Efficient Monte Carlo Inference for Infinite Relational Models

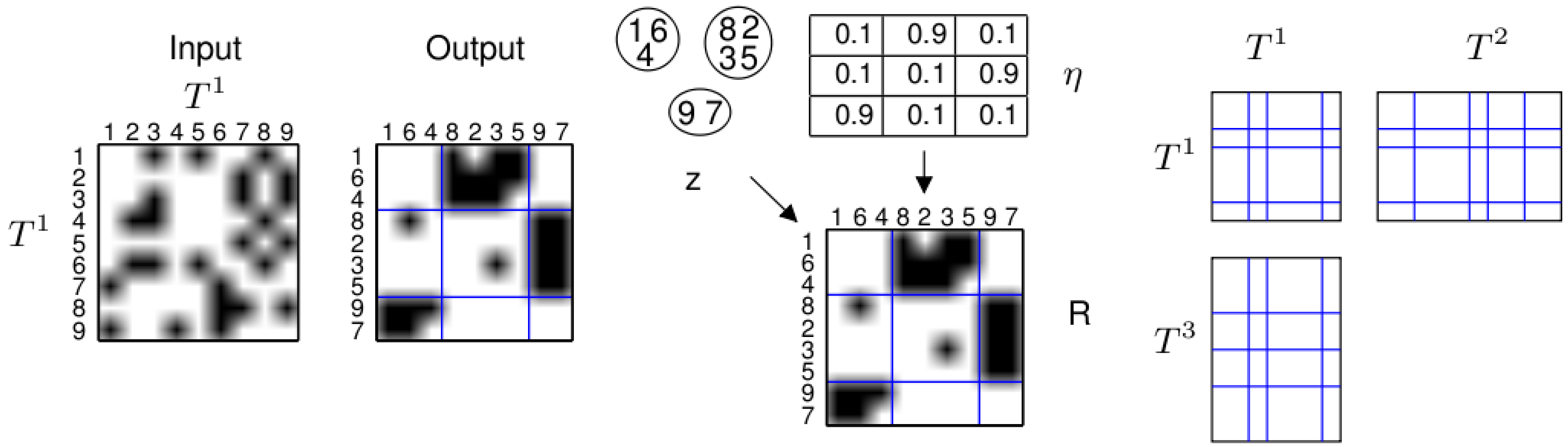
Vikash K. Mansinghka
MIT BCS/CSAIL
Navia Systems, Inc.

Joint work with:
Keith Bonawitz (MIT CSAIL, Navia)
Eric Jonas (MIT BCS, Navia)
Josh Tenenbaum (MIT BCS/CSAIL)

IRM Review - Intuition

- Relational data/knowledge: domains, entities, relations, sparse (~5% typical)
- Entities can be typed; types predict relation values; contains clustering, coclustering, etc
- Build a model with latent types for compression/prediction/exploration
- Nonparametric, relational generalization of stochastic mixtures (and blockmodels)

