Analyzing clickstream sessions as little directed graphs is a powerful way to capture transition‑based behaviors—and then comparing graph‑level features (or even subgraph patterns) between your satisfied vs. dissatisfied groups can reveal what “paths” are most diagnostic of frustration. Below are a few approaches you might try, ordered roughly from simplest to more advanced.

---

1. Global Markov‑Chain Comparison

1. **Build two transition matrices**
    - One for all satisfied sessions, one for all dissatisfied.
    - Each matrix TijT_{ij} counts transitions from page i to page j, normalized to probabilities.
2. **Highlight “hot” edges**
    - Compute the difference ΔT=Tdiss−Tsat\Delta T = T^\text{diss} - T^\text{sat}.
    - Sort edges by ∣ΔTij∣|\Delta T_{ij}| to find transitions much more (or less) common in dissatisfied users.
3. **Statistical testing**
    - For top candidate transitions, do a chi‑square or proportion‑test to confirm significance.

This gives you a ranked list of “most distinctive” page-to-page moves in each group.

---

2. Per-Session Directed-Graph Features

Treat each session as its own directed graph G=(V,E)G=(V,E) where

- VV = the set of pages visited,
- EE = directed edges for each successive hit (with possible edge weights = counts).

For each session-graph compute simple graph metrics, for instance:

- **Number of unique nodes** (breadth of exploration)
- **Total edges** (path length)
- **Average out-degree** (how “branchy” a session is)
- **Diameter** or **longest shortest-path** (how deep/deviated a session goes)
- **Proportion of self-loops** (repeated page hits)
- **Page-rank or in-/out-centrality** of high-traffic nodes

Then compare the distributions of these metrics between satisfied vs. dissatisfied (e.g., via boxplots or a Mann-Whitney U test). Features that differ most can pinpoint behavioral signatures (e.g., dissatisfied users have higher average out-degree, suggesting more aimless clicking).

---

3. Mining Subgraph/Motif Patterns

Rather than mining sequential patterns with PrefixSpan, you can:

- **Enumerate small k-node subgraphs** (e.g., 2- or 3-step paths) using a graph-mining library (like NetworkX’s `triadic_census` or custom code).
- **Count motif frequencies per session**, then do a differential analysis (e.g., t-tests) on motif counts between the two groups.
- This often scales better than full sequence mining, since you limit motif size.

---

4. Sequence-Embedding + Clustering

1. **Embed each session sequence** into a fixed-length vector:
    - Use n-gram TF-IDF on page bigrams/trigrams.

    - Or train a simple sequence embedding (Word2Vec on pages + average, or doc2vec per session).
2. **Cluster embeddings** (e.g., k-means) and label clusters by their satisfaction rate.
    - Clusters dominated by dissatisfied sessions reveal common patterns.
3. **Interpret via top n-grams** or representative sequences per cluster.

---

5. Differential N-Gram Ranking

If full PrefixSpan is too slow, fall back to:

- **Counting all 2- and 3-grams** of pages (sliding window of size 2 or 3) via a single pass through your data.
- For each n-gram compute support in both groups and rank by difference in support.
- This is O(NL)O(NL) (where LL is average session length) and much faster than mining arbitrary-length patterns.

---

6. Session Graph Embeddings (Graph2Vec)

- Represent each session-graph as a point in embedding space using Graph2Vec (or similar).
- Train a simple classifier (e.g., logistic regression) on the embeddings.
- Use the classifier's coefficients or SHAP values to identify which graph "structures" contribute most to predicting dissatisfaction.

