News about Snakemake

Johannes Köster

February 5, 2013
Motivation

bwa

gatk

samtools

refs = defaultdict(list)
with FastaReader(input.fasta) as fr:
    for record in fr:
        name = record.name.split(None, 1)[0]
        namelist = refs[record.sequence]
        if name not in namelist:
            namelist.append(name)
with open(output.out, 'w') as out:
    for seq, name in refs.items():
        seq = seq.replace('U', 'T')
        print('>{}{}'.format('
'.join(name), seq), file=out)
Motivation

bwa

↓

samtools

↓

gatk

```python
refs = defaultdict(list)
with FastaReader(input.fasta) as fr:
    for record in fr:
        name = record.name.split(None, 1)[0]
        namelist = refs[record.sequence]
        if name not in namelist:
            namelist.append(name)
with open(output.out, 'w') as out:
    for seq, names in ref.items():
        seq = seq.replace('U', 'T')
        print(">{}".format(''.join(names)), seq, file=out)
```
Motivation

bwa → samtools → gatk

refs = defaultdict(list)
with FastxReader(input.fastq) as fr:
    for record in fr:
        name = record.name.split(NONE, 1)
namelist = refs[record.sequence]
if name not in namelist:
namelist.append(name)
with open(output.out, 'w') as out:
    for seq, names in ref.items():
        seq = seq.replace('U', 'T')
        print(''.join([seq, file])

refs = defaultdict(list)
with FastxReader(input.fastq) as fr:
    for record in fr:
        name = record.name.split(NONE, 1)
namelist = refs[record.sequence]
if name not in namelist:
namelist.append(name)
with open(output.out, 'w') as out:
    for seq, names in ref.items():
        seq = seq.replace('U', 'T')
        print(''.join([seq, file])

refs = defaultdict(list)
with FastxReader(input.fastq) as fr:
    for record in fr:
        name = record.name.split(NONE, 1)
namelist = refs[record.sequence]
if name not in namelist:
namelist.append(name)
with open(output.out, 'w') as out:
    for seq, names in ref.items():
        seq = seq.replace('U', 'T')
        print(''.join([seq, file])

refs = defaultdict(list)
with FastxReader(input.fastq) as fr:
    for record in fr:
        name = record.name.split(NONE, 1)
namelist = refs[record.sequence]
if name not in namelist:
namelist.append(name)
with open(output.out, 'w') as out:
    for seq, names in ref.items():
        seq = seq.replace('U', 'T')
        print(''.join([seq, file])

refs = defaultdict(list)
with FastxReader(input.fastq) as fr:
    for record in fr:
        name = record.name.split(NONE, 1)
namelist = refs[record.sequence]
if name not in namelist:
namelist.append(name)
with open(output.out, 'w') as out:
    for seq, names in ref.items():
        seq = seq.replace('U', 'T')
        print(''.join([seq, file])

refs = defaultdict(list)
with FastxReader(input.fastq) as fr:
    for record in fr:
        name = record.name.split(NONE, 1)
namelist = refs[record.sequence]
if name not in namelist:
namelist.append(name)
with open(output.out, 'w') as out:
    for seq, names in ref.items():
        seq = seq.replace('U', 'T')
        print(''.join([seq, file])
Motivation

bwa
\downarrow
samtools \ -q1
\downarrow
gatk
\downarrow

refs = defaultdict(list)
with FastaReader(input.fasta) as fr:
    for record in fr:
        name = record.name.split(None, 1)[0]
        namelist = refs[record.sequence]
        if name not in namelist:
            namelist.append(name)
with open(output.out, 'w') as out:
    for seq, names in ref.items():
        seq = seq.replace('U', 'T')
        print('>' + names + format('.join(names), seq, file=out))
Why Snakemake?

GNU Make provided us with...

- a language to write rules to create each output file from input files
- wildcards for generalization
- implicit dependency resolution
- implicit parallelization
- fast and collaborative development on text files
Why Snakemake?

GNU Make provided us with...

- a language to write rules to create each output file from input files
- wildcards for generalization
- implicit dependency resolution
- implicit parallelization
- fast and collaborative development on text files

but we missed...

- easy to read syntax
- simple scripting inside the workflow
- creating more than one output file with a rule
- multiple wildcards in filenames
Structure

1. Idea

2. Scheduling

3. Key Features
Snakemake Idea

Example: for samples \{500, \ldots, 503\} map reads to hg19.
Snakemake Idea

Example: for samples \(\{500,\ldots, 503\}\) map reads to hg19.

```bash
rule map_reads:
  input:  "hg19.fasta", "{sample}.fastq"
  output: "{sample}.sai"
  shell:  "bwa aln {input} > {output}" 
```
Genome Informatics

Snakemake Idea

Example: for samples \{500, \ldots, 503\} map reads to hg19.

```
rule sai_to_bam:
    input:  "hg19.fasta", "{sample}.sai", "{sample}.fastq"
    output:  "{sample}.bam"
    shell:
        "bwa samse {input} | samtools view -Sbh - > {output}"  

rule map_reads:
    input:  "hg19.fasta", "{sample}.fastq"
    output:  "{sample}.sai"
    shell:  "bwa aln {input} > {output}"  
```
Snakemake Idea

Example: for samples \{500, \ldots, 503\} map reads to hg19.

