Mining Quasi-Periodic Communities in Temporal Network (Missing Proves)

Proof of Theorem 2. For each iteration in line 2 of Algorithm 1, the size of $candSet$ is smaller than the number of QPTs of length from 2 to $\sigma-1$ in sub-sequence $(T[0], T[1], ..., T[i-1])$. Based on Theorem 1, $|candSet|$ for any i is smaller than $\sum_{j=2}^{\sigma-1} T[i-1]^2(T[i-1]\epsilon+1)^{j-1} \leq T_{max}^2 \sum_{j=2}^{\sigma-1}(T_{max}\epsilon+1)^{j-1} < T_{max}^2 \frac{(T_{max}\epsilon+1)^{\sigma-1}}{T_{max}\epsilon}$. So the time complexity of Algorithm 1 is $O(T_{max}^3(T_{max}\epsilon+1)^{\sigma-2})$.

In Algorithm 1, QPT and $candSet$ dominate the overall space complexity. QPT is the set of all QPTs of length σ in T, so $|QPT|$ is not greater than $T_{max}^2(T_{max}\epsilon+1)^{\sigma-1}$. Recall that $|candSet| < T_{max}^2 \frac{(T_{max}\epsilon+1)^{\sigma-1}}{T_{max}\epsilon}$, the space complexity of Algorithm 1 is $O(\sigma(|QPT|+|candSet|)) = O(T_{max}^2(T_{max}\epsilon+1)^{\sigma-1})$.

Proof of Theorem 3. For (1), since $D_{min}^{T'}$ is the smallest value in $D^{T'}$ and $T' \subseteq T$, there must be a DAG $DAG_{D_{min}^{T'}} \in \mathcal{DAG}$. Since T' is an (ϵ,σ)-QPT, we have $D_{min}^{T'} \leq T'[i] - T'[i-1] \leq D_{min}^{T'}(\epsilon+1)$ and $(T'[i-1], T'[i]) \in DAG_{D_{min}^{T'}}$ for $i = 1, ..., \sigma-1$. So T' is a path of length σ in $DAG_{D_{min}^{T'}}$.

For (2), it is clear that $u_i \in T, i = 1, 2, ..., \sigma$ and $d \leq u_{i+1} - u_i \leq d(1+\epsilon), i = 1, 2, ..., \sigma-1$, so p is an (ϵ,σ)-QPT in T.

Proof of Theorem 4. In Algorithm 2, there are at most $|T|^2$ iterations in line 2-3. In line 5, there are at most $\frac{T_{max}\epsilon}{\epsilon+1}+1$ iterations. As a result, the time complexity of Algorithm 2 is $O(|T|^2(\frac{T_{max}\epsilon}{\epsilon+1}+1))$.

Since there are at most $|T|$ items in each DAG_d and $|D^T| < T_{max}$, so the space complexity of Algorithm 2 is $O(|T|T_{max})$.

Proof of Theorem 5. In line 2 of Algorithm 3, there are at most T_{max} different DAG_d. In line 3, to reverse a DAG the algorithm needs to traverse the DAG, which needs at most $|T|(d\epsilon+1)$ operations. The algorithm also needs to traverse DAG and $\overline{DAG}$ in line 5-8. In line 9, there are at most $|T|$ key edges. In line 12, the algorithm needs to enumerate pre for at most $\sigma-1$ times. In line 13-18, there are three tasks: conducting hDFS on DAG and $\overline{DAG}$ (line 17, line 16), splicing paths of two directions (line 18). hDFS searches for all paths of length $leftLen$ starting from $start$ and the number of such paths is at most $(d\epsilon+1)^{leftLen-1}$. With the help of $maxLen$ and $\overline{maxLen}$, hDFS never visits vertices where no path of required length starts. For each single path of required length and for each vertex u on the path, hDFS traverses all $(d\epsilon+1)$ neighbors of u in the worst case to find the next vertex of the path, so the time complexity of finding a single path is $(d\epsilon+1)(leftLen-1)$. In line 18, the time complexity of splicing paths of two directions is $\sigma|preAns||postAns| = \sigma(d\epsilon+1)^{\sigma-2}$. Overall, since $leftLen \leq \sigma-1$, the time complexity of line 13-18 is $O(2(\sigma-2)(d\epsilon+1)^{\sigma-1}+\sigma(d\epsilon+1)^{\sigma-2})$. In DeleteKeyEdge, if $\overline{maxLen}[curNode]$ is given $longest+1$ and $longest+1 \geq \sigma-1$ (line 40), then all successors of $curNode$ in DAG do not need to be updated, so in the worst case, there are at most $\sum_{i=1}^{\sigma-1}(d\epsilon+1)^{i-1} = \frac{(d\epsilon+1)^{\sigma-1}-1}{d\epsilon}$ nodes being visited in each call to DeleteKeyEdge, and all their direct successors in $\overline{DAG}$ should also be visited. As a result, the time complexity of DeleteKeyEdge is $O((d\epsilon+1)\frac{(d\epsilon+1)^{\sigma-1}-1}{d\epsilon})$. Overall, since $d < T_{max}$, the time complexity of Algorithm 3 is

