ALGORITHMS FOR
MASSIVE TERRAINS AND GRAPHS
— THE PROGRAM OF THE DAY

— External Memory Pipelining Made Easy With TPIE

Lars Arge, Mathias Rav, Svend C. Svendsen, Jakob Truelsen
— THE PROGRAM OF THE DAY

— External Memory Pipelining Made Easy With TPIE
  Lars Arge, Mathias Rav, Svend C. Svendsen, Jakob Truelsen

— 1D and 2D Flow Routing on a Terrain
  Lars Arge, Aaron Lowe, Svend C. Svendsen, Pankaj K. Agarwal
THE PROGRAM OF THE DAY

— External Memory Pipelining Made Easy With TPIE
   Lars Arge, Mathias Rav, Svend C. Svendsen, Jakob Truelsen

— 1D and 2D Flow Routing on a Terrain
   Lars Arge, Aaron Lowe, Svend C. Svendsen, Pankaj K. Agarwal

— Practical I/O-Efficient Multiway Separators
   Svend C. Svendsen
THE PROGRAM OF THE DAY

- External Memory Pipelining Made Easy With TPIE
  Lars Arge, Mathias Rav, Svend C. Svendsen, Jakob Truelsen

- 1D and 2D Flow Routing on a Terrain
  Lars Arge, Aaron Lowe, Svend C. Svendsen, Pankaj K. Agarwal

- Practical I/O-Efficient Multiway Separators
  Svend C. Svendsen

- Learning to Find Hydrological Corrections
  Lars Arge, Allan Grønlund, Svend Christian Svendsen, Jonas Tranberg
— TERRAIN AND BIG DATA

— Present: Terrain is collected with LiDAR

Source: LiDAR America
TERRAIN AND BIG DATA

Present: Terrain is collected with LiDAR

Denmark - Shuttle Radar Topography Mission
90 meter resolution
4,000,000 points

Source: LiDAR America
— **TERRAIN AND BIG DATA**

— Present: Terrain is collected with LiDAR

— Denmark - Shuttle Radar Topography Mission
  90 meter resolution
  4,000,000 points

— Denmark - Danish Elevation Model
  40 centimeter resolution
  415,000,000,000 points

Source: LiDAR America
— TERRAIN AND BIG DATA

Source: Scalable algorithms for large high-resolution terrain data, Mølhave et al.
I/O-EFFICIENT ALGORITHMS

RAM model

CPU

Random Access

Internal Memory
(RAM)
Capacity: $\infty$
I/O-EFFICIENT ALGORITHMS

- RAM model

- I/O-Model

- Hard drives move blocks of data and are slow

- I/O-Efficient Algorithms: Move as few blocks as possible
--- I/O-EFFICIENT ALGORITHMS

---

--- I/O-Model by Aggarwal and Vitter (CACM 1988)

---

--- $N = \# \text{ of items in input}$

--- $B = \# \text{ of items in a block}$

--- $M = \# \text{ of items in memory (capacity)}$

--- Reading elements: $\text{Scan}(N) = \Theta(N/B)$

--- Sorting elements: $\text{Sort}(N) = \Theta\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right)$
External Memory Pipelining Made Easy With TPIE

Lars Arge, Mathias Rav, Svend C. Svendsen, Jakob Truelsen

IEEE BigData 2017
- **I/O-EFFICIENT ALGORITHMS IN PRACTICE**

- TPIE: The Templated Portable I/O Environment
- Hide low-level details while maintaining performance
- File streams: reading and writing to disk
- Provides implementations of fundamental algorithms
- Used both commercially and in research
TPIE PIPELINING

Imperative-style algorithm

- Read input
- Sort
- Transform
- Sort
- Output
Imperative-style algorithm

Read input → Sort → Transform → Sort → Output

Standard components
Imperative-style algorithm

- **Read input**: $O(\text{Scan}(N))$
- **Sort**: $O(\text{Sort}(N))$
- **Transform**: $O(\text{Scan}(N))$
- **Sort**: $O(\text{Sort}(N))$
- **Output**: $O(\text{Scan}(N))$
— TPIE PIPELINING

