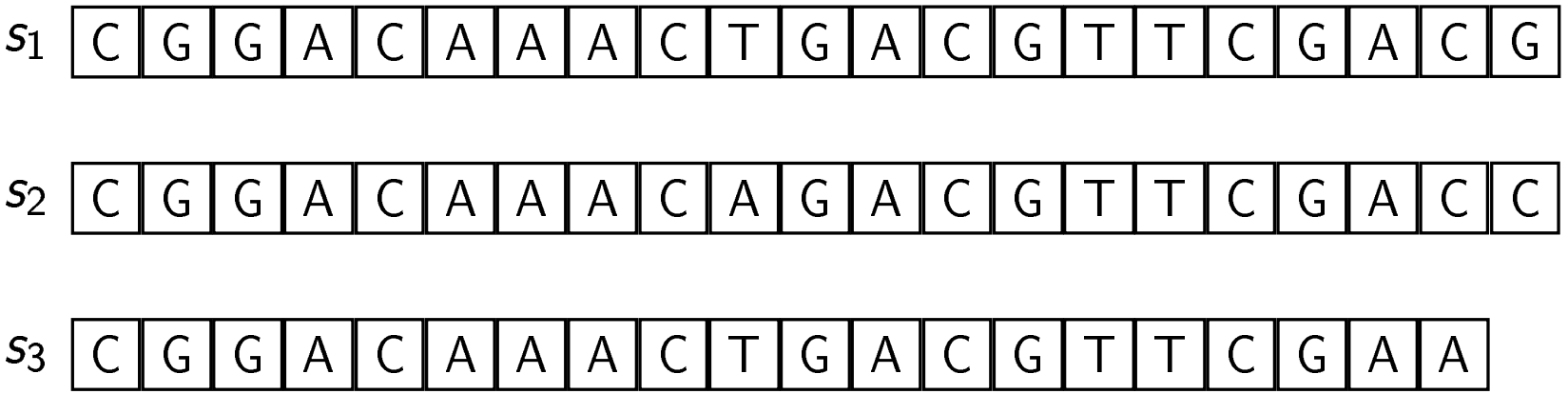


s_1	(12,9,T)	(10,4,T)	(5,5,G)
s_2	(12,9,A)	(10,4,T)	(5,5,C)
s_3	(12,9,T)	(10,4,T)	(5,4,A)

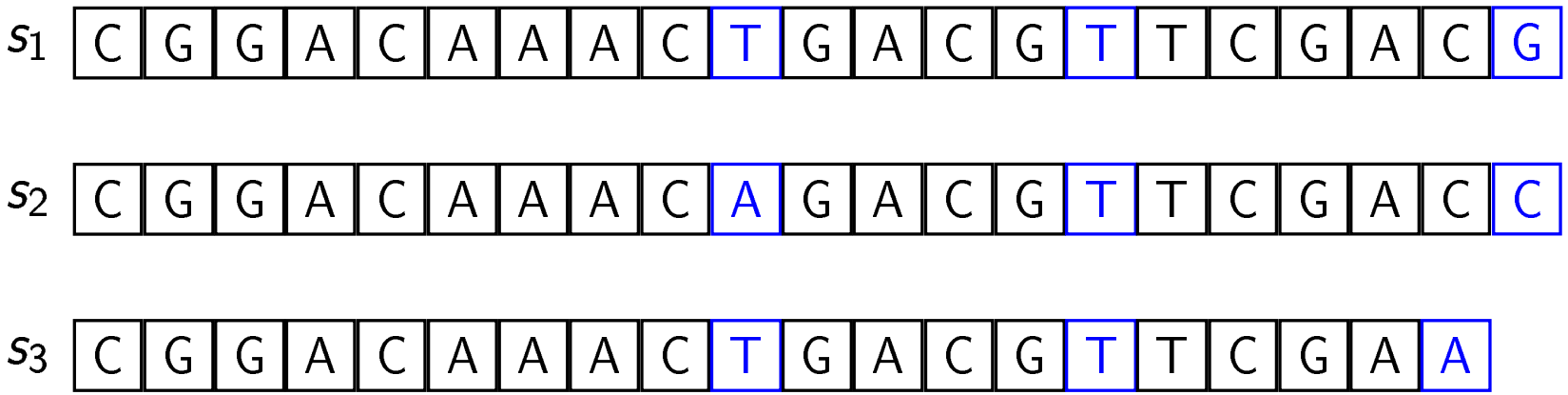
$$\text{search}(ref)^0_{AA} = \{(17, AA), (18, AA)\}$$

$$\text{DBsearch}(\{s_1, s_2, s_3\})^0_{AA} = \{(1, \{(5, AA) , (6, AA)\}) , (2, \{(5, AA), (6, AA)\}), (3, \{(5, AA), (6, AA)\})\}$$

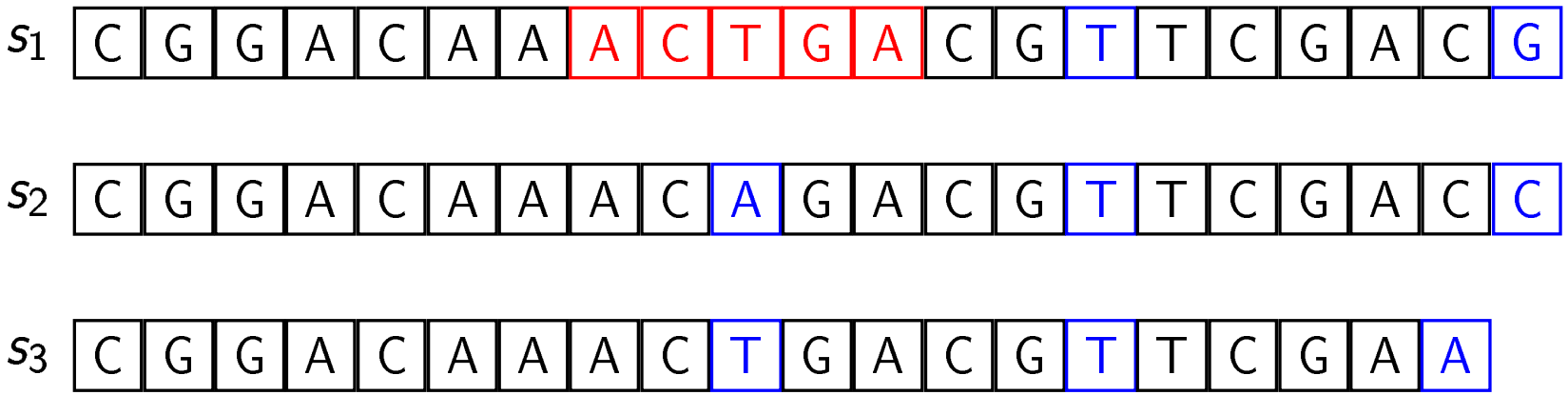
Gesamtalgorithmus (Forts.)



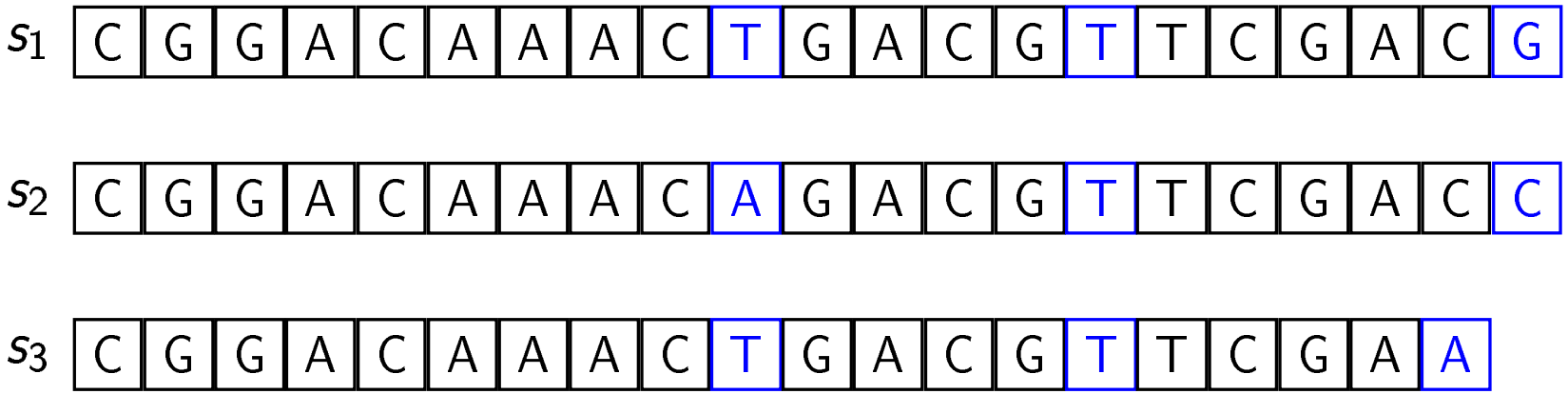
Gesamtalgorithmus (Forts.)



Gesamtalgorithmus (Forts.)

