Methods for big data in medical genomics

Parallel Hidden Markov Models in Population Genetics

Chris Holmes, (Peter Kecskemethy & Chris Gamble)
Department of Statistics and,
Nuffield Department of Medicine (WTCHG)
University of Oxford

January 31, 2013

Outline

- Background to increasing data complexity in modern genomics
- Challenges for statistical models used in medical genomics
- Models that scale: examples using Markov structures
  - Hidden Markov Models (HMMs)
    - Efficient HMM Algorithms via Dynamic Programming
- Parallel computing
  - Parallel computing
  - GPUs
- HMM Parallelisation
  - Simple HMM Parallelisation
  - PAC Likelihood and the Li and Stephens Model (LSM)
  - GPU-LSM
  - Chromosome Painting
  - HMM Parallelisation with Sequence Partitioning

Human genomes

- In 2000 the first draft of the Human Genome was reported
  - Took 10 years to complete
  - Cost approx \$3 billion = 1 dollar per base
- In 2012 the UK government announced plans to sequence 100,000 genomes
  - Cost in region of \$8k per genome (but I hope they're getting a better deal!)
  - Takes around 2 days to sequence a genome
- Alongside this UK Biobank is storing a detailed record of molecular features in blood, and outcome data (phenotypes) on 500,000 individuals
- Within 3 to 5 years it will be routine to have all cancers sequenced in the UK, as well as the patient DNA

Impact on Statistics

- Such developments will have a huge impact on statistics and machine learning
- We will require new methods that can
  - scale to massive (or "Big") data
  - deal with heterogeneity and loosely structured data objects on different scales (genomes to eHealth records)
  - provide robust inference (under known model mispecification)
  - ....on mixed variable types
  - .....and handle missingness
    - ...and new theory to support this
- To do this we will also need to exploit advances in computer hardware to allow us to develop increasingly richer classes of models
  - where the hardware structure is included *within* the design stage of the method
  - e.g. MapReduce, GPGPUs, ...
- Here I will present an illustrative example from a problem in statistical genetics that we're working on

Dependence structures in Genomes

- The genome exhibits complex dependence structures both along a genome within an individual, or cancer cell, and across genomes in populations of individuals or cancer cells in a tumour
- To formally model population and sequential dependence requires graphical models (such as the ancestral-recombination-graph) whose structures prevent computation, i.e. intractable likelihoods
- Hence we require simplifying (approximating) models that capture the major sources of dependence and allow for computation
- Perhaps the most important simplifying structure in statistics is the notion of Markov conditional independence, such that on a set of random variables, $S = \{S_1, \ldots, S_N\}$ we define a joint probability model, $Pr(S)$ that factorises,

$$Pr(S_i|S_1, \ldots, S_{i-1}, S_{i+1}, \ldots, S_N) = Pr(S_i|S_j \in n(i))$$

where $n(i)$ indexes the Markov neighbourhood of i, such that S_i is conditionally independent of variables outside of this neighbourhood

Hidden Markov Models (HMMs)

HMMs are arguably the most widely used probability model in Bioinformatics, where the hidden states refer to classifications of loci such as {coding, non-coding}, or {duplication, deletion} events (in cancer), etc,

HMMs are defined by:

- a number of unobserved "hidden" states $S = \{1, ..., N\}$ and Markov transition probabilities $Pr(S_{t+1}|S_t)$ that define rules for state-transitions
- possible emissions or observations Y_i and emission probabilities (likelihoods) $Pr(Y_i|S_i)$ for each state
- an initial distribution $\pi = Pr(S_1)$

The state sequence S then forms a Markov Chain:

- such that the future $S_{t+1}, ..., S_T$ does not depend on the past $S_1, ..., S_{t-1}$, given the present S_t

Hidden Markov Models (HMMs)

The Markov property is key to the success of HMMs. Dependencies are represented as edges and conditional independences as missing edges in the graph representation.

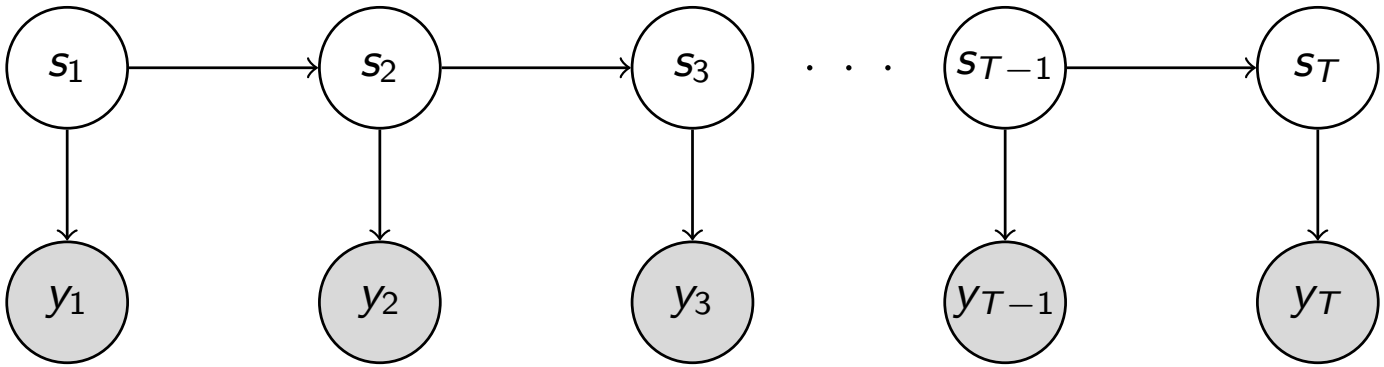


Figure: HMM depicted as a directed graphical model for observation $e_1, ..., e_T$.

The joint distribution for HMMs is written as:

$$Pr(y_1, ..., y_T, s_1, ..., s_T) = Pr(s_1)Pr(y_1|s_1)\prod_{t=2}^{T} Pr(y_t|s_t)Pr(s_t|s_{t-1})$$

Efficient HMM Algorithms with Dynamic Programming

There are three basic problems that can be solved efficiently with HMMs:

- how do we compute the probability of an observation $Y = \{Y_1, \ldots Y_T\}$ given a parameterised HMM?
- how do we find the optimal state sequence corresponding to an observation given a parameterised HMM?
- how do we estimate the model parameters?