Imperative-style algorithm

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read input</td>
<td>$O(\text{Scan}(N))$</td>
</tr>
<tr>
<td>Sort</td>
<td>$O(\text{Sort}(N))$</td>
</tr>
<tr>
<td>Transform</td>
<td>$O(\text{Scan}(N))$</td>
</tr>
<tr>
<td>Sort</td>
<td>$O(\text{Sort}(N))$</td>
</tr>
<tr>
<td>Output</td>
<td>$O(\text{Scan}(N))$</td>
</tr>
</tbody>
</table>

$O(\text{Sort}(N))$
--- TPIE PIPELINING ---

**Imperative-style algorithm**

<table>
<thead>
<tr>
<th>Step</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read input</td>
<td>$O(\text{Scan}(N))$</td>
</tr>
<tr>
<td>Sort</td>
<td>$O(\text{Sort}(N))$</td>
</tr>
<tr>
<td>Transform</td>
<td>$O(\text{Scan}(N))$</td>
</tr>
<tr>
<td>Sort</td>
<td>$O(\text{Sort}(N))$</td>
</tr>
<tr>
<td>Output</td>
<td>$O(\text{Scan}(N))$</td>
</tr>
<tr>
<td></td>
<td>$O(\text{Sort}(N))$</td>
</tr>
</tbody>
</table>

**Pipelined Algorithm**

<table>
<thead>
<tr>
<th>Step</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read input</td>
<td></td>
</tr>
<tr>
<td>Sort</td>
<td>$O(\text{Sort}(N))$</td>
</tr>
<tr>
<td>Transform</td>
<td>$O(\text{Scan}(N))$</td>
</tr>
<tr>
<td>Sort</td>
<td>$O(\text{Sort}(N))$</td>
</tr>
<tr>
<td>Output</td>
<td>$O(\text{Sort}(N))$</td>
</tr>
</tbody>
</table>
def transform(input_file, output_file):
    while input_file.can_read():
        x = input_file.read()
        output_file.write(f(x))

class TransformComponent:
    def push(x):
        dest.push(f(x))
— TPIE PIPELINING
— Blocking Components
— Identifying Phases
— **TPIE PIPELINING**

— Blocking Components

— Identifying Phases

```
        Read input
         ↓
  Transform
         ↓
Sort runs
         ↓
Merge runs
         ↓
Produce Output
```

```
        Transform
         ↓
  Transform
         ↓
Sort runs
         ↓
Merge runs
```
— **TPIE PIPELINING**

— Blocking Components
— Identifying Phases
— Memory Management
— TPIE PIPELINING

— Blocking Components
— Identifying Phases
— Memory Management
— Parallelisation
— Progress Tracking
1D and 2D Flow Routing on a Terrain
Lars Arge, Aaron Lowe, Svend C. Svendsen, Pankaj K. Agarwal
ACM SIGSPATIAL 2020
Invited to ACM TSAS
— FLOOD MODEL

— Single Flow Direction Model: Water on a vertex $v$ flows to a single neighbor $u$ along an edge
— Multiflow Direction Model: Water on a vertex $v$ flows to multiple neighbors
THE PROBLEM

- Rain distribution: $\mathcal{R}(v) : V \to \mathbb{R}_{\geq 0}$

- Terrain-flood query: Given a rain distribution $\mathcal{R}$ and a time $t$, determine which vertices of $\Sigma$ are flooded.

- Flood-time query: Given a rain distribution $\mathcal{R}$, for each vertex $q \in \Sigma$, determine the time $t$ that $q$ becomes flooded.
## STATE OF THE ART

- H: height of the merge tree
- X: number of depressions

<table>
<thead>
<tr>
<th>Model</th>
<th>Flood-time query</th>
<th>Terrain-flood query</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFD RAM-model</td>
<td>$O(N \log N)$ [1]</td>
<td>$O(N \log N)$ [1]</td>
</tr>
<tr>
<td>SFD I/O-model</td>
<td>$O(\text{Sort}(X) \log \frac{X}{M} + \text{Sort } N)$ [2]</td>
<td>$O(\text{Sort}(N) + \text{Scan}(H \cdot X))$ [3]</td>
</tr>
</tbody>
</table>

