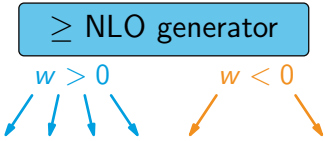
Unbiased Elimination of Negative Weights in Monte Carlo Samples

Jeppe Andersen Andreas Maier

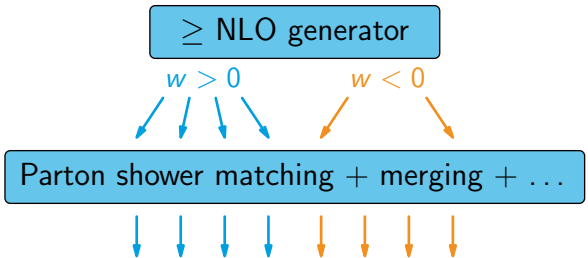
1 December 2021

arXiv:2109.07851

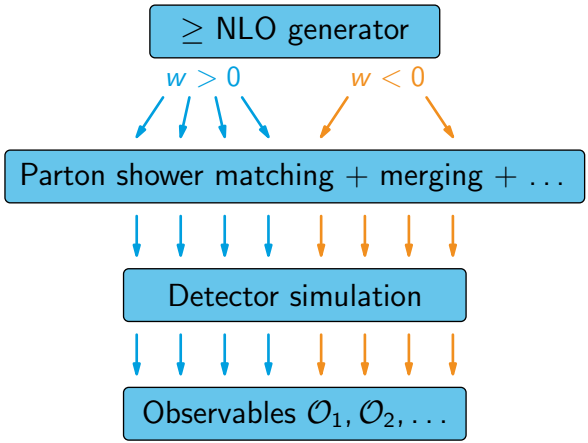
Negative weights in event generation



Negative weights in event generation

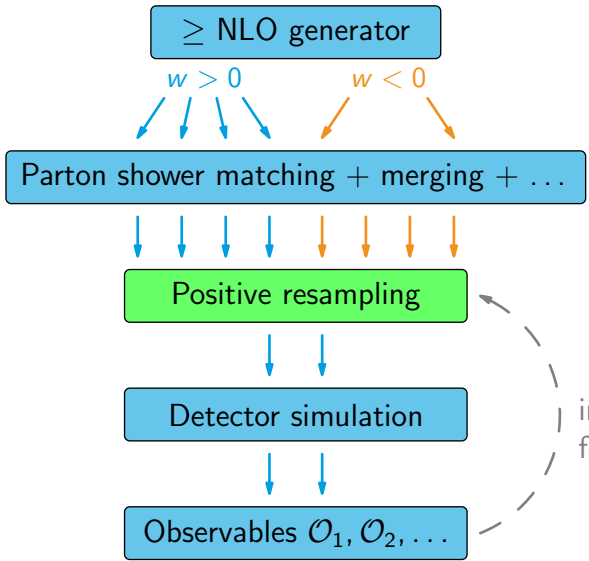


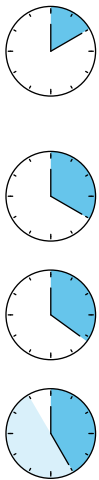
Negative weights in event generation



Positive resampling

[Andersen, Gütschow, Maier, Prestel 2020]





Positive resampling

Goals:

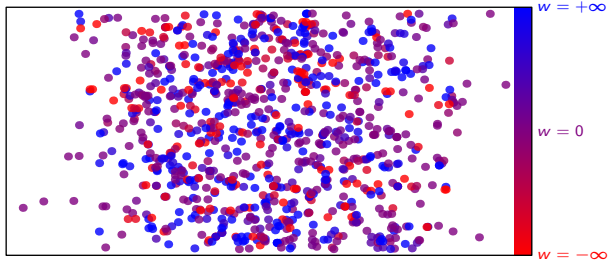
- Eliminate negative weights as much as possible
- Introduce no discernible bias in the observables
- No tuning to specific observables required

Only care about observables that

- can be predicted by the event generator
- can be measured in a real-world experiment limited by
  - detector
  - statistics