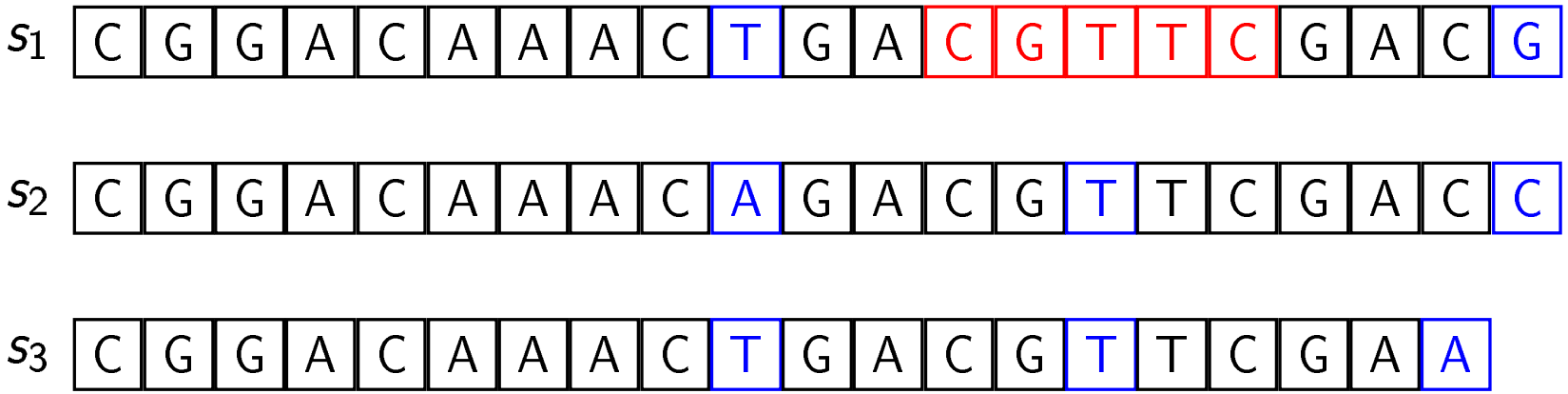


Gesamtalgorithmus (Forts.)



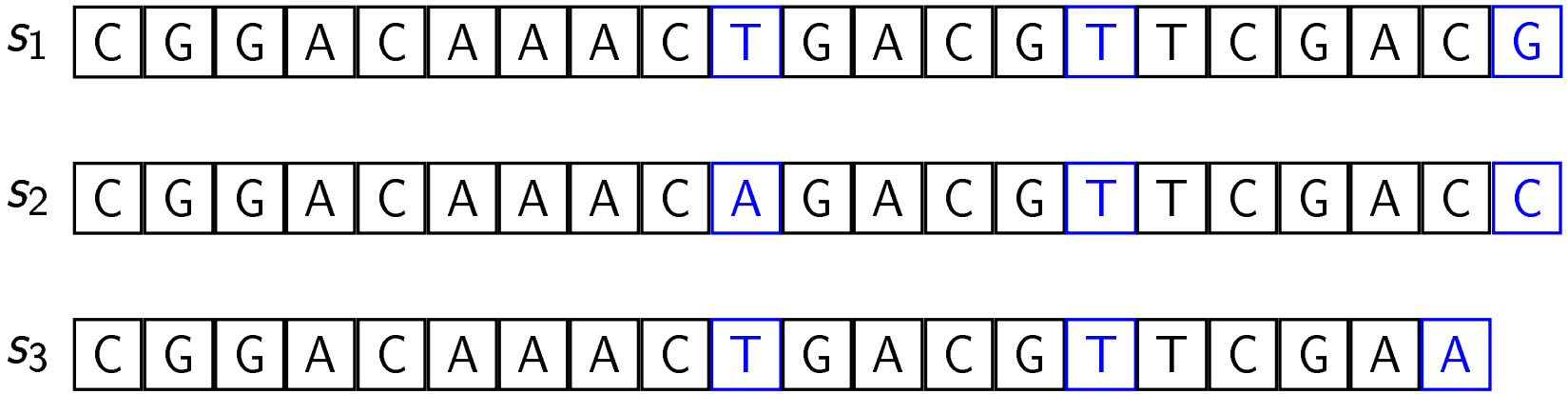
$\text{overlap}(ref, RC_1) = \{(7, ACTGA)$

Gesamtalgorithmus (Forts.)



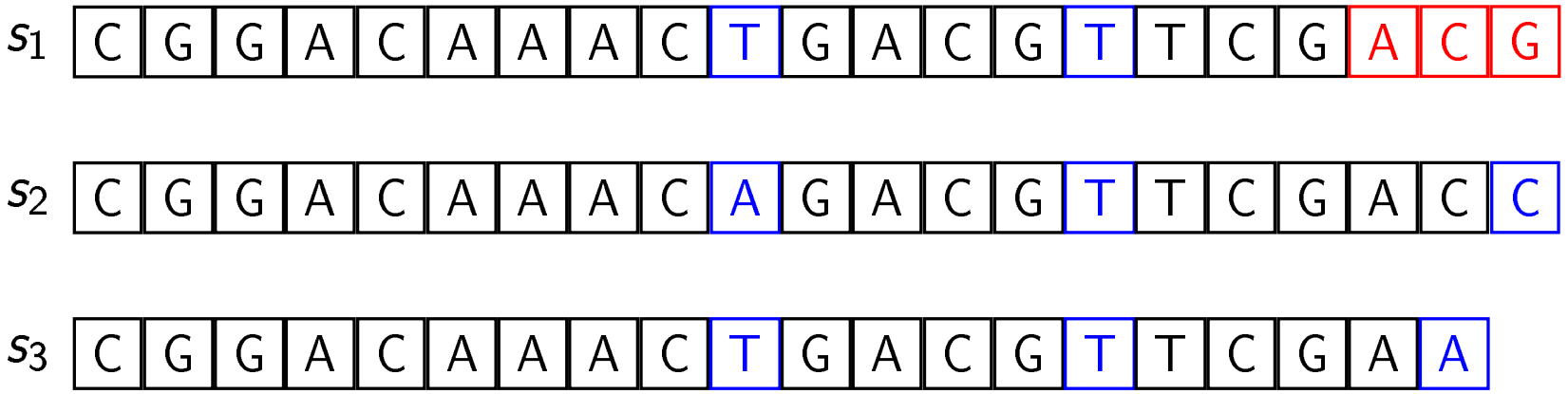
$\text{overlap}(ref, RC_1) = \{(7, ACTGA)$

Gesamtalgorithmus (Forts.)



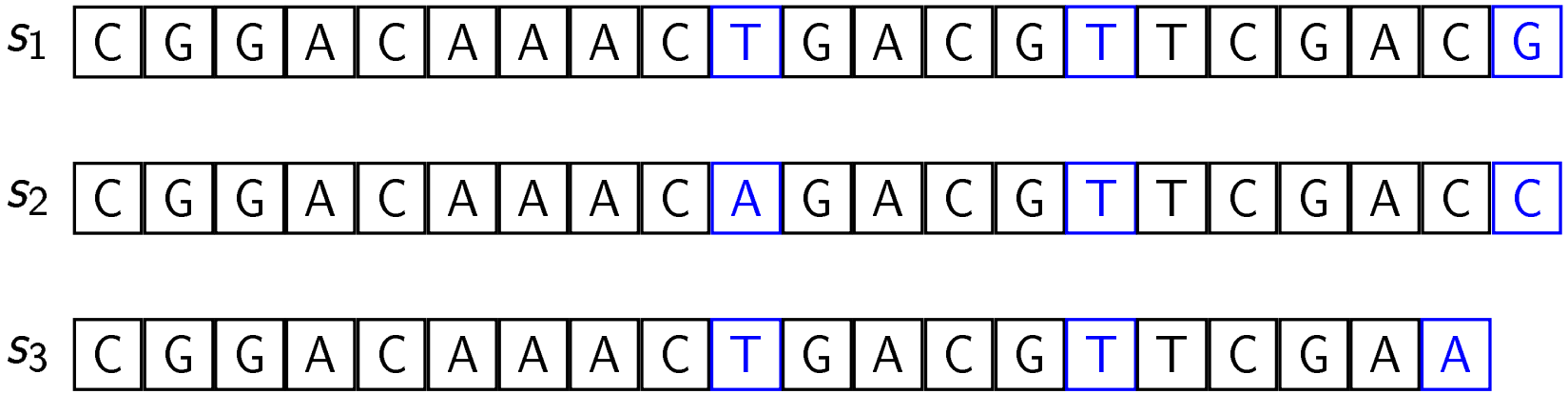
$$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC)$$

Gesamtalgorithmus (Forts.)