IRM Review - Intuition

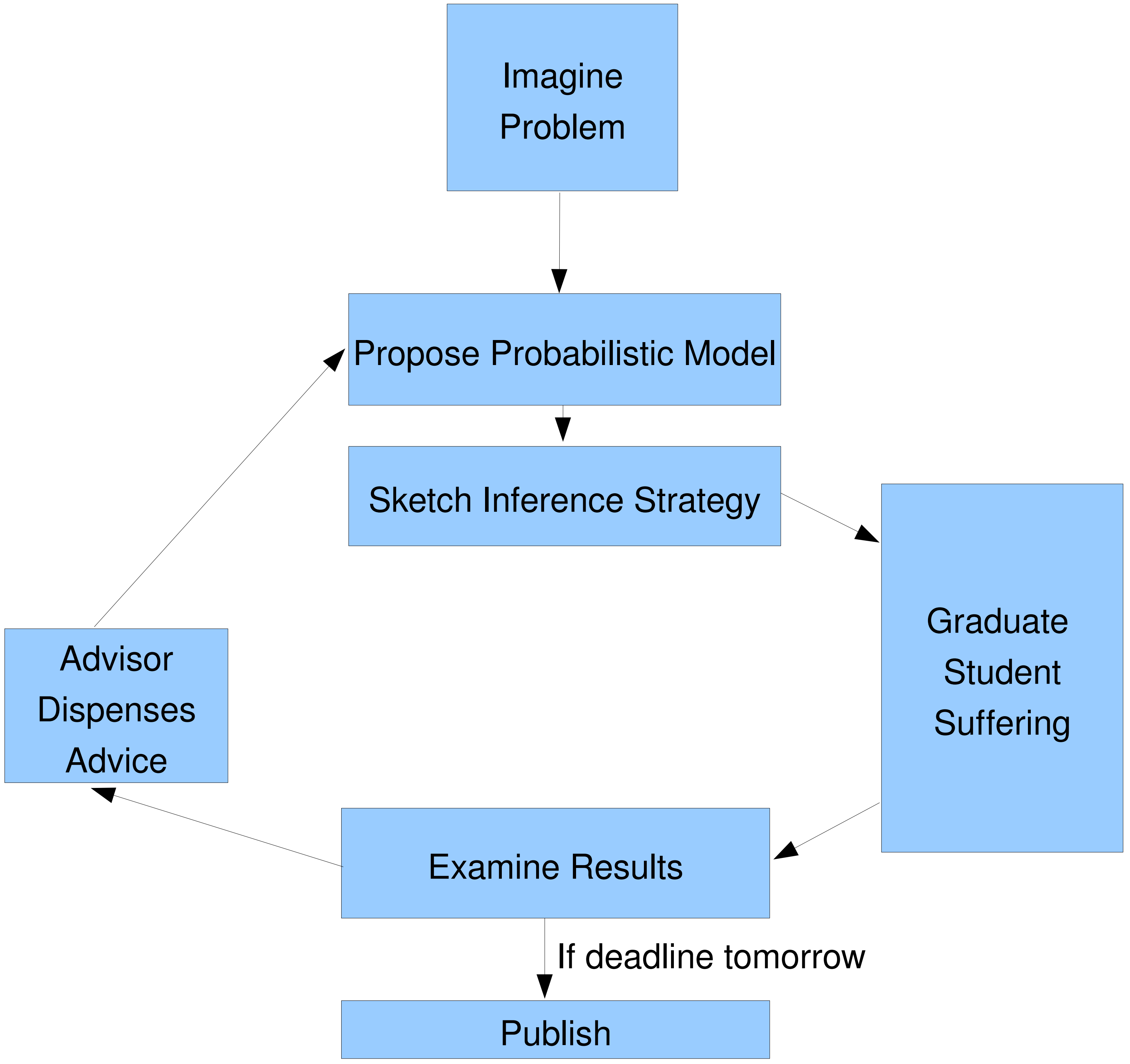


0.1	0.9	0.1
0.1	0.1	0.9
0.9	0.1	0.1

- One CRP per domain; one param per 'block'
- T1xT2 with T2.alpha = infinity is mixture model
- Can model arbitrary SQL schemas
  - Many poorly match modeling assumptions

IRM Review – Context

- LVM inventing #dom discrete nodes *and size*
- Interesting inference microcosm:
  - hard already, getting harder
  - simple methods 'seem' too slow in MATLAB
- Realistic network models much worse:
  - Nonconjugate models (sparsity; missing vals)
  - Types of types of...; latent space with types...


Imagine Problem
Propose Probabilistic Model
Sketch Inference Strategy
Graduate Student Suffering
Examine Results
Advisor Dispenses Advice
If deadline tomorrow
Publish

Rhetorical Point

- MCMC doesn't have to be slow; be careful!
- MC can easily be *online* and *massively parallel*

Technical Content

- Analysis for IRM CRP Gibbs
- Particle filter for IRM
- Parallel Tempering Transform

Monte Carlo is good computer science (fast) and good engineering (composable, modular). The world may work this way (see stat mech). Can we do better (see LDA)?

How to use MCMC

- Design convergent sampler (don't forget floats)
- Estimate (by analysis and measurement):
  - Convergence time
  - Mixing time
- Run sampler for 10 times as long, recording samples via skip. Meditate on Jaynes and Bernoulli while you wait.
- Form Monte Carlo estimate

How to *really* use MCMC

- Make fast sampler (or use tool, like Blaise)
- Run as long as possible
  - Anneal or temper if necessary
- Record (last) k samples
  - Estimate or SMA, if you really want many modes
- "#P problem, O(N) approximation"
- Should be familiar: EM/VB/GD/BP/CG/BP
  - *Risk* of local minima, too narrow posterior. So?

Goal of Fast CRP Gibbs

- Let:
  - G: max number of groups represented this cycle
  - DP: number of datapoints (scalar cells observed)
  - arity: arity of single relation (T1xT1 = T1xT2 = 2)
- Cost of one (cycle) sweep of single site Gibbs: O(G*arity*DP*arity)
- *MCMC should feel like EM/gradient descent:*
  Speedy linear iterations, ~100 is good, ~1k "totally enough"

Strategy for Fast CRP Gibbs

- Apparent problem:
  - Gibbs: conditional posterior (numerical from joint)
  - Joint is global, touches all data; naively quadratic
- Key idea: support fast incremental moves
  - Constant-time score updating (*exchangeability)*
  - Constant-time sufficient statistics updating
  - Exploit sparsity pattern in data (as in matrix*vector)
  - No MATLAB! (matrix*vector isn't everything)

IRM Datastructures (I)

$domain$:

```
    int[] groupcounts
    entity[] entities
    double alpha
```

$entity$:

```
    domain domain
    int group
    datapoint[] datapoints
```

$component$:

```
    double[] suffstats
    double[] hypers
```

IRM Datastructures (II)

datapoint:

```
datum value
component component
entity[] entities
```

irm:

```
double score
domain[] domains
int[] relationsignature
map<int[] groups, component> components
datapoint[] datapoints
```

