Parallel algorithms for degenerate and weighted sequences derived from high throughput sequencing technologies

Costas S. Iliopoulos[1,2], Mirka Miller[1,3,4] and Solon P. Pissis[1]

[1] Dept. of Computer Science, King's College London
[2] Digital Ecosystems & Business Intelligence Institute, Curtin University
[3] School of Electrical Engineering and Computer Science, The University of Newcastle, Callaghan NSW 2308, Australia
[4] Dept. of Mathematics, University of West Bohemia, Pilsen, Czech Republic

September 2, 2009

Overview

Overview

Abstract

Introduction

Preliminaries

Problems definition

Massive Exact Unique Pattern Matching in Parallel
- The Exact Algorithm

Massive Approximate Unique Pattern Matching in Parallel
- The Bit-Vector-Mismatches Algorithm
- The Approximate Algorithm

Conclusion

Abstract

- **High throughput sequencing technologies** have opened new and exciting opportunities in the use of **DNA sequences**.
- We address the problem of **mapping millions** of **degenerate** and **weighted** sequences to **a reference genome** in parallel.
- We formally define and solve the *Massive Exact and Approximate Unique Pattern Matching* problem for **degenerate** and **weighted** sequences.

Introduction

- High throughput sequencing technologies produce **tens of millions** of short reads of currently typical **25-50 bp** in a single run.
- An important problem with these technologies is how to **efficiently and accurately map** these short reads to **a reference genome**.
- The **limitations** of the **equipment** used, or the **natural polymorphisms** that can be observed between individual samples can give rise to **uncertain sequence**s.
- Sequences, where more than one base (A, C, G, T) are possible in certain positions, are called **degenerate**.

Introduction

- Sequences, where the probability of every symbol's occurrence at every location is given, are called **weighted**.
- We address the problem of **mapping millions** of **degenerate** and **weighted** patterns to a reference genome in parallel...
- ...with respect to whether they **occur exactly once** in the genome or not, and...
- ...by taking into consideration **probability scores**.

Preliminaries

- A **degenerate** string is a sequence $s = s[1 \dots n]$, where $s[i] \subseteq \Sigma$ for each i, and Σ is a given alphabet.
- When a position of the string is degenerate, and it can match more than one element from the alphabet Σ, we say that this position has **non-solid** symbol.
- If in a position only one element of the alphabet Σ is present, we refer to this symbol as **solid**.
- A **weighted** string over an alphabet Σ is a sequence $s = s[1 \dots n]$ of sets of couples. In particular, each $s[i]$ is a set $((q_1, \pi_i(q_1)), (q_2, \pi_i(q_2)), \dots, (q_{|\Sigma|}, \pi_i(q_{|\Sigma|}))$, where $\pi_i(q_j)$ is the occurrence probability of character q_j at position i. For every position $1 \leq i \leq n$, $\sum_{j=1}^{|\Sigma|} \pi_i(q_j) = 1$.

Preliminaries

Example

- a *degenerate* string $s = A \begin{pmatrix} A \\ C \\ G \\ T \end{pmatrix} GT \begin{pmatrix} A \\ C \\ T \end{pmatrix} AC$
- a *weighted* string $s = C \begin{pmatrix} C & 0.2 \\ G & 0.2 \\ T & 0.6 \end{pmatrix} GT \begin{pmatrix} A & 0.1 \\ C & 0.1 \\ G & 0.2 \\ T & 0.6 \end{pmatrix} AC$

Problems definition

- **Problem 1.**
  Find whether the degenerate pattern $p_i = p_i[1 \ldots \ell]$, for all $0 \leq i < r$, of length $\ell_{min} \leq \ell \leq \ell_{max}$, with at most μ non-solid symbols, occurs with at most k-mismatches in $t = t[1 \ldots n]$, exactly once.
- **Problem 2.**
  Find whether the weighted pattern $p_i = p_i[1 \ldots \ell]$, for all $0 \leq i < r$, of length $\ell_{min} \leq \ell \leq \ell_{max}$, with at most μ non-solid symbols, occurs with at most k-mismatches in $t = t[1 \ldots n]$, exactly once, with probability at least c, if $\sum_{i=1}^{\ell} \pi_i(q_i) \geq c$.