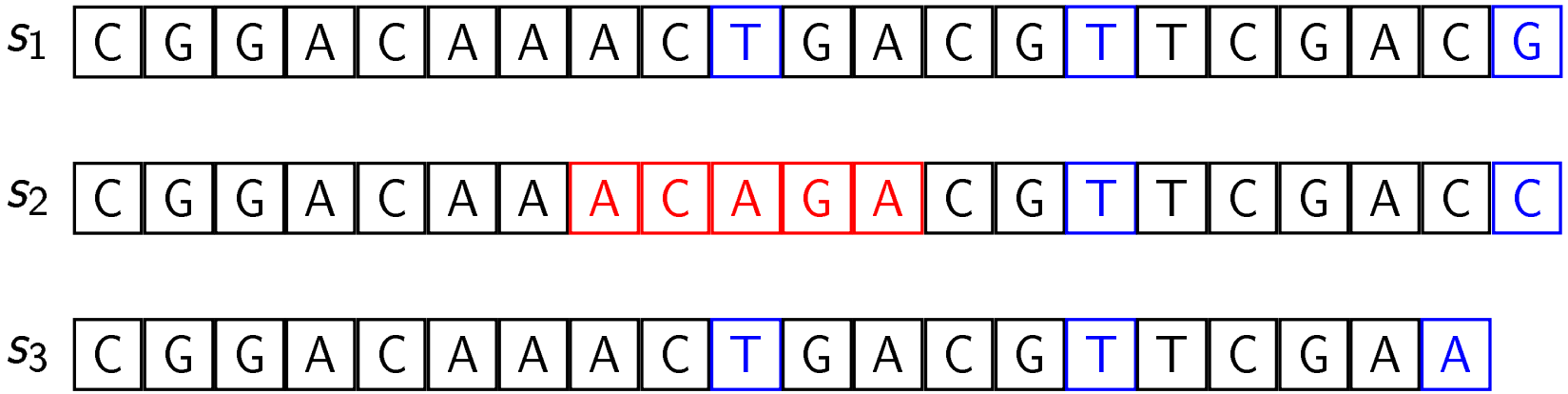
$$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC)$$

Gesamtalgorithmus (Forts.)



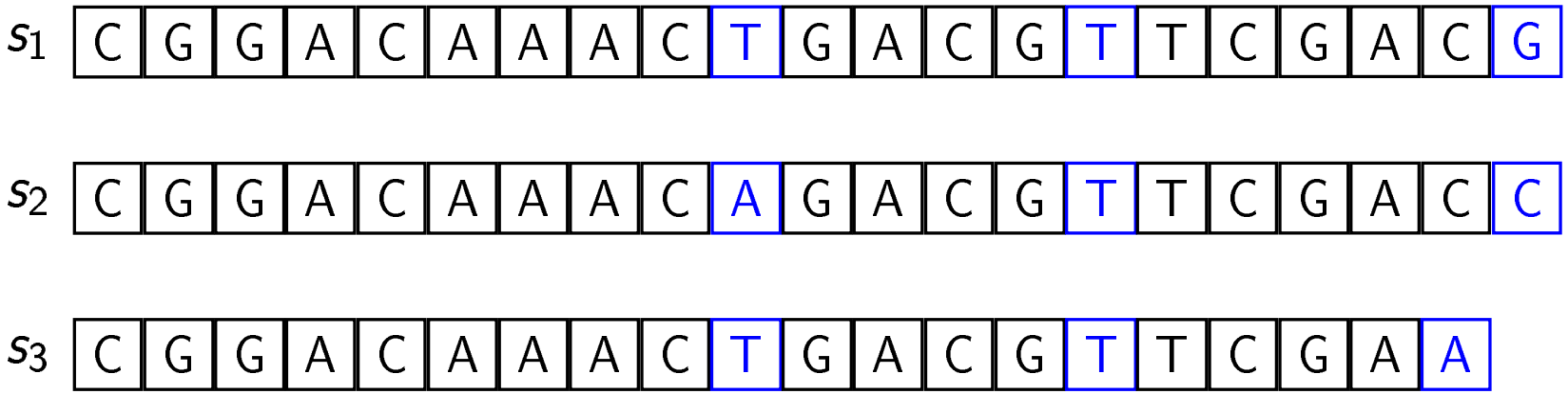
$$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC), (18, ACG)\}$$

Gesamtalgorithmus (Forts.)



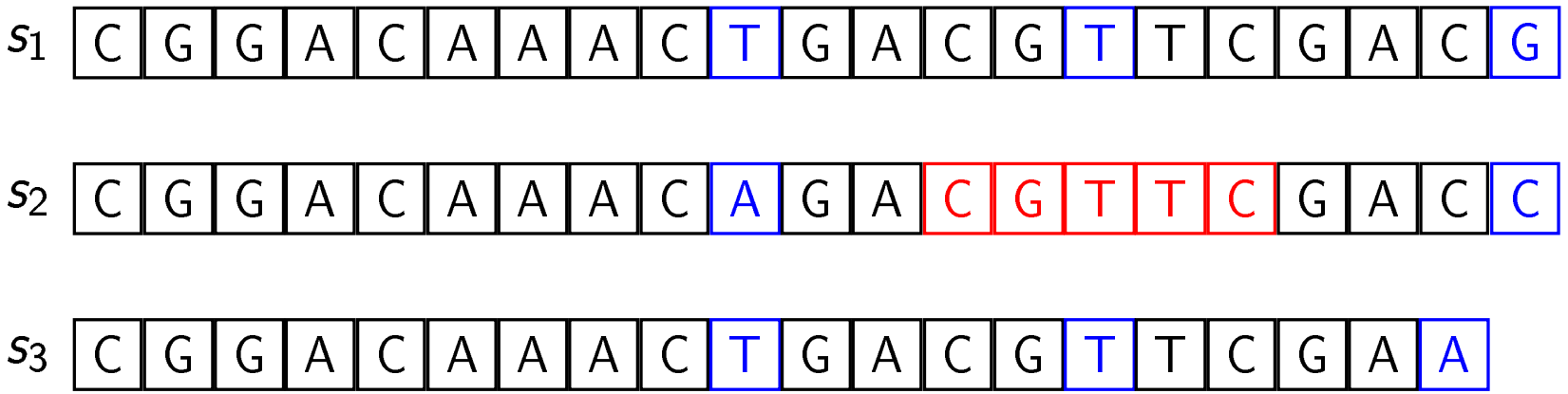
$$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC), (18, ACG)\}$$

Gesamtalgorithmus (Forts.)



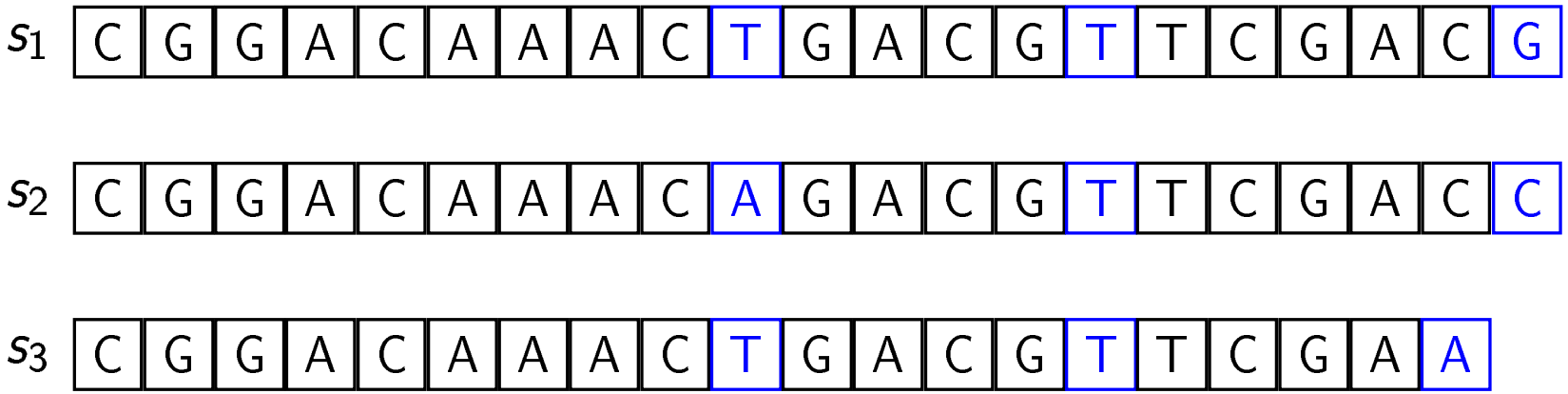
$$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC), (18, ACG)\}$$
$$\text{overlap}(ref, RC_2) = \{(7, ACAGA)\}$$

Gesamtalgorithmus (Forts.)



