

An Efficient Algorithm for Simulating Coalescence with Recombination

Katy L. Simonsen (Statistics)

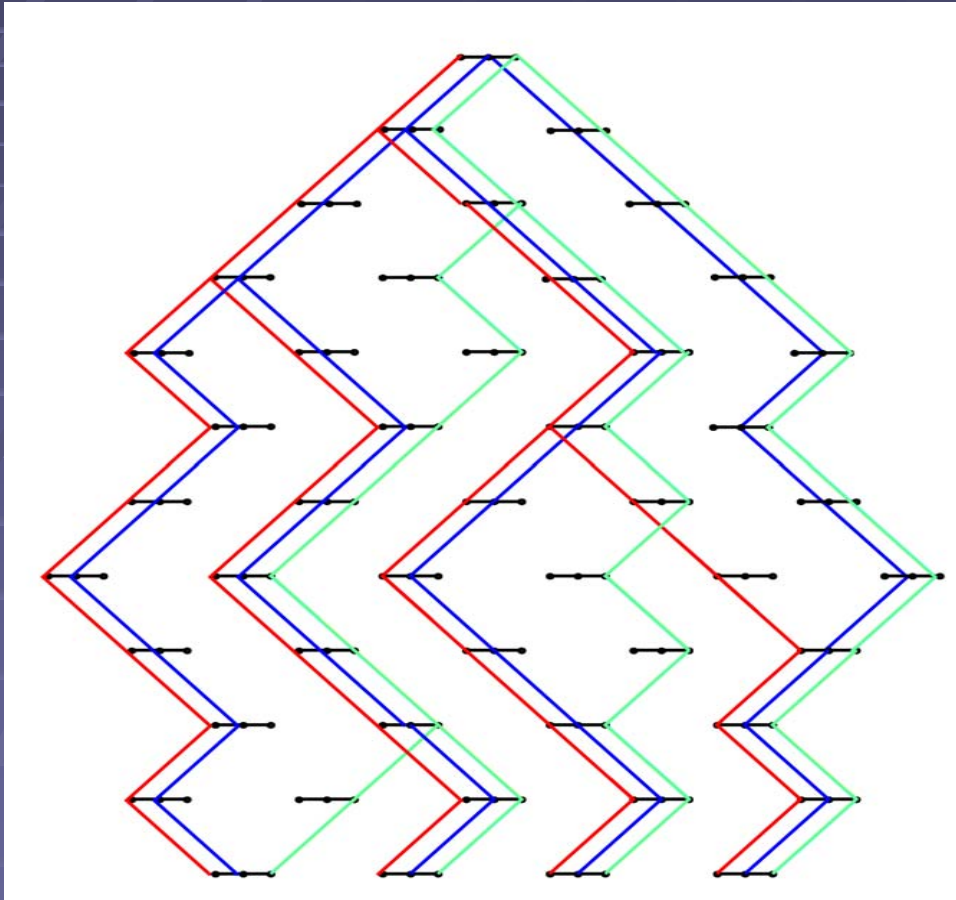
Dan A. Noland (CS and ITaP)

Chinh Le (ITaP),
Purdue University

Coalescence with Recombination

- The coalescent process is a Markov Chain process which models the ancestry of a random sample of DNA sequences from a population
- Recombination allows genetic information from two separate individuals to be combined on a single chromosome
- Tree-like structure describes possibly different ancestry at different loci

Model Parameters



Parameters:

- $n = 4$ sequences
- $m = 3$ loci
- recombination rate $r_i < 0.5$
- effective population size N
- $R_i = 2Nr_i$ for $i = 1 \dots m-1$

Why simulate coalescence?

- Simulating CWR
 - allows the generation of large samples of DNA sequences with linkage
 - statistical properties can empirically be determined
- Realistic levels of linkage disequilibrium are needed to discover effective strategies for mapping genes in natural populations
- Recently popular in human genetics literature

Original algorithm

(Simonsen & Churchill, 1997)

- steps through Markov Chain to build tree
 - series of recombination and coalescent events
- To calculate transition probabilities we need
 - the number of each type of individual present
 - there are 2^m possible types
 - the probability that each type has a recombination event
- Stores a state vector Y of length 2^m
 - Impossible to go beyond $m = 32$ (or 64)
 - Can't even store the indices of the vector let alone the vector!
- Limited by exponential nature
 - Could not simulate more than $m = 24$ loci
 - Very slow, especially for large m and R

Goal

- Want to be able to simulate hundreds, even thousands of loci (**m**)
- Also want large **R** = $2Nr$ where
 - **N** is very large ($10^3 - 10^6$),
 - $0 < r < 0.5$
- as **m** increases, **r** decreases in such a way that **mr** = constant (genome size)

What is taking so much space?