The Markov structure of HMMs allows for dynamic programming.

- the Forward algorithm computes the probability of an observation

Solutions to the second problem depend on the definition of optimality.

- the Viterbi algorithm finds the most probable (MAP) state sequence, maximising $\hat{s} = \arg\max_s Pr(s_{1..T}|y_{1..T})$
- the Forward-Backward algorithm computes the posterior marginal probabilities $Pr(s_t|y_1, \ldots, y_T)$ for each state at every t.

All three algorithms have the computation cost $\mathcal{O}(N^2 T)$, so linear in sequence length T

Efficient HMM Algorithms with Dynamic Programming

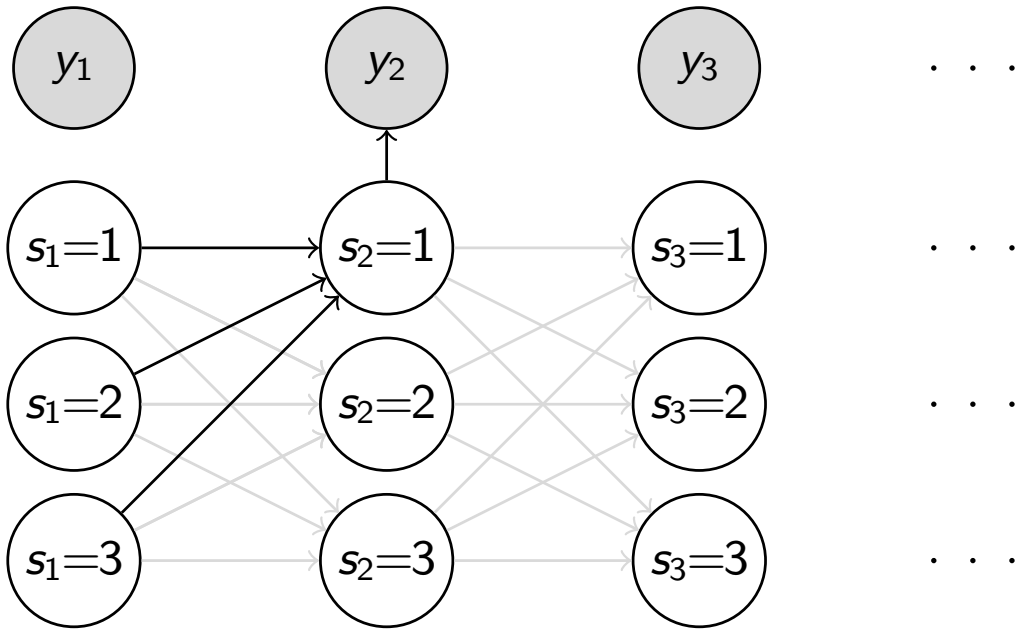


Figure: A single computation step of HMM algorithms.

Each step in the forward recursion means filling in a cell of a dynamic programming table:

$$\alpha(s_t) = Pr(y_t|s_t) \sum_{s_{t-1} \in S} Pr(s_t|s_{t-1})\, \alpha(s_{t-1})$$

Parallel computing

Modern transistor manufacturing has recently reached a power ceiling that limits the computational power of individual processing units.

The future of processing power increase relies on the ability to use multiple computational units in parallel.

Current technology permits parallel computation using

- multiple separated computing nodes in - distributed computing
- multiple local CPUs or CPU cores - Multi-core computing
- multiple specialized processor subunits (e.g. in GPUs) - Many-core computing

Irrespective of the technology used, parallel computing requires parallel algorithm design - the exploration of repetitive operations and operation independences followed by restructuring. Parallel algorithms represent a different paradigm from sequential algorithms - what makes sense in sequential, may not make sense in parallel and vice versa.

Graphical Processing Units (GPUs)

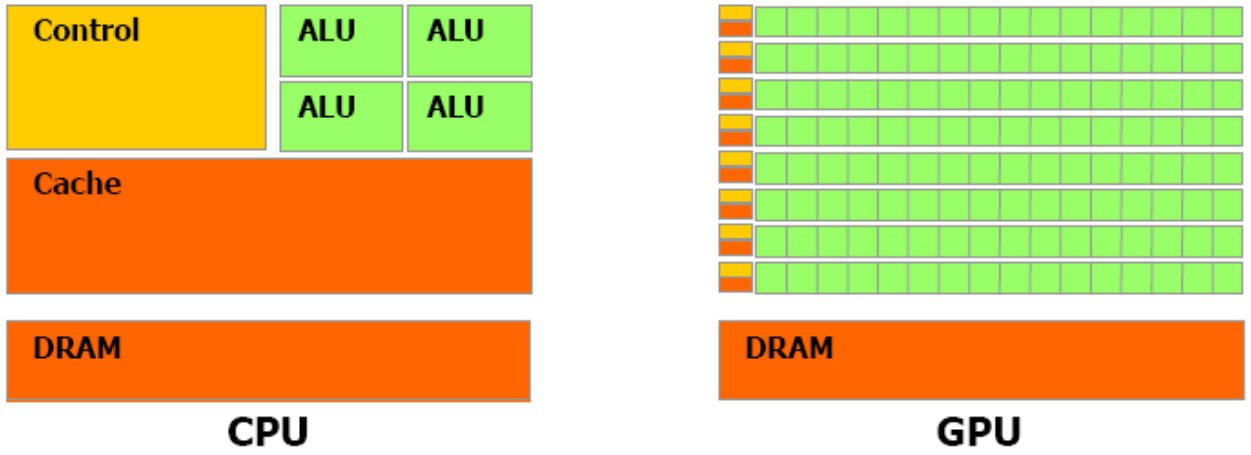


Figure: GPUs devote more transistors to data processing (NVIDIA).