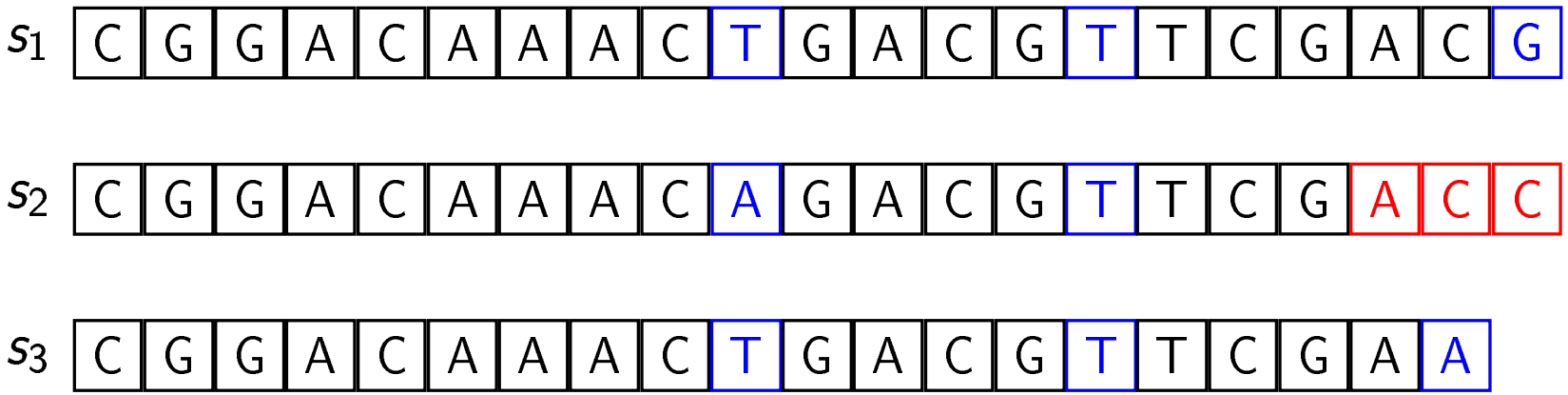
$$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC), (18, ACG)\}$$
$$\text{overlap}(ref, RC_2) = \{(7, ACAGA)\}$$

Gesamtalgorithmus (Forts.)



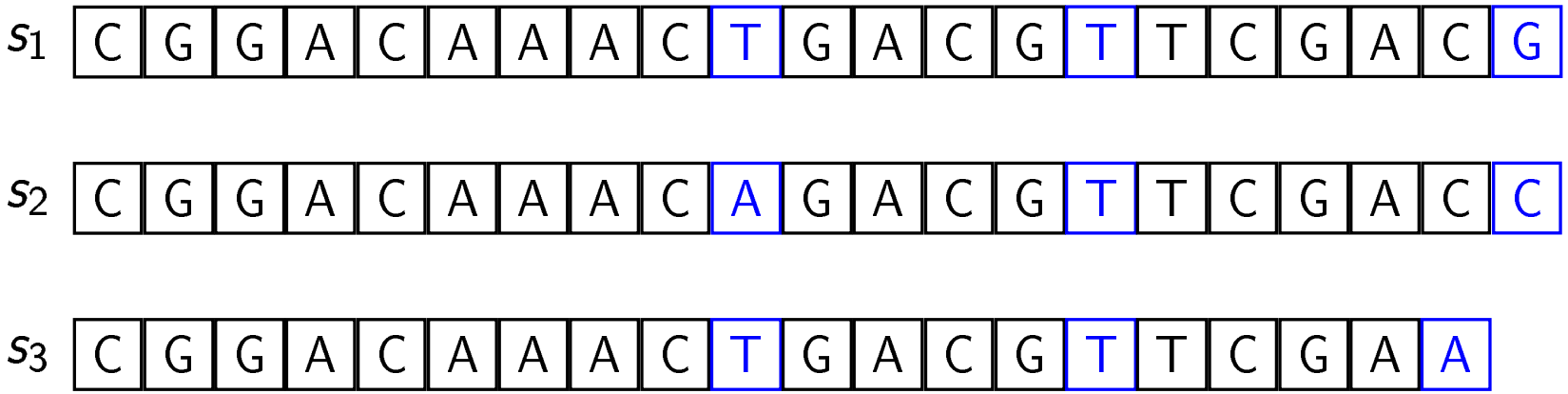
$$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC), (18, ACG)\}$$
$$\text{overlap}(ref, RC_2) = \{(7, ACAGA), (12, CGTTC)\}$$

Gesamtalgorithmus (Forts.)



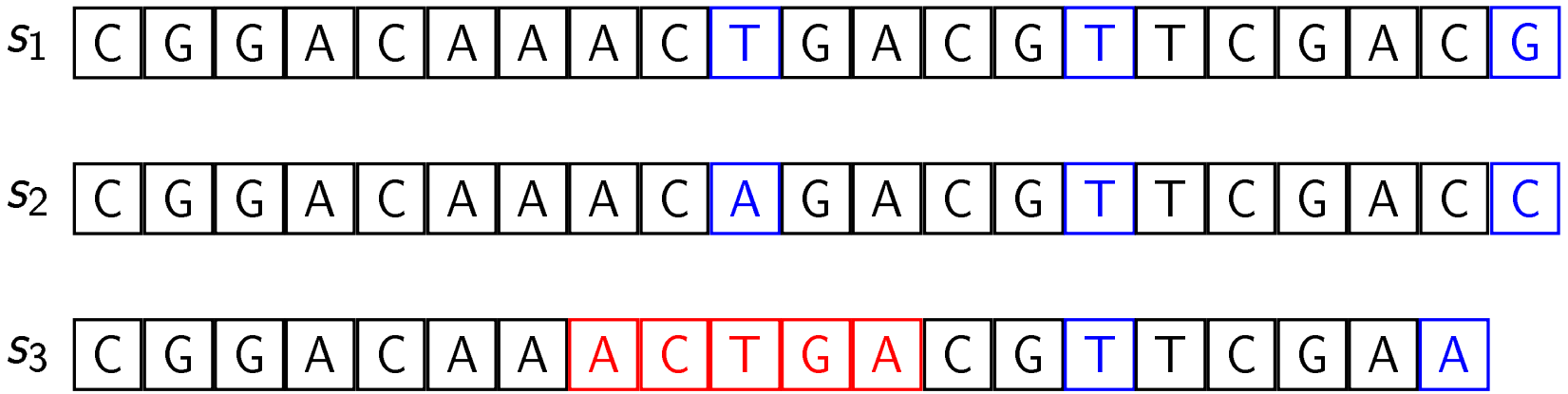
$$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC), (18, ACG)\}$$
$$\text{overlap}(ref, RC_2) = \{(7, ACAGA), (12, CGTTC)\}$$

Gesamtalgorithmus (Forts.)



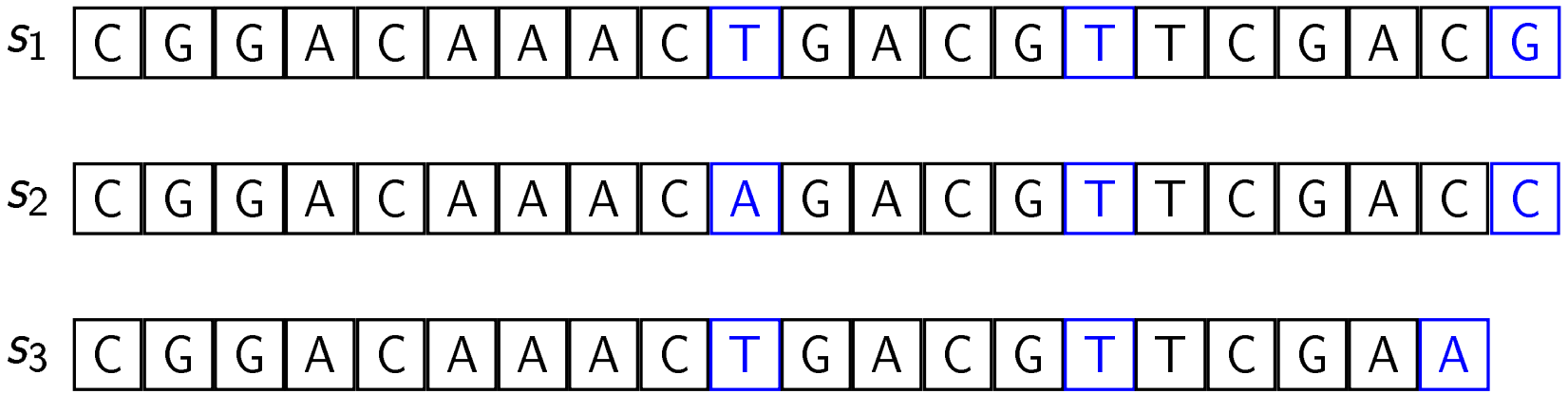
$$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC), (18, ACG)\}$$
$$\text{overlap}(ref, RC_2) = \{(7, ACAGA), (12, CGTTC), (18, ACC)\}$$

Gesamtalgorithmus (Forts.)