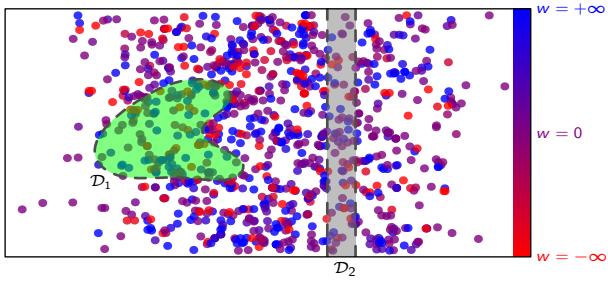
Observables

Events in 2D projection of phase space:



Observables

Events in 2D projection of phase space:



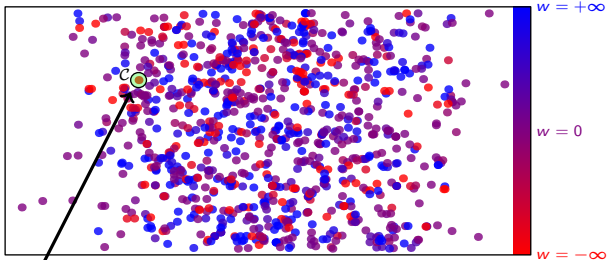
Observables $\mathcal{O}$:

- Select region $\mathcal{D}$ in phase space $\geq$ experimental resolution
- $\mathcal{O} = \sum_{i \in \mathcal{D}} w_i \geq 0$ with sufficient statistics

e.g. histogram bins

Redistribute weights without affecting any observable

Cell resampling

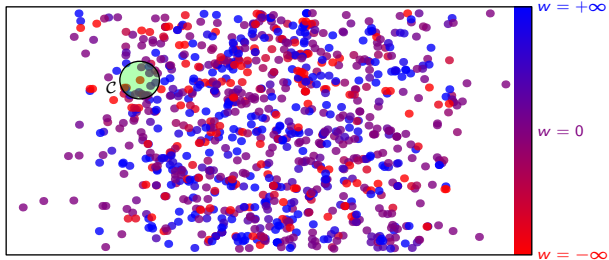


Cell resampling:

Repeatedly

1. Choose seed event with $w < 0$ for cell $\mathcal{C}$

Cell resampling

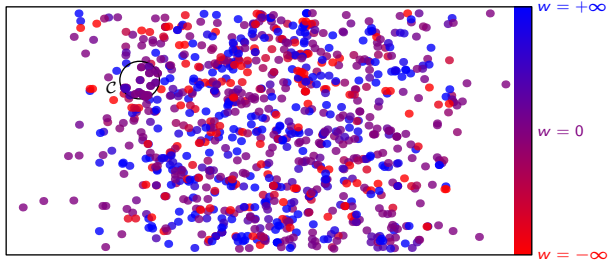


Cell resampling:

Repeatedly

1. Choose seed event with $w < 0$ for cell $\mathcal{C}$
2. Iteratively add nearest event to cell until $\sum_{i \in \mathcal{C}} w_i \geq 0$

Cell resampling

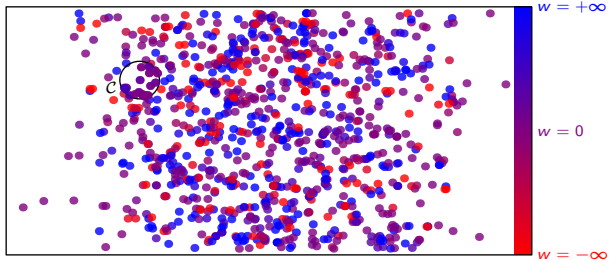


Cell resampling:

Repeatedly

1. Choose seed event with $w < 0$ for cell $\mathcal{C}$
2. Iteratively add nearest event to cell until $\sum_{i \in \mathcal{C}} w_i \geq 0$
3. Reweight: $w_i \to w_i' = \frac{\sum_{j \in \mathcal{C}} w_j}{\sum_{j \in \mathcal{C}} |w_j|} |w_i| \geq 0$

Cell resampling



Cell resampling:

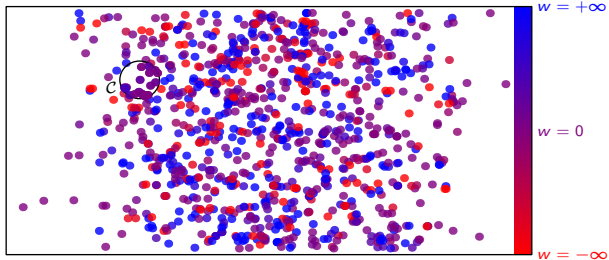
Repeatedly

1. Choose seed event with $w < 0$ for cell $\mathcal{C}$
2. Iteratively add nearest event to cell until $\sum_{i \in \mathcal{C}} w_i \geq 0$
3. Reweight: $w_i \to w_i' = \frac{\sum_{j \in \mathcal{C}} w_j}{\sum_{j \in \mathcal{C}} |w_j|} |w_i| \geq 0$

