A Learning Algorithm for String Assembly

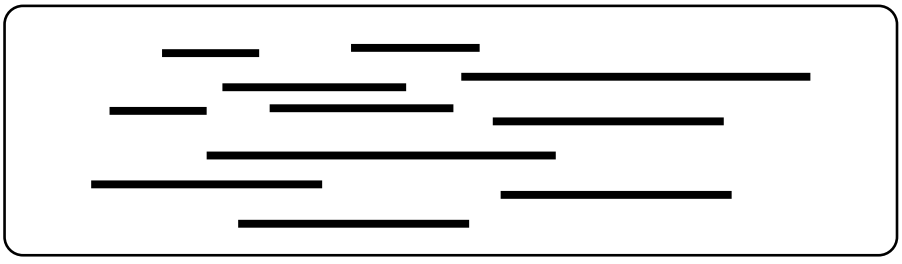
Mark K. Goldberg
Darren T. Lim
Malik Magdon-Ismail
@cs.rpi.edu

August 26, 2001

Problem Formulation

Collection of λ Original Strings

$\mathcal{P}$ (Fragmentation Process)



Collection of Substrings

Problem Statement

Plausible String Assembly Problem (PSAP): Given a collection of strings $\mathcal{C} = \{s_1, \ldots, s_N\}$, construct a physically plausible superstring.

Physically Plausible:

1. Consistent with Input Domain.

2. Consistent with Fragmenting Process.

Efficiency: Within the lifetime of the universe.

Our Approach

Consistency with:

1. Input domain: **Learning**.

2. Fragmenting: **Statistical**.

Efficiency: **Learning**.

Assembly Strategies

1. Greedy Merging.

2. Boosted Greedy Merging.

3. Full Enumeration Search.

4. Partial Enumeration Search.

Greedy Merging

Merge the fragments with longest overlap until only one fragment remains.

Example:

1. C**ATAT** **ATAT**C TATA

2. CATATC TATA

3. CATATCTATA

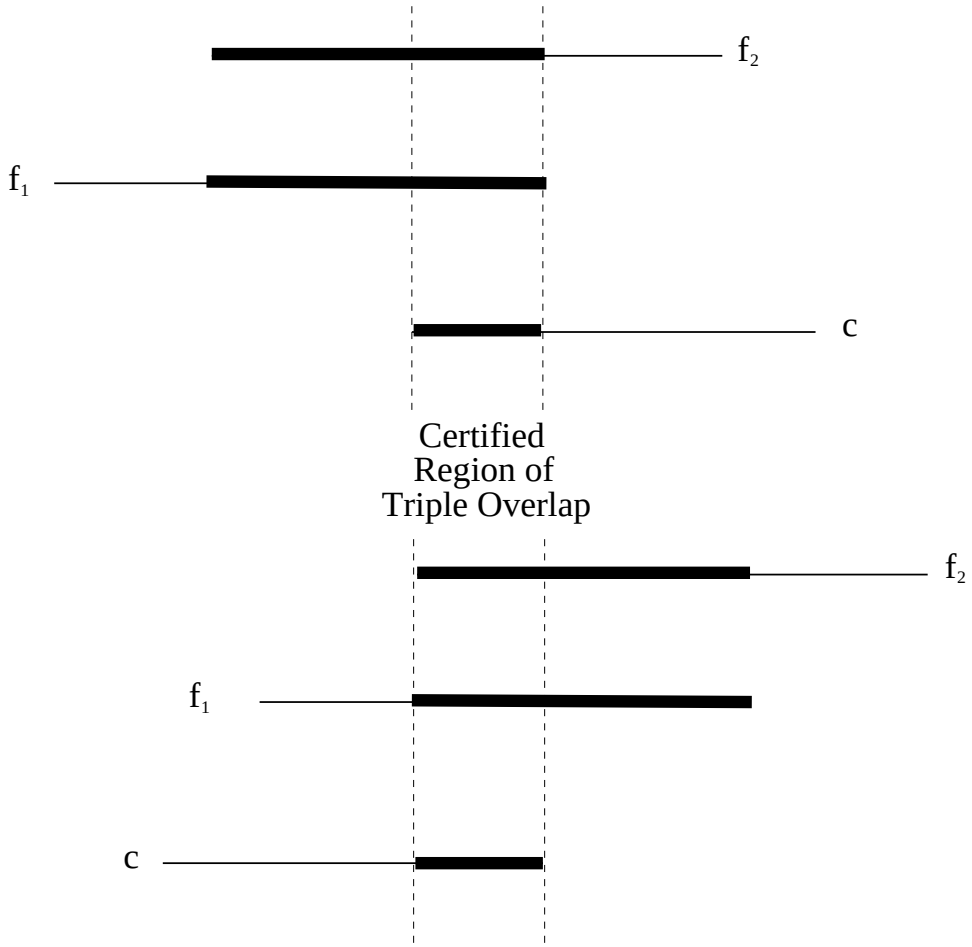
SSP: CATATATC

Greedy merging does not even work for SSP!

