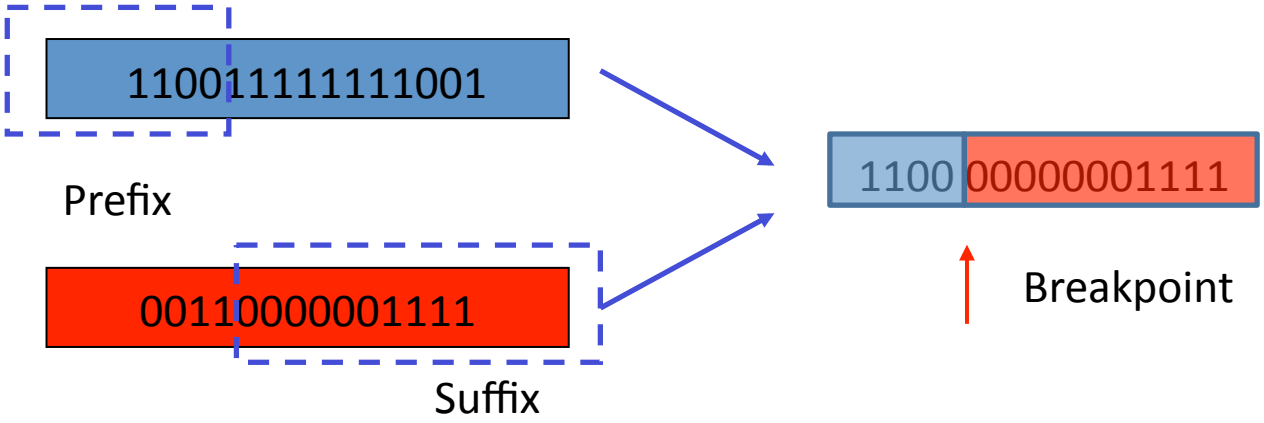
Bounds on the Minimum Mosaic of Population Sequences Under Recombination

Yufeng Wu
Dept. of Computer Science and Engineering
University of Connecticut, USA

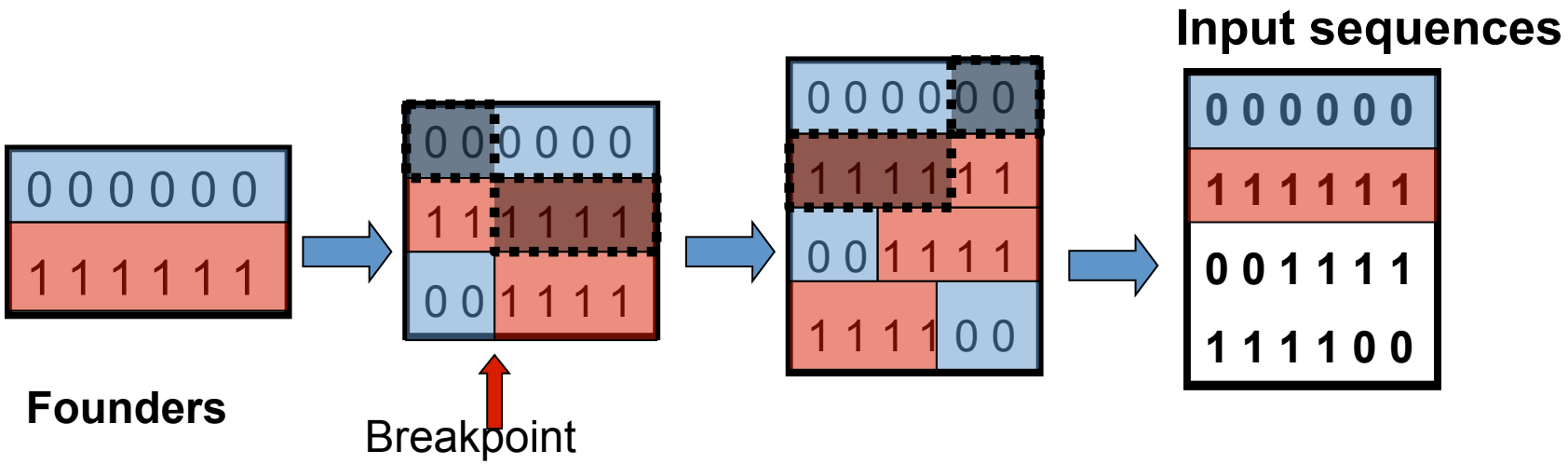
Recombination

- Population sequences: **binary**
- One of the principle genetic forces shaping sequence variations within species
- Two equal length sequences generate a third **new** equal length sequence in genealogy
  - **Spatial** order is important: different parts of genome inherit from different ancestors.