GPUs are low cost, low energy, highly parallel, local devices designed for Single Instruction, Multiple Data (SIMD) processing. They have a highly specialized architecture (with a bespoke language - CUDA) that allows for great performance in certain situations.

With linear algorithm flow, reasonable suitable space requirements and efficient memory use, implementations of suitable algorithms may achieve as high as 30× acceleration compared to multi processors and even more than 100× speedup compared to sequential implementations (Lee *et al* 2010).

Trivial HMM Parallelisation

The parallelisation of HMM algorithms is straightforward (in theory) over multivariate emissions (likelihoods) and over the state-space:

- calculations corresponding to different observations are trivially parallelisable and is perfectly suited even for distributed computation.
- the standard HMM algorithms all repeat the same operations for each state s_t. Moreover, the calculation of the $\alpha(s_t)$ (or beta, etc.) values are independent and hence the calculations are suitable for parallelisation on GPUs.

Theoretically, the above parallelisations can reduce the overall runtime of the HMM algorithms for K multivariate observations, $y_i = \{y_{i1}, \ldots, y_{iK}\}$, to $\mathcal{O}(TN)$ from $\mathcal{O}(KTN^2)$. The number of compute operations remains $\mathcal{O}(KTN^2)$.

Trivial HMM Parallelisation

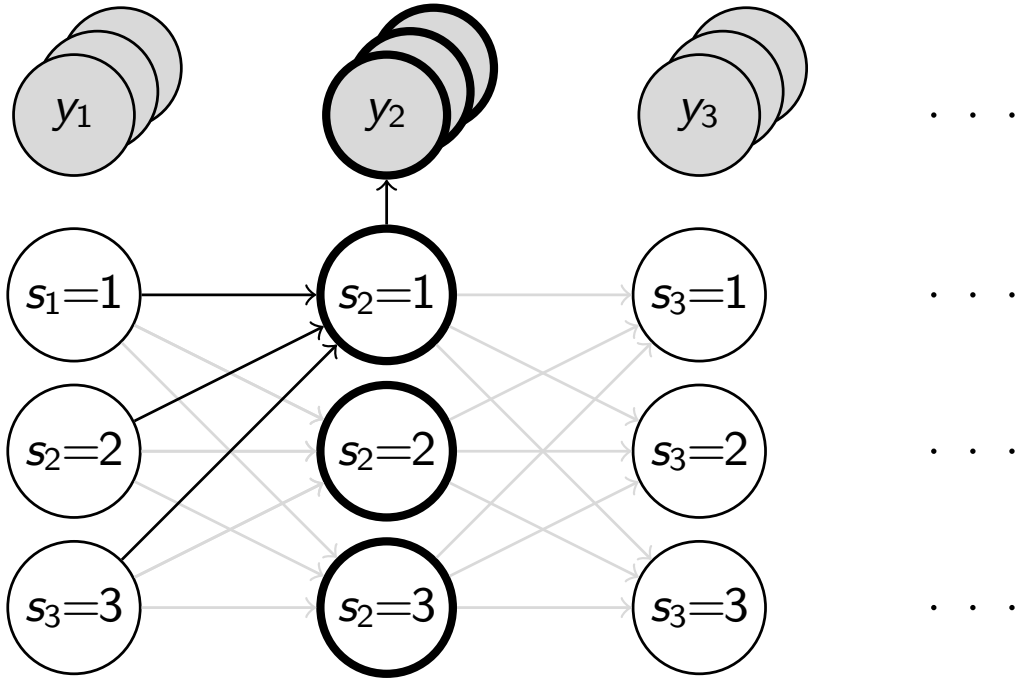


Figure: The calculation over the observations and the calculations of $\alpha(s_2 = 1)$, $\alpha(s_2 = 2)$ and $\alpha(s_2 = 3)$ can be performed in parallel.

- Such computations are known as embarrassingly parallel or trivially parallel.
- The algorithm, and number of compute operations, remains the same. You just exploit additive redundancy (`for loops`).

HMM Parallelisation with Sequence Partitioning

Genetic datasets are generally large and the length of sequences is much greater than the state space ($T >> N$). The natural question is whether it is possible to design new parallel-HMM algorithms (rather than parallelising existing algorithms)?

We have been investigating GPU algorithms exploiting parallel reduction along the sequence

The algorithm works by partitioning the sequence into blocks:

- assume the sequence is partitioned into B blocks each of length T_b
- the starting (e.g. $\alpha(s_{k \times T_b})$) values are not available for each block at the beginning of each recursion
- hence run the algorithm N-times, from each possible state, each time conditioning on a different starting state. Naturally these sets of N conditional runs may be also run in parallel.
- when the computation is done for all blocks, they can be merged sequentially updating each conditional run with its corresponding starting value