STATE OF THE ART

- H: height of the merge tree
- X: number of depressions

<table>
<thead>
<tr>
<th>Model</th>
<th>Flood-time query</th>
<th>Terrain-flood query</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFD RAM-model</td>
<td>$O(N \log N)$ [1]</td>
<td>$O(N \log N)$ [1]</td>
</tr>
<tr>
<td>SFD I/O-model</td>
<td>$O(\text{Sort}(X) \log \frac{X}{M} + \text{Sort } N)$ [2]</td>
<td>$O(\text{Sort}(N) + \text{Scan}(H \cdot X))$ [3]</td>
</tr>
</tbody>
</table>


*: assuming merge tree fits in memory
## STATE OF THE ART

- **H**: height of the merge tree
- **X**: number of depressions

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Flood-time query</th>
<th>Terrain-flood query</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFD RAM-model</td>
<td>$O(N \log N)$ [1]</td>
<td>$O(N \log N)$ [1]</td>
</tr>
<tr>
<td>SFD I/O-model</td>
<td>$O(\text{Sort}(X) \log \frac{X}{M} + \text{Sort } N)$ [2]</td>
<td>$O(\text{Sort}(N) + \text{Scan}(H \cdot X))$ [3]</td>
</tr>
<tr>
<td>MFD RAM-model</td>
<td>**$O\left(N(</td>
<td>\mathcal{R}</td>
</tr>
<tr>
<td>MFD I/O-model</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Assuming merge tree fits in memory
** $O(NX + N \log N)$ pre-processing

--- STATE OF THE ART ---

- H: height of the merge tree
- X: number of depressions

<table>
<thead>
<tr>
<th></th>
<th>Flood-time query</th>
<th>Terrain-flood query</th>
</tr>
</thead>
<tbody>
<tr>
<td>SFD RAM-model</td>
<td>$O(N \log N)$ [1]</td>
<td>$O(N \log N)$ [1]</td>
</tr>
<tr>
<td>SFD I/O-model</td>
<td>$O(\text{Sort}(X) \log \frac{X}{M} + \text{Sort} N)$ [2]</td>
<td>$O(\text{Sort}(N) + \text{Scan}(H \cdot X))$ [3]</td>
</tr>
<tr>
<td>MFD RAM-model</td>
<td>$**O\left( N(</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>*** $O(</td>
<td>\phi</td>
</tr>
<tr>
<td></td>
<td></td>
<td>**: $O(NX + N \log N)$ pre-processing</td>
</tr>
<tr>
<td>MFD I/O-model</td>
<td>* $O(\text{Sort}(N +</td>
<td>\phi</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[5] 2021, Arge, Lowe, Svendsen, Agarwal,</td>
</tr>
</tbody>
</table>

H: height of the merge tree
X: number of depressions
FLOW FUNCTIONS

- Rain distribution: $\mathcal{R}(v, t) : \mathbb{V} \times \mathbb{R} \rightarrow \mathbb{R}_{\geq 0}$
  piece-wise constant changing at times $\{t_0, t_1, \ldots, t_K\}$

- Flow function $\phi_v$: the flow rate over a vertex $v$

- $\phi_v$ is a piece-wise constant function

- $\phi_v$ changes only at spill events and when the rain distribution changes
— **SADDLES AND NON-SADDLES**

— Saddle Vertex: $v_i$

— Sink Vertex: $u_i$

— Maximal Depression: $\alpha_i$

— Non-maximal Depression $\beta_1$
ALGORITHM FOR COMPUTING FLOW FUNCTIONS

Preprocessing:

- For all \( v \in \Sigma \), compute the maximal depression containing \( v \) \( \text{(Sort}(N) \ [1]) \)
- For all \( v \in \Sigma \), compute the volume of the depression \( \alpha_v \) \( \text{(Sort}(N) \ [2]) \)
- For each maximal depression \( \beta \), compute the amount of rain falling directly in \( \beta \) \( O(\text{Sort}(N) + \text{Sort}(|\mathcal{R}|)) \)

[1] 2009, Arge and Revsbaek
ALGORITHM FOR COMPUTING FLOW FUNCTIONS

Sweep:

- At height \( l \) maintain depressions \( \alpha_i \) in the sublevel set \( h < l \)
ALGORITHM FOR COMPUTING FLOW FUNCTIONS

Sweep:

- At height $l$ maintain depressions $\alpha_i$ in the sublevel set $h < l$
— ALGORITHM FOR COMPUTING FLOW FUNCTIONS

— Sweep:
  — At height $l$ maintain depressions $\alpha_i$ in the sublevel set $h < l$
ALGORITHM FOR COMPUTING FLOW FUNCTIONS

Sweep:

- At height $l$ maintain depressions $\alpha_i$ in the sublevel set $h < l$
- For each $\alpha_i$: maintain
  - $E(\alpha_i)$: the edges crossing the sweep line into $\alpha_i$
  - For each $e \in E(\alpha_i)$: maintain $\phi_e(t)$
  - $F_{\alpha_i}(t)$: fill-rate function of $\alpha_i$
— **NON-SADDLE VERTICES**

— For each $\alpha_i$: maintain
  
  — $E(\alpha_i)$: the edges crossing the sweep line into $\alpha_i$
  
  — For each $e \in E(\alpha_i)$: maintain $\phi_e(t)$
  
  — $F_{\alpha_i}(t)$: fill-rate function of $\alpha_i$

— Whenever we cross a non-saddle:
  
  — Compute $\phi_v = R(v, t) + \sum_{e \in E(\alpha)} \phi_e(t)$
  
  — For each outgoing edge $e$: Compute $\phi_e(t) = w_e \cdot \phi_v(t)$
  
  — Update $E(\alpha)$: remove incoming edge, add outgoing edges
— **SADDLE VERTICES**

— For each $\alpha_i$: maintain
  
  — $E(\alpha_i)$: the edges crossing the sweep line into $\alpha_i$
  
  — For each $e \in E(\alpha_i)$: maintain $\phi_e(t)$
  
  — $F_{\alpha_i}(t)$: fill-rate function of $\alpha_i$
  
— Whenever we cross a saddle:

  — Compute $\phi_e(t)$ for outgoing edges as before
  
  — Partition $E(\alpha)$ into $E(\beta_1)$ and $E(\beta_2)$
— SADDLE VERTICES

— For each \( \alpha_i \): maintain
  — \( E(\alpha_i) \): the edges crossing the sweep line into \( \alpha_i \)
  — For each \( e \in E(\alpha_i) \): maintain \( \phi_e(t) \)
  — \( F_{\alpha_i}(t) \): fill-rate function of \( \alpha_i \)

— Whenever we cross a saddle:
  — Compute \( \phi_e(t) \) for outgoing edges as before
  — Partition \( E(\alpha) \) into \( E(\beta_1) \) and \( E(\beta_2) \)
  — Compute fill-rate functions for \( \beta_1 \) and \( \beta_2 \)
  — \( F_{\beta_1}(t) = R(\beta_1, t) + \sum_{e \in E(\beta_1)} \phi_e(t) \)
— SADDLE VERTICES

— For each $\alpha_i$: maintain

  — $E(\alpha_i)$: the edges crossing the sweep line into $\alpha_i$
  — For each $e \in E(\alpha_i)$: maintain $\phi_e(t)$
  — $F_{\alpha_i}(t)$: fill-rate function of $\alpha_i$

— Whenever we cross a saddle:

  — Compute $\phi_e(t)$ for outgoing edges as before
  — Partition $E(\alpha)$ into $E(\beta_1)$ and $E(\beta_2)$
  — Compute fill-rate functions for $\beta_1$ and $\beta_2$
  — $F_{\beta_1}(t) = R(\beta_1, t) + \sum_{e \in E(\beta_1)} \phi_e(t)$
  — Assume $\beta_1$ spills first: Add the spill from $\beta_1$ to $\phi_v$
  — Update $\phi_e$ for outgoing edges $e$ and update $E(\beta_1)$, and $E(\beta_2)$
— COMBINING EVERYTHING

— Total: $O(Sort(N + |\phi|))$
COMBINING EVERYTHING

- Total: $O(\text{Sort}(N + |\phi|))$

- For each $v \in \Sigma$, we precomputed the volume of $\beta_v$.