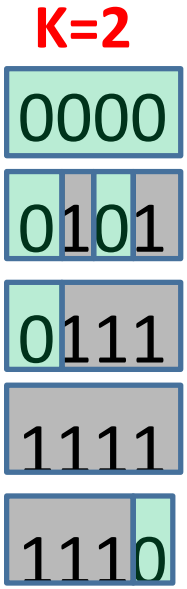
Founders of a Population

- *Current* sequences are descendents of a **small** number of **founders**.
  - A current sequence is composed of exact *blocks* from the founders, due to recombination.
  - Breakpoints: **invisible**.

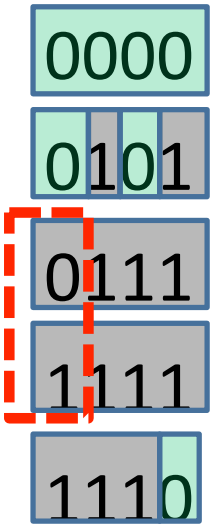


Coloring in the Mosaic Model

Mosaic model: given input sequences **M** and integer K, assume M is descendent of **K** (unknown) founder sequences



Total 5 breakpoints



Not consistent

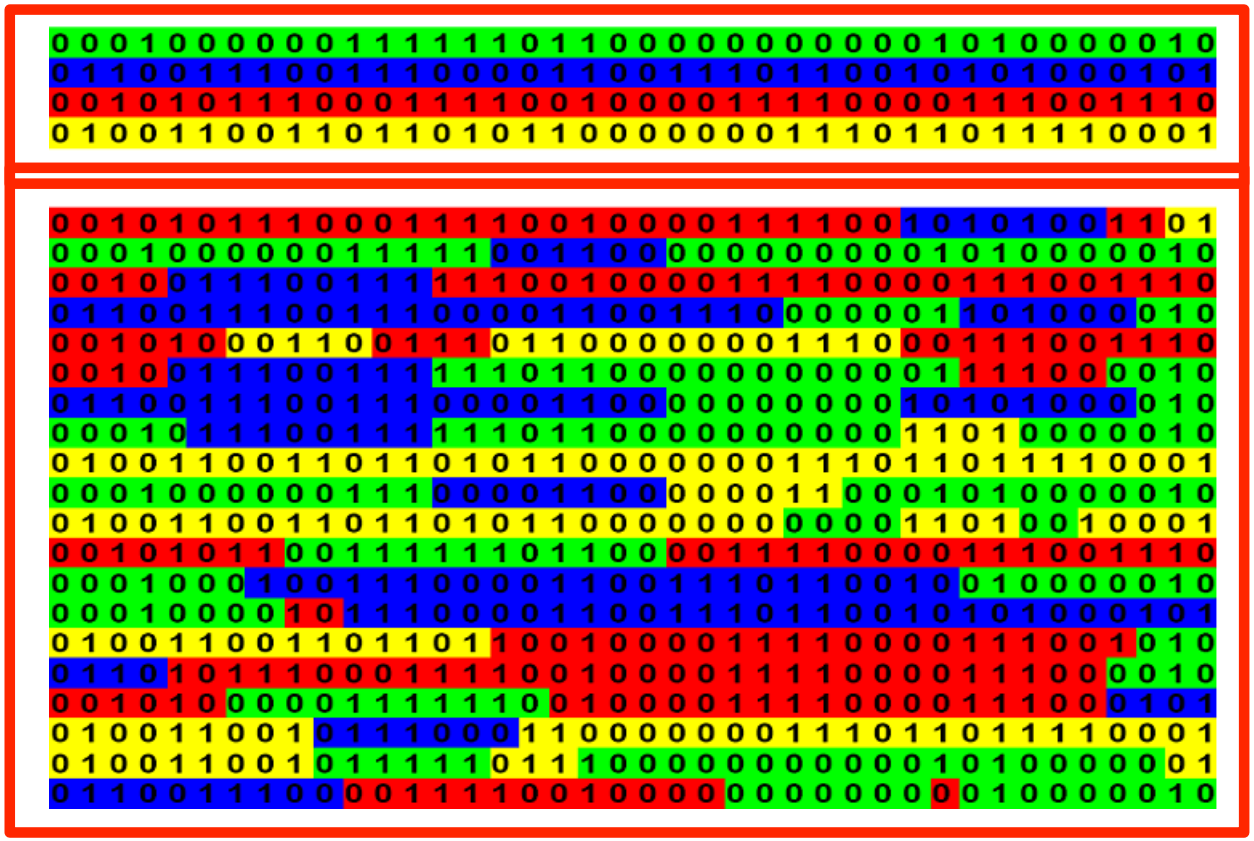
breakpoint

- **Coloring**: assign which position of a sequence is from which of K founder (color).
- **Consistency** of coloring: two sequences with distinct values at a position can **not** be colored the same

The Minimum Mosaic Problem

Given binary sequences M and the number of founder K, find a K-coloring of M to **minimize** the number of color change (recombination breakpoints)

- And find the K founder sequences (**not** part of input)



Inferred founders

Data from Rastas and Ukkonen

20 sequences