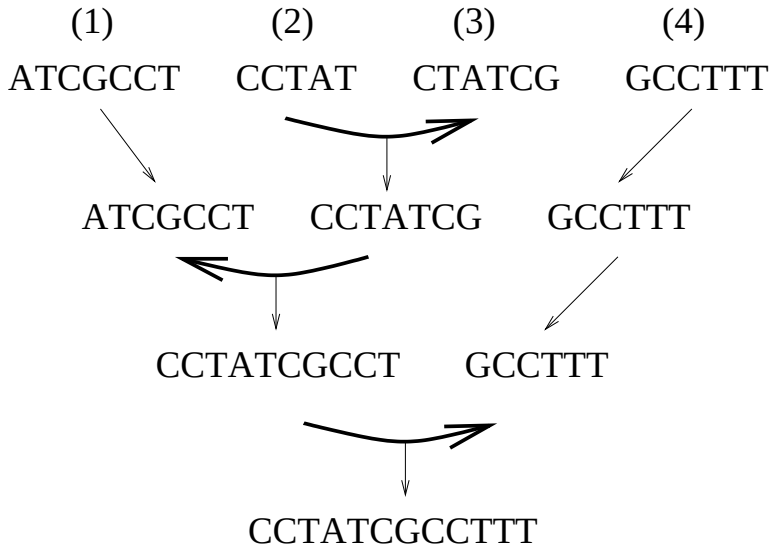
–SSP is **NP-Hard**.

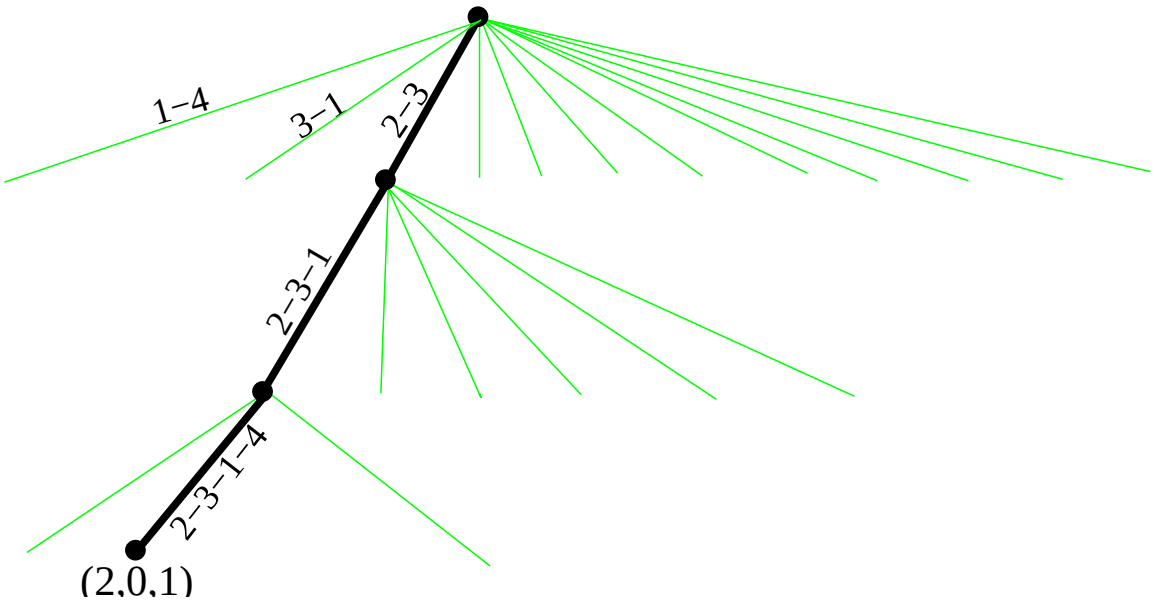
Boosted Greedy Merging

Merge the fragments with the longest certified overlap.



Full Enumeration – The Assembly Tree





Use backtracking coordinates, $\mathbf{b}$, to enumerate.