Forward-Backward algorithm, serial and parallel versions

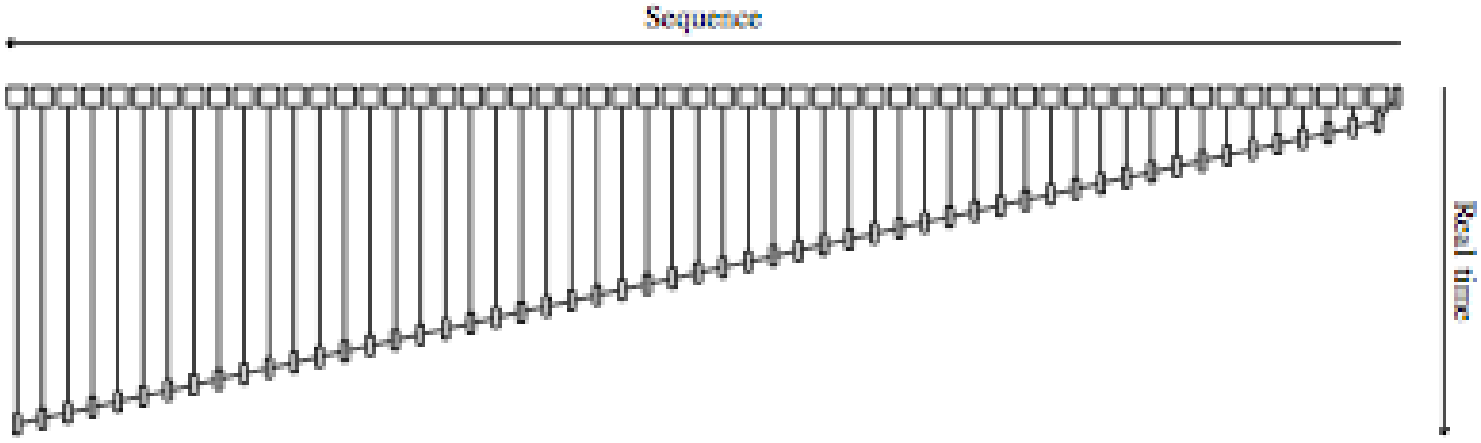


Figure 2. The traditional forward algorithm, as described by Rabiner [1]. The rectangles represent matrices and vectors. The black lines denote dependencies. The top row is the C_i matrices.

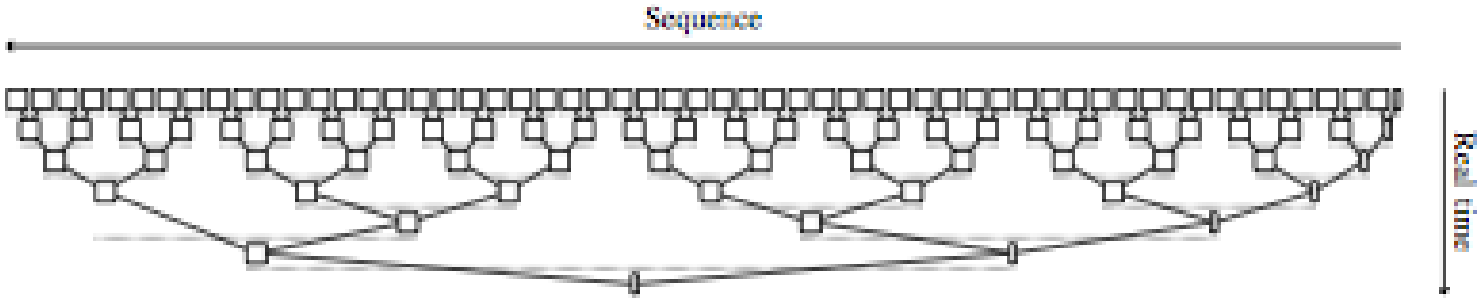


Figure 3. Using parallel reduction on the forward algorithm. The rectangles represent matrices and vectors. The black lines denote dependencies, while the horizontal dotted ones denote synchronization. The top row is the C_i matrices.

Figure: Serial and Parallel representation of FB algorithm (Neilsen&Sand, 2011).

HMM Parallelisation with Sequence Partitioning

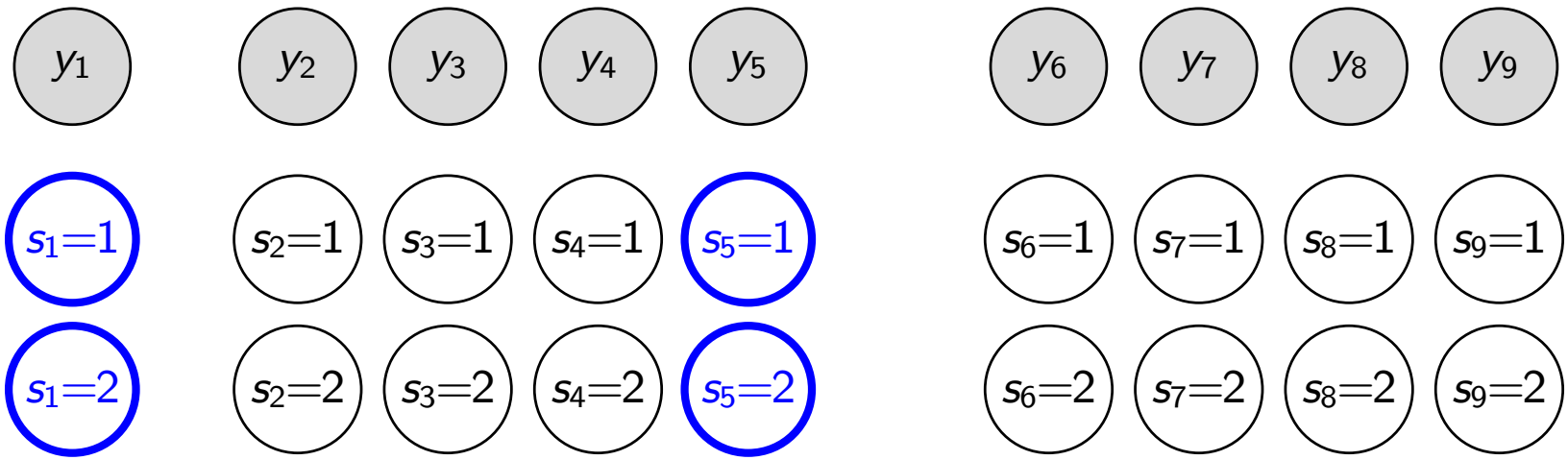


Figure: Parallelisation of the Viterbi algorithm with simple observation partitioning. The number of blocks is $B = 2$, the length of blocks is $T_b = 4$.

