Relational databases for lattice data analysis

Daniel C. Hackett, William I. Jay, Ethan T. Neil

Physics
UNIVERSITY OF COLORADO **BOULDER**

Overview

Idea:

Centrally store and organize all data and analysis results in a relational database

Keep results of **all** intermediate stages of analysis

[e.g. fitting a correlator – keep fits to all fit ranges, not just the best fit range]

No black boxes: structure DB so that analysis can be traced from raw data to final output

Why?

Enforces standardization for collaborations – everyone uses a common format, all data and results live in one place. [Prevents errors converting formats!]

Structure helps avoid "stupid mistakes" like associating data from different trajectories, working with an incomplete dataset, using outdated results, etc.

Reproducibility & provenance – entire analysis kept in DB, easy to "check your work". Can keep metadata (e.g. code version, run date) associated without it getting in the way.

Automated analysis becomes straightforward

Relational Databases

In practice, this means: SQL databases
We use PostgreSQL [open source!]

PostgreSQL

Data is stored in **tables**
Key columns provide unique specifiers for each entry
Tables can refer to each other's key columns
→ Use this to encode structure in data

Must specify table structure and relations at initialization. After, any data put in must conform to these types and relations [Feature, not a bug: enforces conventions!]

Retrieve data from DB using SQL query language

people.id is a key, uniquely specifies people

people

id	first	last
0	Daniel	Hackett
1	William	Jay

car.owner_id refers to **people.id**

cars

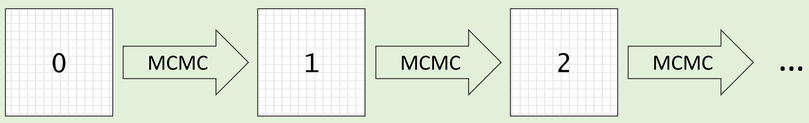
license_plate	owner_id	color
ASD123	0	mauve
JKL456	1	chartreuse

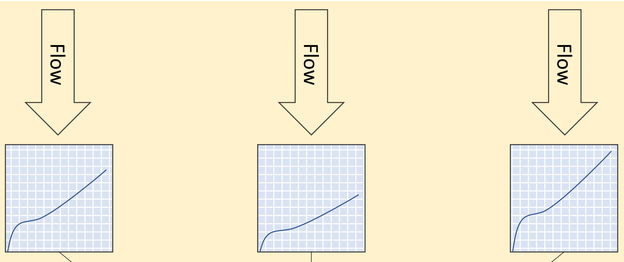
license_plate is a (non-integer) key for cars

Abstract Structure

MCMC

Map from gauge config to next gauge config
⇒ Ensemble ~ configs generated with same physical & MCMC parameters





Gradient flow

Map from gauge configs to measurements of flowed observables like $t^2E(t)$

...

Make equilibration cut, average observables, solve e.g. $\langle t^2E(t)\rangle = M$

$\frac{t_0}{a^2}$ $\frac{w_0}{a^2}$ ...

Scale estimation

Reduce from ensemble of measurements to estimates of observables

Reduction produces different estimates for different

- ensembles
- equilibration cuts
- scale observables
  [e.g. $\langle t^2E\rangle$, $t\partial_t\langle t^2E\rangle$, ...]
- "magic number" M
  [e.g. 0.3 for t_0, w_0]

Example Lattice Database

map_mcmc and **map_physical** describe map from gauge config **prev_id**
Convenient split: all configs generated with same physical parameters share a **phys_id**

map_physical

id	beta	nf	kappa	...
4	5.85	2	0.127	

gauge_config

id	prev_id	phys_id	mcmc_id	ens_id	traj	nc	ns	nt	...
0	NULL	4	17	121	0	3	24	6	
1	0	4	18	121	1	3	24	6	
2	1	4	17	121	2	3	24	6	

map_mcmc

id	nsteps	code	git	...
17	10	milc	54ff825	
18	20	milc	54ff825	

ensemble

id	nc	ns	nt	beta	nf	kappa	...
121	3	24	6	5.85	2	0.127	

Map from configs to measurements on configs:
result_flow contains results of applying gradient flow with parameters in **map_id** row of **map_flow** to config **gc_id**

