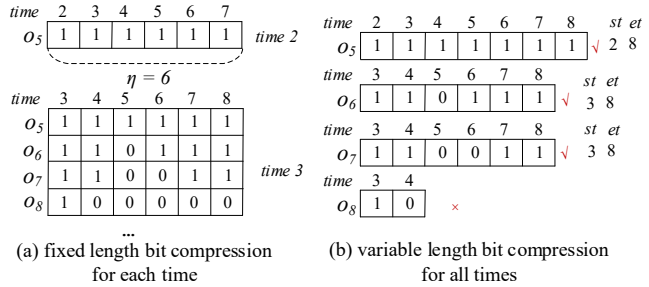

(a) fixed length bit compression for each time

(b) variable length bit compression for all times

Figure 9: Bit Compression for Subtask of o_4 in Fig. 2

2–6), and it inserts the valid o_i whose $B[o_i]$ satisfies the (K, L, G) constraints into C (lines 7–8). Next, the algorithm enumerates all subsets of C in S with the subset's cardinality (i.e., S.level) equaling $M - 2$ (line 9). In the sequel, a while loop is used to incrementally increase the level of subsets, and output the valid patterns (lines 10–17). If S is empty or S.level $> |C|$, the while loop stops.

Time Complexity and Storage Cost. Given a partition $P_t(o)$, the bit compression reduces the storage cost from $O(2^{|P_t(o)|})$ to $O(\eta \times |P_t(o)|)$. Let R be the final pattern result set in $P_t(o)$. Using the candidate set C, the time complexity of pattern enumeration is reduced from $O(2^{|P_t(o)|})$ to $O(|R| \times |C| + C_{|C|}^{M-1})$. Here, $O(C_{|C|}^{M-1})$ is the cost of enumerating all patterns O from C with $|O| = M - 1$ in the first step, and $O(|R| \times |C|)$ is the total enumeration cost of incrementally increasing the pattern cardinality.

6.3 Variable Length Bit Compression Method

Both Baseline and FBA find valid patterns every η snapshots, resulting in the same snapshot being involved in multiple verifications. Consider the example in Fig. 2, where $[S_1, S_6]$ is verified for snapshot S_1, $[S_2, S_7]$ is verified for snapshot S_2, and so on. In this example, S_2 is verified twice. To address this, we develop a variable length bit compression method that verifies each snapshot only once.

Variable Length Bit Compression. The partitioning method is the same as that in the Baseline and Bit Compression based Algorithm, i.e., a subtask is created for each trajectory. However, instead of using a fixed length bit string for every trajectory in $P_t(o)$ of a specific time t, we use a variable length bit string to represent each trajectory assigned to the subtask of o over all times.

DEFINITION 14. **(Variable Length Bit String)** *Given a trajectory o_i assigned to the subtask of trajectory o, o_i can be represented as a variable length bit string $\langle st_i, et_i, B[o_i]\rangle$, where st_i is the start time, et_i is the end time, and $B[o_i][t - st_i] = 1$ means that o and o_i belong to the same cluster at times $t \in [st_i, et_i]$, while $B[o_i][t - st_i] = 0$ indicates that o and o_i belong to different clusters at times $t \in [st_i, et_i]$.*

Fig. 9 shows the two different bit compression methods, where Fig. 9(a) uses fixed length bit strings for each time, while Fig. 9(b) uses variable length bit strings over all times. More specifically, for the subtask of o_4, o_5 is represented as the variable length bit string $\langle 2, 8, 1111111\rangle$, while o_5 is represented as two fixed length bit strings (i.e., '111111') at both times 2 and 3. Hence, the storage cost is further reduced.

THEOREM 1. *Given a subtask of trajectory o, three (K, L, G) constraints, and the total number n of occurrences for trajectories at any time $t \geq 1$ of this subtask, the total storage cost for the subtask is $O(n\frac{G+L}{L})$ when using variable length bit strings, and the total storage cost of the subtask is $O(n \times (\lceil \frac{K}{L} \rceil - 1) \times (G - 1) + K + L - 1))$ when using fixed length bit strings.*

PROOF. The fixed length bit compression method needs $O(\eta)$ space to store the fixed length bit string for every occurrence of a trajectory at any time $t \geq 1$, where $\eta = (\lceil \frac{K}{L} \rceil - 1) \times (G - 1) + K + L - 1$. Hence, the storage cost is $O(n \times (\lceil \frac{K}{L} \rceil - 1) \times (G - 1) + K + L - 1))$.