\texttt{SAMPLES = ["500", "501", "502", "503"]}

\texttt{rule all:}

\hspace{1cm} input: expand("{sample}.bam", sample=SAMPLES)

\texttt{rule sai_to_bam:}

\hspace{1cm} input:  "hg19.fasta", 
\hspace{1cm} "{sample}.sai", 
\hspace{1cm} "{sample}.fastq"

\hspace{1cm} output:  "{sample}.bam"

\hspace{1cm} shell:
\hspace{1cm} "bwa samse {input} | samtools view -Sbh \rightarrow {output}"

\texttt{rule map_reads:}

\hspace{1cm} input:  "hg19.fasta", 
\hspace{1cm} "{sample}.fastq"

\hspace{1cm} output:  "{sample}.sai"

\hspace{1cm} shell:  "bwa aln \{input\} \rightarrow \{output\}"
Snakemake Idea

Example: for samples \(\{500, \ldots, 503\}\) map reads to hg19.

```python
SAMPLES = ["500", "501", "502", "503"]
rule all:
    input: expand("{sample}.bam", sample=SAMPLES)

rule sai_to_bam:
    input:  "hg19.fasta", "{sample}.sai", "{sample}.fastq"
    output: protected("{sample}.bam")
    shell:
        "bwa samse {input} | samtools view -Sbh - -> {output}"

rule map_reads:
    input:  "hg19.fasta", "{sample}.fastq"
    output: "%sample}.sai"
    shell:  "bwa aln {input} > {output}"
Snakemake Idea

Example: for samples \{500, \ldots, 503\} map reads to hg19.

SAMPLES = ["500", "501", "502", "503"]

rule all:
   input: expand("\{sample\}.bam", sample=SAMPLES)

rule sai_to_bam:
   input: "hg19.fasta", "\{sample\}.sai", "\{sample\}.fastq"
   output: protected("\{sample\}.bam")
   shell:
      "bwa samse {input} | samtools view -Sbh - > {output}"

rule map_reads:
   input: "hg19.fasta", "\{sample\}.fastq"
   output: temp("\{sample\}.sai")
   shell: "bwa aln {input} > {output}"
Snakemake Idea

Example: for samples \{500, \ldots, 503\} map reads to hg19.

```
rule all
  500.bam, 501.bam, 502.bam, 503.bam
```

```
rule sai_to_bam:
  input: "hg19.fasta", "{sample}.sai", "{sample}.fastq"
  output: protected("{sample}.bam")
  shell:
    "bwa samse {input} | samtools view -Sbh -> {output}" 
```

```
rule map_reads:
  input: "hg19.fasta", "{sample}.fastq"
  output: temp("{sample}.sai")
  shell: "bwa aln {input} > {output}" 
```
Snakemake Idea

Example: for samples \(\{500, \ldots, 503\}\) map reads to hg19.

```python
rule all:
    500.bam, 501.bam, 502.bam, 503.bam

rule sai_to_bam:
    500.sai

rule map_reads:
    input: "hg19.fasta", "{sample}.fastq"
    output: temp("{sample}.sai")
    shell: "bwa aln {input} > {output}"```
Snakemake Idea

Example: for samples \{500, \ldots, 503\} map reads to hg19.

```
rule all:
input: expand("{sample}.bam", sample=SAMPLES)
rule sai_to_bam:
input: "hg19.fasta", "{sample}.sai", "{sample}.fastq"
output: protected("{sample}.bam")
shell: "bwa samse {input} | samtools view -Sbh - > {output}"
rule map_reads:
input: "hg19.fasta", "{sample}.fastq"
output: temp("{sample}.sai")
shell: "bwa aln {input} > {output}"
```

500.bam, 501.bam, 502.bam, 503.bam

rule sai_to_bam
500.sai

rule map_reads
500.fastq
Snakemake Idea

Example: for samples \{500, \ldots, 503\} map reads to hg19.

```
rule all:
  input: expand("{sample}.bam", sample=SAMPLES)
  output: protected("{sample}.bam")
  shell: "bwa samse {input} | samtools view -Sbh - > {output}"

rule sai_to_bam:
  input: "hg19.fasta", "{sample}.sai", "{sample}.fastq"
  output: temp("{sample}.sai")
  shell: "bwa aln {input} > {output}"

rule map_reads:
  input: "hg19.fasta", "{sample}.fastq"
  output: temp("{sample}.sai")
  shell: "bwa aln {input} > {sample}.sai"
```

