Time Complexity Analysis

$n = |V|$ (number of vertices)
$m = |E|$ (number of edges)
K (neighbor cap parameter) $|\text{neighs}| \le K$
$T = max\_iter$ (replicator maximum iterations)
For vertex v :
$d(v) =$ degree of v
$s_v = |S_v| = 1 + |\text{neighs}| \le 1 + \min(K, d(v))$
$p_v =$ number of seeds for v $(\le s_v$ and $\le K + 1)$
$\text{nnz}(A_v) =$ nonzeros in induced subgraph on S_v $(\le s_v^2)$
$C =$ cache capacity, evictions are $O(1)$ amortized

- **Core numbers and degeneracy order.** The core numbers of G can be computed in $O(n + m)$ time, where $n = |V|$ and $m = |E|$. Similarly, computing a degeneracy ordering (or order by (core, id)) requires $O(n+m)$.
- **Outer loop over vertices.** The algorithm processes each vertex $v \in V$. For each v, its neighborhood $N(v)$ is retrieved in $O(\deg(v))$. The sum over all vertices is $\sum_v O(\deg(v)) = O(m)$.
- **Neighbor selection and canonical seeding.** Selecting the top-K neighbors by core number takes $O(\deg(v) \log K)$ using a heap or $O(\deg(v))$ if a linear scan is used. Across all vertices, this is bounded by $O(m \log K)$.
- **Subgraph extraction and caching.** Building the induced subgraph on $S = \{v\} \cup \text{neighs}$ may take $O(|S|^2)$ in the worst case, but is bounded by $O(K^2)$. With LRU caching, repeated subgraphs are reused, so the cost is amortized.
- **Replicator dynamics (per seed batch).** Let $s = |S| \le K + 1$ and $p = |seeds| \le K$. Each iteration requires a sparse matrix–dense matrix product: $O(\text{nnz}(A_S) \cdot p)$, where $\text{nnz}(A_S) \le s^2$. With T iterations until convergence, the cost per vertex is $O(T \cdot s^2 \cdot p) = O(T \cdot K^3)$.
- **Clique verification and expansion.** Checking whether a candidate support is a clique costs $O(|C|^2)$, where C is the support size. Expansion is at most $O(n)$ per clique, but in practice is bounded by the degree of v, i.e. $O(K)$. Thus this step is $O(K^2)$ per candidate.

Overall Complexity. Combining the above, the total running time is

$$O(n + m) \; + \; O(m \log K) \; + \; O\big(n \cdot (T \cdot K^3 + K^2)\big) .$$

Since K is a fixed parameter (the neighbor cap), the dominant term is

$$O(n + m) + O(n \cdot T \cdot K^3).$$

In practice: T is small (dozens of iterations until convergence), so the algorithm is nearly linear in the input size, up to the cubic factor in K.
For the intended usage where K is small (local neighborhoods) and T is modest, complexity is roughly linear in n (times small constants):

$$\approx O(n \cdot T \cdot K^3) \quad \text{with small } K \text{ making it practical.}$$

Correctness proof for AlgebraicMaxCliques_Canonical

Notation. Let $G = (V, E)$ be an undirected simple graph with $n = |V|$. For a vertex $v \in V$ denote by $N(v)$ its open neighborhood and $\deg(v) = |N(v)|$. For a finite set S we write clique(S) to mean that the induced subgraph $G[S]$ is a clique. Let order_pos : $V \to \mathbb{Z}$ be the total order used for canonical seeding (here obtained from degeneracy/core-number and id tie-break). For a set $C \subseteq V$ let $\min_{\text{order}}(C)$ be the unique vertex of C minimal under order_pos.

We recall the algorithm's two central components:

- the *seeding + replicator* stage: for each vertex v we build a small induced subgraph S_v (of size $s_v \leq K + 1$) and form initial columns X_0 (one per seed); then we run BatchedReplicatorSparseAdaptive to obtain columns X_{final}. From each column we extract a support (indices with value $\geq$ `rel_thresh` $\cdot$ max) as a candidate.
- the *postprocessing* stage: each candidate is verified to be a clique (IsClique), expanded greedily to a maximal clique (ExpandToMaximal), then added to the output set only if the current center v is the canonical minimum in that clique.