Problems definition

We mainly focus on the following classes of both problems:

- **Class 1.** ρ_i occurs in t once
- **Class 2.** ρ_i occurs, with at most 1-mismatch in t, once
- **Class 3.** ρ_i occurs, with at most 2-mismatches in t, once

We assume that the data is derived from **high quality** sequencing methods and therefore we will consider patterns with **at most** $\mu = 3$ **non-solid symbols**.

Massive Exact Unique Pattern Matching in Parallel

- In order for the procedure to be efficient we will make use of **word-level parallelism** by compacting strings into single computer words that we call **signatures**.
- We get the signature $\sigma(x)$ of a string x, by transforming it to its binary equivalent using **2-bits-per-base encoding** of the DNA alphabet, and storing its **decimal value** into **a computer word**.
  $A \rightarrow 00,\ C \rightarrow 01,\ G \rightarrow 10,\ T \rightarrow 11$

  Example
  $\sigma(ACGT) = 00011011_2 = 27_{10}$

The Exact Algorithm

We use a **data decomposition** approach to partition the text t with the sliding window mechanism into a set of substrings $z_1, z_2, \ldots, z_{n-\ell+1}$, where $z_i = t[i \ldots i + \ell - 1]$, for all $1 \leq i \leq n - \ell + 1$.

An outline of the algorithm, for all $\ell_{min} \leq \ell \leq \ell_{max}$, is as follows.

- **1.** We **distribute** $z_1, z_2, \ldots, z_{n-\ell+1}$ evenly among the p **available processors**. We denote $z_{first_q}, \ldots, z_{last_q}$ as the set of the **allocated substrings** of processor ρ_q.
- **2.** Each processor ρ_q **transforms** each allocated **substring** z_i, for all $first_q \leq i \leq last_q$, into a **signature** $\sigma(z_i)$, **packs** it in a couple $(i,\ \sigma(z_i))$, and **adds** the couple to a local list Z_q.

The Exact Algorithm

- **3.** We **sort** the local lists Z_q based on the **signature's field**, in parallel, using *Parallel Sorting by Regular Sampling* (PSRS), a practical parallel deterministic sorting algorithm.
- **4.** Each processor ρ_q runs sequentially through its sorted list Z_q and **checks** whether the signatures in $Z_q[x]$ and $Z_q[x+1]$ are equal, for all $0 \leq x < |A_q| - 1$. If they are equal, then ρ_q adds $Z_q[x]$ to a **new list** L_q. If not, then $Z_q[x]$ is added to a **new list** L'_q.

The Exact Algorithm

- **5.** Each processor ρ_q, for all $1 \leq q < p$, sends the **first element** in Z_q to the neighbour processor ρ_{q-1}. Then, each processor ρ_q, for all $0 \leq q < p-1$, compares the signature of the **last element** in Z_q, to the signature of the element received from processor ρ_{q+1}. If they are equal, then processor ρ_q adds the element to the list L_q, else it is added to the list L'_q. **(Boundary comparison)**
- **6.** Processor ρ_0 perform a **gather** operation to **collect** and **combine** L_q and L'_q to Λ_ℓ and Λ'_ℓ, respectively. Processor ρ_0 performs a **one-to-all broadcast** to **send both lists** Λ_ℓ and Λ'_ℓ to **all other processors**.