- rule sai_to_bam
  - 500.sai
- rule sai_to_bam
  - 501.sai
- rule sai_to_bam
  - 502.sai
- rule sai_to_bam
  - 503.sai
- rule map_reads
  - 500.fastq
- rule map_reads
  - 501.fastq
- rule map_reads
  - 502.fastq
- rule map_reads
  - 503.fastq

500.bam, 501.bam, 502.bam, 503.bam
Snakemake Idea

- DAG of jobs
- each path needs to be executed serially
- two disjoint paths can be executed in parallel
Building the DAG

an edge between two jobs A,B if input of A is matched by output of B, e.g.

"500.bam" matches "{sample}.bam"

\[ \Leftrightarrow \]

"500.bam" ∈ L(".+\.bam")

In case of ambiguity:

- Constrain wildcards: "{sample,[0-9]+}.bam"
- Order rules: ruleorder: sai_to_bam > sort_bam
Scheduling

execute the set of jobs $E^*$ among all $E \subseteq J$ that maximizes under lexicographical order

$$\sum_{j \in E} (p_j, i_j)$$

such that

$$\sum_{j \in E} t_j \leq I$$

$J$ set of jobs ready to execute
$T$ provided cores
$I$ idle cores
$t_j$ threads of job $j$
$p_j$ priority of job $j$
$i_j$ input size of job $j$
rule plot_coverage_histogram:
    input: "{sample}.bam"
    output: hist="{sample}.coverage.pdf"
    run:
        plt.hist(np.fromiter(
            shell("samtools mpileup {input} | cut -f4",
                iterable=True),
            dtype=int))
        plt.savefig(output.hist)
rule plot_coverage_histogram:
    input: ...
    output: ...
    run:
        R(""
        # some R code
        "")
Dynamic Output

In case of unknown number of output files... dynamically update the DAG to process these
Dynamic Output

In case of unknown number of output files... dynamically update the DAG to process these

```
rule all:
    input: dynamic("{cluster}.pdf")

rule plot:
    input: "{cluster}.csv"
    output: "{cluster}.pdf"
    shell: "gnuplot ..."

rule cluster:
    input: ...
    output: dynamic("{cluster}.csv")
    shell: "cluster ..."
```
Dynamic Output

In case of unknown number of output files... dynamically update the DAG to process these

```
rule all
dynamic("{cluster}.pdf")
```

```
rule plot:
    input: "{cluster}.csv"
    output: "{cluster}.pdf"
    shell: "gnuplot ...
```

```
rule cluster:
    input: ...
    output: dynamic("{cluster}.csv")
    shell: "cluster ..."
```
Dynamic Output

In case of unknown number of output files... dynamically update the DAG to process these

rule all
dynamic("" {cluster}.pdf")

rule plot

rule cluster:
  input: ...
  output: dynamic("{cluster}.csv")
  shell: "cluster ..."
Dynamic Output

In case of unknown number of output files... dynamically update the DAG to process these

```
rule all
dynamic("" {cluster}.pdf"")
```

```
rule plot
input: "{cluster}.csv"
output: "{cluster}.pdf"
shell: "gnuplot ..."
```

```
rule cluster
input: ...
output: dynamic("" {cluster}.csv"")
shell: "cluster ..."
```

rule plot
rule cluster
Dynamic Output

In case of unknown number of output files... dynamically update the DAG to process these

rule all
0.pdf 1.pdf 2.pdf 3.pdf

rule plot

rule cluster
Dynamic Output

In case of unknown number of output files... dynamically update the DAG to process these

rule all
  0.pdf 1.pdf 2.pdf 3.pdf

rule plot
  rule plot
  rule plot
  rule plot

rule cluster
rule report:
    input: "table.csv", "plot.pdf"
    output: "report.html"
run:
    report(""
            ================
            Report of some project
            ================
    
    Some text containing a formula
    :math:`\sum_j \in E t_j \leq I`
    and embedding a table F1_ and a figure F2_.
    """, output[0], F1=input[0], F2=input[1])
Report of some project

Some text containing a formula $\sum_{j\in E} t_j \leq I$ and embedding a table $F_1$ and a figure $F_2$. 

2013-01-31
Combining expressions of IncRNAs measured by qPCR with HuEx exon arrays

HuEx and qPCR datasets and their combination

Two datasets were given: An IncRNA assay using qPCR (11) and 274 primary tumors analyzed with Affymetrix HuEx exon arrays within the NRC. ENSEMBL Gene IDs for IncRNAs were extracted from given IncRNA qPCR assay description (12). For these, exon loci were derived from the ENSEMBL hgu19 v5 annotation track. Affymetrix HuEx 1.0 probe that is within these loci were identified, and combined to meta-probesets (13). Here, each row depicts an IncRNA given as ENSEMBL ID together with all the HuEx probes that should measure the expression of one of its exons.