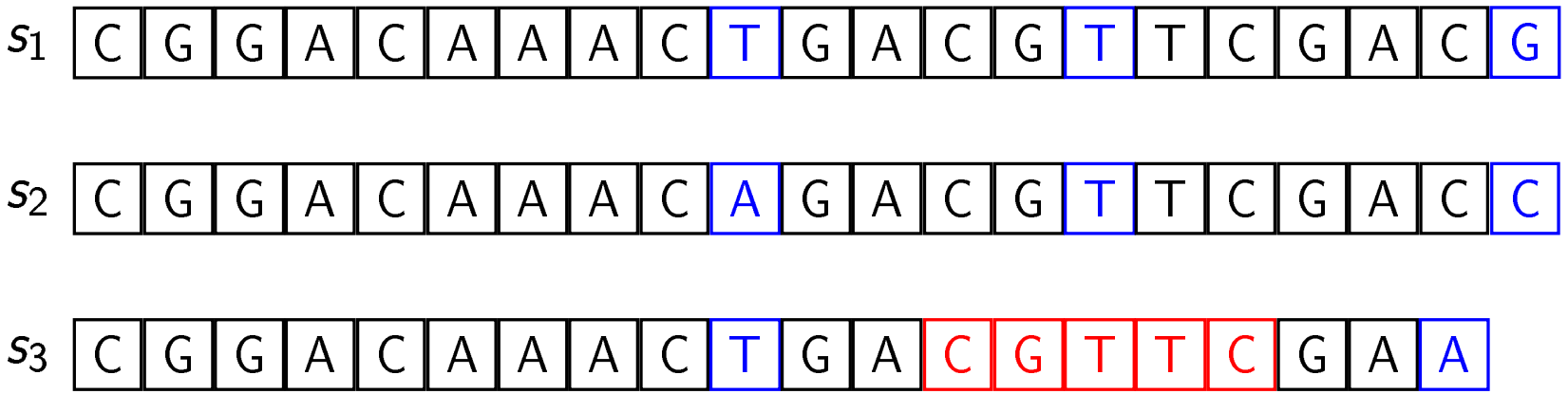
$$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC), (18, ACG)\}$$
$$\text{overlap}(ref, RC_2) = \{(7, ACAGA), (12, CGTTC), (18, ACC)\}$$

Gesamtalgorithmus (Forts.)



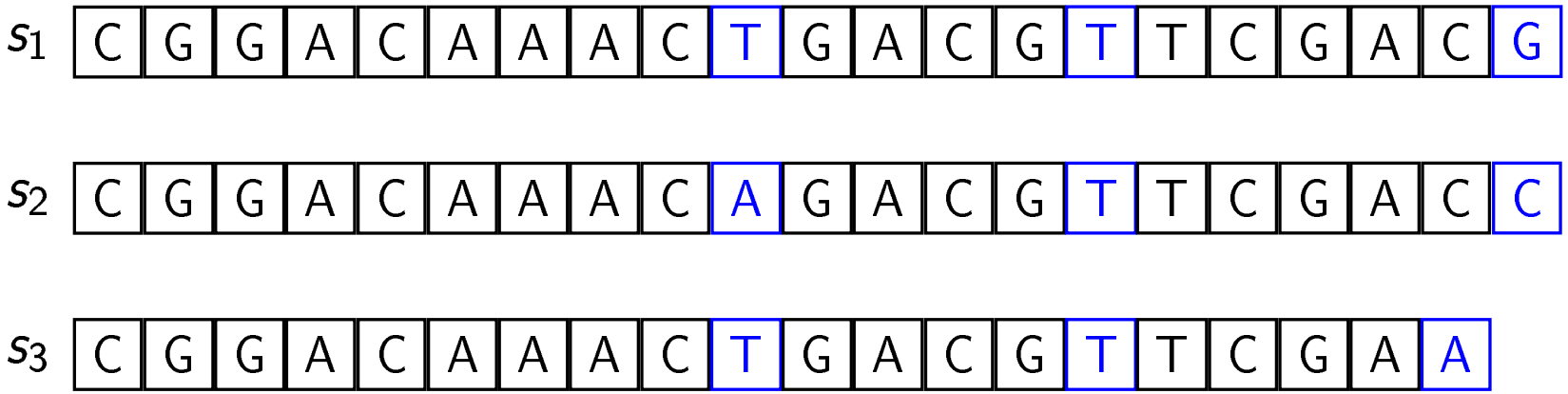
$$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC), (18, ACG)\}$$
$$\text{overlap}(ref, RC_2) = \{(7, ACAGA), (12, CGTTC), (18, ACC)\}$$
$$\text{overlap}(ref, RC_3) = \{(7, ACTGA)$$

Gesamtalgorithmus (Forts.)



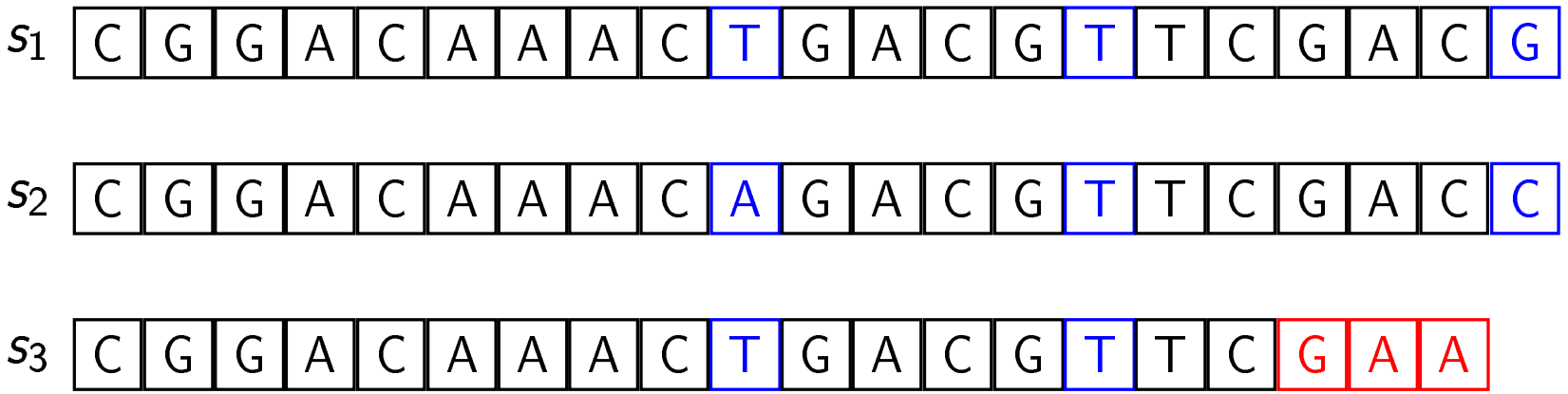
$$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC), (18, ACG)\}$$
$$\text{overlap}(ref, RC_2) = \{(7, ACAGA), (12, CGTTC), (18, ACC)\}$$
$$\text{overlap}(ref, RC_3) = \{(7, ACTGA)$$

Gesamtalgorithmus (Forts.)



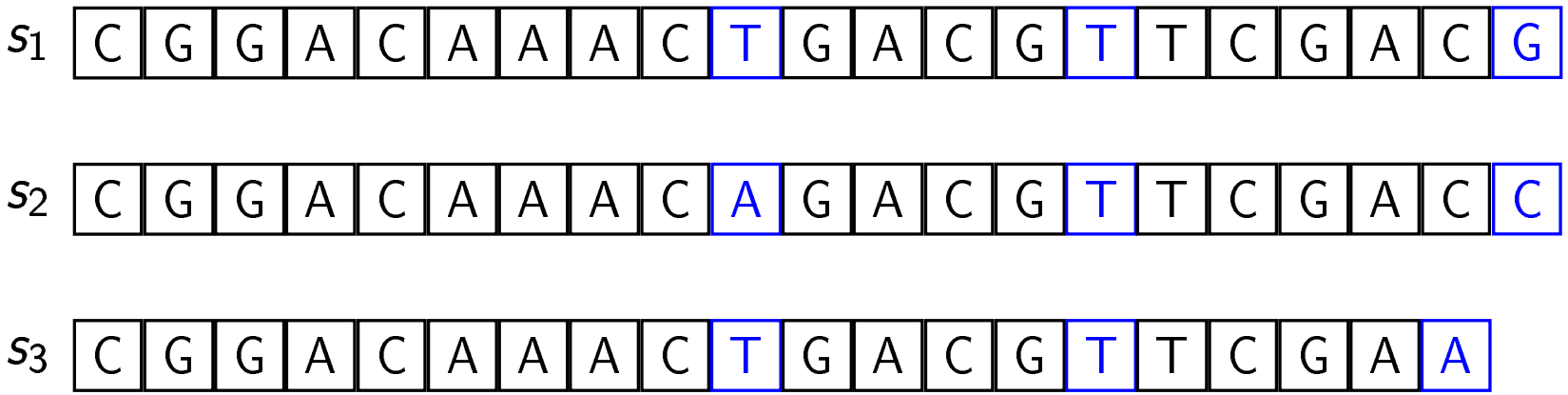
$$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC), (18, ACG)\}$$
$$\text{overlap}(ref, RC_2) = \{(7, ACAGA), (12, CGTTC), (18, ACC)\}$$
$$\text{overlap}(ref, RC_3) = \{(7, ACTGA), (12, CGTTC)$$

Gesamtalgorithmus (Forts.)