Sufficient statistics: cell size $<$ experimental resolution

Otherwise: limit cell size, accept $w_i' < 0$

Cell resampling



Cell resampling:

Repeatedly

1. Choose seed event with $w < 0$ for cell $\mathcal{C}$
2. Iteratively add nearest event to cell until $\sum_{i \in \mathcal{C}} w_i \geq 0$
   What does “nearest” mean?
3. Reweight: $w_i \to w_i' = \frac{\sum_{j \in \mathcal{C}} w_j}{\sum_{j \in \mathcal{C}} |w_j|} |w_i| \geq 0$

Sufficient statistics: cell size $<$ experimental resolution

Otherwise: limit cell size, accept $w_i' < 0$

Distances in phase space

Need distance function $d(e, e')$ between events e, e'

- Essential: $d(e, e')$ small $\Rightarrow$ e, e' look similar in detector or differ only in properties the event generator can't predict
- Desirable: $d(e, e')$ large $\Rightarrow$ e, e' look different in detector

Distances in phase space

Need distance function $d(e, e')$ between events e, e'

- Essential: $d(e, e')$ small $\Rightarrow$ e, e' look similar in detector or differ only in properties the event generator can't predict
- Desirable: $d(e, e')$ large $\Rightarrow$ e, e' look different in detector

Example: infrared safety

- $d(e, e')$ unaffected by collinear splittings with $\Theta \to 0$
- $d(e, e')$ unaffected by soft particles with $p \to 0$

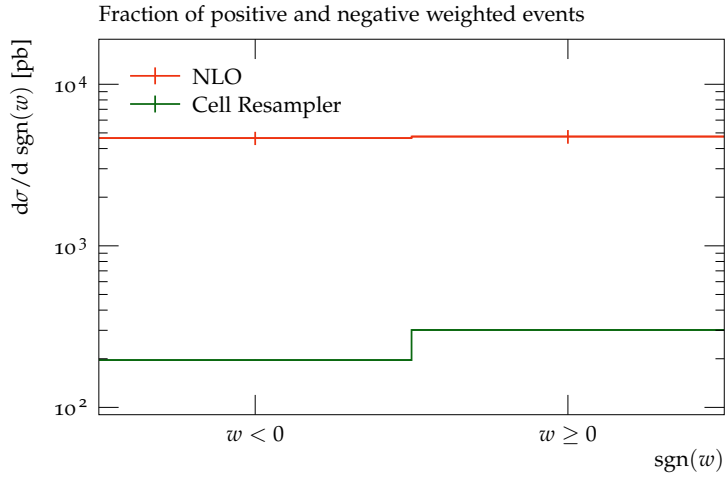
Here: Fixed-order (QCD) event generator, use jet clustering

Application: W + 2 jets at NLO

- Generate $61 \cdot 10^6$ weighted NLO events for $pp \to (W^- \to l\bar{\nu}_l) + 2$ jets (Sherpa + OpenLoops)
- Apply cell resampling:
  - Divide into subsamples with $9 \cdot 10^6$ events
  - $\tau = 10$ (sensitivity to $\Delta p_\perp$)
  - Require cell size of $d(e_s, e) < 100\,\text{GeV}$
- Analyse with Rivet `MC_WINC, MC_WJETS, MC_XS, ATLAS_2014_I1319490`: $< 10\%$ of events pass