Partial Enumeration Search

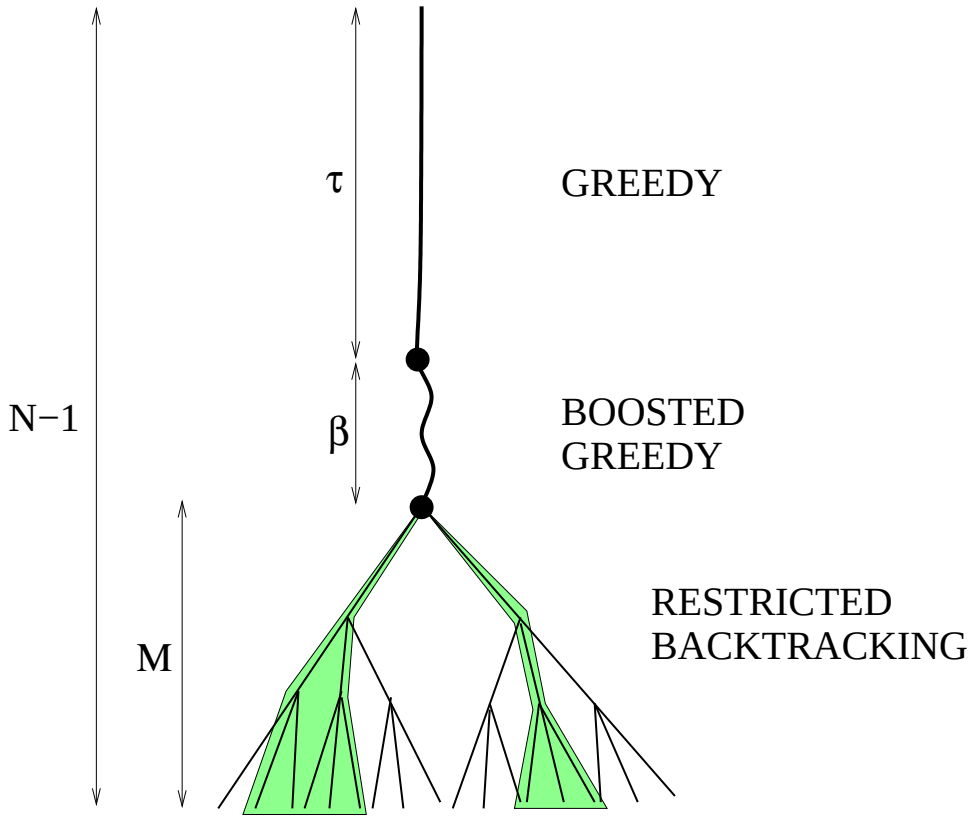
- Full enumeration is practically unfeasible.
- Only enumerate a subset of the paths:

  Restricted Backtracking

- Which subset:

  Learn the subset

Parameterized Assembly Algorithms



To be learned:

1. τ, β, $\mathbf{M}$;

2. The restricted search space, $\{\mathbf{b}\}$.

Learning Data

We need a training set of the form:

$$\mathcal{D} = \{\mathcal{C}_i, \tau_i, \beta_i, \mathbf{b}_i\}$$

$\mathcal{C}_i$**:** A collection of fragments from the input domain of interest.

$\tau_i, \beta_i, \mathbf{b}_i$**:** The correct values for the particular assembly problem.

It appears that we need an (exponentially slow) **oracle** to generate learning data.

Inverse Algorithm Engineering

No need for the exponentially slow oracle:

1. Generate string, S, from input domain.

2. Use $\mathcal{P}$ to generate fragments.

3. Obtain correct parameters $\tau_i, \beta_i, \mathbf{b}_i$.

Learning

$$M = \max_i \ length(\mathbf{b}_i)$$

$$\beta = \max_i \beta_i$$

$$\tau = (N-1) - M - \beta$$

$$\mathbf{x} \in \{\mathbf{b}\} \iff \mathbf{x} < \mathbf{b}_i \qquad \text{for some } i$$

The set $\left\{\tau, \beta, M, \{\mathbf{b}\}\right\}$ defines the **learned assembly algorithm**.

Output: Set of candidate superstrings.

Statistical Selection

Given a candidate superstring, compute:

Expected frequency of letters (δ)

Compare with

Observed frequency of letters (Δ)

Select the candidate that minimizes the discrepancy:

$$\kappa = \sum_{a \in \Sigma} (\delta_a - \Delta_a)^2$$

Experiments

$\lambda = 5$	Test on:	
	Human	H. Pylori
Human	1.007|92%	1.009|84%
H. Pylori	1.008|88%	1.003|95%

$\lambda = 6$	Test on:	
	Human	H. Pylori
Human	1.006|92%	1.007|89%
H. Pylori	1.008|92%	1.003|97%

Concluding Remarks

1. Learn on one DNA; apply to another DNA.

2. Gaps and Errors.

3. Parametric algorithm design

   —Other parameterizations?

4. More efficient clustering for backtracking.

5. Learning on one problem size and scaling up to bigger problem sizes.

6. $k-mer$ technology.

`www.cs.rpi.edu/~magdon`