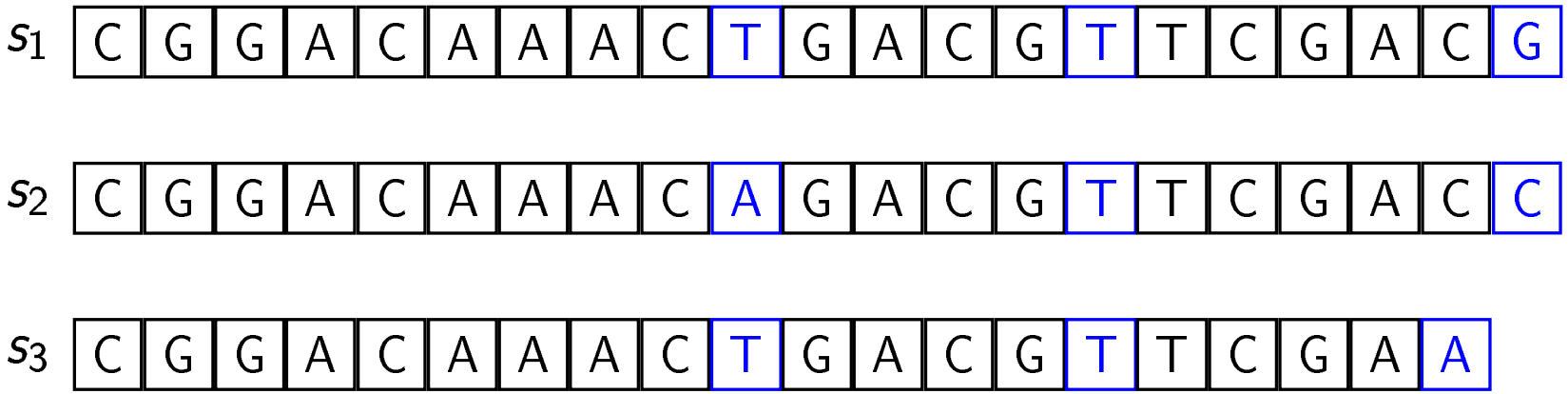
$$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC), (18, ACG)\}$$
$$\text{overlap}(ref, RC_2) = \{(7, ACAGA), (12, CGTTC), (18, ACC)\}$$
$$\text{overlap}(ref, RC_3) = \{(7, ACTGA), (12, CGTTC)$$

Gesamtalgorithmus (Forts.)



$$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC), (18, ACG)\}$$
$$\text{overlap}(ref, RC_2) = \{(7, ACAGA), (12, CGTTC), (18, ACC)\}$$
$$\text{overlap}(ref, RC_3) = \{(7, ACTGA), (12, CGTTC), (17, GAA)\}$$

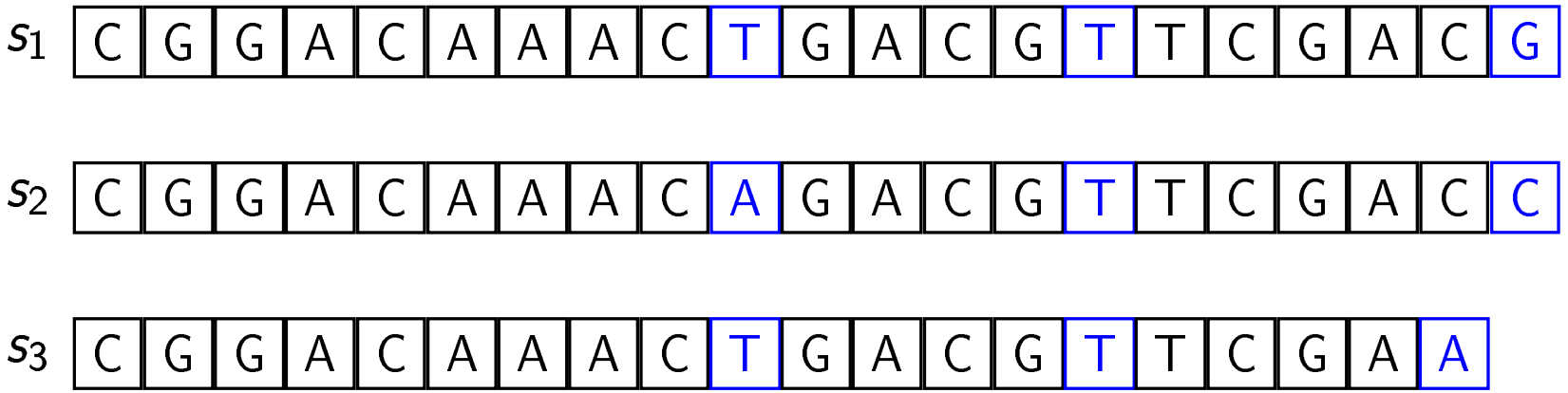
Gesamtalgorithmus (Forts.)



$$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC), (18, ACG)\}$$
$$\text{overlap}(ref, RC_2) = \{(7, ACAGA), (12, CGTTC), (18, ACC)\}$$
$$\text{overlap}(ref, RC_3) = \{(7, ACTGA), (12, CGTTC), (17, GAA)\}$$

Einziges Vorkommen an $(18, AA)$ in s_3.

Gesamtalgorithmus (Forts.)



$$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC), (18, ACG)\}$$
$$\text{overlap}(ref, RC_2) = \{(7, ACAGA), (12, CGTTC), (18, ACC)\}$$
$$\text{overlap}(ref, RC_3) = \{(7, ACTGA), (12, CGTTC), (17, GAA)\}$$

Einziges Vorkommen an $(18, AA)$ in s_3.

Endergebnis:
$\text{DBsearch}(\{s_1, s_2, s_3\})^0_{AA} = \{(1, \{(5, AA), (6, AA)\}), (2, \{(5, AA), (6, AA)\}), (3, \{(5, AA), (6, AA), (18, AA)\})\}$

Bemerkungen zur Überlappungsmenge

Beobachtung Überlappungsbereiche kommen mehrfach vor
$\rightarrow$ Einsparpotenzial für große Genomdatensätze

Suche in Überlappungsbereichen in der Praxis:

Bemerkungen zur Überlappungsmenge

Beobachtung Überlappungsbereiche kommen mehrfach vor
$\rightarrow$ Einsparpotenzial für große Genomdatensätze

Suche in Überlappungsbereichen in der Praxis:

$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC), (18, ACG)\}$

Bemerkungen zur Überlappungsmenge

Beobachtung Überlappungsbereiche kommen mehrfach vor
$\rightarrow$ Einsparpotenzial für große Genomdatensätze

Suche in Überlappungsbereichen in der Praxis:

$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC), (18, ACG)\}$

Anhängen an Referenzgenom:

Bemerkungen zur Überlappungsmenge

Beobachtung Überlappungsbereiche kommen mehrfach vor
$\rightarrow$ Einsparpotenzial für große Genomdatensätze

Suche in Überlappungsbereichen in der Praxis:

$\text{overlap}(ref, RC_1) = \{(7, ACTGA), (12, CGTTC), (18, ACG)\}$

Anhängen an Referenzgenom:

Referenzgenom	*	A	C	T	G	A	*	C	G	T	T	C	*	A	C	G

„$*$“ : Trennzeichen (für $k > 0$ entsprechend mehr Trennzeichen)

Experimentelle Ergebnisse

Verwendete Testsysteme

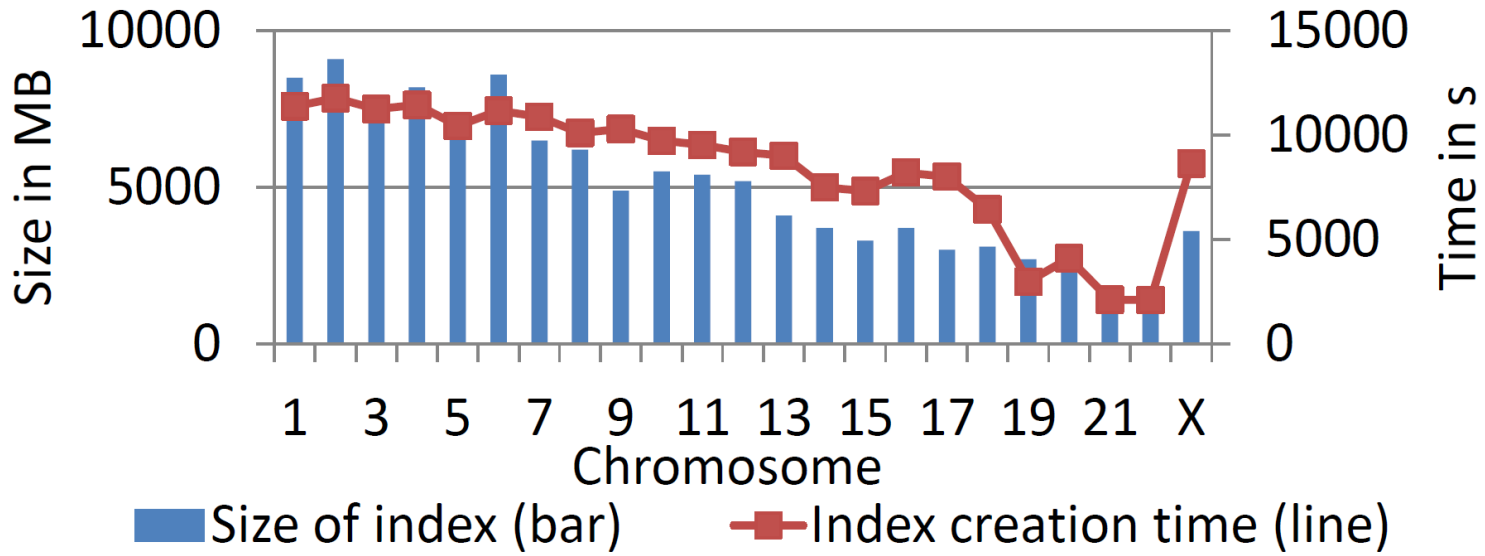
- Notebook mit Quadcore-Prozessor und, 16 GB RAM
- Server mit 4 Prozessoren á 10 Kernen, 1 TB RAM