IRM CRP Gibbs Pseudocode (I)

```
ASSIGN-DATAPOINT($irm, datapoint, cmpt$)
1  $irm.score \leftarrow irm.sore +$ COMPONENT-PREDICTIVE($cmpt.hypers, cmpt.suffstats, datapoint.value$)
2  ADD-SUFFSTATS($cmpt.suffstats, datapoint.value$)
3  $datapoint.component \leftarrow cmpt$

REMOVE-DATAPOINT($irm, datapoint$)
1  $cmpt \leftarrow datapoint.component$
2  REMOVE-SUFFSTATS($cmpt.suffstats, datapoint.value$)
3  $irm.score \leftarrow irm.score -$ COMPONENT-PREDICTIVE($cmpt.hypers, cmpt.suffstats, datapoint.value$)

COMPUTE-COMPONENT($irm, datapoint$)
1  for $i \leftarrow 1$ to $datapoint.entitites.length$
2       do
3            $groups[i] \leftarrow datapoint.entities[i].group$
4  return HASHMAP-GET($irm.components, groups$) $\triangleright$ Allocates new if nonexistent.
```

IRM CRP Gibbs Pseudocode (II)

```
ASSIGN-ENTITY($irm, domain, entity, group$)
1   $entity.group = group$
2   for $i \leftarrow 1$ to $entity.datapoints.length$
3       do
4           $datapoint \leftarrow entity.datapoints[i]$
5           $component \leftarrow$ COMPUTE-COMPONENT($irm, datapoint$)
6           ASSIGN-DATAPOINT($irm, datapoint, component$)
7   $irm.score \leftarrow irm.score +$ CRP-PREDICTIVE($domain, group$)
8   $domain.groupcounts[group]++$
```

```
REMOVE-ENTITY($irm, domain, entity$)
1   for $i \leftarrow 1$ to $entity.datapoints.length$
2       do
3           $datapoint \leftarrow entity.datapoints[i]$
4           REMOVE-DATAPOINT($irm, datapoint$)
5   $domain.groupcounts[entity.group]--$
6   $irm.score \leftarrow irm.score -$ CRP-PREDICTIVE($domain, entity.group$)
7   $entity.group = -1$
```

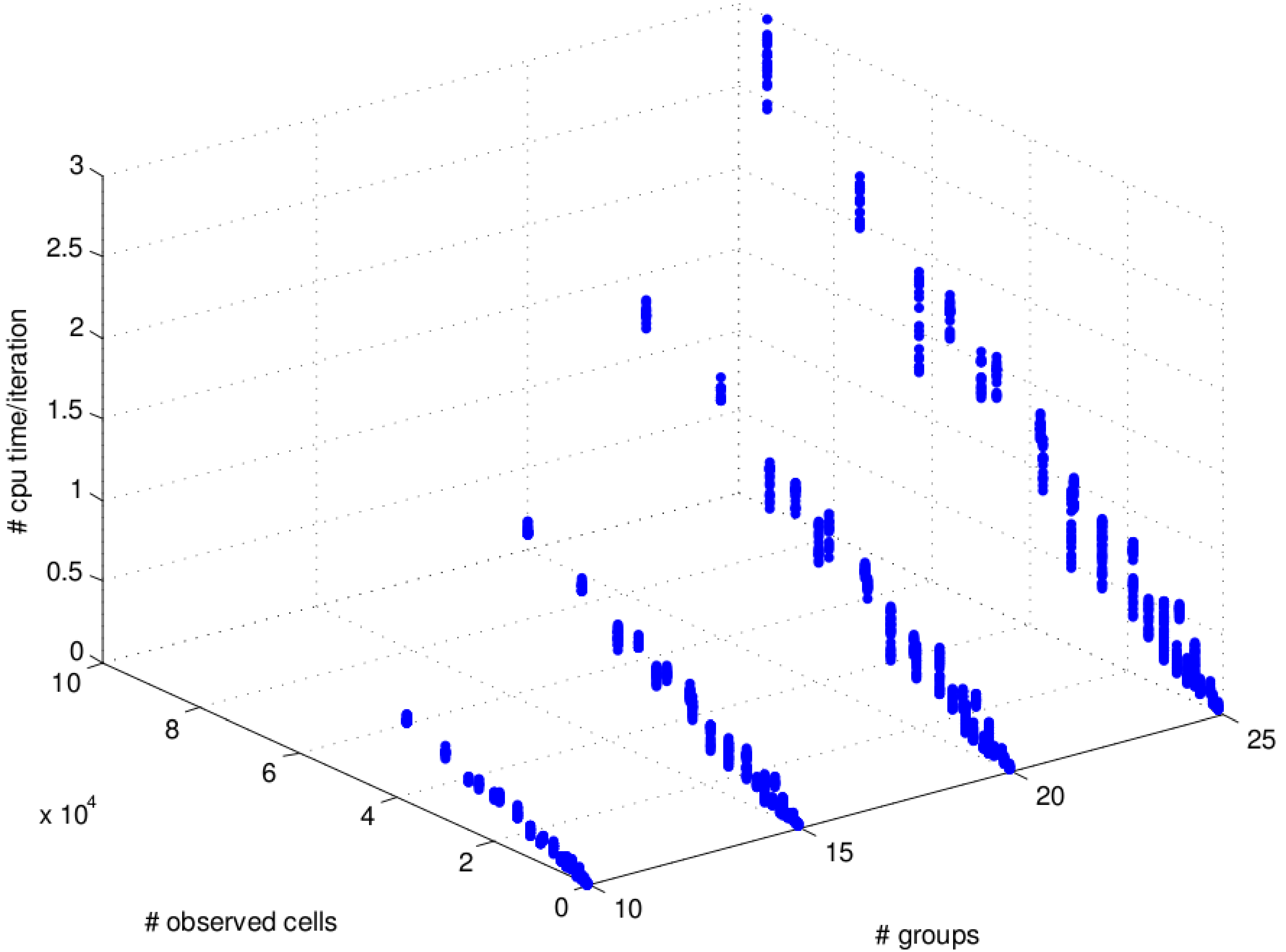
IRM CRP Gibbs Pseudocode (III)

```
CYCLE-GIBBS-SWEEP($irm, domain$)
for $i \leftarrow 1$ to $domain.entities.length$
    do
        $entity \leftarrow domain.entitites[i]$
        REMOVE-ENTITY($irm, domain, entity, entity.group$)
        $consideredempty \leftarrow 0$
        for $g \leftarrow 1$ to $domain.groups.length$
            do
                if $domain.groupcounts[g] == 0$ & $consideredempty == 1$
                  then
                        $scores[g] \leftarrow 0$
                  else
                        ASSIGN-ENTITY($irm, domain, entity, domain.groups[g]$)
                        $scores[g] \leftarrow irm.score$
                        REMOVE-ENTITY($irm, domain, entity$)
                        if $domain.groupcounts[g] == 0$
                          then
                                $consideredempty \leftarrow 1$
        $gnew \leftarrow$ SAMPLE-UNNORMALIZED($scores$)
        ASSIGN-ENTITY($irm, domain, entity, gnew$)
```