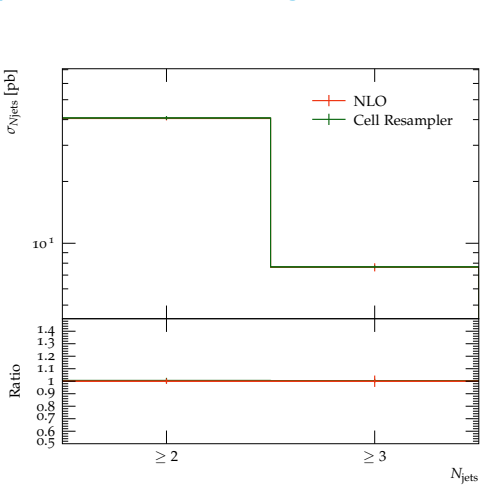
Application: W + 2 jets at NLO

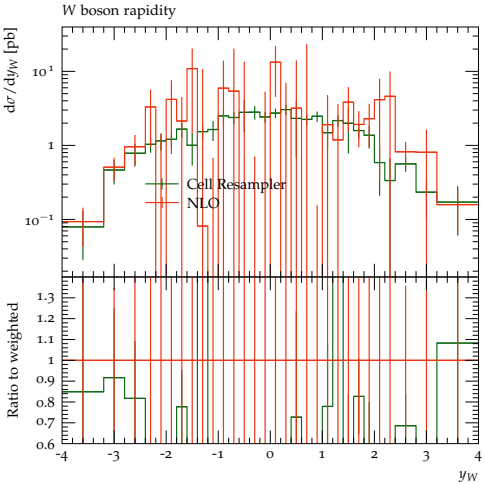
Negative weight reduction



- Amount of cancellation between $+$ and $-$ reduced by more than one order of magnitude

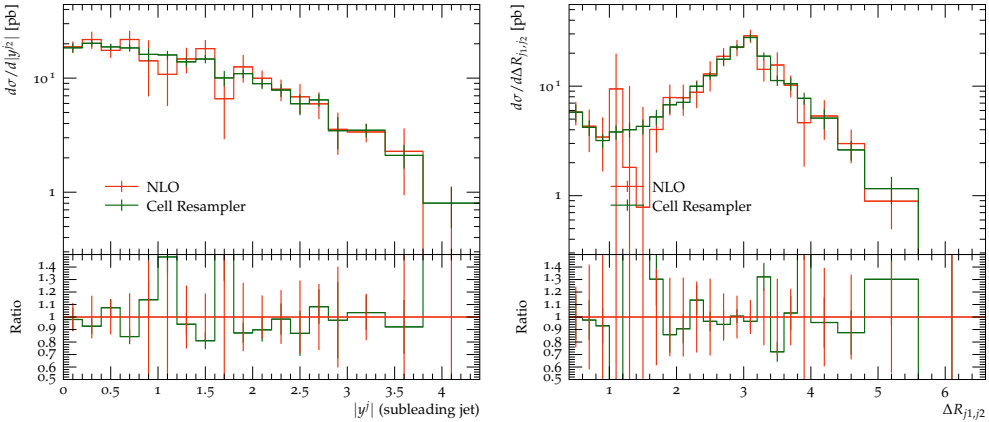
Application: W + 2 jets at NLO





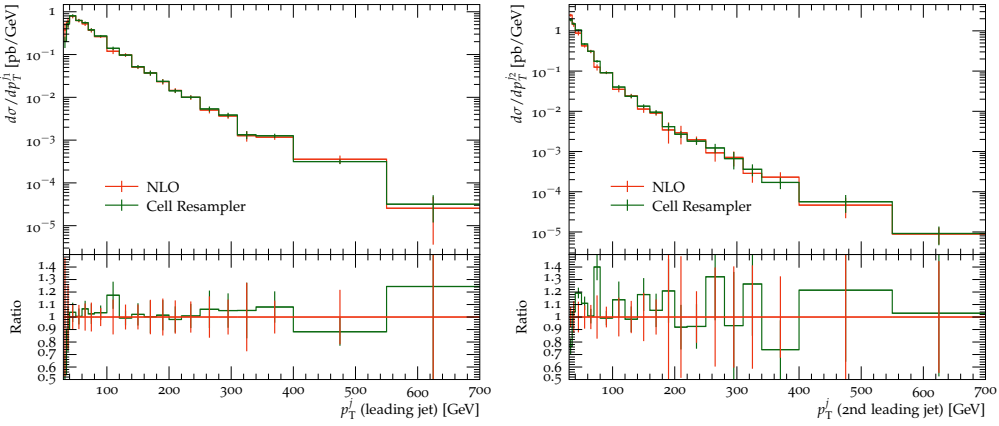
- Central prediction retained by resampling
- Resampling decreases nominal Monte Carlo error and bin-to-bin fluctuations