- Vector Y changes at each step
- Matrix Λ (0/1 indicator) size $2^m \times m-1$ (fixed)
 - whether recombination can happen for each type
- Need all entries of $Y \Lambda R$ at every step
 - to calculate transition probabilities for next step
- number of steps in the chain is random:
 - $G = 2H + n - 1$ where r.v. H depends on m, n, R
 - End result is a tree with G “events”
- Have to store all this: takes **memory**
- Have to calculate all this: takes **time**

How can we fix it?

- Don't want to store **Y**
- Don't want to store **$\wedge R$**
- Don't want to do all that multiplication
- Don't want to store the big tree with **G** events

Exploit structure: Y

- Y contains integers in $[0, n]$
- Sum of entries in Y is $\leq nm$
- At most nm non-zero elements out of 2^m
- Y starts with one non-zero element ($=n$) and ends with one non-zero element ($=1$)
- Y must be very sparse!
- At most 3 entries change at any one step
- Can store Y cleverly to exploit this structure

Exploit structure: ΛR

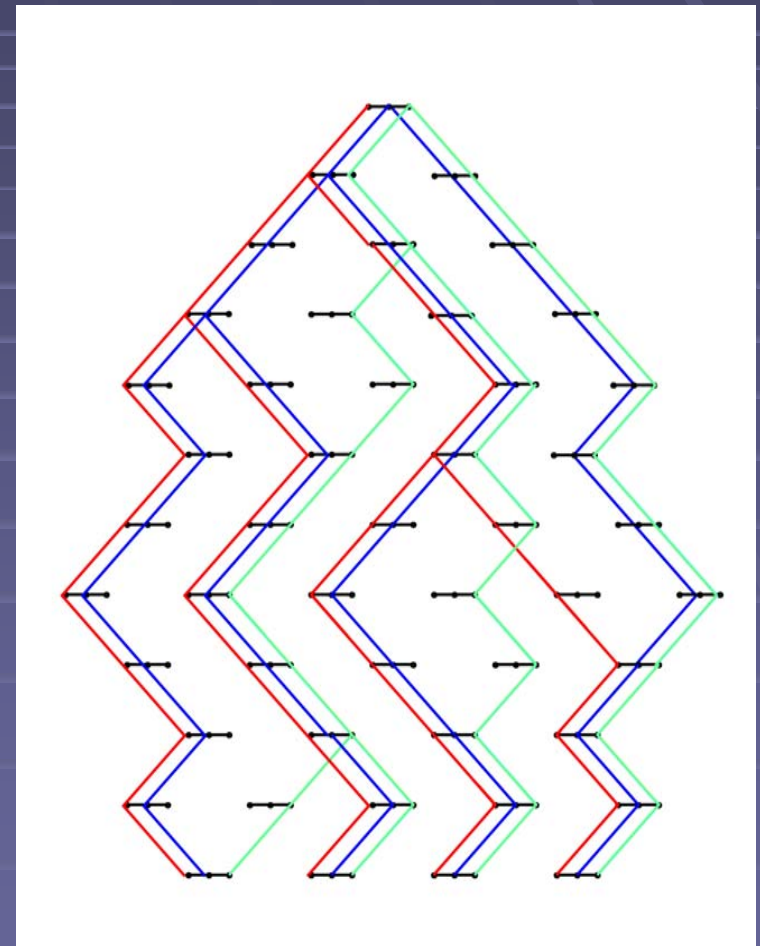
- Λ turns out not to be sparse
 - Indicates where recombination is relevant (0 – 1)
 - Looks kind of sparse for small m
 - Becomes more dense with m , and is completely dense in the limit (Lei Liu)
 - Can't get away with clever storage here
- BUT since Y multiplies ΛR we don't need it all
 - Only need rows of ΛR with >0 entries of Y
- Will recalculate ΛR “on the fly” as entries of Y become >0
 - Seems inefficient but trades recalculation for space
 - Use bitwise calculations for speed