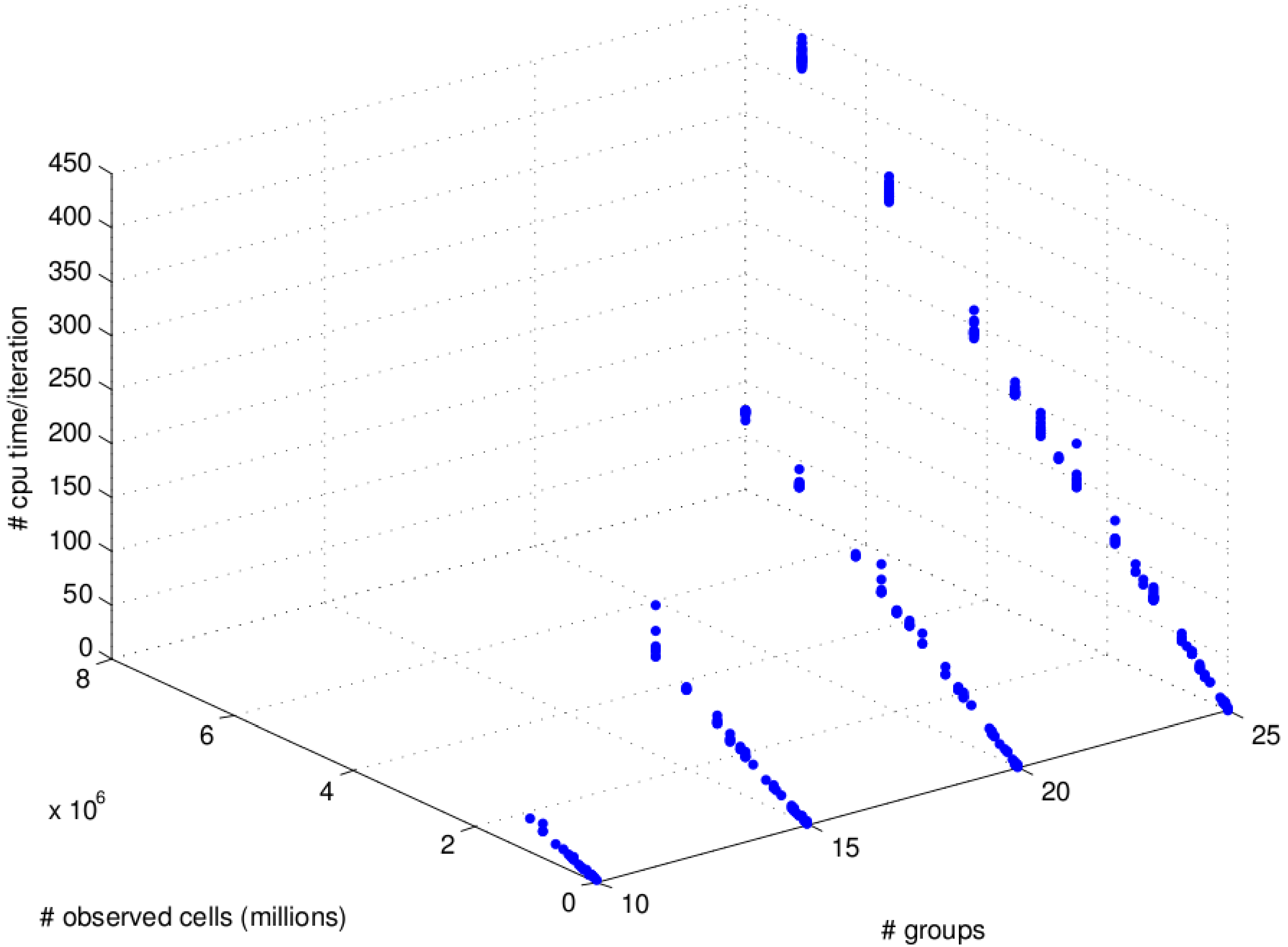
Algorithm Analysis

- Datapoint manipulation O(1) (*exchangeability)*
- Computing component O(arity) (hash key)
- Entity assignment O(datapoints touching entity)
  - Amortized; naively, O(DP/(#entities)) “on average”
- 1-dom: Called O(#entities * G) times per sweep
- O(arity) domains, so: O(G * arity * DP * arity)
- Entities drop out; only cells matter