Application: W + 2 jets at NLO



- Shapes retained within statistical error

Application: W + 2 jets at NLO



- Shapes retained within statistical error

Runtime scaling

- Runtime for generating N events: $\mathcal{O}(N)$
- Naive cell resampling: $\mathcal{O}(N^2)$
  - Expect: #cells $\propto$ (#events with $w < 0$) $\propto$ #events
  - Naive (k) nearest neighbour search: $\mathcal{O}(N)$

Need faster nearest-neighbour search

Runtime scaling

- Runtime for generating N events: $\mathcal{O}(N)$
- Naive cell resampling: $\mathcal{O}(N^2)$
  - Expect: #cells $\propto$ (#events with $w < 0$) $\propto$ #events
  - Naive (k) nearest neighbour search: $\mathcal{O}(N)$

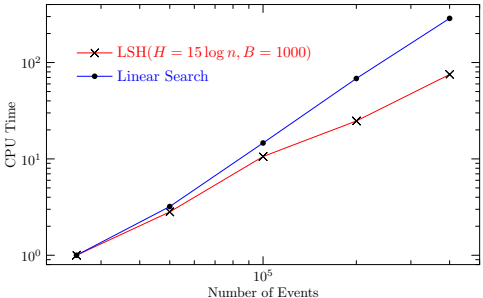
Need faster nearest-neighbour search

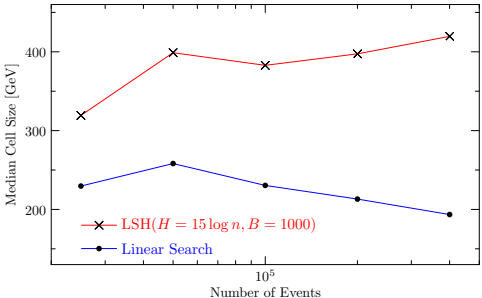
Locality sensitive hashing (LSH):

- H random hash tables (histograms) with B events in each bucket (bin)
- Nearest neighbour candidates: events that share many buckets

Locality-sensitive hashing

Performance





Properties of LSH-based nearest-neighbour search:

- \+ Better runtime scaling
- − No systematic reduction in cell size

Conclusion

Cell resampling:

- Local cancellation of negative weights
- Independent of observables
- Good performance for non-trivial example:
  $W + 2$ jets at NLO
- Implementation on `https://cres.hepforge.org`

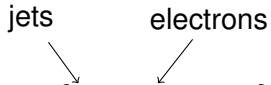
Open questions:

- Better nearest-neighbour search
- Alternative distance functions
  e.g. energy-mover's distance [Komiske, Metodiev, Thaler 2019]
- Accurate estimate of statistical error? (cf. [Nachman, Thaler 2020])

Backup

Distances in phase space

Concrete implementation



1. Collect all “particles” in event e into sets $\{ s_1 , s_2 , \ldots , s_T \}$

$$d(e, e') = \sum_{t=1}^{T} d(s_t, s'_t)$$

Distances in phase space

Concrete implementation

jets electrons

1. Collect all "particles" in event e into sets $\{ s_1 , s_2 , \ldots , s_T \}$

$$d(e, e') = \sum_{t=1}^{T} d(s_t, s'_t)$$

2. Particles in s_t have four-momenta $(p_1 , \ldots \ldots \ldots , p_P)$

   Particles in s'_t have four-momenta $(q_1 , \ldots , q_Q, 0, \ldots , 0)$

$$d(s_t, s'_t) = \min_{\sigma \in S_P} \sum_{i=1}^{P} d_t(p_i, q_{\sigma(i)})$$

Distances in phase space

Concrete implementation

jets electrons

1. Collect all "particles" in event e into sets $\{ s_1 , s_2 , \ldots , s_T \}$

$$d(e, e') = \sum_{t=1}^{T} d(s_t, s'_t)$$

2. Particles in s_t have four-momenta ($p_1 , \ldots \ldots \ldots , p_P$)

   Particles in s'_t have four-momenta ($q_1 , \ldots , q_Q, 0, \ldots , 0$)