- For each maximal depression $\alpha$, we computed the fill function $F_\alpha(t)$

- We use this to compute the fill-time of $v$!
— COMBINING EVERYTHING

— Total: $O(\text{Sort}(N + |\phi|))$

— For each $v \in \Sigma$, we precomputed the volume of $\beta_v$.

— For each maximal depression $\alpha$, we computed the fill function $F_\alpha(t)$

— We use this to compute the fill-time of $v$!
OPEN PROBLEMS

- Can we $O(N \log N)$ instead of $O(|\phi| \log |\phi|)$ in the RAM model?
- Can we get $\text{Sort}(\phi)$ in the I/O model with no assumptions on $M$?
- Output sensitive algorithm for the I/O-model?
Practical I/O-Efficient Multiway Separators

Svend C. Svendsen

Manuscript
— PLANAR SEPARATOR THEOREM
— PLANAR SEPARATOR THEOREM
— PLANAR SEPARATOR THEOREM
--- PLANAR SEPARATOR THEOREM ---

Lipton and Tarjan 1979:

\[ \frac{1}{3}N \leq |A|, |B| \leq \frac{2}{3}N \]

\[ |S| = O(\sqrt{N}) \]

\[ O(N) \text{ time} \]
— MULTIWAY SEPARATOR
MULTIWAY SEPARATOR
— MULTIWAY SEPARATOR
— MULTIWAY SEPARATOR
- **MULTIWAY SEPARATOR**

- Frederickson (1953): $r$-way separator
  - Divide a graph into $r$ regions
  - Each region has $O(N/r)$ vertices
  - $O(\sqrt{Nr})$ boundary vertices
--- MULTIWAY SEPARATOR ---

- Frederickson (1953): $r$-way separator
  - Divide a graph into $r$ regions
  - Each region has $O(\frac{N}{r})$ vertices
  - $O(\sqrt{Nr})$ boundary vertices
- Useful in the I/O-model: $N/M$-separator
- Can solve problems such as SSSP, DFS, finding strongly connected components, and topological sorting [1,2]

[1] 2003, Arge, Toma, and Zeh
## STATE OF THE ART

<table>
<thead>
<tr>
<th>I/Os</th>
<th>Internal Computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maheshwari and Zeh (2008)</td>
<td>$O(\text{Sort}(N))$</td>
</tr>
<tr>
<td>Arge, Walderveen, and Zeh (2013)</td>
<td>$O(\text{Sort}(N))$</td>
</tr>
</tbody>
</table>
— DISK PACKINGS

— Koebe (1936): every planar graph can be embedded as a disk packing
DISK PACKINGS

Koebe (1936): every planar graph can be embedded as a disk packing

Miller, Teng, Thurston, Vavasis (1997):
- At most $\frac{3}{4}N$ disks inside
- At most $\frac{3}{4}N$ disks outside
- At most $O(\sqrt{N})$ disks crossing
— DISK PACKINGS

— Koebe (1936): every planar graph can be embedded as a disk packing
— Miller, Teng, Thurston, Vavasis (1997):
  — At most $\frac{3}{4}N$ disks inside
  — At most $\frac{3}{4}N$ disks outside
  — At most $O(\sqrt{N})$ disks crossing
  — Given a disk packing: $O(\text{Scan}(N))$ I/Os
How do we compute an $r$-way separator for $r = \frac{N}{M}$?

Naively computing an $\frac{N}{M}$-way separator: $O(\text{Scan}(N) \log \frac{N}{M})$. 

COMPUTING MULTIWAY SEPARATORS ON DISK PACKINGS

AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE
COMPUTING MULTIWAY SEPARATORS ON DISK PACKINGS

- How do we compute an $r$-way separator for $r = \frac{N}{M}$?
- Naively computing an $\frac{N}{M}$-way separator: $O(\text{Scan}(N) \log \frac{N}{M})$
- We want $O(\text{Sort}(N)) = O(\text{Scan}(N) \log_{\frac{M}{B}} \frac{N}{B})$
— **COMPUTING MULTIWAY SEPARATORS ON DISK PACKINGS**

— How do we compute an \( r \)-way separator for \( r = \frac{N}{M} \)?