$$O\left(T_{max}\left(4|T|(T_{max}\epsilon+1)+|T|\left\{(\sigma-1)\left[2(\sigma-2)(T_{max}\epsilon+1)^{\sigma-1}+\sigma(T_{max}\epsilon+1)^{\sigma-2}\right]+(T_{max}\epsilon+1)\frac{(T_{max}\epsilon+1)^{\sigma-1}-1}{T_{max}\epsilon}\right\}\right)\right),$$

which can be abbreviated as $O(T_{max}^2(T_{max}\epsilon+1)^{\sigma-1})$.

Clearly, the main space consumption of Algorithm 3 is caused by $\mathcal{DAG}$ and the set of QPTs QPT. As a result, the space complexity of Algorithm 3 is $O(T_{max}^2(T_{max}\epsilon+1)^{\sigma-1})$.

Proof of Theorem 6. For any MQPCore $\mathcal{C} = (C, T)$ and $C = (V_C, E_C)$, suppose $u \in V_C$, it is clear that for $t \in T$, $deg_{SN_t}(u) \geq k$, so $T \in QPTSET_u$. Let (G_S, T) be a $QPCS_u$. If $C \nsubseteq G_S$, then it is easy to construct a larger QPCS (G'_S, T) that $G_S \subset G'_S$, which breaks the precondition. So $C \subseteq G_S$.

Proof of Theorem 7. We first prove that any MQPCore can be enumerated by Algorithm 4. For any MQPCore $\mathcal{C} = (C, T)$, suppose $C = (V_C, E_C)$, it is clear that $C \subseteq G_c$ (G_c is defined in line 2 of Algorithm 4). Before the vertex in V_C that will be traversed first in line 5 of Algorithm 4 is traversed, no vertex in V_C will be deleted, i.e., be added into X in line 14 and line 22 (since $\forall v \in V_C, deg_C(v) \geq k$). For the first vertex u in V_C being traversed in line 5, T must be a sub-sequence of $t_k(u)$, because u is in C and $\mathcal{C}$ is a MQPCore on time sequence T. As a result, $T \in QPTSET_u$. In line 9 of Algorithm 4, $(T, u) \in L$ indicates that u has been visited as part of a $QPCS_{u'}$ $(G_{u'}, T)$ (u' is a vertex which is traversed before u). Clearly, $C \subseteq G_{u'}$ because $G_{u'}$ is a maximal connected subgraph according to Definition 9. As a result, $\mathcal{C} = (C, T)$ can be generated in the second stage in traversing u'. If $(T, u) \notin L$ in line 9, then (G_u, T) (G_u is computed in line 10) is the $QPCS_u$. So $\mathcal{C} = (C, T)$ can be generated in the second stage in traversing u.

Next we prove that any tuple (C, T) generated by Algorithm 4 is a MQPCore in $\mathcal{G}$. Let (C, T) be one of the connected k-cores in line 12 of Algorithm 4, and suppose (C, T) is not a MQPCore in $\mathcal{G}$, there must be a MQPCore in $\mathcal{G}$, (C', T), where $C \subset C'$. Recall the previous proof, let the first vertex in C' being traversed in line 5 of Algorithm 4 is u', if $(T, u') \in L$ in line 9, then some QPCS containing C' has being generated before, and for each vertex v in C', (T, v) has been recorded in L (line 11) and will be skipped later. So we can not obtain (C, T) but only (C', T), which contradicts the conditions before. It is the same that only (C', T) will be generated if $(T, u') \notin L$.