40 columns

55 breakpoints: **minimum** number of breakpoints

The Minimum Mosaic Problem

- Introduced by Ukkonen (2002)
- Main known results
  - An exponential-time algorithm (Ukkonen 2002)
  - An **exact** method that works for relatively small K and modest-sized data (Wu and Gusfield, 2007): a few seconds for 20 by 40 data with K=4 shown before. Not working well when K is large.
  - Haplovisual program and other extensions by Rastas and Ukkonen (2007).
  - Heuristic algorithm by Roli and Blum (2009)
- No polynomial-time algorithm known for K 3.
- This paper: **bounds** for the minimum number of breakpoints needed
  - **Range** of optimal solution
  - How good is a heuristic solution?

CH Bound: A Known Lower Bound

K=3

0011

1001

0010

0101

0001

Question: *at least* how many breakpoints do we need?

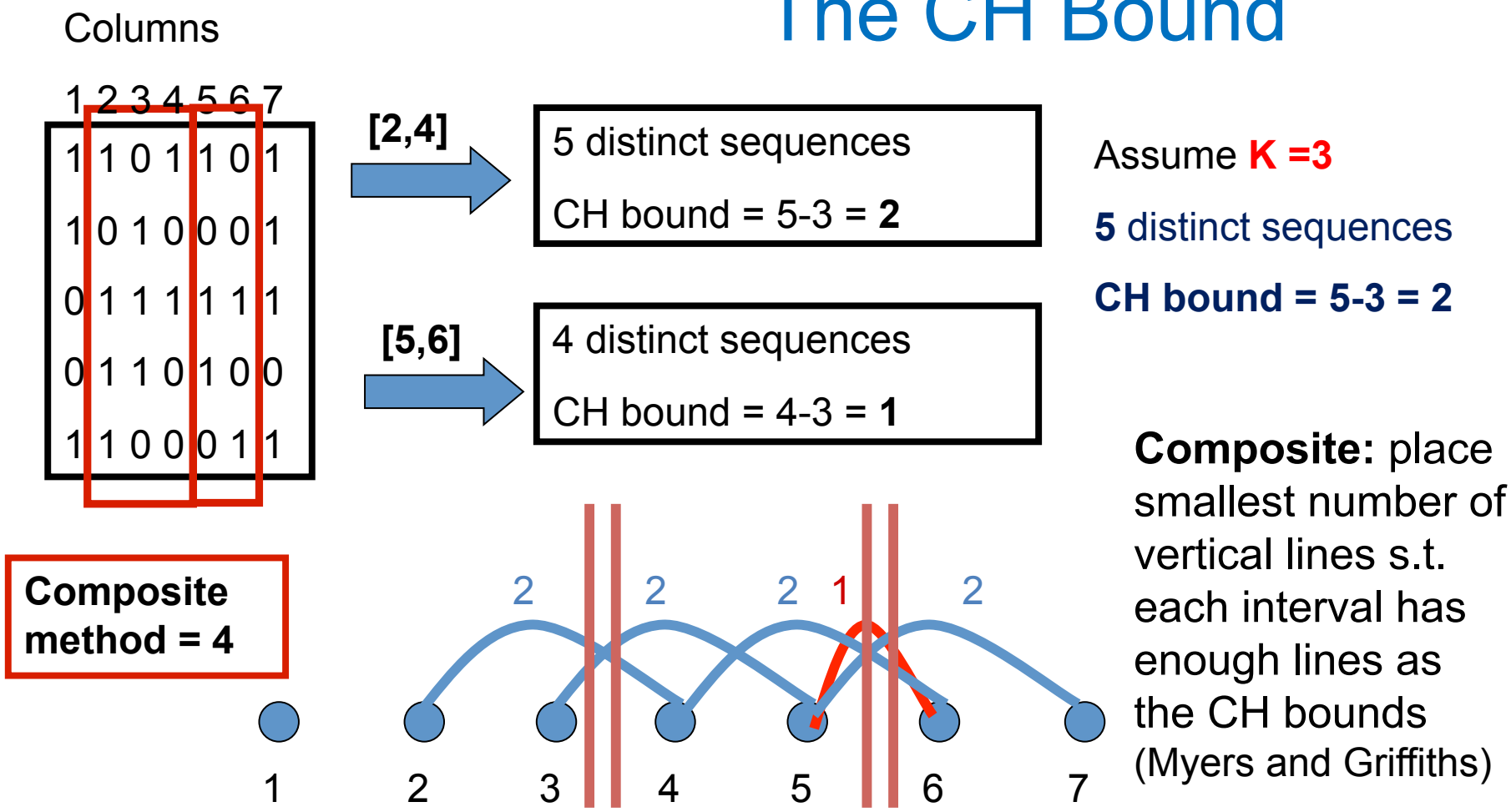
Observation: two distinct sequences can not be colored the **same** if they contain **no** breakpoints (e.g. 0011 and 1001 can not be both colored red if no breakpoints appear in either of them)

