Below I give: (A) high-level app architecture and UI/state flow, (B) algorithm + how matrix math is used (concrete), (C) fusion-of-board-games logic, (D) Cursor IDE prompt you can paste, and (E) a recommended file tree. I kept it compact; want a verbose deep-dive or starter code next? Estimated tokens for this reply: ~650.

instructions

A — App overview (one-paragraph)

A desktop/web app that runs a timed scan window (e.g., 6 hours) over the SHR/Murder Accountability dataset to compute pairwise similarity between cases, detect matches (high similarity), link chains (graph walkability), and clusters (communities). The UI is an animated, colorful stage panel with big stage chip (Idle → Scanning → Matches → Links → Clusters → Report). Main visualization: an interactive matrix / heatmap (cases × cases) that morphs into a graph view (nodes = cases, edges = links) with geographic timeline lanes. Provide progressive feedback: live counts, top candidate chains, and a “20-Questions” guided filter sidebar to narrow searches.

B — Algorithm / matrix-math (concrete)

Data input: rows from CSV with columns you gave (ID,CNTYFIPS,Ori,State,...,MSA). Preprocess into numeric feature vectors.

Feature encoding

Categorical → one-hot / embedding (Victim sex/race, Weapon, Relationship, Circumstance, Agency, MSA).

Numerical/time → normalized (Year, Month, VicAge, OffAge).

Text fields (Incident, Subcircum, Ori/Source) → TF-IDF or simple bag-of-words vector.

Create vector v\_i ∈ R^d for each case i.

Similarity matrix (matrix multiplication)

Stack vectors into matrix V with shape (n\_cases × d).

Compute raw similarity via matrix multiplication: S\_raw = V · V^T (this yields dot products).

To get cosine similarities: normalize rows (L2) then S = V\_norm · V\_norm^T.

Complexity note: for n large, compute in blocks (streaming) or use sparse representations.

Thresholding → adjacency

Choose threshold t\_match (e.g., 0.7 cosine). Create binary adjacency A\_ij = 1 if S\_ij ≥ t\_match else 0.

Also compute time and geography masks: only allow links where |year diff| ≤ Y\_window and distance(MSA/CNTYFIPS) ≤ G\_window (configurable).

Link-finding & transitive closure

Links = edges of graph G(A). Use matrix powers or graph BFS:

Transitive closure / reachable sets: compute R = A + A^2 + ... (matrix multiplication) limited to small k to find chains.

For short chains, compute A^k to find length-k paths (matrix powers).

Clustering

Use spectral clustering on Laplacian L = D − A (eigenvectors = matrix ops) or community detection (Louvain on graph).

Present top clusters (sorted by cluster score: density × unsolved ratio).

Scoring & ranking

Score edges by weighted sum: score = α·similarity + β·(1 - solved\_flag) + γ·1/time\_proximity + δ·geog\_proximity.

Produce chain score (sum / normalization) and rank candidate chains.

C — UI & state machine (visual ideas)

Top bar stage chip (large pill) — color map:

Idle = faded gray

Scanning = animated teal pulse + spinner

Found matches = bright yellow

Found links = orange

Clusters = magenta

Reporting = green

Center pane: dual-view toggle

Matrix heatmap: rows/cols = filtered cases. Hotter = higher similarity. Hover shows case summary.

Graph view: nodes sized by VicCount/OffCount, colored by Solved status; edges thickness = score.

Left sidebar: “20-Questions” filter builder (attributes presented as question tiles). Each question selection reduces cases; app shows remaining candidate count like Battleship hits.

Bottom strip: real-time log (events found), timer countdown, “Pause/Resume/Export” buttons.

Right panel: chain inspector (drop-in clone of Guess Who: progressive elimination of suspects), and Clue-like reasoning box showing likely motive/weapon intersections.

D — Fusion of games (how each helps design logic)

20 Questions → attribute-driven narrowing UI. The app asks or the analyst picks a binary attribute (e.g., victim female? weapon=sharp?). Each answer filters V.

Clue → causal inference cards: Weapon, Relationship, Circumstance — use to propose hypothesis sets (e.g., stranger + strangulation + female).

Battleship → geographical/time grid: user fires 'pings' (filter) at MSA/time bins; hits reveal clusters/density; repeated hits reveal a pattern.

Guess Who → progressive elimination of suspects/attributes from a candidate set; used for prioritizing investigative leads.