CRP Mixture (T1xT2) Results



Relational (T1xT1) Results



Take Home Messages

- MCMC: A little care goes a long way
- CRP Gibbs competitive (per iteration) with
  - Truncated variational (Blei&Jordan, Teh et al.)
  - Relational extensions (Tresp et al)
  - A* search (Daume)
- Large scale (10^6) IRM fitting is straightforward
- What if naïve CRP Gibbs gets stuck badly?
  How can we measure useful work per iteration?

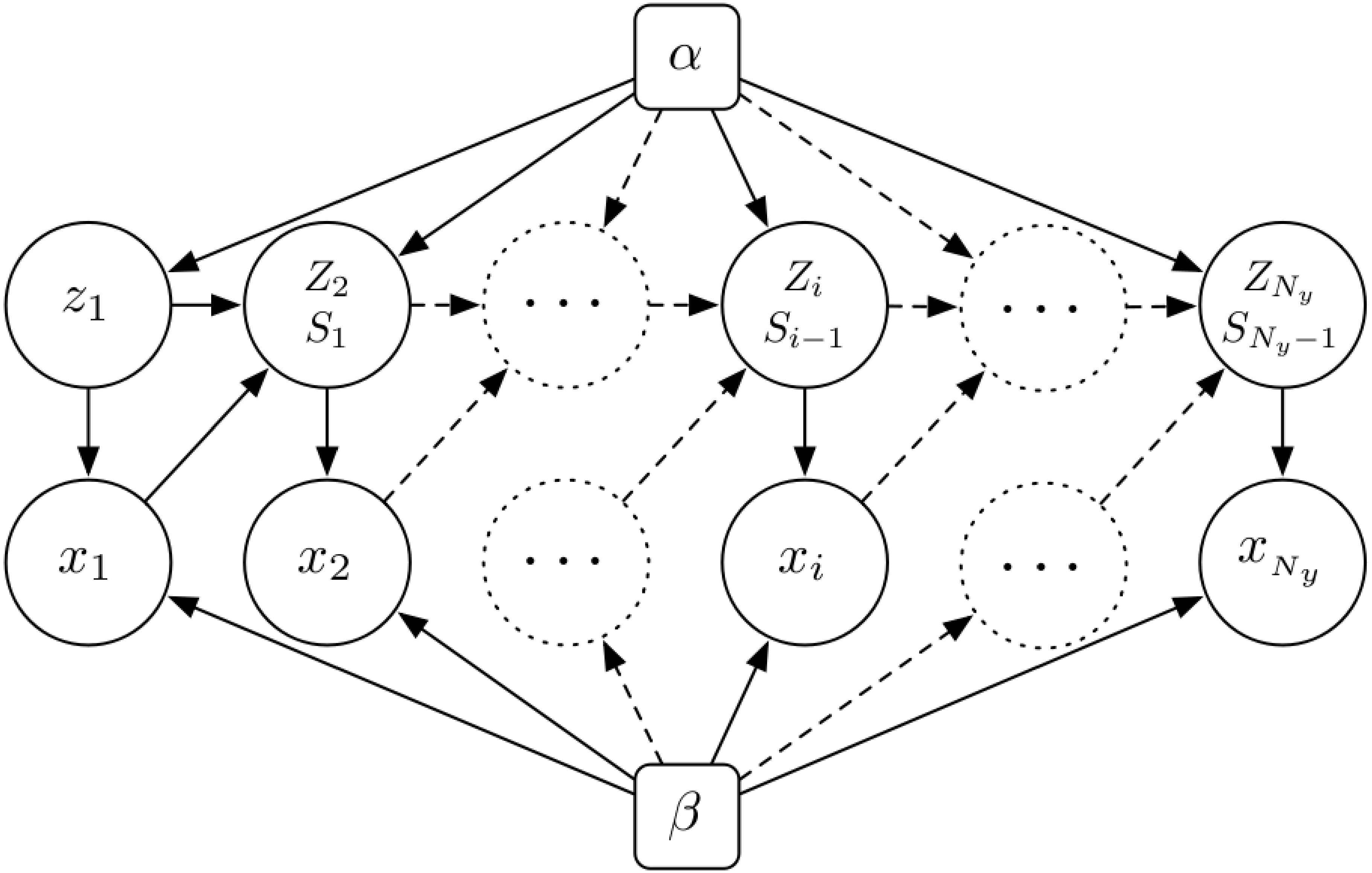
Rhetorical Point	Technical Content
• MCMC doesn't have to be slow; be careful!	• Analysis for IRM CRP Gibbs
• MC can easily be *online* and *massively parallel*	• Particle filter for IRM • Parallel Tempering Transform

Monte Carlo is good computer science (fast) and good engineering (composable, modular). The world may work this way (see stat mech). Can we do better (see LDA)?