With the variable length bit compression method, every occurrence of a trajectory at one particular time $t \geq 1$ is represented by a bit '1' in an variable length bit string. For a variable length bit string B that satisfies the (K, L, G) constraints, we can get that $n_o < n_1 \frac{G}{L}$, where n_0 is the number of '0's in B and n_1 is the number of '1's in B. Thus, given n '1's in all the variable length bit strings, the total storage cost is $O(n\frac{G+L}{L})$. □

Pattern Enumeration. To further reduce the cost of enumeration, we use a candidate list C to only store the bit strings $\langle st_i, et_i, B[o_i]\rangle$ with maximal pattern time sequences.

DEFINITION 15. **(Maximal Pattern Time Sequence)** *T is a maximal pattern time sequence for a pattern O, iff (i) T satisfies the (K, L, G) constraints, and (ii) no T' exists such that $T \cup T'$ also satisfies the (L, G, K) constraints.*

Next, we develop a lemma to help obtain bit strings with maximal pattern time sequences.

LEMMA 7. *Given a variable length bit string $\langle st_i, et_i, B[o_i]\rangle$ in the subtask of o that satisfies the (K, L, G) constraints, if $B[o_i][et_i + j] = 0$ $(1 \leq j \leq (G + 1))$, then $T = \{t | t \in [st_i, et_i] \wedge B[t - st_i] = 1\})$ is a maximum pattern time sequence.*

PROOF. The proof is simple due to Definition 15 and the G constraint, and thus, it is omitted. □

For example, in Fig. 9, assume that objects o_5, o_6, and o_7 do not belong to the same cluster as o_4 at future times 9, 10, and 11. Given $L = G = 2$ and $K = 4$, then $\langle 2, 8, B[o_5] = 1111111\rangle$, $\langle 3, 8, B[o_6] = 110111\rangle$, and $\langle 3, 8, B[o_7] = 110011\rangle$ are three maximal pattern time sequences.

After obtaining a new candidate bit string s that has a maximal pattern time sequence, we first enumerate all possible patterns in $s \cup C$ (C is the global candidate set), and then, we insert s into C. The enumeration method is similar to that discussed in Section 6.2. However, since the bit strings have variable lengths, a new lemma is developed for enabling the punning of unqualified combinations.

LEMMA 8. *Given m variable length bit strings $\langle st_i, et_i, B[o_i]\rangle$ $(1 \leq i \leq m)$ and a K constraint, if $min_{i=1}^{m}\{et_i\} - max_{i=1}^{m}\{st_i\} < K$, we can prune the combination $\{o_i | 1 \leq i \leq m\}$.*

PROOF. The proof is straightforward due to the K constraint, and hence, it is omitted. □

Based on Lemmas 7 and 8, we propose Variable Length Bit Compression based Algorithm (VBA). The pseudo-code is shown in Algorithm 5. VBA takes as inputs a partition $P_t(o) = \{o_i | 1 \leq i \leq |P_t(o)|\}$, a global hashmap H, a global candidate list C, and (M, K, L, G) constraints. Initially, it initializes an empty local candidate list C_l (line 1). Then, it updates (lines 2–12) or creates new variable length bit strings (lines 13–14) for trajectories in $P_t(o)$. It first updates the bit strings already in H. More specifically, for each object o_i in the global H, if $o_i \in P_t(o)$, VBA appends a bit '1' to the bit string $H[o_i].B$, and removes o_i from $P_t(o)$ (lines 3–5). Otherwise, it appends a bit '0' to the bit string $H[o_i].B$ (line 7). An isVaild function is called to determine whether the bit string is a maximal pattern time sequence according to Lemma 7