A MEMORY BOUND ANALYSIS

We next provide an analysis of the memory bound of T-FSM. Recall from Figure 6 on Page 6 that T-FSM that has 4 places that consumes space: (1) the input graph G, **the space cost of which is denoted by** $O(|G|)$; (2) the list of active patterns $\mathcal{L}_{active}$ currently under evaluation with tasks; (3) the stack of candidate patterns C_{pat} extended from examined patterns but not yet started evaluation; (4) the space used by compers for task evaluation. We next analyze the memory space costs of (2)–(4).

First consider (2) $\mathcal{L}_{\text{active}}$. Recall from Section 3 that we only allow $\mathcal{L}_{active}$ to contain at most n^{max}_{active} active patterns, where $n^{max}_{active} = 32$ by default. Each pattern S in $\mathcal{L}_{active}$ is kept as a pattern capsule that consists of three parts: (i) pattern status as shown in Figure 7 on Page 7, and tasks of S as shown in Figure 8, with (ii) regular tasks generated from the domain table being kept in Q_{reg}, and (iii) timeout tasks being kept in $\mathcal{L}_{timeout}$. We now analyze their memory cost.

- For (i), as Figure 7 shows, the dominant space cost is by the domain table, which gives $O(\sum_{i=1}^{k} |D(u_i)|)$ where $k = |V^S|$ is the number of vertices in pattern S. Note that it usually holds that $|D(u_i)| \ll |V^G|$ so the space cost is small. Another space cost is by the auxiliary structure $\mathcal{A}_S$ as illustrated by Figure 5 on Page 5 which is also typically small for the same reason. Overall, patterns status of S takes $O(\sum_{i=1}^{k} |D(u_i)| + |\mathcal{A}_S|)$.
- For (ii), as the second paragraph of Section 3.3 has indicated, Q_{reg} holds at most $(n^{min}_{reg} + n_{batch})$ tasks in memory at any time. Each task in Q_{reg} simply corresponds to a vertex in the domain table that it begins subgraph matching, so the space cost is O(1). So the space cost of Q_{reg} is $O(n^{min}_{reg} + n_{batch})$.
- For (iii), let the maximum number of timeout task at any moment be $n^{max}_{timeout}$, which tends to be small since the number of timeout tasks if very limited according to our experiments, and we prioritize already decomposed tasks for task fetching. Since each timeout task of pattern S needs to keep the current partial match which occupies space $O(k)$, the total memory cost if $\mathcal{L}_{timeout}$ is $O(k \cdot n^{max}_{timeout})$.

Overall, the memory cost of a pattern capsule for S is $O\big(\sum_{i=1}^{k} |D(u_i)| + |\mathcal{A}_S| + n^{min}_{reg} + n_{batch} + k \cdot n^{max}_{timeout}\big)$. If we denote the average size of $|D(u_i)|$ by D_{avg}, and the average size of $|\mathcal{A}_S|$ by $\mathcal{A}_{avg}$, then the memory cost of a pattern capsule for S can be simplified as $O\big(k(D_{avg} + n^{max}_{timeout}) + \mathcal{A}_{avg} + n^{min}_{reg} + n_{batch}\big)$. Let the maximum number of vertices in a pattern be k_{max}, then the memory cost of $\mathcal{L}_{active}$ is bounded by $O\big(n^{max}_{active}\big(k_{max}(D_{avg} + n^{max}_{timeout}) + \mathcal{A}_{avg} + n^{min}_{reg} + n_{batch}\big)\big)$.

Next consider (3) C_{pat}. Recall from Figure 4 that our pattern extension uses gSpan's depth-first pattern-growth scheme with rightmost path extension on DFS code tree. Also recall from Figure 6 on Page 6, step ④, that if an active pattern with m edges in $\mathcal{L}_{active}$ is evaluated to be frequent, it will be extended to patterns of size $(m + 1)$ which are pushed to the stack C_{pat}. Finally, as Section 3.1 has indicated, C_{pat} is organized as a stack so that only the leftmost available pattern in the pattern-growth tree will be fetched from C_{pat} at a time by a comper in step ⑥ for evaluation.

Figure 21 illustrates a snapshot of the pattern-growth tree along with the content of $\mathcal{L}_{active}$ and C_{pat}. Here, we assume $n^{max}_{active} = 3$ and we have 3 compers processing subgraph patterns 4, 5 and 6 in $\mathcal{L}_{active}$, respectively, while patterns 1, 2 and 3 have been evaluated, extended and deleted from $\mathcal{L}_{active}$. We also show the content of stack C_{pat}, where, for example, the extension of pattern 1 pushes d to C_{pat}, the extension of pattern 2 pushes c to C_{pat}, and the extension of pattern 3 pushes a and b to C_{pat}.