The Exact Algorithm

- **7.** Each processor ρ_q is allocated a **fair amount** of query **patterns** from the set $p_0, p_1, ..., p_{r-1}$.
- **8.** We **extend** the set of patterns $\rho_0, \rho_1, ..., \rho_{r-1}$ to a new set $\rho'_0, \rho'_1, ..., \rho'_{r'-1}$, $r < r'$, as follows.
    1. **Problem 1.** For each **degenerate** pattern ρ_i of length ℓ with λ non-solid symbols, such that $\lambda \leq \mu$, we create $\prod_{j=1}^{\ell} |\rho[j]|$ **new patterns**, each differing in λ **positions**.
    2. **Problem 2.** For each **weighted** pattern ρ_i of length ℓ with λ non-solid symbols, such that $\lambda \leq \mu$, we create $\prod_{j=1}^{\ell} |\rho[j]|$ **new patterns**, each differing in λ **positions**. We select each of those patterns, say $s = s[1...\ell]$, with $s[1] = (q_1, \pi_1(q_1))$, $s[2] = (q_2, \pi_2(q_2))$,...,$s[\ell] = (q_\ell, \pi_\ell(q_\ell))$, that satisfy $\prod_{j=1}^{\ell} \pi_j(q_j) \geq c$, where c is a constant.

The Exact Algorithm

Example

- **Problem 1.** if $\rho'_i = A \begin{pmatrix} A \\ C \end{pmatrix} GT \begin{pmatrix} G \\ T \end{pmatrix} AC$ then we construct $AAGTGAC$, $ACGTGAC$, $AAGTTAC$, $ACGTTAC$
- **Problem 2.** if $\rho'_i = C \begin{pmatrix} A & 0.05 \\ G & 0.95 \end{pmatrix} CT \begin{pmatrix} A & 0.1 \\ T & 0.9 \end{pmatrix} TC$ and $c = 0.3$ then we only construct $CGCTTTC$

Notice that, since $\mu = 3$ and $|\Sigma| = 4$, the number of the new created patterns is treated as **constant**.

The Exact Algorithm

Assuming that the two sets of lists $\Lambda_{\ell_{min}}, \ldots, \Lambda_{\ell_{max}}$ and $\Lambda'_{\ell_{min}}, \ldots, \Lambda'_{\ell_{max}}$ are already created...

- **9.** We **transform** each **pattern** ρ'_i, for all $0 \leq i < r'$, into a **signature**.
- **10.** We can **determine**, by using a binary search, whether a pattern ρ'_i of length ℓ occurs in t exactly once.
  1. If $\sigma(\rho'_i) \in \Lambda'_\ell$, then ρ'_i is a **unique pattern**, and the algorithm returns its matching position in t.
  2. If $\sigma(\rho'_i) \in \Lambda_\ell$, then ρ'_i occurs in t **more than once**.
  3. If $\sigma(\rho'_i) \notin \Lambda_\ell$ and $\sigma(\rho'_i) \notin \Lambda'_\ell$, then ρ'_i **does not occur** in t.

The Exact Algorithm

- $\mathcal{O}(\lceil \ell_{max}/w \rceil (\frac{n}{p} \log \frac{n}{p} + \frac{r}{p} \log n))$ **computation** time
- $\mathcal{O}(n \log p + r)$ **communication** time

Massive Approximate Unique Pattern Matching in Parallel

- We make use of **word-level parallelism**, and apply a **bit-vector algorithm** for efficient approximate string matching with mismatches.
- The **fixed-length approximate string matching with at most k-mismatches** problem: given a text t of length n, a pattern ρ of length m and an integer ℓ, find all substrings of ρ of length ℓ that match any contiguous substring of t of length ℓ with at most k-mismatches.
- If we assign $\rho{=}t$, we can extract all **unique** and **duplicate** substrings of length ℓ of t with **at most** k-mismatches.

Massive Approximate Unique Pattern Matching in Parallel

- The focus is on computing **matrix** M, which contains the number of **mismatches** of all substrings of pattern ρ of length ℓ and any contiguous substring of the text t of length ℓ.
- We maintain the **bit-vector** $B[i,j] = b_\ell \ldots b_1$, where $b_\lambda = 1,\ 1 \leq \lambda \leq \ell$, if there is a mismatch of a contiguous substring of the text $t[i-\ell+1 \ldots i]$ and $t[j-\ell+1 \ldots j]$ in the λ^{th} position. Otherwise we set $b_\lambda = 0$.
- Given the restraint that the integer ℓ is less than the length of the computer word w, then the bit-vector operations allow to update each entry of the matrix B in **constant time** (using "shift"-type of operation on the bit-vector).