Key data representations: $Y \wedge R$

- Store Y as a linked list
 - only the non-zero entries
- Along with each element of Y , store its corresponding $\wedge R$ row
- Store index of Y (type) as a multi-precision integer since this is still of length 2^m
 - use the Gnu Multiprecision (gmp) Library (Dan's idea)
- At each step, only recalculate $\wedge R$ row for “new” Y entries

Exploit structure: Trees

- Trees are complicated when mingled but simple if looked at one by one
- Store the “marginal” trees instead of the “joint” tree:



Key data representations: Trees

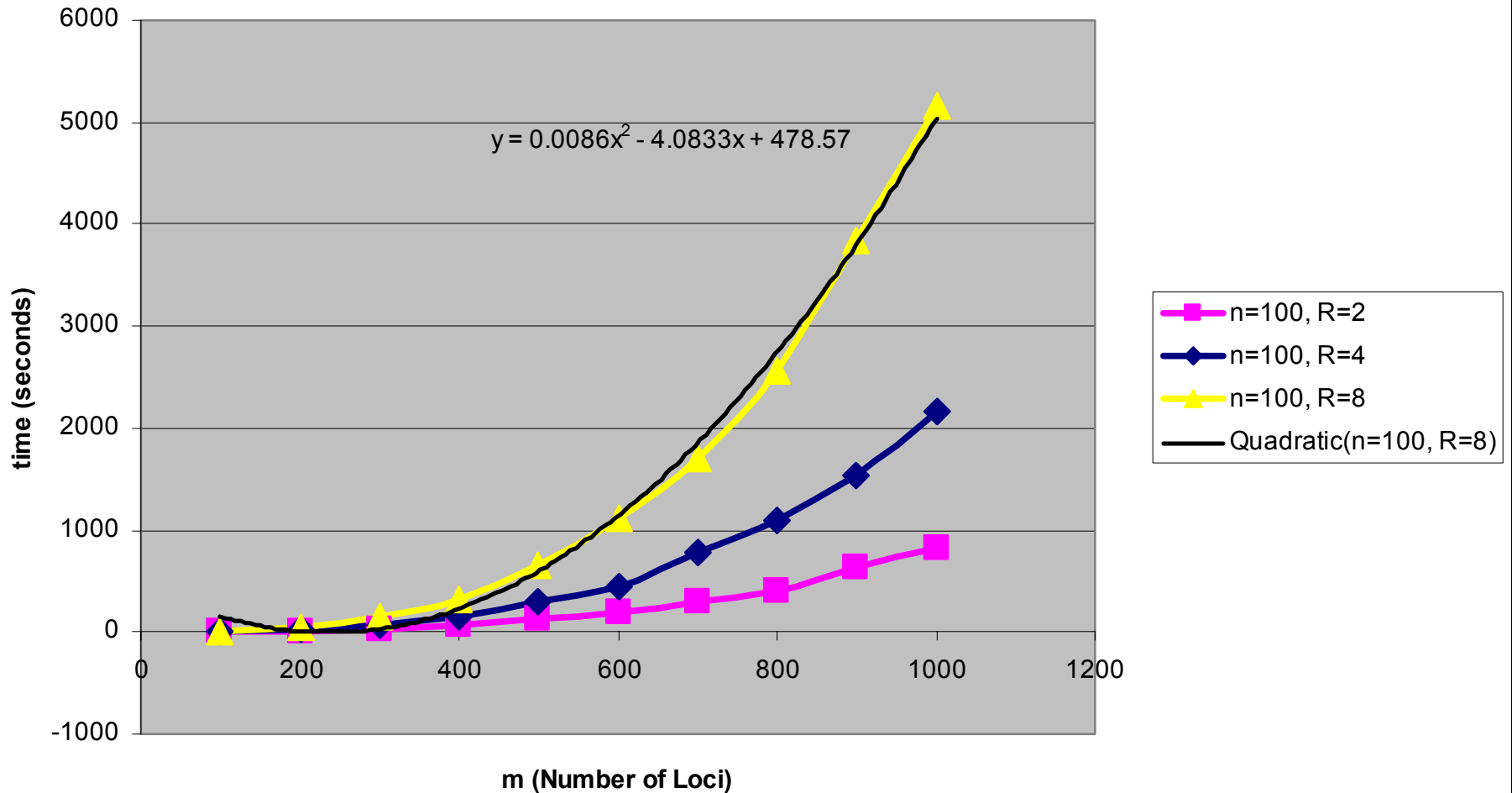
- Instead of storing combined tree, just store m trees.
- These are just binary trees so data size is FIXED at $m(2n-1)$
- Storage is independent of G which is random and could be huge
- this does lose information which could be stored separately