Monte Carlo in a Parallel Universe

- MCMC is 'stochastic local search'; SMC is 'stochastic systematic search' (in progress)
  - Particle filtering natural for many 'offline' models; very cheap, good when data dense (like A*)
  - Use reactively: SMC new data, MCMC when bored
    - Can always initialize this way: “shoot first, then relax”
- (Essentially) all MC algorithms can be parallel tempered to help avoid local minima

Sequentializing Batch Problems



Particle Filter Datastructures

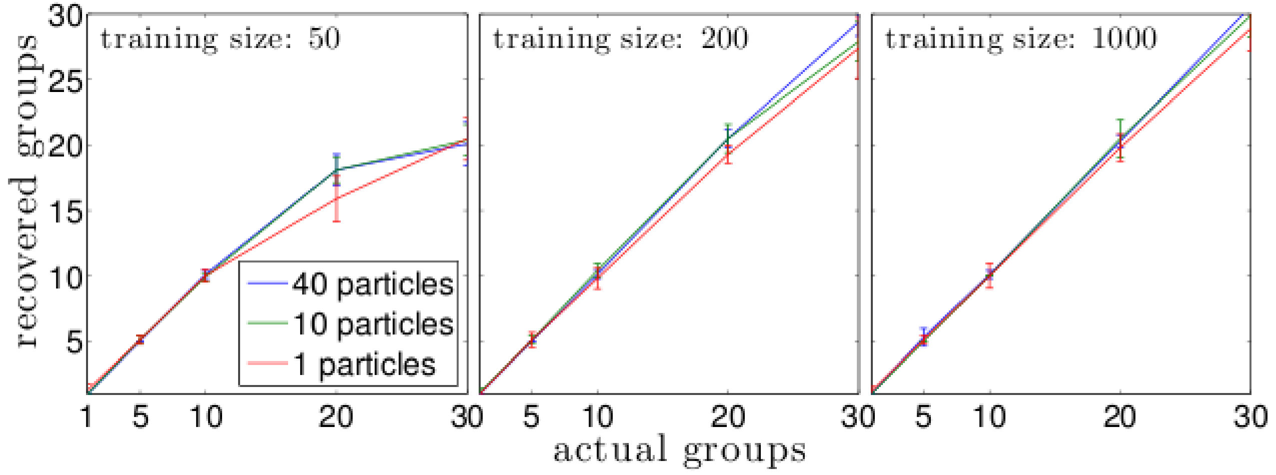
irm is now state for each particle, with a weight $irm.weight$
$entity.id$ is a unique identifier for each entity
$irm.known$ is a hash from $entity.id$s to entities
$irm.incorporated$ is a hash from $entity.id$s to entities
$datapoint.ids[i]$ is the id of the ith entity for this datapoint
$datapoint.numoutstanding$ is the number of non-incorporated entities touching $datapoint$
$filter.particles[]$ is an array of irm particles

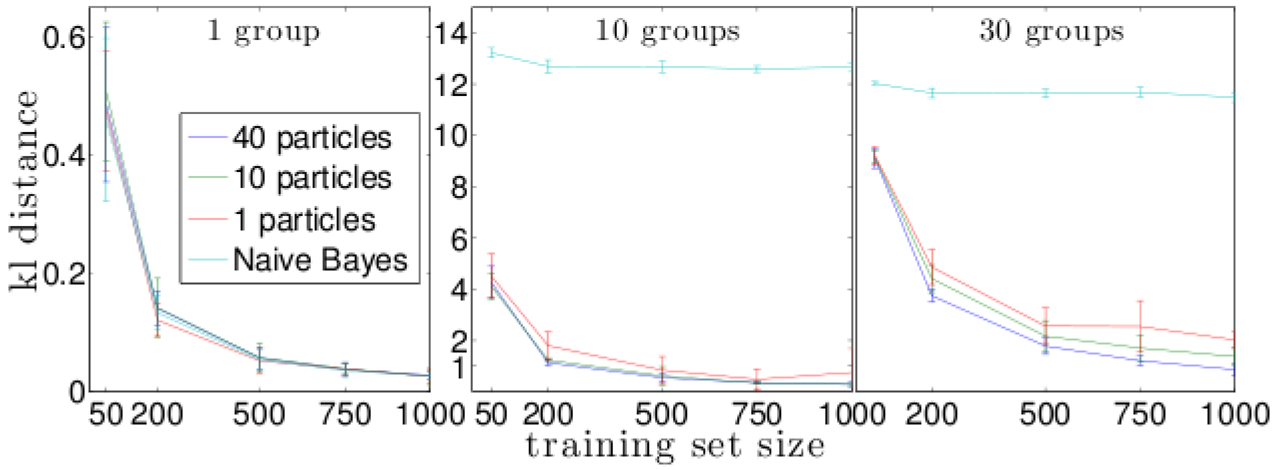
Key Particle Filter Pseudocode

```
OBSERVE-DATAPOINT($irm, datapoint$)
  $datapoint.numoutstanding \leftarrow datapoint.relationsignature.length$
  for $i \leftarrow 1$ to $datapoint.entities.length$
      do
        $id \leftarrow datapoint.ids[i]$
        $entity \leftarrow$ HASHMAP-GET($irm.known, id$)
        if $entity == NULL$
          then
              $entity \leftarrow$ NEW-ENTITY($irm.domain[datapoint.relationsignature[i]], id$)
              HASHMAP-PUT($irm.known, entity$)
        if HASHMAP-CONTAINS($irm.incorporated, entity$)
          then
              $datapoint.numoutstanding$–
        APPEND($entity.datapoints, datapoint$)
        APPEND($datapoint.entities, i$)

COMPUTE-DPSET($entity$)
  for $i \leftarrow 1$ to $entity.datapoints$
      do
        $entity.datapoints[i].numoutstanding$–
        if $entity.datapoints[i].numoutstanding == 0$
          then
              APPEND($dpset, entity.datapoints[i]$)
  return $dpset$
```