$$d(s_t, s'_t) = \min_{\sigma \in S_P} \sum_{i=1}^{P} d_t(p_i, q_{\sigma(i)})$$

Distances in phase space

Concrete implementation

jets electrons

1. Collect all "particles" in event e into sets $\{ s_1 , s_2 , \ldots , s_T \}$

$$d(e, e') = \sum_{t=1}^{T} d(s_t, s'_t)$$

2. Particles in s_t have four-momenta $(p_1 , \ldots \ldots \ldots , p_P)$

   Particles in s'_t have four-momenta $(q_1 , \ldots , q_Q, 0, \ldots , 0)$

$$d(s_t, s'_t) = \min_{\sigma \in S_P} \sum_{i=1}^{P} d_t(p_i, q_{\sigma(i)})$$

Distances in phase space

Concrete implementation

jets electrons

1. Collect all "particles" in event e into sets $\{ s_1 , s_2 , \ldots , s_T \}$

$$d(e, e') = \sum_{t=1}^{T} d(s_t, s'_t)$$

2. Particles in s_t have four-momenta $(p_1 , \ldots \ldots \ldots , p_P)$
   Particles in s'_t have four-momenta $(q_1 , \ldots , q_Q, 0, \ldots , 0)$

$$d(s_t, s'_t) = \min_{\sigma \in S_P} \sum_{i=1}^{P} d_t(p_i, q_{\sigma(i)})$$

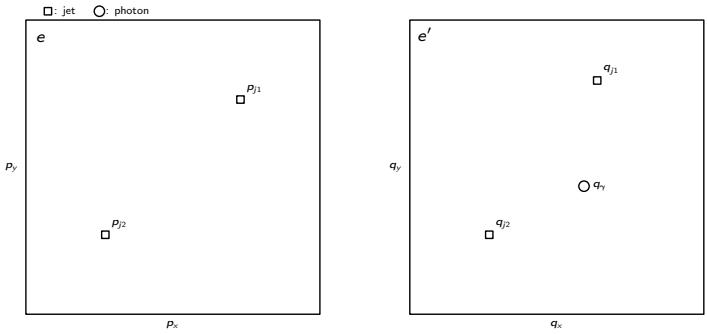
3. Choose distance function between particle momenta
   Here: independent of particle type t

$$d_t(p, q) = \sqrt{\sum_{i=1}^{3} (p_i - q_i)^2 + \tau^2 (p_\perp - q_\perp)^2}$$