Massive Approximate Unique Pattern Matching in Parallel

		0	1	2	3	4	5	6	7
		ϵ	G	G	G	T	C	T	A
0	ϵ	0	0	0	0	0	0	0	0
1	G	1	0	0	0	1	1	1	1
2	G	2	1	0	0	1	2	2	2
3	G	3	2	1	0	1	2	3	3
4	T	3	3	2	1	0	2	2	3
5	C	3	3	3	2	2	0	3	2
6	T	3	3	3	3	2	3	0	3
7	A	3	3	3	3	3	2	3	0

Table: Matrix M for $t = p = GGGTCTA$ and $\ell = 3$.

Massive Approximate Unique Pattern Matching in Parallel

		0	1	2	3	4	5	6	7
		ϵ	G	G	G	T	C	T	A
0	ϵ	0	0	0	0	0	0	0	0
1	G	1	0	0	0	1	1	1	1
2	G	11	10	00	00	01	11	11	11
3	G	111	110	100	000	001	011	111	111
4	T	111	111	101	001	000	011	110	111
5	C	111	111	111	011	011	000	111	101
6	T	111	111	111	111	110	111	000	111
7	A	111	111	111	111	111	101	111	000

Table: Matrix B for $t = p = GGGTCTA$ and $\ell = 3$.

Massive Approximate Unique Pattern Matching in Parallel

The **maintenance** of the **bit-vector** is done via **operations** defined as follows:

1. $\mathit{shiftc}(x)$: shifts and truncates the leftmost bit of x.
2. $\delta_H(x, y)$: returns the minimum number of replacements required to transform x into y

The Bit-Vector-Mismatches Algorithm

```
Bit-Vector-Mismatches
▷Input: $t$, $n$, $\rho$, $m$, $\ell$
▷Output: $B$, $M$
1   begin
2       ▷ Initialization
3       $B[0...m, 0] \leftarrow \min(i, \ell)$ 1's; $B[0, 0..n] \leftarrow 0$
4       for $i \leftarrow 1$ until $m$ do
5           for $j \leftarrow 1$ until $n$ do
6               $B[i, j] \leftarrow shiftc(B[i-1, j-1])$ OR $\delta_H(\rho[i], t[j])$
7               $M[i, j] \leftarrow ones(B[i, j])$
8   end
```

Figure: The Bit-Vector-Mismatches algorithm

The Approximate Algorithm

We use a **functional decomposition** approach, in which the initial focus is on the computation that is to be performed rather than on the data manipulated by the computation. We partition the problem of computing matrix B (and M) into a set of diagonal vectors $\mathit{\Delta}_0, \mathit{\Delta}_1, ..., \mathit{\Delta}_{n+m}$.

$$\Delta_\nu[x] = \begin{cases} B[\nu - x,\ x] \ : \ 0 \leq x \leq \nu, & \text{(a)} \\ B[m - x,\ \nu - m + x] \ : \ 0 \leq x < m + 1, & \text{(b)} \\ B[m - x,\ \nu - m + x] \ : \ 0 \leq x < n + m - \nu + 1, & \text{(c)} \end{cases} \tag{1}$$

where,
(a) if $0 \leq \nu < m$, (b) if $m \leq \nu < n$, (c) if $n \leq \nu < n + m + 1$.

The Approximate Algorithm

An outline of the algorithm, for all $\ell_{min} \leq \ell \leq \ell_{max}$, is as follows. In each diagonal $\varDelta_0, \varDelta_1, \dots, \varDelta_{n+m}$

- **1. Each processor** is allocated with $|\Delta_\nu|/p$ cells (without loss of generality) and **computes** each **allocated cell** using the Bit-Vector-Mismatches.
- **2. If** $M[i, j] = 0$ **and** $i = j$, then substring $t[i - \ell + 1 \dots i]$ occurs in t at least once. We **transform** substring $t[i - \ell + 1 \dots i]$ into a signature $\sigma(t[i - \ell + 1 \dots i])$, **pack** it in a couple $(i - \ell + 1, \sigma(t[i - \ell + 1 \dots i]))$, and **add** the couple to a new list Z_q.