HMM Parallelisation with Sequence Partitioning

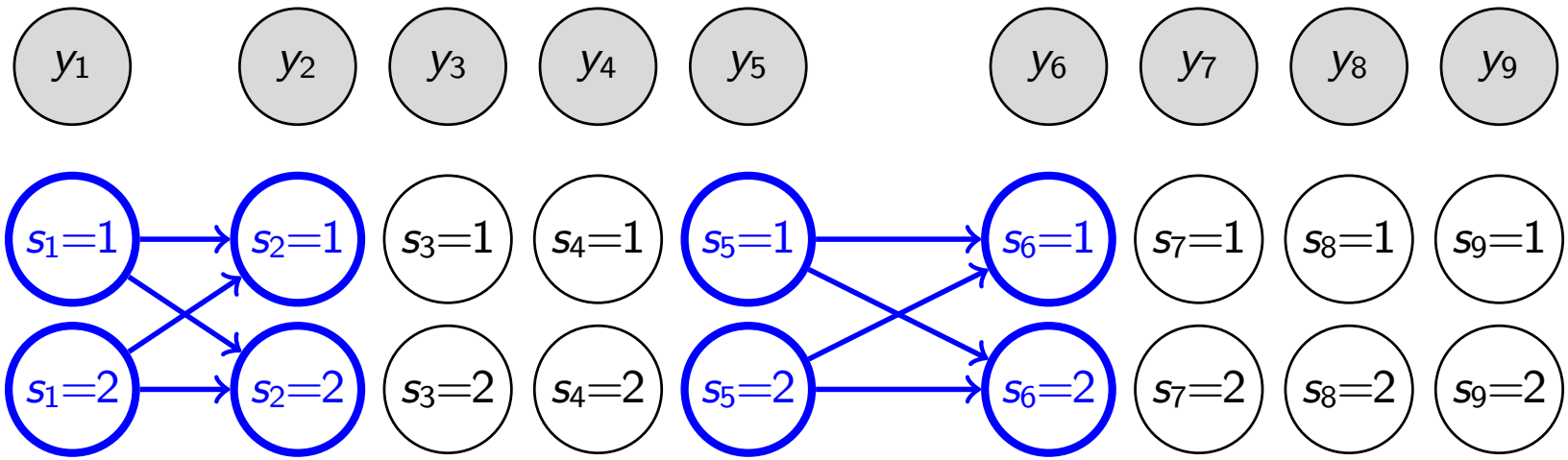


Figure: Parallelisation of the Viterbi algorithm with simple observation partitioning. The number of blocks is $B=2$, the length of blocks is $T_b=4$.

HMM Parallelisation with Sequence Partitioning

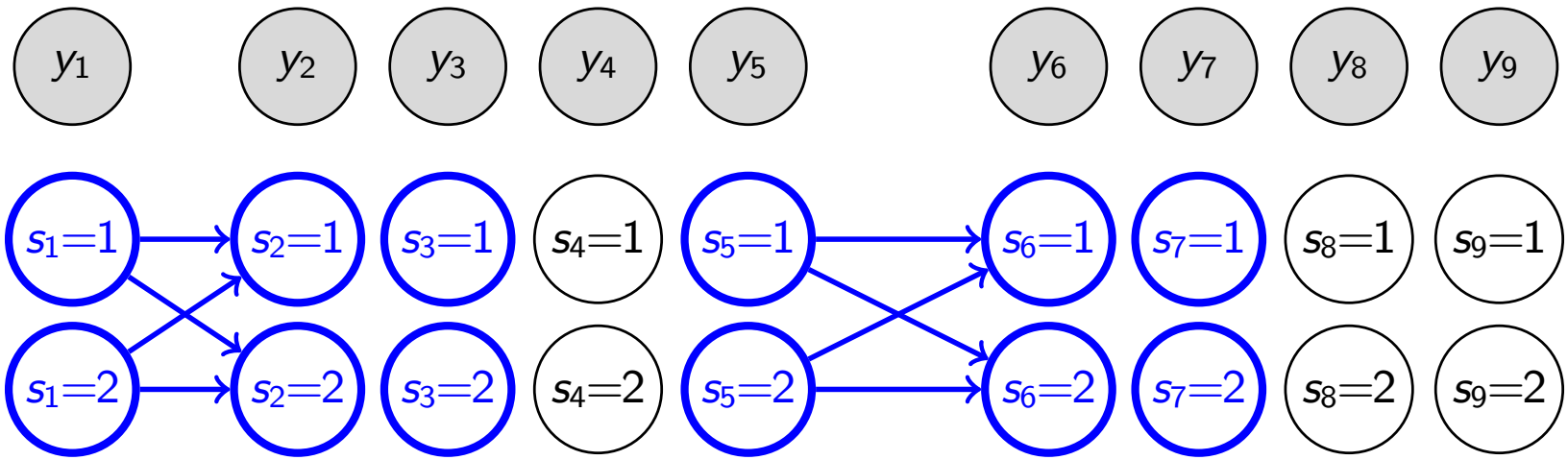


Figure: Parallelisation of the Viterbi algorithm with simple observation partitioning. The number of blocks is $B = 2$, the length of blocks is $T_b = 4$.