result_flow

id	gc_id	map_id	t2e	...
0	0	0	$[0, 0.11, \dots]$	
1	1	0	$[0, 0.12, \dots]$	
2	2	0	$[0, 0.09, \dots]$	

map_flow

id	epsilon	...
0	0.01	

result_scale

id	ens_id	map_id	reduce_id	scale	...
0	121	0	0	$(0.89, 0.01)$	
1	121	0	1	$(0.77, 0.02)$	

reduce_scale

id	scale_obs	m	...
0	t2e	0.3	
1	bmw	0.3	

Reduce from ensemble of measurements on configs to an observable:
result_scale contains results of computing e.g. $\langle t^2E\rangle = M$ with E calculated using parameters specified by **map_id** and $\langle \dots \rangle$ taken over group of configs specified by **ens_id**

Smart Updating with Hashes

To start: assign a unique **output_hash** to each gauge config [e.g. checksum, **gauge_config.id**]

Every result has an **input_hash** and an **output_hash**.

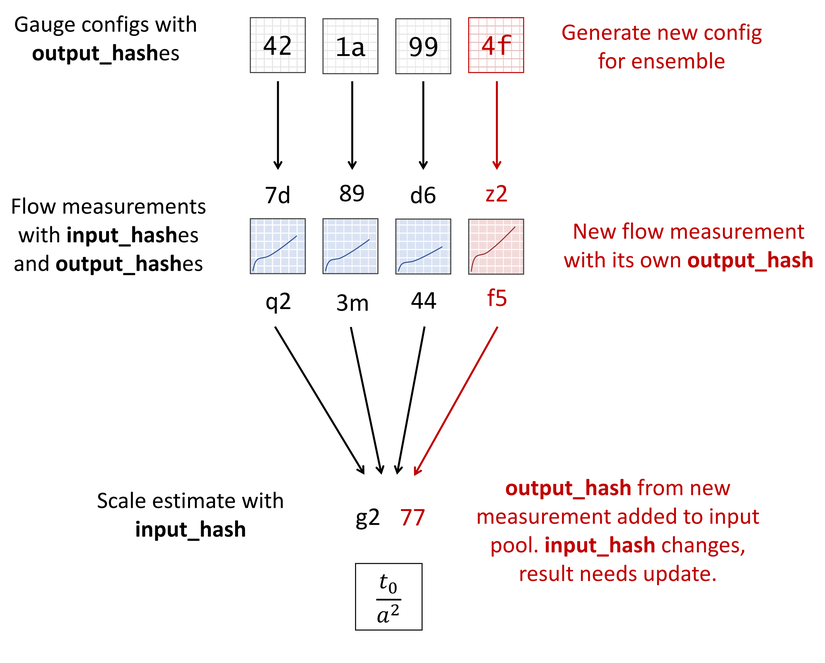
input_hash: Hash together **output_hash**es of all inputs & **map_id**s/**reduce_id**s

output_hash: Hash together all outputs of a measurement/reduction with the **input_hash**

Compute hashes outwards from **gauge_config** to determine which analyses need update

input_hash changes
→ something upstream has changed – analysis needs update

output_hash changes but input_hash does not
→ analysis has been corrupted – easy integrity checking



Automation

Run "bulk analysis" scripts N times daily:

- Check for newly-generated raw data and sync to DB
- Determine which analyses need to be updated/new analyses to perform
- Perform necessary analyses (e.g. computing scales, fitting correlators, picking best fits)

No human intervention required to keep analysis up-to-date (wake up to fresh results every morning)

Closing the automation loop:

- **Workflow manager** runs simulations
  [e.g. taxi github.com/dchackett/taxi]
- Data stored and automatically analyzed in database
- Automated run-specifier looks at analysis in DB, tells workflow manager to launch new simulations
- Applications:
  Tuning lattice spacing, m_q, etc
  Phase diagram exploration (ask about video!)