Verwendete Datensätze

- 1092 Genome aus dem „1000 Genomes Project“
- für einige Tests nur Chromosome statt komplettes Genom verwendet
- Vergleichstests mit ersten 10MB von Chromosome 1 ($\tilde{1}$000 Stück)
- Reads aus Genommenge generiert, Länge $\tilde{1}$20-170

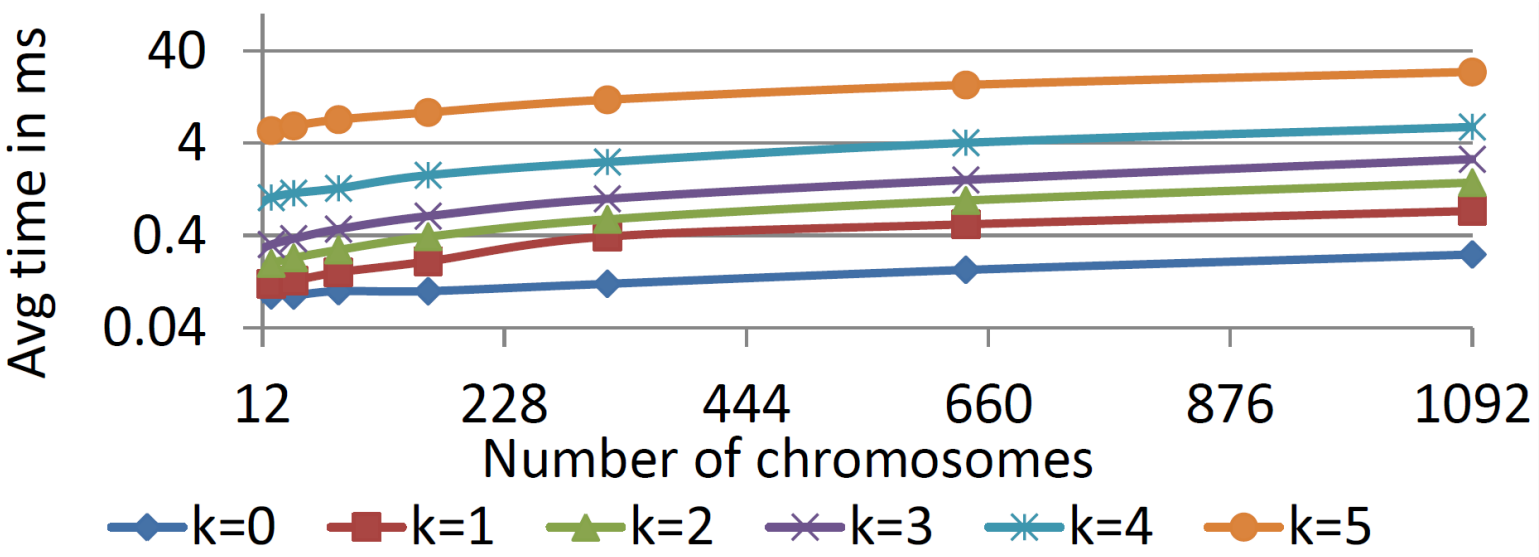
Bilder sind aus [Wandelt et al., 2013] entnommen!

Indexgröße und Rechenzeit



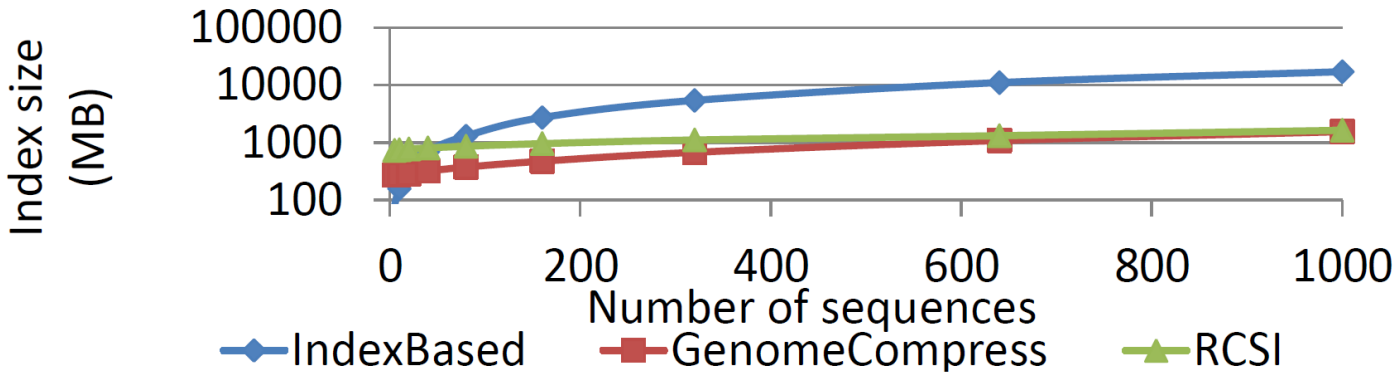
- Rechendauer auf Notebook: 54 Stunden (+8 Stunden für Kompression)
- 3 TB Rohdaten $\rightarrow$ 115 GB Index

Rechenzeit - Skalierung

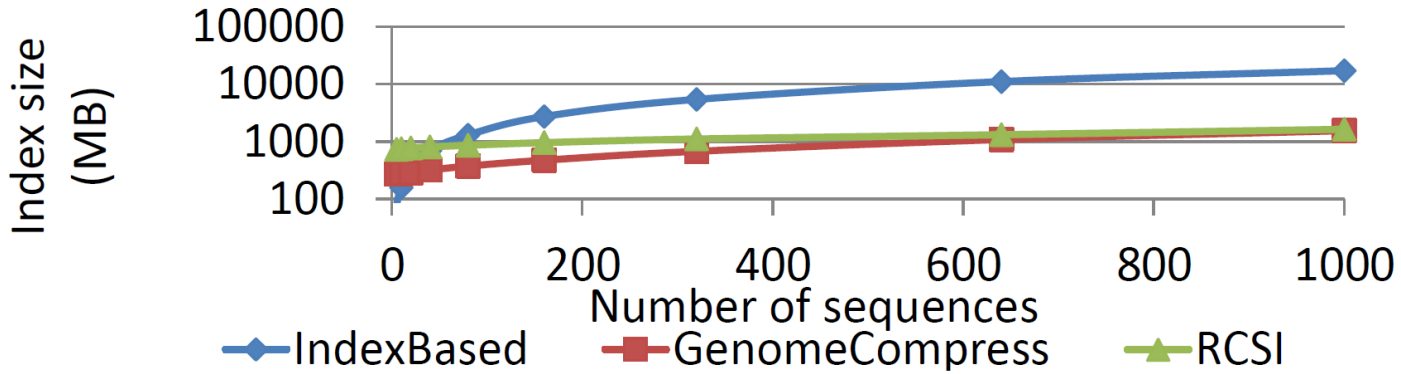


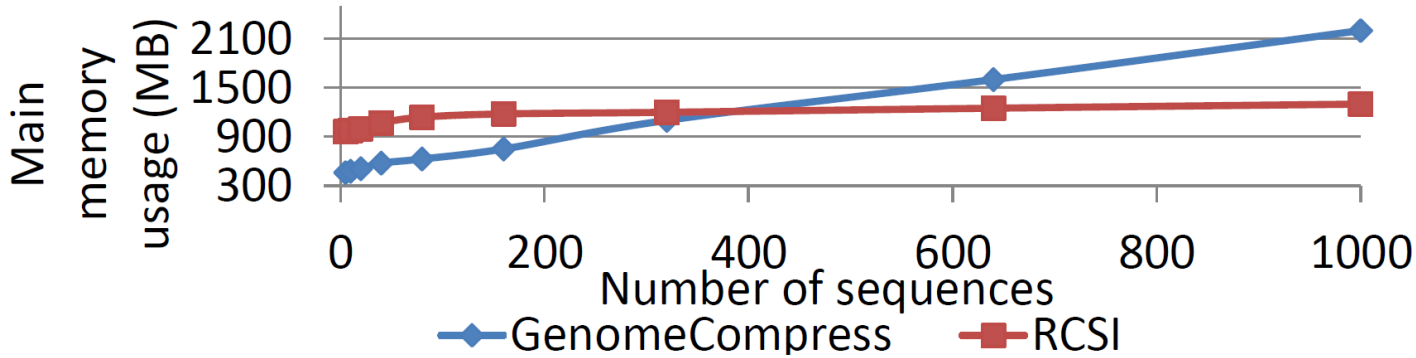
- Durchgeführt auf Notebook
- Gute Parallelisierung, wenn Suche nach Chromosom aufgeteilt (auf Testserver)