HMM Parallelisation with Sequence Partitioning

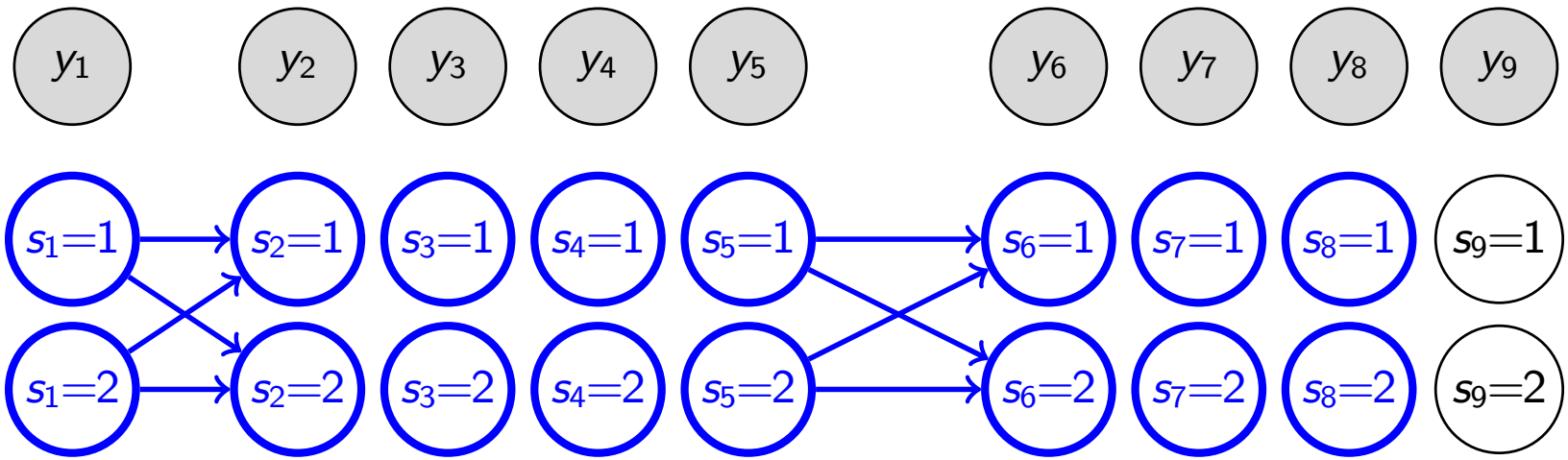


Figure: Parallelisation of the Viterbi algorithm with simple observation partitioning. The number of blocks is $B=2$, the length of blocks is $T_b=4$.

HMM Parallelisation with Sequence Partitioning

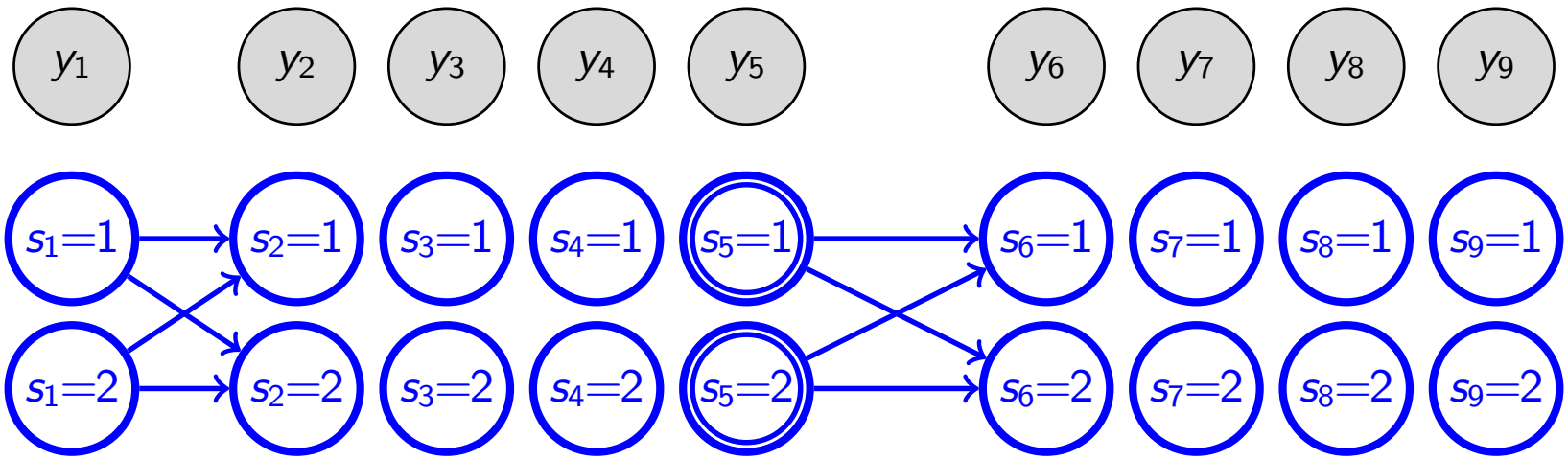


Figure: Parallelisation of the Viterbi algorithm with simple observation partitioning. The number of blocks is $B = 2$, the length of blocks is $T_b = 4$.

HMM Parallelisation with Sequence Partitioning

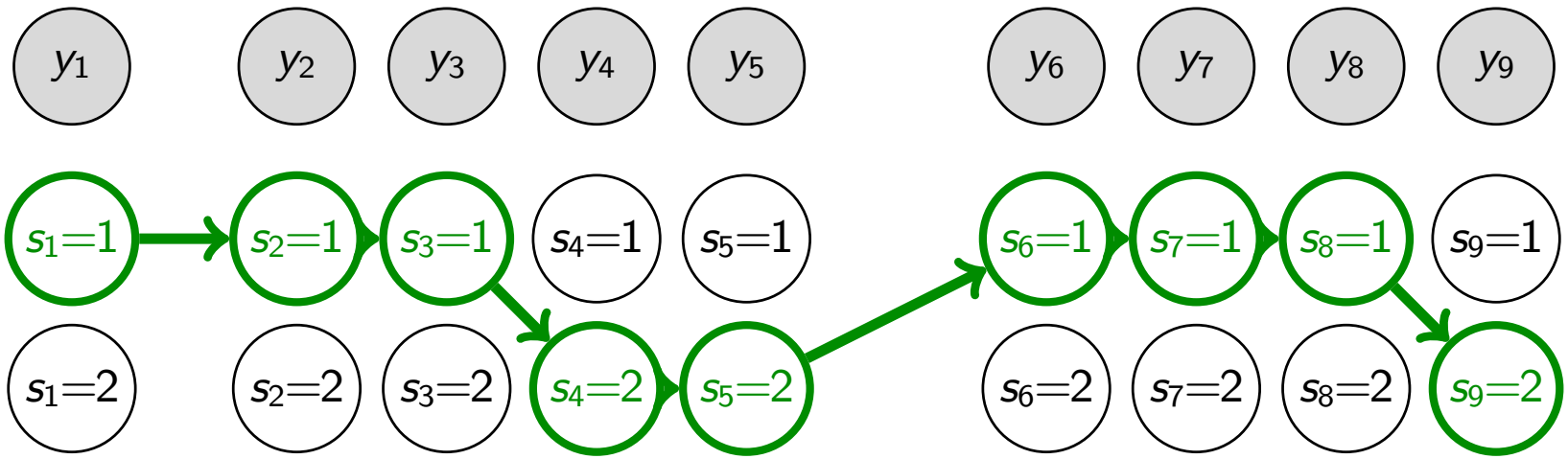


Figure: Parallelisation of the Viterbi algorithm with simple observation partitioning. The number of blocks is $B=2$, the length of blocks is $T_b=4$.