Timing Results

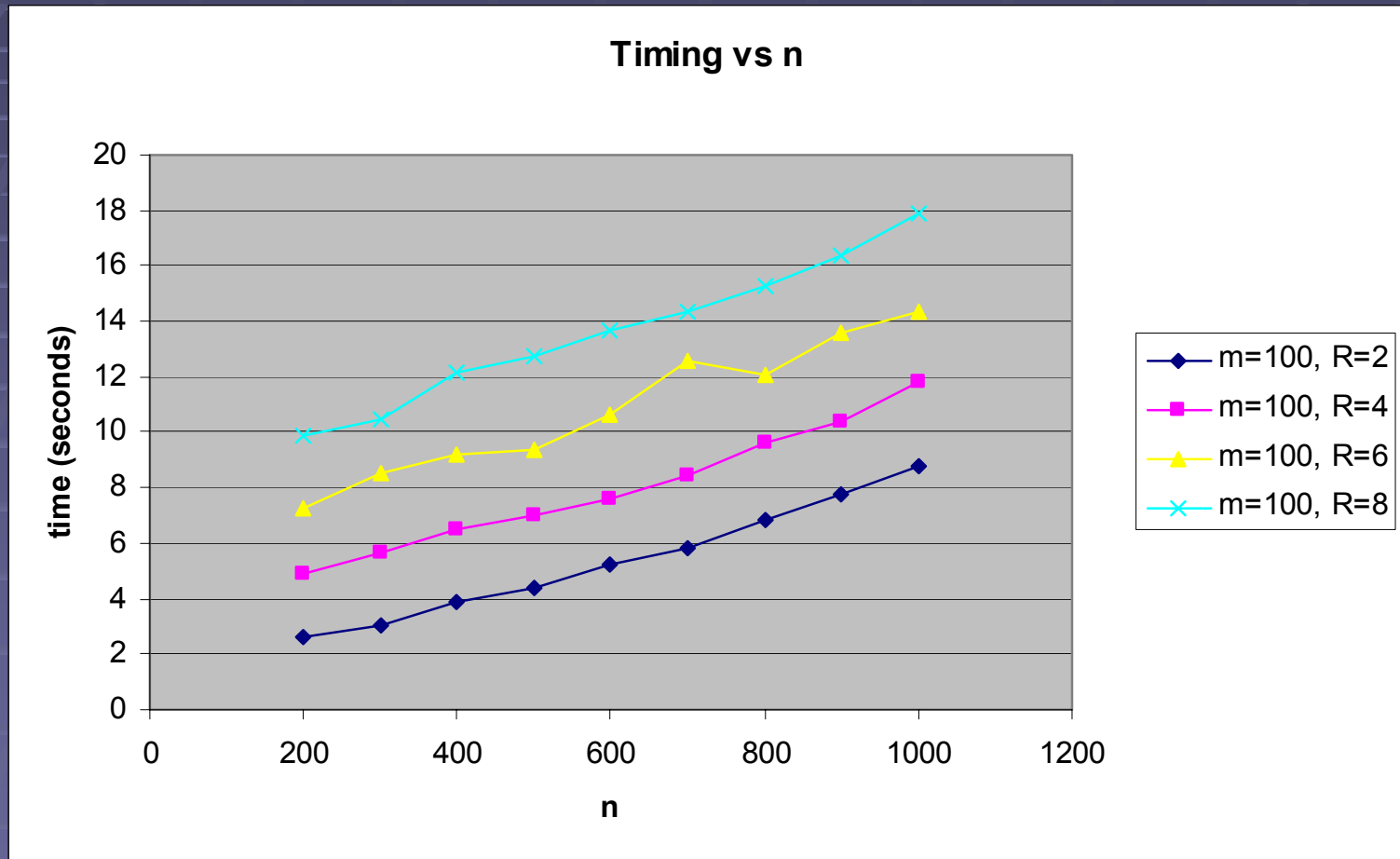
- Time is polynomial instead of exponential
 - Quadratic in m
 - Linear in n
 - Linear in R (indirectly through G)
 - $O(nm^2R)$
- Memory is $O(mn)$ and *not random*