Each of these meta-probesets summarizes the expression of one IncRNA. We calculated and normalized the expressions for the given 274 primary tumors. This was done with the Affymetrix Power Tools implementation of RMA with default parameters. It remains to be investigated if the RMA normalization has successful removed batch effects since the tumor data comes from different labs.

Estimation of regulated IncRNAs in the qPCR dataset

We estimate the consistency between the two controls by calculating the fold-change and throwing away all IncRNAs that exceed a threshold in this test. For the remaining IncRNAs the fold-change between treatment and the mean of the two controls is calculated.

Table 23 shows unregulated IncRNAs sorted by strength of fold-change. Table 24 shows the sense for downregulated IncRNAs. Figure 21 shows the histogram of fold-changes.

Counting tumors expressing the regulated IncRNAs in the HuEx dataset

We assume IncRNAs with an absolute logarithmic fold-change greater than 0.69 to be regulated.

We only consider those IncRNAs that can be measured by consensus probes (see Table 26). For these we calculate from the HuEx setup described above the number of tumors with a minimum probe-set expression of 6. Table 27 shows the results. Figure 25 shows a histogram of the observed counts.

Mathematical background

The provided qPCR analysis yielded \( -\log_{2} \) values that are on a logarithmic scale compared to the real molecule counts. This is because each PCR cycle in theory doubles the amount of molecules. Since this rate is not reached in practice we assume a factor of 1.8 here.

Consequently, a fold-change on these logarithmic values has to be computed as subtraction instead of a quotient. Further, a non-logarithmic fold-change of 1 corresponds to \( \log_{2} 1 \) in logarithmic scale. It has to be noted that at the moment it is not yet clear whether the normalization applied before may have an effect on the assumed factor.

We denote the different -\( \log_{2} \) values as \( y_{i}^{T} \) and \( y_{i}^{d} \) for a given IncRNA \( i \). The consistency between the two controls (see above) is now computed as

\[
|y_{i}^{T} - y_{i}^{d}| > 0.5
\]

The fold-change between treatment and the mean of controls is calculated as

\[
y_{i}^{T} - \frac{1}{2} (y_{i}^{T} + y_{i}^{d})
\]

Since the latter -\( \log_{2} \) values are still logarithmic, the mean here corresponds to the geometric mean of the real molecule counts. This is intended since it avoids diminution of the mean by the higher -\( \log_{2} \) value.
Visualization

Using the dot language of graphviz:

$ \text{snakemake} --\text{dag} \mid \text{dot} \mid \text{display}$
Using the dot language of graphviz:

$ snakemake --dag | dot | display
Visualization

- `build_bwa_index`
  - prefix: hgref/hg1kv37.fasta.gz

- `run_cutadapt_on_dataset`
  - ds: 552

- `run_cutadapt_on_dataset`
  - ds: 557

- `map_ds_against_hg`

- `download_hgtrack`

- `map_ds_against_hg`

- `annotate`

- `annotate`

- `rnatypes_of_dataset`

- `rnatypes_of_dataset`

- `correlate_seq_with_rtpcr`
  - method: quantile

- `summarize_matypes`

- `plot_rnatypes`

- `tabulate_cutread_lengths`

- `tabulate_cutread_lengths`

- `correlate_seq_with_rtpcr`
  - method: raw

- `summarize_readlengths`

- `plot_readlengths`
  - path: cutreads
  - name: 552

- `map_ds_against_mirbase`
  - sort_a_bam_file

- `download_mirbase`

- `extract_mature_hsa`

- `xls_to_colon_separated_csv`
  - name: rtpcr/CT -values-original

- `download_hgtrack`

- `extract_mature_hsa`

- `xls_to_colon_separated_csv`
  - name: mirbase/mature-all-2

- `download_mirbase`

- `download_mirbase_rtpcr`

- `xls_to_colon_separated_csv`
  - name: rtpcr/CT -values-original

- `xls_to_colon_separated_csv`
  - name: mirbase/mature-all-2
Conclusion

Snakemake is a new workflow system that provides:

- an easy pythonic textual representation
- multiple wildcards in filenames
- dynamic update of job DAG
- implicit parallelization and dependency resolution
- job scheduling considering threads, priorities and input size
- cluster and batch support

https://bitbucket.org/johanneskoester/snakemake

depends on Python ≥ 3.2

Early adopters:
Shirley Liu’s lab, Broad Institute
Genome of the Netherlands Project