— Naively computing an \( \frac{N}{M} \)-way separator: \( O(\text{Scan}(N) \log \frac{N}{M}) \)

— We want \( O(\text{Sort}(N)) = O(\text{Scan}(N) \log \frac{N}{M/B}) \)

— Solution:
  — Given a disk packing \( P \), sample \( S \subseteq P \)
  — Compute multiway separator on \( S \)
  — Split \( P \) using the multiway separator (hopefully)
COMPUTING MULTIWAY SEPARATORS ON DISK PACKINGS

How do we compute an $r$-way separator for $r = \frac{N}{M}$?

Naively computing an $\frac{N}{M}$-way separator: $O(\text{Scan}(N) \log \frac{N}{M})$

We want $O(\text{Sort}(N)) = O(\text{Scan}(N) \log \frac{N}{M/B})$

Solution:
- Given a disk packing $P$, sample $S \subseteq P$
- Compute multiway separator on $S$
- Split $P$ using the multiway separator (hopefully)
- $O(\text{Sort}(N))$ but no bound on boundary vertices
COMPUTING MULTIWAY SEPARATORS ON DISK PACKINGS

How do we compute an $r$-way separator for $r = \frac{N}{M}$?

Naively computing an $\frac{N}{M}$-way separator: $O(\text{Scan}(N) \log \frac{N}{M})$

We want $O(\text{Sort}(N)) = O\left(\text{Scan}(N) \log \frac{N}{M/B}\right)$

Solution:
- Given a disk packing $P$, sample $S \subseteq P$
- Compute multiway separator on $S$
- Split $P$ using the multiway separator (hopefully)

$O(\text{Sort}(N))$ but no bound on boundary vertices

Upper bound on boundary vertices if

$$\log^3 \frac{M}{B} \log \log \frac{M}{B} \log N = O(\sqrt{M})$$
— APPLYING TO TRIANGULATIONS

— Disk Packings are difficult to compute

— Use circumcircles
— APPLYING TO TRIANGULATIONS

— Disk Packings are difficult to compute
— Use circumcircles
APPLYING TO TRIANGULATIONS

- Disk Packings are difficult to compute
- Use circumcircles
— APPLYING TO TRIANGULATIONS

— Disk Packings are difficult to compute
— Use circumcircles
— Miller, Teng, Thurston, Vavasis (1997):
  If at most $k$ disks overlap in one point,
  the separator has size $O(\sqrt{kN})$
— APPLYING TO TRIANGULATIONS

— Disk Packings are difficult to compute
— Use circumcircles
— Miller, Teng, Thurston, Vavasis (1997):
  If at most $k$ disks overlap in one point,
  the separator has size $O(\sqrt{kN})$
— This works well in practice on terrain!
— Triangulation are fast to compute ($\text{Sort}(\mathcal{N})$ [1][2])

[1] 1993, Goodrich, Tsay, Vengroff, and Vitter
OPEN PROBLEMS

- Can we get a bound on the boundary size?
- Can we do better on circumcircles?
Learning to Find Hydrological Corrections
Lars Arge, Allan Grønlund, Svend C. Svendsen, Jonas Tranberg
ACM SIGSPATIAL 2019
WHAT ARE HYDROLOGICAL CORRECTIONS?
WHAT ARE HYDROLOGICAL CORRECTIONS?
INPUT DATA

- Digital Elevation Model
  - 415 billion cells
- Road and River Networks
- Terrain Flood-Time Computation
- List of Corrections
CREATING TILE DATA
CREATING TILE DATA
— CREATING TILE DATA
— CREATING TILE DATA
— TRAINING THE ALGORITHM
EFFICIENTLY COMPUTING FILL FUNCTIONS

- Partition $E(\alpha)$ into $E(\beta_1)$ and $E(\beta_2)$
- Assume w.l.o.g. $|E(\beta_1)| < |E(\beta_2)|$
- $F_{\beta_1}(t) = R(\beta_1, t) + \sum_{e \in E(\beta_1)} \phi_e(t)$
- $F_{\alpha}(t) = F_{\beta_1}(t) + F_{\beta_2}(t)$