Vergleich mit GenomeCompress - Speicherbedarf



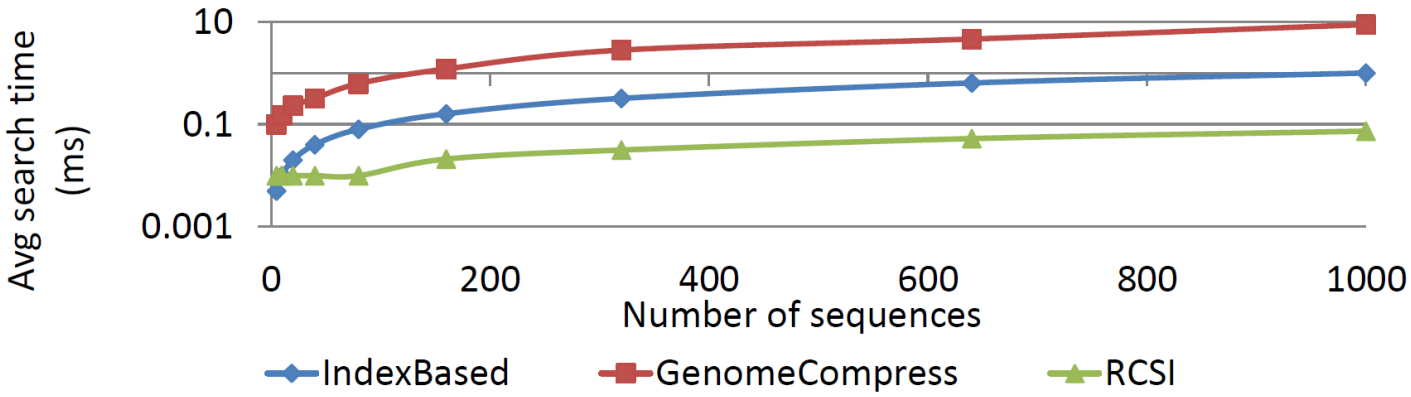
Vergleich mit GenomeCompress - Speicherbedarf





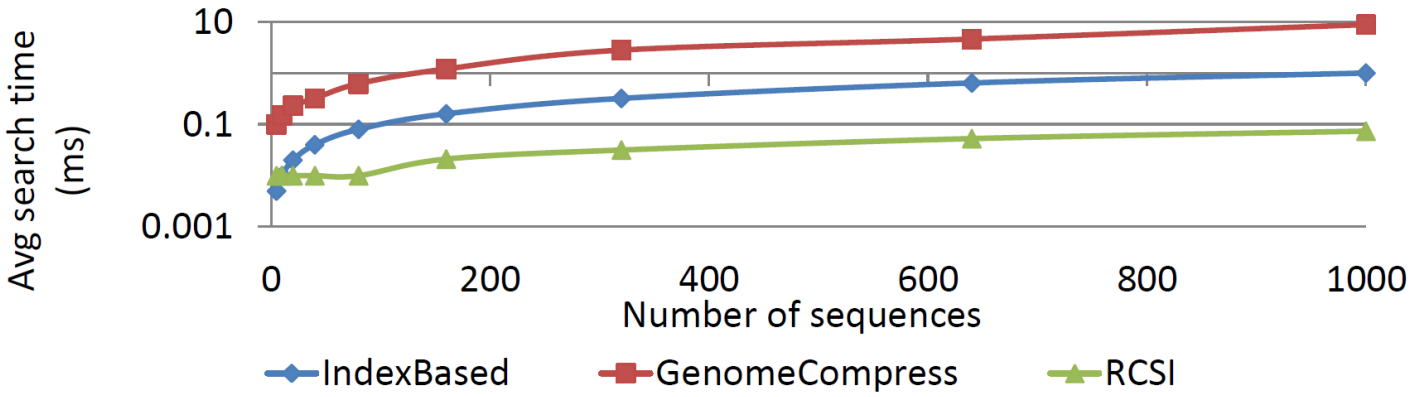
Vergleich mit GenomeCompress - Suchzeit

Exakte Suche:

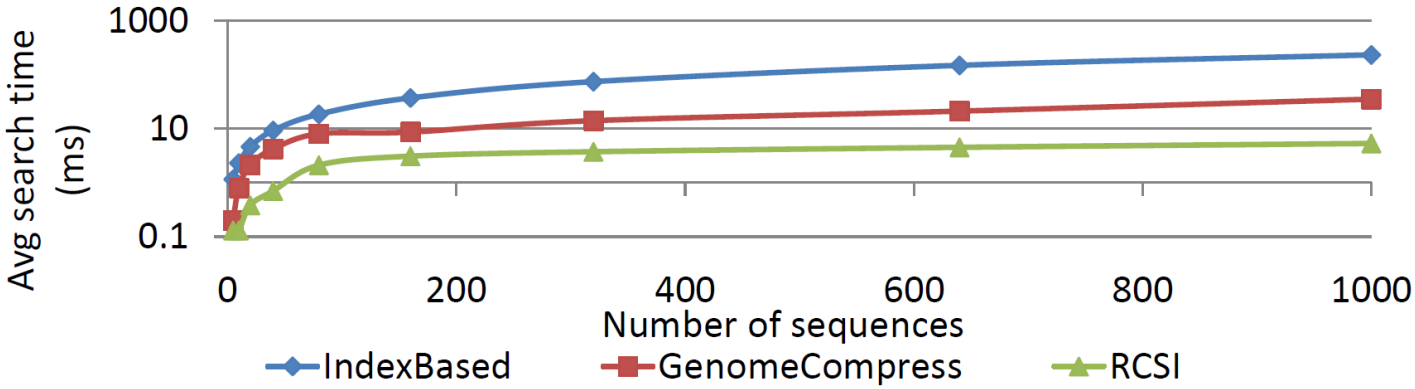


Vergleich mit GenomeCompress - Suchzeit

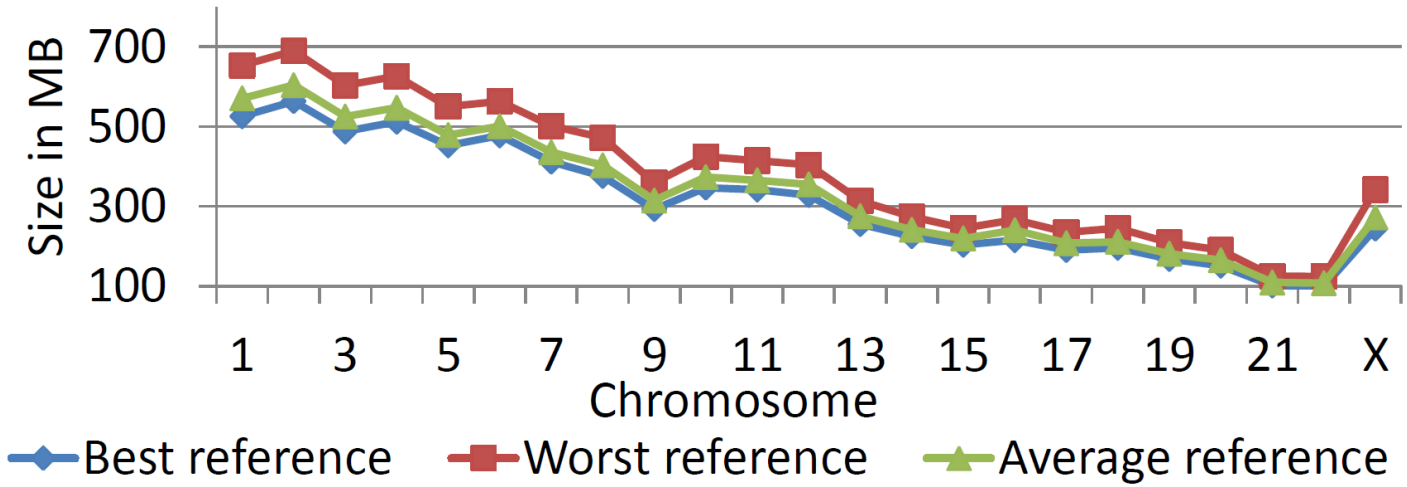
Exakte Suche:



Approximative Suche ($k = 3$):



Wahl des Referenzgenoms



Fazit zu RCSI-Algorithmus

- Grundidee: Komprimiere Referenzmenge für schnellere Suche
- Skaliert gut mit zunehmender Anzahl an Genomen
- Sehr hohe Kompressionsraten (>400:1 für reine Daten)
- Suchzeit sehr gering für exakte Suchen
- Approximative Suche deutlich länger, aber immer noch schnell
- Deutlich schneller als untersuchter Referenzalgorithmus
- Sehr aufwändiges Preprocessing

→ gut für große Datenmengen geeignet
→ steht fällt mit der Ähnlichkeit zwischen den Genomen

BWBBLE-Algorithmus

Weiter geht's mit dem BWBBLE-Algorithmus

Literatur


Huang, L., Popic, V., and Batzoglou, S. (2013).
Short read alignment with populations of genomes.
Bioinformatics, 29(13):361–370.


Wandelt, S., Starlinger, J., Bux, M., and Leser, U. (2013).
Rcsi: Scalable similarity search in thousand(s) of genomes.
PVLDB, 6(13):1534–1545.