---

7. Putting It All Together: Sample Workflow in Python

Here's a high-level sketch of how you might code approach (2) with NetworkX:

```
import networkx as nx
import pandas as pd
from scipy.stats import mannwhitneyu

# assume df has columns ['session_id', 'page_sequence', 'satisfaction']
features = []
for sid, group in df.groupby('session_id'):
    seq = group['page_sequence'].iat[0]         # a list of pages
    sat = group['satisfaction'].iat[0]
    G = nx.DiGraph()
    # build edges
    for u, v in zip(seq, seq[1:]):
        if G.has_edge(u, v):
            G[u][v]['weight'] += 1
        else:
            G.add_edge(u, v, weight=1)
    # compute metrics
    num_nodes = G.number_of_nodes()
    num_edges = G.number_of_edges()
    avg_out_deg = sum(d for _, d in G.out_degree()) / num_nodes
    diameter = nx.diameter(G.to_undirected()) if num_nodes > 1 else 0
    features.append({
        'session_id': sid,
        'satisfaction': sat,
        'num_nodes': num_nodes,
        'num_edges': num_edges,
        'avg_out_deg': avg_out_deg,
        'diameter': diameter
    })

feat_df = pd.DataFrame(features)

# Compare distributions
metrics = ['num_nodes','num_edges','avg_out_deg','diameter']
for m in metrics:
    sat_vals = feat_df.loc[feat_df.satisfaction==1, m]
    dis_vals = feat_df.loc[feat_df.satisfaction==0, m]
    stat, p = mannwhitneyu(sat_vals, dis_vals, alternative='two-sided')
    print(f"{m}: U={stat:.1f}, p={p:.3g}")
```

✦ From here you'd pick the metrics with the most significant p-values as your key behavioral differences.

Recommendations

- **Start simple**: try the global Markov‑chain edge‑difference method (Section 1). It’s fast, highly interpretable, and often surfaces the biggest “aha” transitions.
- **Augment with graph features** (Section 2) to capture broader navigation behaviors.
- If you need more nuance, dive into motif mining or embedding-based clustering.

By combining these techniques you’ll not only avoid the scalability issues of PrefixSpan on long sessions, but also gain richer, graph-based insights into what frustrated users do differently.

Global Markov-chain comparison

Here’s a self‑contained Python example showing how to do a global Markov-chain comparison on a tiny toy dataset. You can easily extend the same logic to your full clickstream.

```
import pandas as pd

# 1) Toy data: each row is one session’s page sequence and its satisfaction label
data = [
    {"session": "s1", "pages": ["Home", "Search", "Product", "Cart"],       "satisfaction": 1},
    {"session": "s2", "pages": ["Home", "Search", "Product", "Checkout"],   "satisfaction": 1},
    {"session": "s3", "pages": ["Home", "Search", "Search", "Product"],     "satisfaction": 0},
    {"session": "s4", "pages": ["Home", "Help",   "Home",   "Search"],      "satisfaction": 0},
]
df = pd.DataFrame(data)

# 2) Build transition‑count matrices for each group
def build_transition_counts(seqs):
    counts = {}
    for seq in seqs:
        for u, v in zip(seq, seq[1:]):
            counts[(u, v)] = counts.get((u, v), 0) + 1
    return counts

sat_seqs = df[df.satisfaction == 1].pages
dis_seqs = df[df.satisfaction == 0].pages

sat_counts = build_transition_counts(sat_seqs)
dis_counts = build_transition_counts(dis_seqs)
```

```
# 3) Turn counts into DataFrames
all_edges = set(sat_counts) | set(dis_counts)
rows = []
for (u, v) in all_edges:
    rows.append({
        "from": u, "to": v,
        "count_sat": sat_counts.get((u, v), 0),
        "count_dis": dis_counts.get((u, v), 0)
    })
edges_df = pd.DataFrame(rows)

# 4) Normalize to get probabilities
total_sat = sum(sat_counts.values())
total_dis = sum(dis_counts.values())
edges_df["p_sat"] = edges_df["count_sat"] / total_sat
edges_df["p_dis"] = edges_df["count_dis"] / total_dis

# 5) Compute difference and sort
edges_df["delta_p"] = edges_df["p_dis"] - edges_df["p_sat"]
edges_df = edges_df.sort_values("delta_p", key=abs, ascending=False)

print(edges_df)
```

What this does

1. **Collects transition counts** separately for satisfied vs. dissatisfied sessions.
2. **Builds a DataFrame** listing each possible directed edge (u→v)(u \to v) with counts in each group.
3. **Normalizes** by the total number of transitions in each group to get transition probabilities puvsatp^\text{sat}_{uv} and puvdisp^\text{dis}_{uv}.
4. **Computes** Δpuv=puvdis−puvsat\Delta p_{uv} = p^\text{dis}_{uv} - p^\text{sat}_{uv} and sorts by ∣Δp∣|\Delta p|.

The top rows of the resulting table are the page-to-page moves that differ most between dissatisfied and satisfied users. In your real data you'd just replace the toy `data` list with your full clickstream DataFrame and this will scale to thousands of sessions.