Claim: *at least* 2. Why? 5 distinct rows. At most three with no breakpoints (founders). Called the **CH** bound (see Wu and Gusfield, related to Haplotype bound by Myers and Griffiths)

Composite method (Myers and Griffiths): giving higher bound

- Compute the CH bounds for each interval [a,b]
- Combine these *local* bounds into an overall bound

The CH Bound



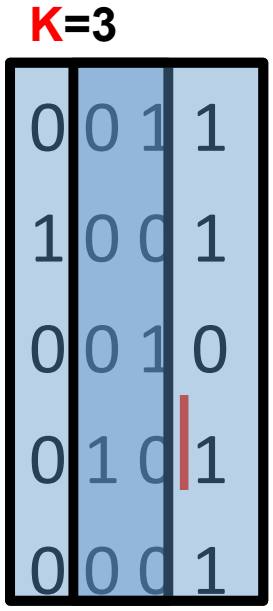
CH bound (with composite method): fast to compute but often under-estimate.

Now the **new** C bound, which is often higher.

C Bound: Placing the Breakpoints

Observation: enough breakpoints must be added s.t. within any **interval**, there are at most K distinct sequences.

When CH bound = b, it is often **not** feasible to place b breakpoints in M to achieve this property.



4 distinct sequences within [2,3]: 001,010,000,100

First improvement

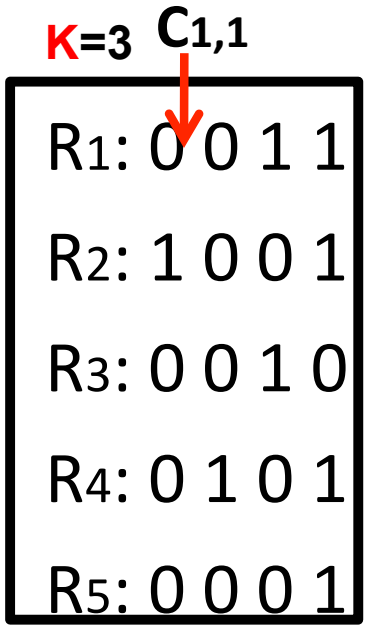
Place enough breakpoints directly in M s.t. there is at most K distinct sequences with **no** breakpoints within any interval. The **C** bound = the **minimum** number of breakpoints needed.