Speed Up

- In theory with $P = T/2$ processors the runtime is of order $\mathcal{O}(N^3 \log T)$, in comparison with the traditional $\mathcal{O}(N^2 T)$, leading to a speed up of
$$R = \frac{N \log T}{T}$$
which for $T >> N$ is hugely significant.
- The number of compute operations is $\mathcal{O}(N^3 T)$, so involves a factor of N additional computations relative to the traditional HMM
- In practice we can implement serial merging and $P << T$ leading to loss of efficiency but the runtime order is close to linear in the block size

HMM Parallelisation with Sequence Partitioning

- If we have even more computational power (some $KT/2N^2$ processing units), it is possible to combine all parallelisation approaches to achieve $\mathcal{O}(N \log T)$ runtime for $\mathcal{O}(KTN^3)$ computation.
- Just for comparison, state of the art GPUs have 2688 cores and the supercomputer Titan (National Oak Ridge) has 18,688 Nvidia Tesla K20X GPUs with a total of $\sim$50million cores. Emerald has 372 Tesla cards. Present datasets in population genetics typically have K and N in the thousands and T in the millions.

Example

- So where can we exploit such methods.....

The Li and Stephens Model (LSM)

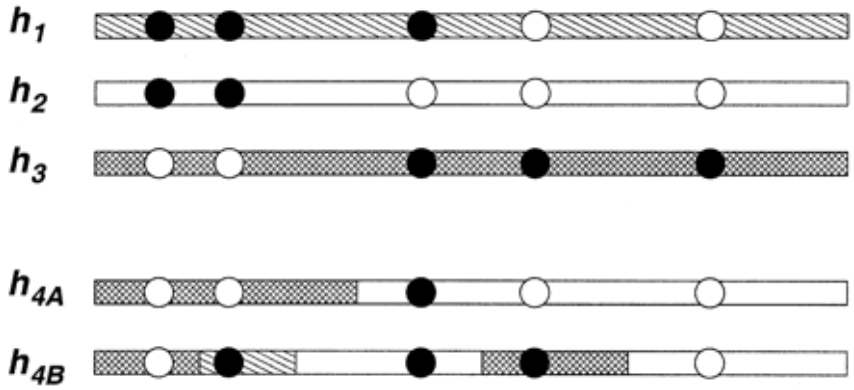


Figure: The "imperfect mosaic" modeling (Li and Stephens 2003).

The Li and Stephens Model

- is one of the most widely used models since its development.
- is an HMM-based approximation to the coalescent with recombination
- models the complex correlation structure between genetic loci (linkage disequilibrium) by treating each genome as an imperfect mosaic made of the other genomes
- defines a joint model over a collection of sequences as a Product of Approximate Conditionals (PAC) likelihood

Application of the LSM - Chromosome Painting

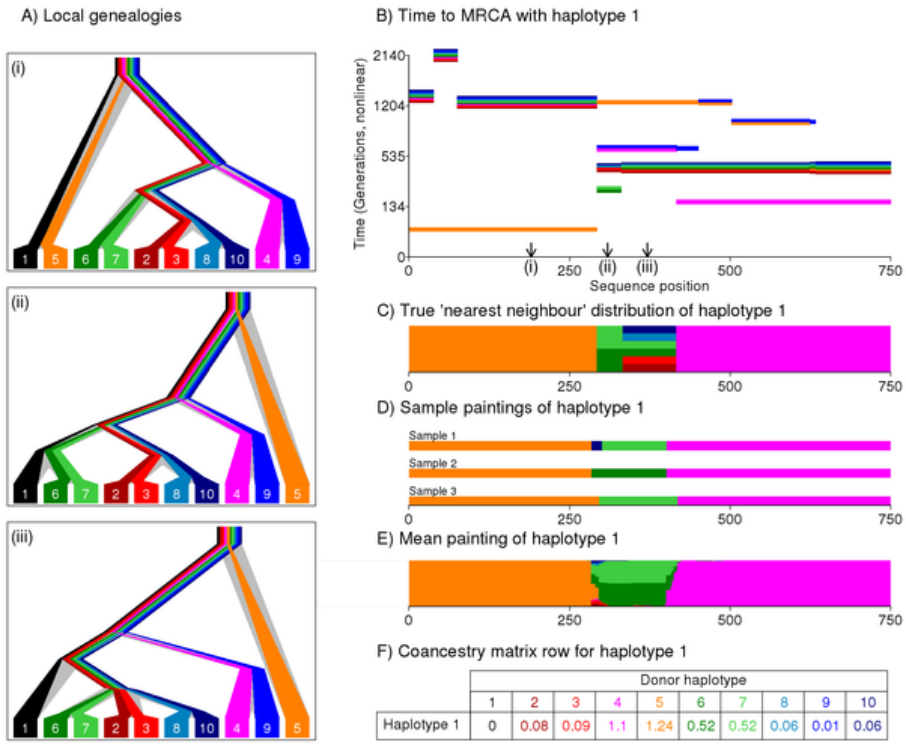


Figure: Chromosome painting and derivation of coancestry (Lawson *et al.*).

The GPU-LSM is currently being applied to Chromosome painting, which

- is a method of relating stretches of DNA sequences to one another
- is a crucial step in producing coancestry matrices when inferring population structure from dense haplotype data

GPU-LSM

Parallelisation of the LSM is less trivial then normal HMMs.

- the simplifications that make the LSM so efficient actually result in complications for parallel programming. The summations can be performed as parallel reduction, but they can not be hidden in any existing loop, they need to be run separately. This increases the cost of parallelisation with a $\log N$ factor to $\mathcal{O}(KTN^2 \log N)$.
- the datasets are generally big, hence achieving memory efficiency is not straightforward

Present implementation of the Viterbi algorithm under the LSM achieves $\times 50 - \times 80$ acceleration compared to optimized sequential C code.