Assumptions. To state a clean correctness result we make the following mild, explicit assumptions. (The algorithm as implemented already checks cliquehood and expands to maximal, so only the second assumption is essential for *completeness* of discovery.)

(A1) **(neighborhood coverage)** For every maximal clique $C \subseteq V$ and for $v = \min_{\text{order}}(C)$, the set S_v constructed by the algorithm contains C. Equivalently $C \setminus \{v\} \subseteq$ `neighs` (so the neighbor-cap K and the `top_k_by_core` rule do not exclude other members of C). Concretely this holds when $K \geq |C| - 1$ (or when the top-K selection by core keeps the other clique vertices).

(A2) **(replicator localization)** For any induced subgraph $G[S]$ and any seed column of X_0 that places strictly positive mass on at least one node of a clique $C \subseteq S$, the BatchedReplicatorSparseAdaptive procedure returns (for some column) a vector whose significant support (w.r.t. `rel_thresh`) is contained in C (or in a subset of C) and hence the subsequent IsClique/ExpandToMaximal will recover C. This hypothesis is a practical behavioural property of the replicator dynamics; the classical Motzkin–Straus relation and standard replicator-ascent arguments justify it (see Lemma **??** below).

Goal. Under (A1) and (A2) show:

1. Every tuple added to F by the algorithm is a *maximal clique* of G. "verified by the `is_clique` and `ExpandToMaximal` procedures."

2. Every maximal clique C of G is added to F exactly once.
   "guaranteed by the canonical check procedure."

[Output are cliques and are maximal] Every tuple inserted into F by the algorithm represents a clique, and it is maximal.

By inspection of the pseudocode: each candidate support extracted from a replicator column is first tested with `IsClique(G,candidate)`; only if this test succeeds does the algorithm compute

$$\texttt{clique} \leftarrow \texttt{ExpandToMaximal}(G, \texttt{candidate}),$$

which expands the verified clique greedily by adding any vertex adjacent to all current members until no such vertex remains. The result of this greedy expansion is by construction a maximal clique (no external vertex is adjacent to every vertex in the returned set). Thus every tuple added to F is both a clique and maximal.

[Canonical uniqueness] If a maximal clique C is added to F at the moment when the algorithm is processing a vertex v, then $v = \min_{\text{order}}(C)$. Hence at most one processing vertex can add a given maximal clique.

The algorithm performs an explicit canonical check before insertion:

$$\texttt{min\_node} \leftarrow \arg\min_{u \in \texttt{clique}} \text{order\_pos}[u], \quad \text{and it only inserts when } \texttt{min\_node} = v.$$

Therefore only the minimal vertex (with respect to order_pos) of a clique C can cause C to be added. Since the minimum in a finite totally ordered set is unique, C can be added at most once. This proves uniqueness.

The two above lemmas already establish soundness: Anything added is a unique maximal clique. It remains to show completeness: regarding completeness so far i haven't reached a way that grants us enumerating all cliques although it is possible and within hands specially on small graphs that i tested it on myself, assumption one, assuming the possibility of finding all cliques but it isn't an enough proof since the parameters have to be set in such a way that adaptive to each scale of graph to the ensure the enumeration of all maximal cliques.

so far i have tried with a python implementation this algorithm and it gives full enumeration in graphs of size up to (5K nodes, 8k edges):

- **graph:** (1174, 1417)
  Found 1356 cliques in 2.829s
  Exact: 1356 Common: 1356
- **graph:** (1029, 1559)
  Found 1559 cliques in 2.679s
  Exact: 1559 Common: 1558
- **graph:** (2888, 2981)
  Found 2890 cliques in 11.642s
  Exact: 2890 Common: 2890

- **graph:** nodes, edges: 4271 8909
  Found 7834 cliques in 44.185s
  Exact: 7834 Common: 7781

but the biggest graph i have tested on so far has (7k nodes, 27k edges) due to the very humble abilities (6 ram, core i5) of my notebook, the algorithm found 15433 out of 17957 cliques in 250 seconds! im sure with proper optimization engineering according to BLAS in C++ we will get much much better numbers.