E — Cursor IDE prompt (pasteable) Project: CaseLinker — SHR serial-pattern scanner

Goal: Build a Python/React app that scans a provided SHR/ Murder Accountability CSV feed for a timed window (configurable, default 6 hours), computes pairwise case similarity using matrix operations, detects matches/links/clusters, and displays results in a colorful interactive UI with stage states: Idle, Scanning, Matches, Links, Clusters, Reporting.

Requirements:

* Backend: Python 3.10+, FastAPI
  + Modules: data\_loader.py, features.py, matcher.py, graph\_engine.py, clusterer.py, scan\_controller.py
  + Use numpy, scipy, scikit-learn, networkx (or igraph), pandas.
  + Streaming/batching support (process n rows at a time to avoid memory blowups).
  + REST endpoints:
    - POST /scan/start {duration\_hours, thresholds, filters}
    - GET /scan/status
    - GET /scan/results?view=heatmap|graph|clusters
    - GET /case/{id}
* Frontend: React + Tailwind (or simple stylesheet)
  + Main components: StageChip, MatrixHeatmap (canvas/webgl), GraphView (d3), FilterPanel (20-questions), ChainInspector, LogStrip
  + Live updates via WebSockets for progress events.
* Persistence: local disk JSON for scan sessions and export CSV/JSON reports.
* Tests: unit tests for features and matrix ops.

Deliverables for first sprint:

1. data\_loader.py: read CSV using pandas, normalize columns listed (ID,CNTYFIPS,...,MSA).
2. features.py: transformer functions with tests.
3. matcher.py: produces similarity matrix S using block matrix multiplication and normalization.
4. graph\_engine.py: threshold to adjacency, compute A^2 and up to A^k, BFS chain listing.
5. minimal FastAPI server exposing endpoints.
6. React shell showing StageChip and a static heatmap placeholder that updates via websocket events.

Extra notes:

* Respect privacy: do not map names (none present), follow data use rules.
* Include config file for windows: year\_window, geo\_window, similarity\_threshold, chain\_length\_k.

Implementation hints:

* Use numpy.matmul for core matrix ops. For large n, implement chunked inner products: for i in batches: S\_block = V\_block @ V.T
* For spectral clustering, use sparse Laplacian in scipy.sparse.
* Provide a CLI "scan\_local" that runs the same pipeline headless and writes results to /outputs

Please scaffold the repo, producing the file tree and stubs for each Python file with TODOs and a simple unit test for similarity correctness.

F — Suggested file tree (copy/paste) caselinker/ ├─ backend/ │ ├─ app.py # FastAPI entry │ ├─ scan\_controller.py # manages timed scan session │ ├─ data\_loader.py # CSV parsing, cleaning (columns provided) │ ├─ features.py # Vectorize/encode cases │ ├─ matcher.py # similarity matrix (V @ V.T), thresholding │ ├─ graph\_engine.py # adjacency, path/chain detection (A^k) │ ├─ clusterer.py # spectral / louvain clustering │ ├─ report.py # export results │ ├─ config.py # thresholds & windows │ └─ tests/ │ └─ test\_matcher.py ├─ frontend/ │ ├─ src/ │ │ ├─ App.jsx │ │ ├─ components/ │ │ │ ├─ StageChip.jsx │ │ │ ├─ MatrixHeatmap.jsx │ │ │ ├─ GraphView.jsx │ │ │ ├─ FilterPanel.jsx │ │ │ └─ ChainInspector.jsx │ │ └─ ws.js # websocket client │ └─ package.json ├─ data/ │ └─ sample\_shr.csv ├─ docs/ │ └─ murderdata\_instructions.md # include original file (:contentReference[oaicite:1]{index=1}) └─ README.md

G — UX notes & operational tips

6-hour scan: run scan\_controller.start(duration=6h) which streams CSV in chunks, emits partial heatmaps so UI shows progressive filling.

Explainability: for each match/edge, show contributing features and weights so analyst sees why they matched (weapon+age+MSA+month).

Safety/ethics: label everything as hypotheses, not accusations. Provide export with metadata and confidence scores. Add large warning banner.

Performance: if n > 20k, use approximate nearest neighbor (ANN) for candidate pairs; matrix multiplication scales O(n^2) so filter by time/MSA first.

## Additional AI suggestions

If you want I can:

produce a small starter backend matcher.py skeleton (only the key lines) and a minimal FastAPI endpoint (no fluff), or

generate the exact Cursor prompt for a code agent to create the repo with stubs.

Which do you want next — starter code (brief), or the Cursor agent prompt to scaffold the repo?