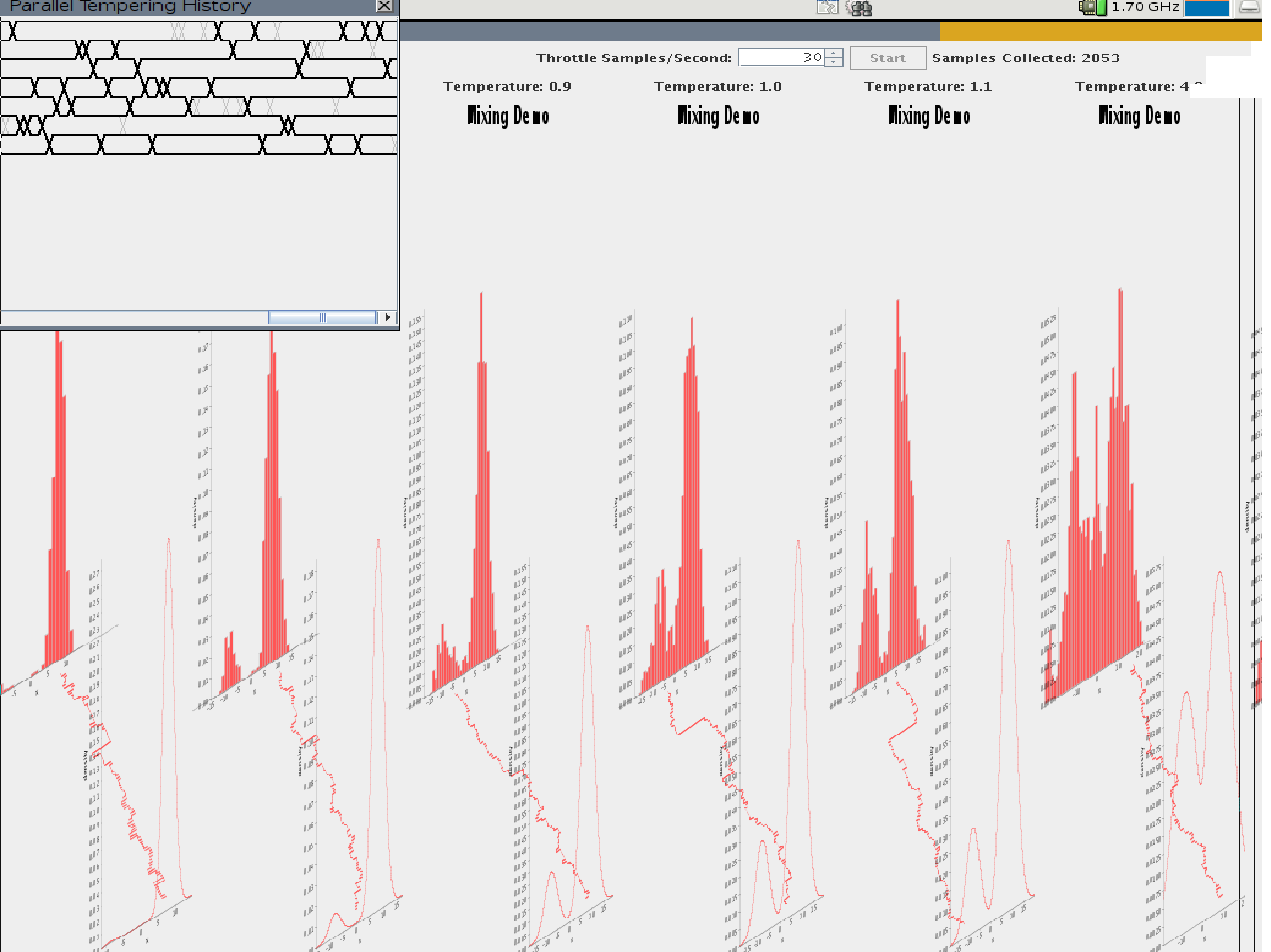
Particle Filter Results





Parallel Tempering Idea

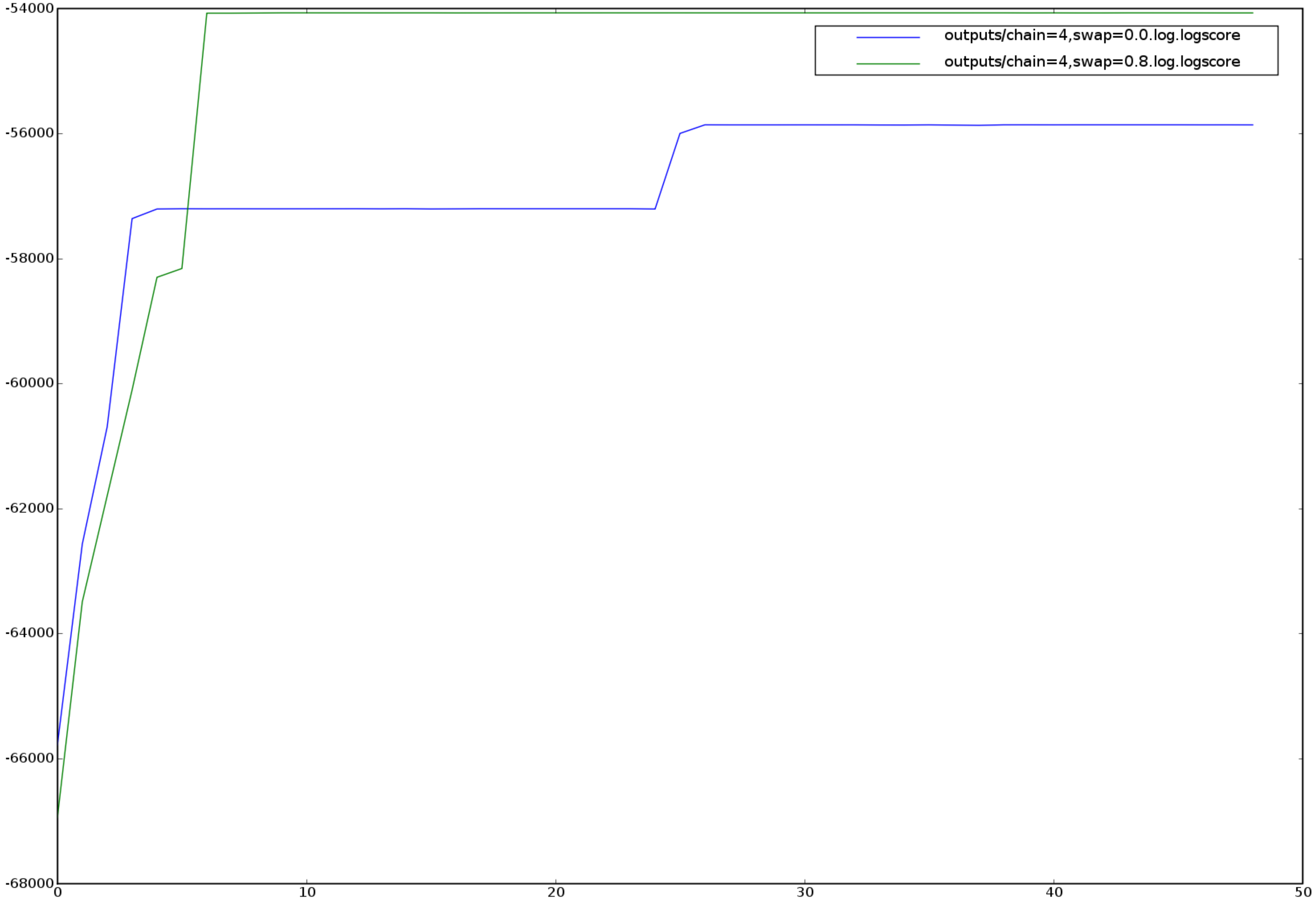
- T=1: Target. T=0: Delta on MAP. T>>1: Uniform
- Annealing: Start high, cool slowly, fall into good mode.
- Tempering:
  - Run parallel chains at different temperatures.
  - Periodically propose swaps (via M-H).
  - Implement by transferring temperatures; collect at end.


Parallel Tempering History

Throttle Samples/Second: 30

Start

Samples Collected: 2053


-54000
-56000
-58000
-60000
-62000
-64000
-66000
-68000
0
10
20
30
40
50
outputs/chain=4,swap=0.0.log.logscore
outputs/chain=4,swap=0.8.log.logscore

Rhetorical Point

- MCMC doesn't have to be slow; be careful!
- MC can easily be *online* and *massively parallel*

Technical Content

- Analysis for IRM CRP Gibbs
- Particle filter for IRM
- Parallel Tempering Transform

Monte Carlo is good computer science (fast) and good engineering (composable, modular). The world may work this way (see stat mech). Can we do better (see LDA)?

Questions for Discussion

- (Bayesian) Probability lets us build big, **accurate** models out of simple pieces. Key across AI/ML, including network learning.
  - Monte Carlo: solve/implement analogously:
    - Kernels compose via cycles, mixtures, (less known) conditionals, ...
    - Programmed well (NO MATLAB), can be competitive
    - Can naturally exploit massively parallel computers
- Tradeoff: Give up conceptual simplicity and composability for:
  - Simple models: non prob. or point estimates, so OR applies
  - Fancy math: OR relaxations; variational methods