τ: tunable parameter

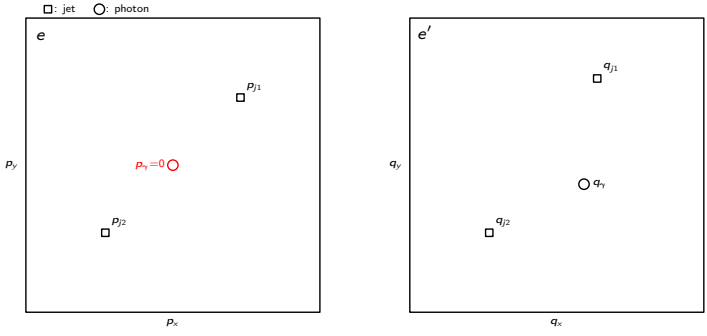
Distances in phase space

Example



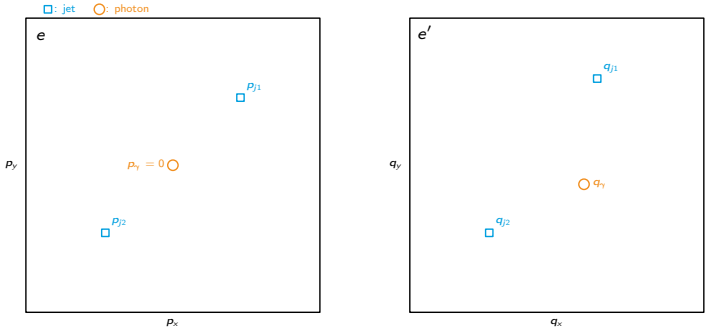
Distances in phase space

Example



Distances in phase space

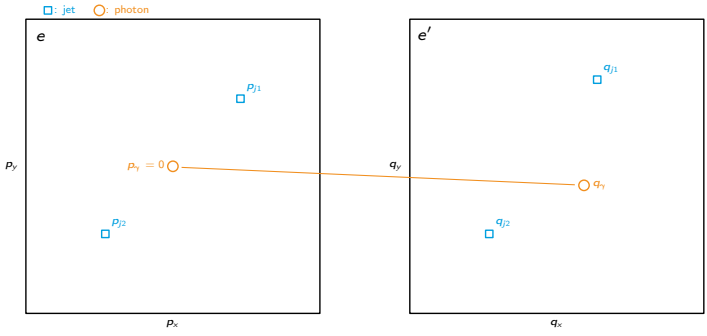
Example



$$d(e, e') = d(s_j, s'_j) + d(s_\gamma, s'_\gamma)$$

Distances in phase space

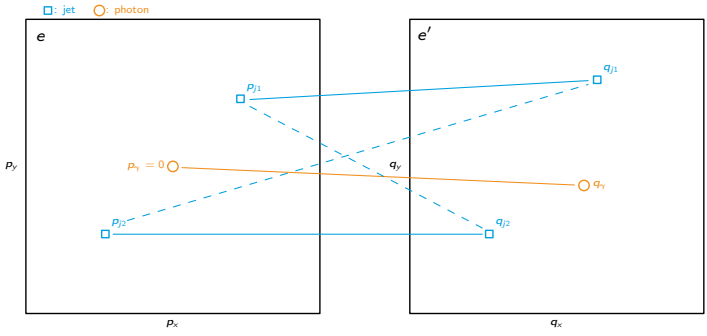
Example



$$d(e, e') = d(s_j, s_j') + d(s_\gamma, s_\gamma')$$
$$= d(s_j, s_j') + d(p_\gamma, q_\gamma)$$

Distances in phase space

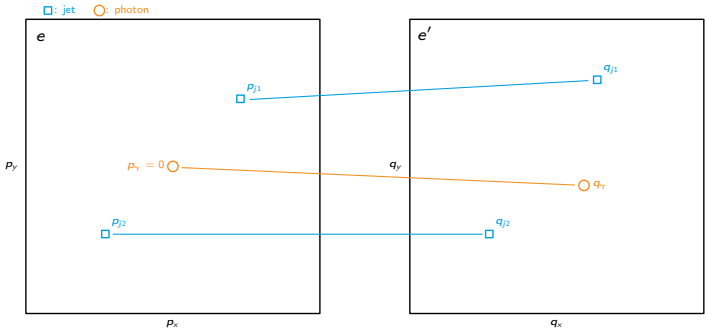
Example



$$d(e, e') = d(s_j, s_j') + d(s_\gamma, s_\gamma')$$
$$= \min\left[d(p_{j1}, q_{j1}) + d(p_{j2}, q_{j2}), d(p_{j1}, q_{j2}) + d(p_{j2}, q_{j1})\right] + d(p_\gamma, q_\gamma)$$

Distances in phase space

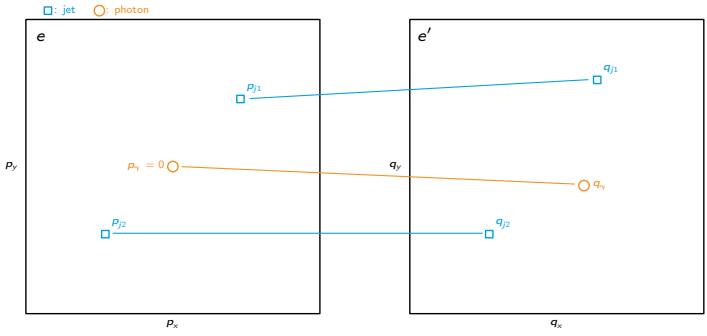
Example



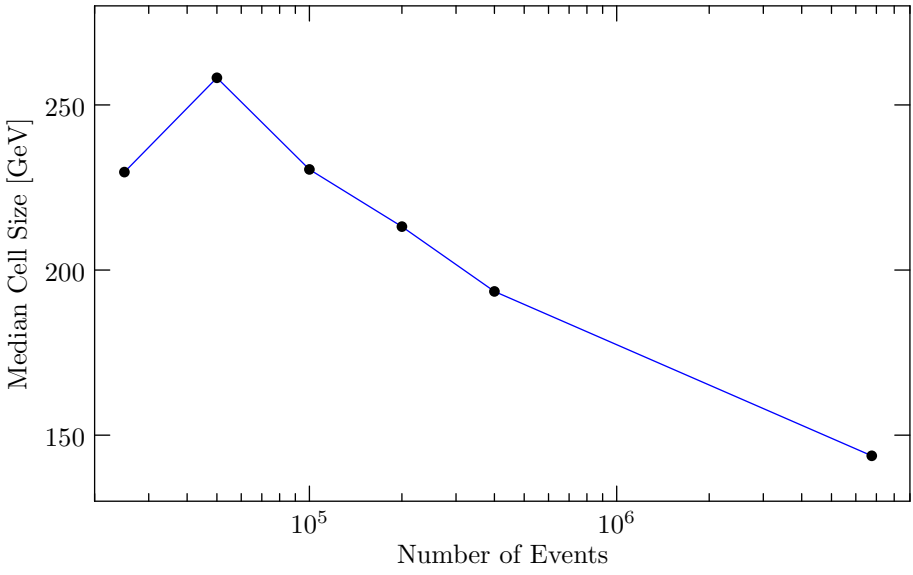
$$d(e, e') = d(s_j, s'_j) + d(s_\gamma, s'_\gamma)$$
$$= d(p_{j1}, q_{j1}) + d(p_{j2}, q_{j2}) + d(p_\gamma, q_\gamma)$$