Proof of Theorem 8. In line 2, the time complexity of computing G_c is $O(|V| + |E|)$, and $|V_c| \leq |V|, |E_c| \leq |E|$. In line 5, the cost for sorting is $O(|V| log(|V|))$. In line 7, $|t_k(u)| \leq |T| \leq T_{max}$, and ComputeQPTSET is implemented using QPT+, so the total time complexity of ComputeQPTSET is $O(|V| T_{max}^2 (T_{max}\epsilon + 1)^{\sigma-1})$. According to Theorem 1, the size of $QPTSET_u$ (line 7) is smaller than $T_{max}^2 (T_{max}\epsilon + 1)^{\sigma-1}$. For each qpt in line 8 and in the worst case, the total size of G_u is $|V| + |E|$. In computing G_u, the time complexity of checking whether $qpt \subseteq ST_{\mathcal{G}}((u', v'))$ in line 10 is $O(|qpt|) = O(\sigma)$. Combining line 11-13 with the previous lines, the overall time complexity of line 8-13 is $O(T_{max}^2 (T_{max}\epsilon + 1)^{\sigma-1} (|V| + |E|))$. The total cost in line 14-22 is at most $O(|V| + |E|)$. As a result, the complexity of Algorithm 4 is $O(T_{max}^2 (T_{max}\epsilon + 1)^{\sigma-1} (|V| + |E|))$.

In Algorithm 4, since $|QPTSET_u| \leq T_{max}^2 (T_{max}\epsilon + 1)^{\sigma-1}$, L stores at most $T_{max}^2 (T_{max}\epsilon + 1)^{\sigma-1} |V|$ tuples, so the space complexity of L is $O(T_{max}^2 (T_{max}\epsilon + 1)^{\sigma-1} |V|)$. In the worst case, the total space consumed by all (C', qpt) in line 13 for the same qpt is $|V| + |E|$, so the space complexity of $\mathcal{M}$ is $O(T_{max}^2 (T_{max}\epsilon + 1)^{\sigma-1} (|V| + |E|))$. Together with space consumed by ComputeQPTSET and X, Q, the space complexity of Algorithm 4 is $O(T_{max}^2 (T_{max}\epsilon + 1)^{\sigma-1} (|V| + |E|))$.

Proof of Theorem 10. We first prove that any MQPClique can be enumerated by Algorithm 5. For any MQPClique $\mathcal{C} = (C, T)$ and $C = (V_C, E_C)$, it is clear that $C \subseteq G_c$. As in the proof of Theorem 7, suppose the first vertex in V_C being visited in line 5 is u, no vertex in V_C will be deleted before u is visited. When u is visited, $T \in (\epsilon, \sigma, k-1)$-$QPTSET_u$ since $\mathcal{C}$ is a MQPClique on time sequence T, so T will be traversed in line 8. In line 15, P represents the set of remaining vertices in $QPNS_u$ excluding u on the pruned graph (pruning rule 1), and it serves as the candidate set for the BKPivot procedure. As a result, $\mathcal{C}$ can be enumerated by BKPivot.

Then we prove that any tuple $\mathcal{C} = (C, T) \in \mathcal{M}$ (originally referred to as (R, qpt) in line 28) generated by Algorithm 5 is a MQPClique in $\mathcal{G}$. Suppose $\mathcal{C}$ is obtained in traversing u, it is clear that $\mathcal{C}$ is a quasi-periodic clique, because $\mathcal{C}$ is enumerated in $QPNS_u$. In line 28, $X = \emptyset$ indicates that $\mathcal{C}$ is also a maximal quasi-periodic clique, i.e., a MQPClique.

Proof of Theorem 11. In Algorithm 5, in the worst case we need to compute $QPNS_u$ for each vertex u and each $qpt \in QPTSET_u$. The total time complexity of the above process is $O(|V| T_{max}^2 (T_{max}\epsilon + 1)^{\sigma-1} (|V| + |E|))$. The time complexity of enumerating maximal cliques in a static graph with vertex set V is $O(3^{|V|/3})$ [8]. So the total time complexity of BKPivot procedure is $O(|V| T_{max}^2 (T_{max}\epsilon + 1)^{\sigma-1} 3^{|V|/3})$. Considering ComputeQPTSET in line 7 and the sorting in line 5, the total time complexity of Algorithm 5 is $O(T_{max}^2 (T_{max}\epsilon + 1)^{\sigma-1} |V| 3^{|V|/3})$.