No polynomial time algorithm known for computing the C bound.
Use integer linear programming (**ILP**) to compute the exact C bound.

Integer Linear Programming for the C Bound

$C_{i,j}$: binary variable for row R_i and column j, 1 if there is a breakpoint between columns j and j+1 in R_i.

Optimization objective: minimize $\sum C_{i,j}$ for all R_i and j.
Constraints: ensure no more than K distinct sequences with no breakpoints within an interval (i.e. Ci,j=0 for all j inside the interval). *Omitted*.



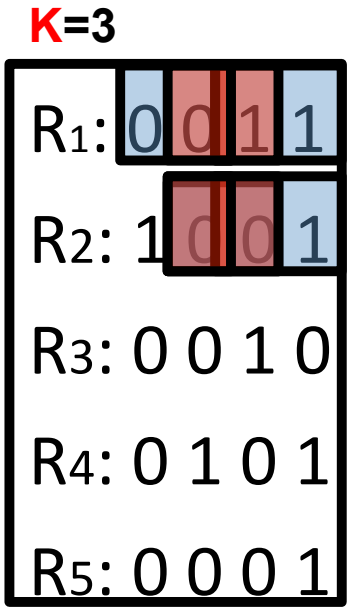
In practice, may use linear programming **relaxation** to obtain a weaker C bound (polynomial-time computable).

Theorem. Even the weaker C bound is higher than (or equal to) the CH bound (w/ composite method). That is, an efficiently computable version of the C bound is **provably** higher than the CH bound.

Incompatible Intervals

Accuracy of the C bound: often higher than the CH bound by 10-20% for certain range of data, but still much less than the exact minimum.

Overlapping row-intervals (ORI): interval [a,b] of row R_i and interval [c,d] of row R_j s.t. [a,b] and [c,d] are **not** disjoint.



Incompatible ORIs: overlapping interval [a,b] of row R_i and interval [c,d] of row R_j s.t. R_i and R_j are **not identical** within the *overlapped* region.

Overlapped region

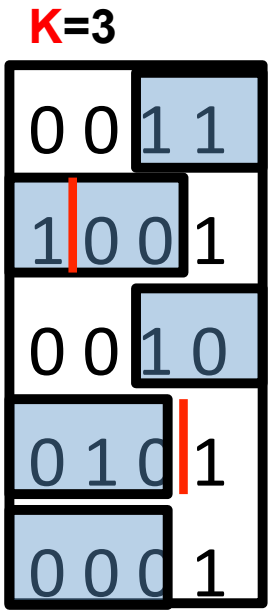
R1[3,4] and R2[2,3]: Incompatible

Lemma. Two incompatible overlapping row-intervals with no breakpoints can **not** be colored the same.

Improving C Bound with Incompatible Overlapping Row-intervals

The new C bound: the minimum number of breakpoints needed s.t. there exists no K+1 **pairwise** incompatible row-intervals with **no** breakpoints.

Lemma: the new C bound is higher than (or equal to) the original C bound.



Example: are these two breakpoints enough?

These 5 row-intervals are **pairwise**:

- Overlap and
- **Not** identical within overlapped region

But only 4 out of 5 row-intervals have **no** breakpoint. So these two breakpoints are **not** enough. This can give **higher** lower bound.

Improved C Bound

Finding IORIs: want to find incompatible ORIs (IORIs) that are not contained inside other IORIs

- The **shorter** the intervals are, the more **restrictive** they are

No polynomial-time algorithm is known for finding the overall shortest incompatible overlapping row intervals.

Heuristic algorithms: find good IORIs in a greedy way

Improved C bound: **faster** to compute using integer linear programming

- Fewer intervals are needed since the number of good IORIs are often not very large

Analytical Upper Bound

A different question: for **any** n by m binary matrix and K founders, at most how many breakpoints do we need (i.e. no matter what input matrix is given)?