Quadratic in m

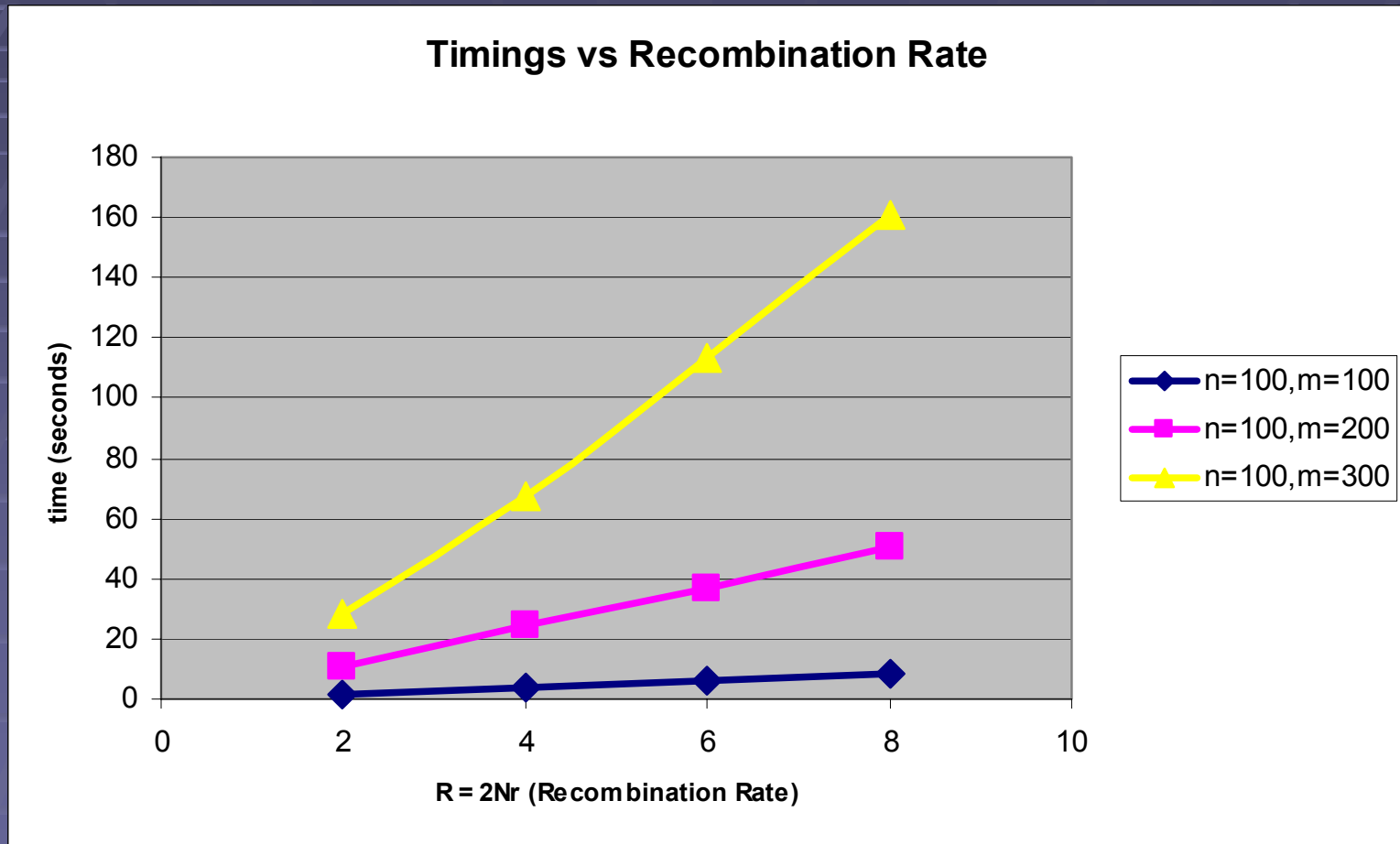
Timings vs Number of Loci



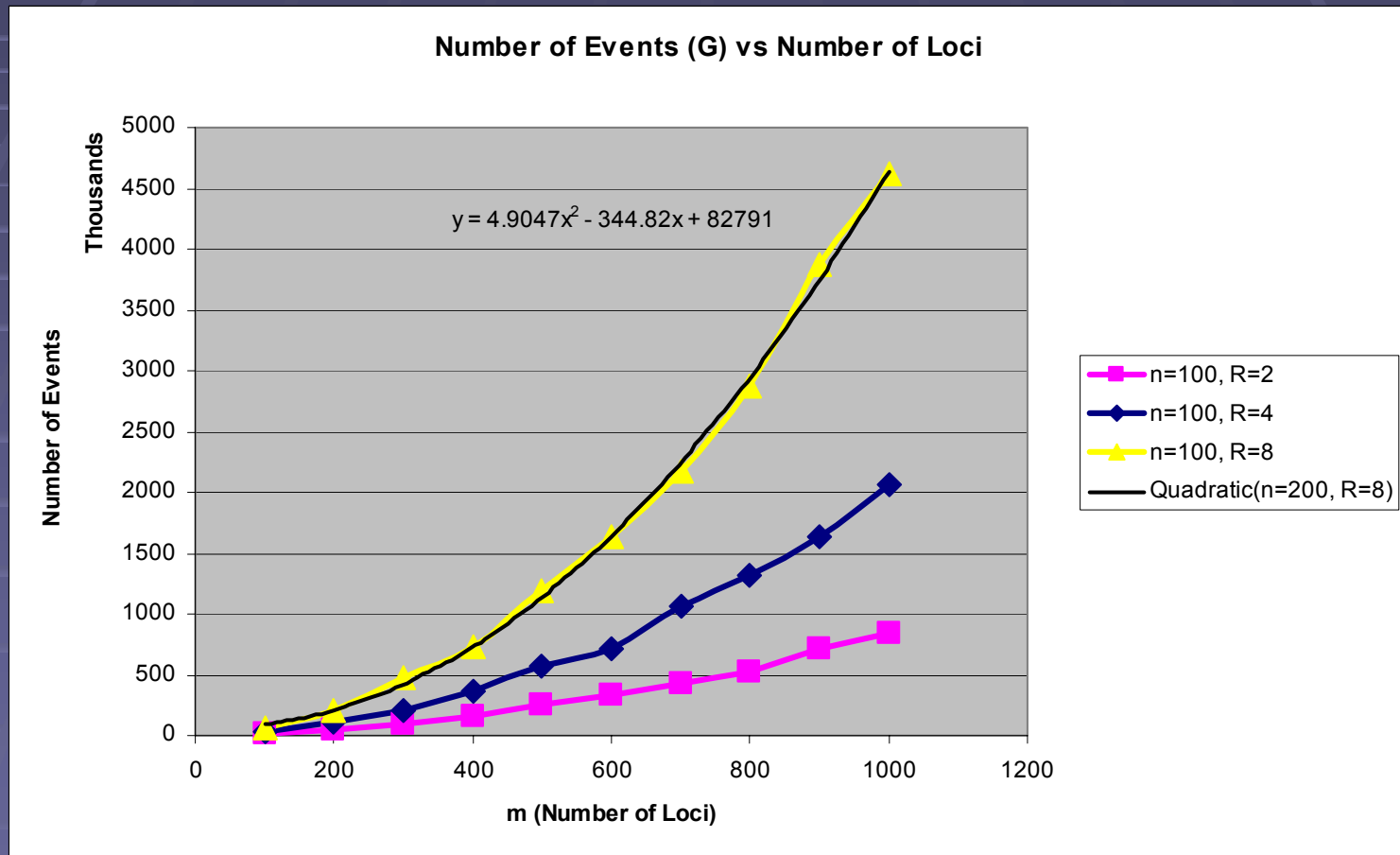
Linear in n



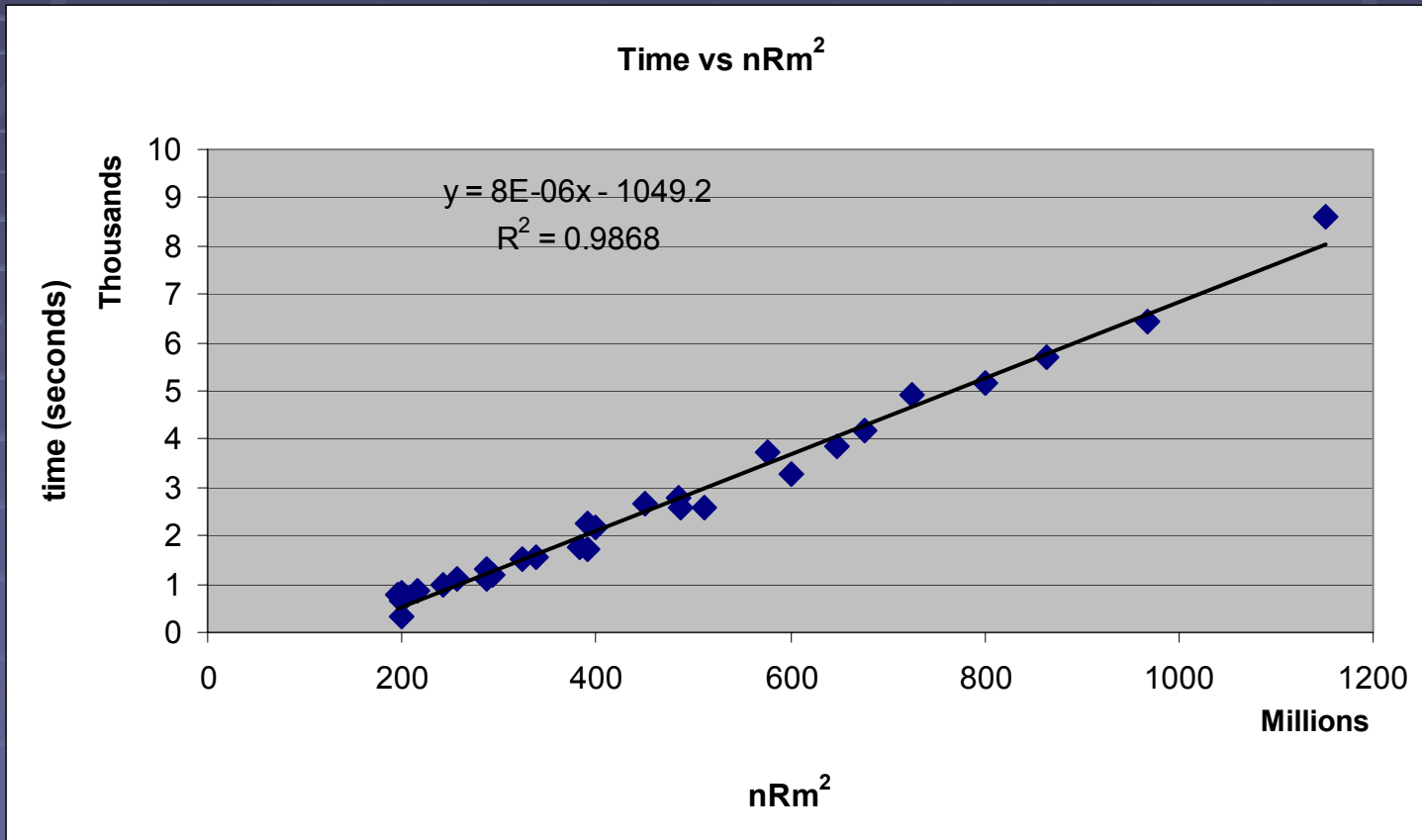
Linear in R



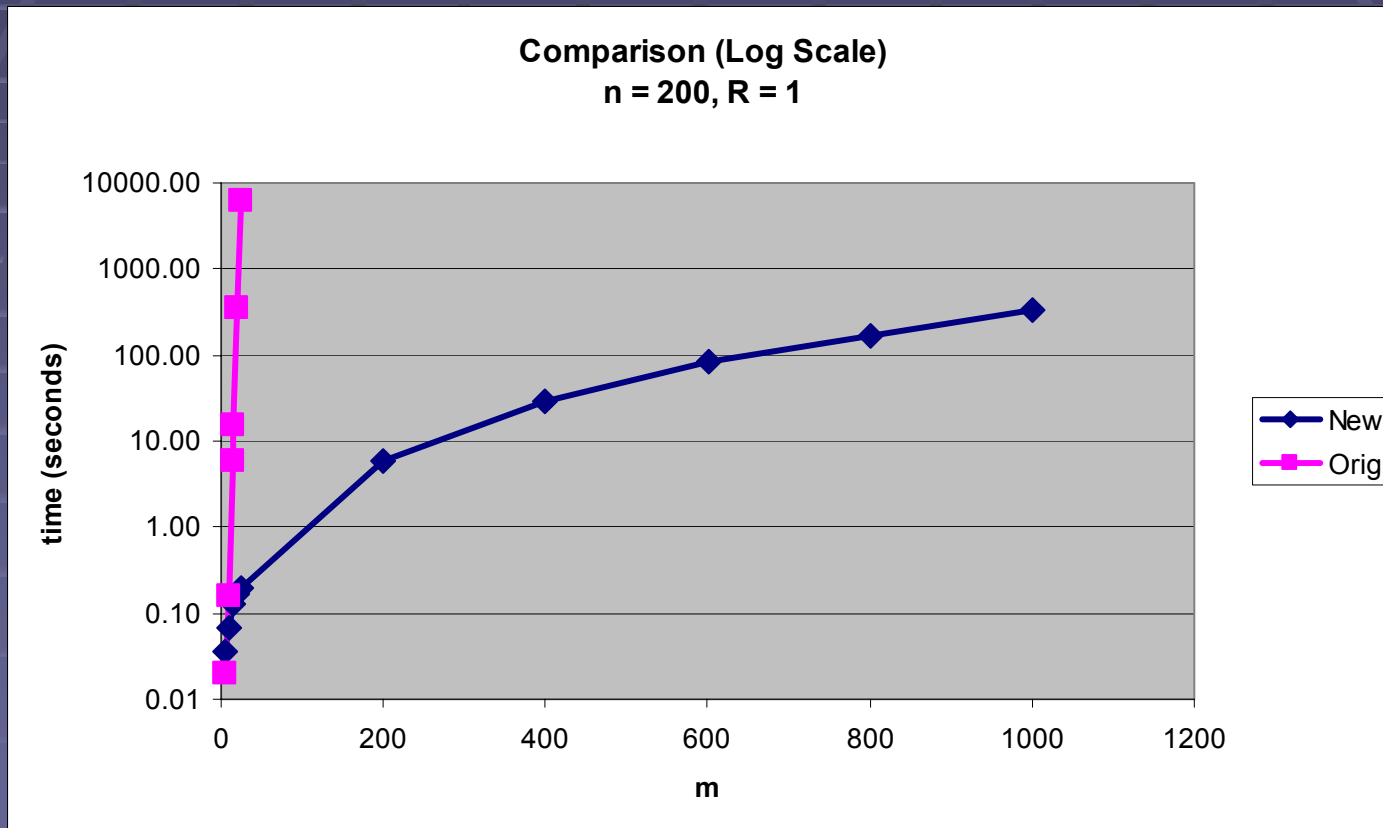
No. Events **G** is Quadratic in **m**



Time vs. nm^2R



Comparison: $\text{Log}(\text{time})$ vs. m



Limits

- So far have simulated up to $m = 5000$ loci (took 19GB)