Distances in phase space

Example



$$d(e, e') = d(s_j, s_j') + d(s_\gamma, s_\gamma')$$
$$\overset{\tau=0}{=} |\vec{p}_{j1} - \vec{q}_{j1}| + |\vec{p}_{j2} - \vec{q}_{j2}| + |\vec{p}_\gamma - \vec{q}_\gamma|$$


Median Cell Size [GeV]
250
200
150
10^5
10^6
Number of Events

Runtime scaling

- Runtime for generating N events: $\mathcal{O}(N)$
- Naive cell resampling: $\mathcal{O}(N^2)$
  - Expect: #cells $\propto$ (#events with $w < 0$) $\propto$ #events
  - Naive (k) nearest neighbour search: $\mathcal{O}(N)$

Need faster nearest-neighbour search

Runtime scaling

- Runtime for generating N events: $\mathcal{O}(N)$
- Naive cell resampling: $\mathcal{O}(N^2)$
  - Expect: #cells $\propto$ (#events with $w < 0$) $\propto$ #events
  - Naive (k) nearest neighbour search: $\mathcal{O}(N)$

Need faster nearest-neighbour search

Small number of dimensions:

- Histograms
- Voronoi cells ($\rightarrow$ `fastjet`)

Here: locality-sensitive hashing (LSH)

Locality-sensitive hashing

Preparation

Go to high-dimensional Euclidean space

For each particle type t, sort momenta and arrange in vector:

$$V(e) = \begin{pmatrix} \vec{p}_1 \\ \vec{p}_2 \\ \vdots \\ \vec{k}_1 \\ \vec{k}_2 \\ \vdots \\ \vdots \end{pmatrix} \begin{matrix} \\ \\ \\ \} \, t = 1 \\ \\ \} \, t = 2 \\ \\ \\ \end{matrix}$$

$$\left|V(e) - V(e')\right| \text{ small} \Rightarrow d(e, e') \text{ small}$$

Locality-sensitive hashing

Hash table initialisation

1. Random projections:
   Projectors Π_h onto $H = \mathcal{O}(\log N)$ random hyperplanes

Locality-sensitive hashing

Hash table initialisation

1. Random projections:
   Projectors Π_h onto $H = \mathcal{O}(\log N)$ random hyperplanes
2. Take first component:
   $$V_e \mapsto c_{e,h} = (\Pi_h V_e)_1$$

Johnson-Lindenstrauss Lemma suggests

$$d(V_e, V_{e'}) \text{ small} \Leftrightarrow |c_{e,h} - c_{e',h}| \text{ small}$$

with high probability

Locality-sensitive hashing

Hash table initialisation

1. Random projections:
   Projectors Π_h onto $H = \mathcal{O}(\log N)$ random hyperplanes
2. Take first component:
   $$V_e \mapsto c_{e,h} = (\Pi_h V_e)_1$$
   Johnson-Lindenstrauss Lemma suggests
   $$d(V_e, V_{e'}) \text{ small} \Leftrightarrow |c_{e,h} - c_{e',h}| \text{ small}$$
   with high probability
3. H hash tables (histograms) from coordinates
   $B = \mathcal{O}(\log N)$ entries in each bucket (bin).

Locality-sensitive hashing

Nearest-neighbour search

For cell seed event e_s

1. Select candidates:
   events that share the most hash table buckets (histogram bins) with e_s
2. Compute $d(e_s, e'_s)$ for all candidates
3. Fill up cell using actual nearest neighbours

Runtime scaling: $\mathcal{O}(HBN) = \mathcal{O}(N \log^2 N)$