Analytical upper bound: a closed-form formula

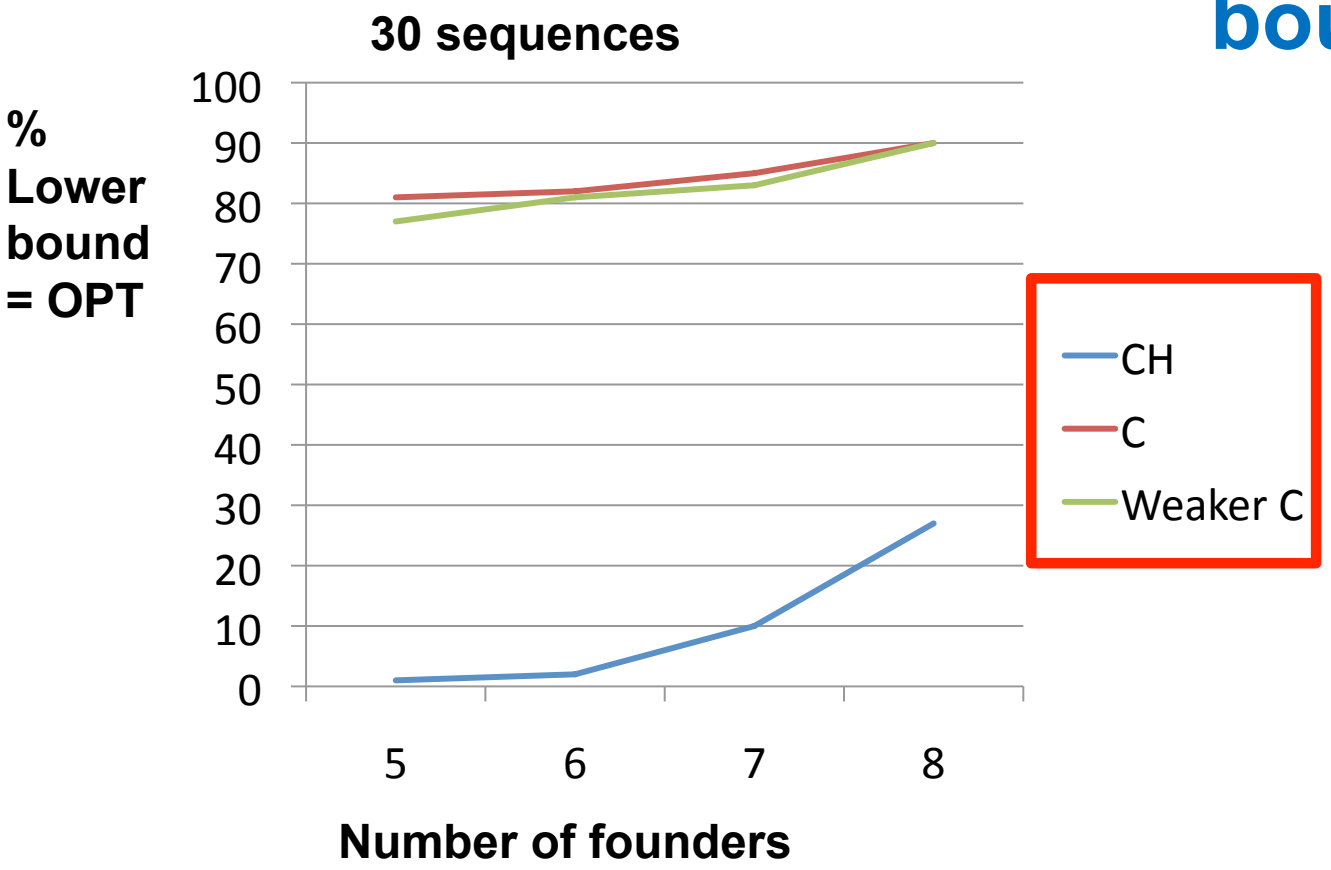
- Helps to tell the range of solution.