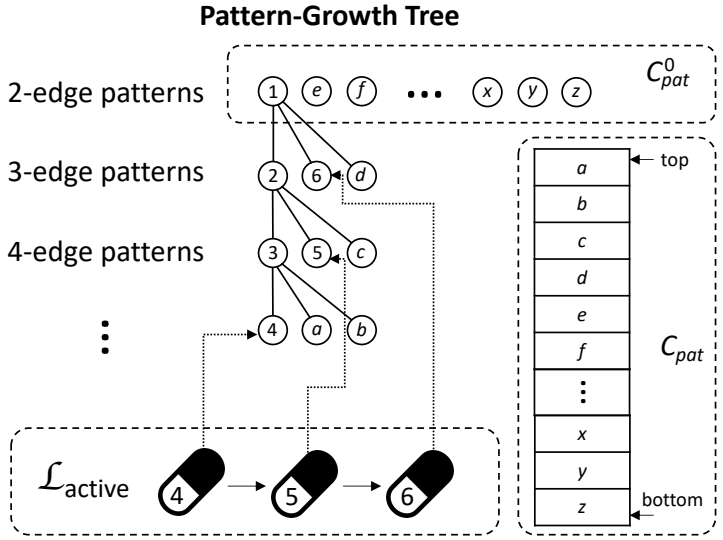

Fig. 21. A Snapshot of Pattern-Growth Tree

Let us denote the average fanout of a pattern extension by c^{avg}_{fanout}, which tends to be small since gSpan only extends a pattern along the rightmost path of its DFS code tree. Then for an active pattern S in $\mathcal{L}_{active}$ with m edges, each ancestor of S in the pattern-growth tree has been extended and added around c^{avg}_{fanout} child-patterns into C_{pat}. Since there are $(m-2)$ ancestors, it creates at most $O\left((m-2)\cdot c^{avg}_{fanout}\right) = O\left(m\cdot c^{avg}_{fanout}\right)$ patterns.

Let us denote the maximum number of edges in a pattern be m_{max}. Since there are at most n^{max}_{active} active patterns, the number of extended patterns in C_{pat} is bounded by $O\left(n^{max}_{active}\cdot m_{max}\cdot c^{avg}_{fanout}\right)$, and this is actually a loose upper bound since some ancestor paths share common ancestors. For example, patterns 4 and 5 in Figure 21 share the same ancestors 1 and 2 so their extension is double counted. Counting the initial patterns C^0_{pat}, the total pattern number of C_{pat} is bounded by $O\left(n^{max}_{active}\cdot m_{max}\cdot c^{avg}_{fanout} + \left|C^0_{pat}\right|\right)$, and since each pattern has at most m_{max} edges, the total memory cost of C_{pat} is bounded by $O\left(m_{max}\left(n^{max}_{active}\cdot m_{max}\cdot c^{avg}_{fanout} + \left|C^0_{pat}\right|\right)\right) = O\left(n^{max}_{active}\cdot c^{avg}_{fanout}\cdot m^2_{max} + \left|C^0_{pat}\right|\cdot m_{max}\right)$.

Finally, consider (4) memory space required by compers for task evaluations. A comper simply fetches subgraph-matching tasks one after another, where each subgraph-matching task for a pattern S runs an Ullmann-style recursive algorithm on $\mathcal{A}_S$ (recall Algorithm 1 on Page 8), which consumes $O(k)$ space ($k = |V^S|$) to maintain the data structures for backtracking (since only one DFS path is active at a time in the recursion tree). As a result, the memory consumption by each comper is $O(k_{max})$. Let the number of compers be n_{comper}, then the total memory consumption by all compers is $O(n_{comper}\cdot k_{max})$.

The total memory cost by (1)–(4) is obtained by adding up the above four bounds. In practice, since n^{max}_{active}, k_{max}, m_{max}, D_{avg}, $n^{max}_{timeout}$, $\mathcal{A}_{avg}$, n^{min}_{reg}, n_{batch}, c^{avg}_{fanout} and n_{comper} are all small values, the memory cost by tasks and patterns is actually comparable to $O(|G|)$, and is relatively stable (and more related to k_{max} than $|G|$). This has been verified by our empirical experiments. For example,

in Figure 15 on Page 12, the memory space consumed by tasks is only around 3 GB even though the graph *Patent* is three orders of magnitude larger than *Yeast*, and the memory to hold *Patent* alone is around 5GB. We have tested on all our graphs and this observation holds on all of them.

Received July 2022; revised October 2022; accepted November 2022