- **Time** is proportional to the number of events (can't get around that)
- **Memory** is linear in m and n , independent of R , and not random

Applications

- Whole genome simulations
 - $m = 1000$ is the scale of markers now in use
 - $m = 10,000?$ 100,000
- Human Populations
 - understanding LD and haplotype blocks
- Natural Populations
 - statistical mapping methods
 - power calculations

Refinements

- more complex models
 - population size changes 😊
 - Windows GUI 😊
 - complex mutation models
 - population structure
 - selection

Thank You

- Dan Noland (programming, ideas)
 - Chinh Le (profiling, timing, advice)
 - Ahmed Sameh, David Moffett, and ITaP
 - Koen Verhoeven, Lei Liu
-
- <http://www.stat.purdue.edu/~simonsen/Argos/>

Human Genome Parameters (very approximate)

- 3×10^9 base pairs (3 Gb) in 23 chromosomes
- 10^7 SNPs $\Rightarrow m = 5 \times 10^5$ SNPs / chrom
- $r \approx .01/\text{Mb}$ (highly variable)
- $\approx 1\text{-}2$ SNP /Kb $\Rightarrow r \approx 10^{-5}$ between SNPs
- Historical $N \approx 10^4 - 10^5$ so $R \approx 1$
- sample sizes $n \approx 1000$ or as large as necessary (power)