This paper: a general analytical upper bound (generalized the K=2 case in Rastas and Ukkonen)

Proposition 2. $B_{min} \leq (1 - \frac{1}{K})(\frac{m}{\lceil log_2(K\ ) \rceil} - 1)n$, *for any* K

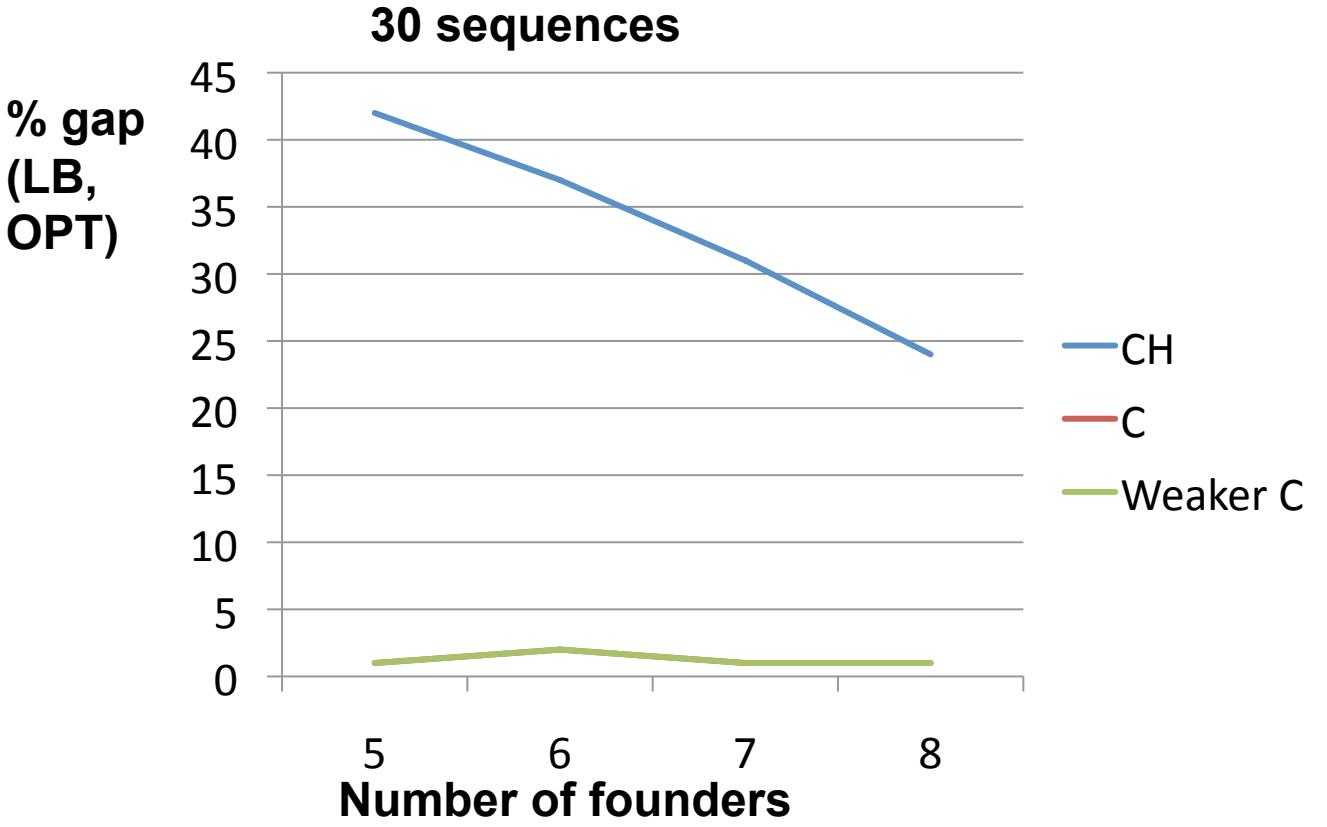
Simulation Results: how often lower bounds match OPT?

Simulated data: use Hudson's program ms to generate datasets with from 20 to 50 sequences.
Vary K (number of founders)



Computed using linear programming relaxation

Simulation Results: the gap between the lower bounds and OPT?



Running time: the C bound is slower but often computable (especially with CPLEX), or use the weaker C bound

The C bound (and even the weaker C bound) is significantly better than the previous known CH bound, and is often practically computable.

Usage of the C Bound

Application 1. Quantifying the range of OPT: especially useful when K is larger and/or the data size is larger.

Application 2. Branch and bound. Using the lower bound to trim the search space.

- Exact method in Wu and Gusfield (2007): enumerate the configuration of founders in an incremental way.
- In principle, the search can be sped up with the better lower bound (to terminate earlier)

Currently, there is some speed up but not very significant

- May need better upper bounds.

Acknowledgement

- More information available at: **http://www.engr.uconn.edu/~ywu**
- Research supported by National Science Foundation