- reduce days of runtime to hours

Using LSM to detect signals of "Natural Selection"

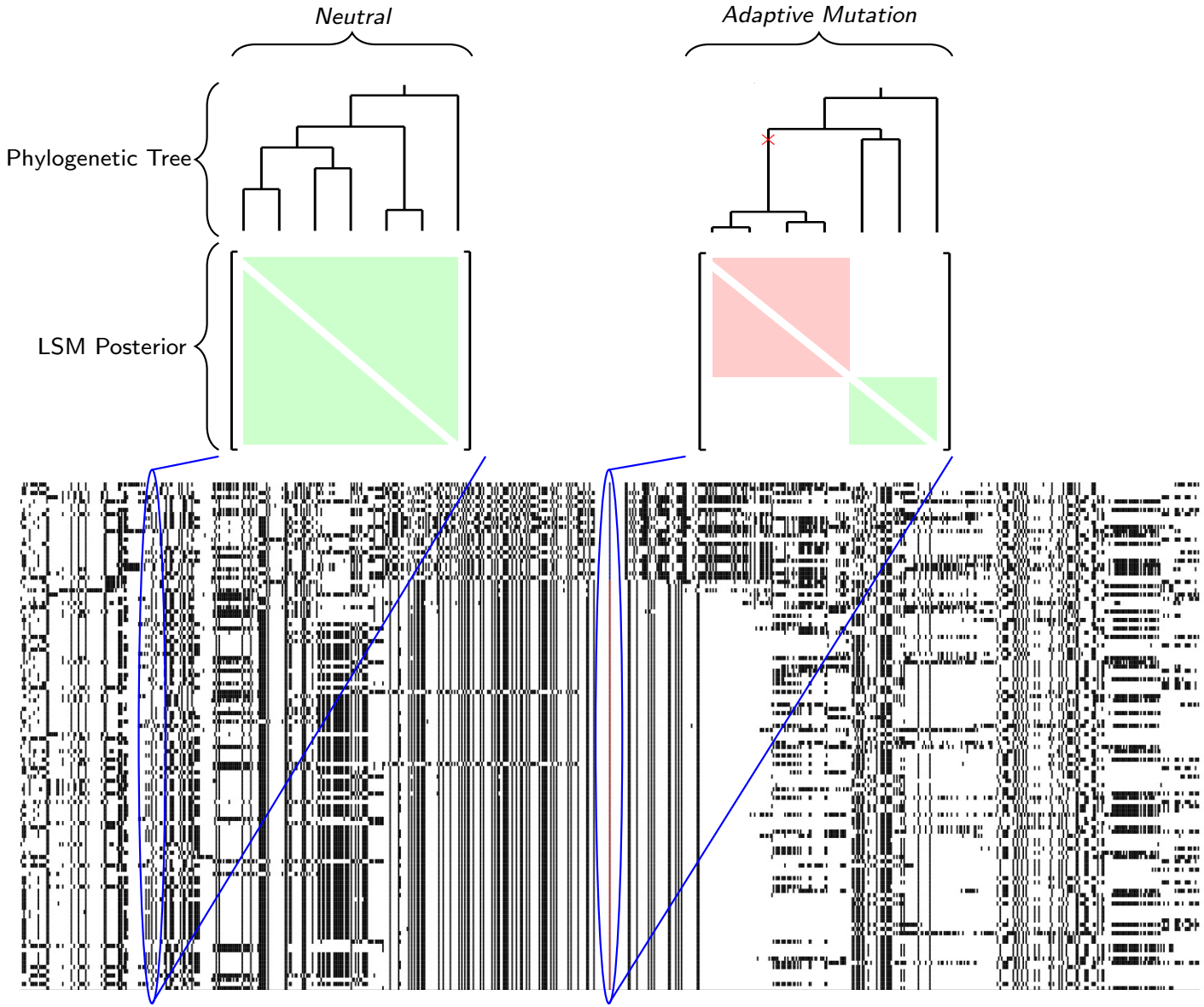
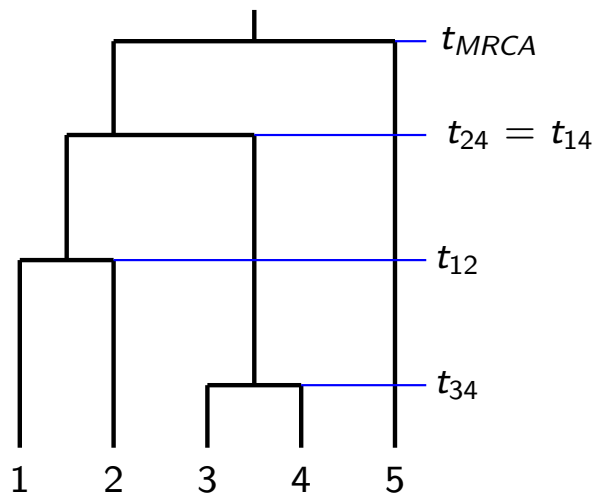


Figure: Effect of Natural Selection on Haplotypes - LCT:2q21.3 HapMap

LSM to Detect Signals of Natural Selection

- Possible to express the HMM-LSM posterior copying probabilities $Pr(S_i = j|\cdot)$ as $f(t_{ij})$.
- Using some Linear Algebra we can obtain t_{ij} up to constants of proportionality.
- By looking for regions with an excess of young mutations we can establish whether a mutation is under "selection".



Applications

- The LSM provides an efficient and relatively accurate approach to detecting signals of "natural selection".
- One application we're looking to apply this model to is in monitoring drug resistance mutations in populations of Malaria parasite, Plasmodium falcaparum.
- The ability to detect the emergence of new drug resistant traits before they become fixed in the population, will provide important insight in disease monitoring and population health.

Conclusions

- Medical genetics and genomics will produce vast data sets over the next few years
- We need statistical methods that can scale to handle
  - dimension
  - heterogeneity
  - model misspecification
- To do so, exploiting parallel computation within the algorithm design will be key, both for model development and model fitting