The Approximate Algorithm

If $M[i,j] \leq k$ **and** $i \neq j$, then substrings $t[i-\ell+1 \dots i]$ and $t[j-\ell+1 \dots j]$ are considered to be duplicates with at most k-mismatches. We **transform** both substrings into the signatures $\sigma(t[i-\ell+1 \dots i])$ and $\sigma(t[j-\ell+1 \dots j])$, **pack** them in couples $(i-\ell+1, \sigma(t[i-\ell+1 \dots i]))$ and $(j-\ell+1, \sigma(t[j-\ell+1 \dots j]))$, and **add** the couples to the list Z_q.

- **3. Processors communication** involving point-to-point **boundary cells swaps**.

The Approximate Algorithm

- **4.** Assume that the diagonal supersteps $\varDelta_0, \varDelta_1, ..., \varDelta_{n+m}$ are executed. The local lists Z_q are constructed, and so, we **follow the steps 3-9 of the exact algorithm**.
- **5.** We can determine, by using a binary search, whether a pattern ρ'_i of length ℓ occurs in t exactly once.
    1. If $\sigma(\rho'_i) \in \varLambda'$, then ρ'_i is a **unique pattern** with at most k-mismatches.
    2. If $\sigma(\rho'_i) \in \varLambda_\ell$, then ρ'_i occurs in t **more than once** with at most k-mismatches.
    3. If $\sigma(\rho'_i) \notin \varLambda_\ell$ and $\sigma(\rho'_i) \notin \varLambda'_\ell$, then **we can check** whether the k-mismatches occur inside ρ'_i.

The Approximate Algorithm

1. **Class 2 and Class 3.** We **construct** a new set of patterns x_j, for all $0 \leq j < |\Sigma|.\ell$, differing from ρ'_i in one position, we compact each x_j into a signature $\sigma(x_j)$, and do the binary search in Λ' and Λ.
2. **Class 3.** We **construct** a new set of patterns y_j, for all $0 \leq j < |\Sigma|^2.\binom{\ell}{2}$, differing from ρ'_i in two positions, we compact each y_j into a signature $\sigma(y_j)$, and do the binary search in Λ' and Λ.

In general, for the problem of k-mismatches, for each pattern ρ'_i of length ℓ that does not occur in t, we construct k new sets of patterns, each containing $|\Sigma|^\lambda.\binom{\ell}{\lambda}$ patterns differing from ρ'_i in λ positions, for all $1 \leq \lambda \leq k$.

The Approximate Algorithm

- $\mathcal{O}(\lceil \ell_{max}/w \rceil (\frac{n^2}{p} + \frac{\ell_{max}^2 r}{p} \log p))$ **computation** time
- $\mathcal{O}(n \log p + r)$ **communication** time

Conclusion

- The **new technologies** produce a **huge number** of very **short** sequences and these sequences need to be **classified, tagged and recognised** as parts of a **reference genome**.
- The proposed algorithms can manipulate this data for **degenerate** and **weighted** sequences for Massive Exact and Approximate Unique Pattern Matching in Parallel.
- We have already **implemented** and **tested** the **exact** case, getting very promising results, comparable to **more traditional** mapping programs.
- Our **immediate target** is to implement **the approximate case** and explore the possibility of **a faster algorithm**.

Conclusion

Biosciences and Computing are not just trying to bridge their gaps.

Most importantly, they remind us all, that **Science** is, mainly, **a social tool**, which therefore must be used for **humanistic purposes**.

Thank you!

Questions

Look and you shall find it; for what is unsought will go undetected.

Sophocles (496 BC - 406 BC)