## A SQG construction algorithm

Algorithm 1 Build Semantic Query GraphInput: Node set V, Relation Extraction model $RE()$ , Reward Function $\gamma()$ Output: The final Semantic Query Graph1: for each pair $(u, v) \in V \times V$ do2: $RE(u, v)$ 3: end for4: Initialize priority queue H5: $SQG s_0=\{V, E=\emptyset\}$ 6: H.add $(s_0, \gamma(s_0))$ 7: while H is not empty do8: $s, r = H.pop()$ 9: if isValidSQG(s) then10: return s11: end if12: for operate node $u \in S.V$ do14: if checkConstraint $(op, u)$ then15: $s' = TS(s, op, u)$ 16: if s' is a new state then17: H.add $(s', \gamma(s'))$ 18: end if19: end if20: end for21: end for22: end while	
$RE()$ , Reward Function $\gamma()$ <b>Output:</b> The final Semantic Query Graph1: for each pair $(u, v) \in V \times V$ do2: $RE(u, v)$ 3: end for4: Initialize priority queue H5: $SQG s_0 = \{V, E = \emptyset\}$ 6: H.add $(s_0, \gamma(s_0))$ 7: while H is not empty do8: $s, r = H.pop()$ 9: if isValidSQG $(s)$ then10: return s11: end if12: for operation $op$ do13: for operate node $u \in S.V$ do14: if checkConstraint $(op, u)$ then15: $s' = TS(s, op, u)$ 16: if s' is a new state then17: H.add $(s', \gamma(s'))$ 18: end if19: end if20: end for21: end for	Algorithm 1 Build Semantic Query Graph
Output: The final Semantic Query Graph 1: for each pair $(u, v) \in V \times V$ do 2: $RE(u, v)$ 3: end for 4: Initialize priority queue $H$ 5: $SQG s_0 = \{V, E = \emptyset\}$ 6: $H.add(s_0, \gamma(s_0))$ 7: while $H$ is not empty do 8: $s, r = H.pop()$ 9: if isValidSQG( $s$ ) then 10: return $s$ 11: end if 12: for operation $op$ do 13: for operate node $u \in S.V$ do 14: if checkConstraint( $op, u$ ) then 15: $s' = TS(s, op, u)$ 16: if $s'$ is a new state then 17: $H.add(s', \gamma(s'))$ 18: end if 19: end if 20: end for 21: end for	<b>Input</b> : Node set V, Relation Extraction model
1: for each pair $(u, v) \in V \times V$ do 2: $RE(u, v)$ 3: end for 4: Initialize priority queue $H$ 5: SQG $s_0 = \{V, E = \emptyset\}$ 6: $H.add(s_0, \gamma(s_0))$ 7: while $H$ is not empty do 8: $s, r = H.pop()$ 9: if isValidSQG( $s$ ) then 10: return $s$ 11: end if 12: for operation $op$ do 13: for operate node $u \in S.V$ do 14: if checkConstraint( $op, u$ ) then 15: $s' = TS(s, op, u)$ 16: if $s'$ is a new state then 17: $H.add(s', \gamma(s'))$ 18: end if 19: end if 20: end for 21: end for	$RE()$ , Reward Function $\gamma()$
2: $RE(u, v)$ 3: end for 4: Initialize priority queue $H$ 5: $SQG s_0 = \{V, E = \emptyset\}$ 6: $H.add(s_0, \gamma(s_0))$ 7: while $H$ is not empty do 8: $s, r = H.pop()$ 9: if isValidSQG( $s$ ) then 10: return $s$ 11: end if 12: for operation $op$ do 13: for operate node $u \in S.V$ do 14: if checkConstraint( $op, u$ ) then 15: $s' = TS(s, op, u)$ 16: if $s'$ is a new state then 17: $H.add(s', \gamma(s'))$ 18: end if 19: end if 20: end for 21: end for	Output: The final Semantic Query Graph
3: end for 4: Initialize priority queue $H$ 5: SQG $s_0 = \{V, E = \emptyset\}$ 6: $H.add(s_0, \gamma(s_0))$ 7: while $H$ is not empty do 8: $s, r = H.pop()$ 9: if isValidSQG( $s$ ) then 10: return $s$ 11: end if 12: for operation $op$ do 13: for operate node $u \in S.V$ do 14: if checkConstraint( $op, u$ ) then 15: $s' = TS(s, op, u)$ 16: if $s'$ is a new state then 17: $H.add(s', \gamma(s'))$ 18: end if 19: end if 20: end for 21: end for	1: for each pair $(u, v) \in V \times V$ do
4: Initialize priority queue $H$ 5: SQG $s_0 = \{V, E = \emptyset\}$ 6: $H.add(s_0, \gamma(s_0))$ 7: while $H$ is not empty do 8: $s, r = H.pop()$ 9: if isValidSQG( $s$ ) then 10: return $s$ 11: end if 12: for operation $op$ do 13: for operate node $u \in S.V$ do 14: if checkConstraint( $op,u$ ) then 15: $s' = TS(s,op,u)$ 16: if $s'$ is a new state then 17: $H.add(s', \gamma(s'))$ 18: end if 19: end if 20: end for 21: end for	2: $RE(u, v)$
5: SQG $s_0 = \{V, E = \emptyset\}$ 6: <i>H</i> .add $(s_0, \gamma(s_0))$ 7: while <i>H</i> is not empty do 8: $s, r = H.pop()$ 9: if isValidSQG $(s)$ then 10: return <i>s</i> 11: end if 12: for operation <i>op</i> do 13: for operate node $u \in S.V$ do 14: if checkConstraint $(op, u)$ then 15: $s' = TS(s, op, u)$ 16: if <i>s'</i> is a new state then 17: <i>H</i> .add $(s', \gamma(s'))$ 18: end if 19: end if 20: end for 21: end for	3: end for
6: $H.add(s_0, \gamma(s_0))$ 7: while $H$ is not empty do 8: $s, r = H.pop()$ 9: if isValidSQG( $s$ ) then 10: return $s$ 11: end if 12: for operation $op$ do 13: for operate node $u \in S.V$ do 14: if checkConstraint( $op, u$ ) then 15: $s' = TS(s, op, u)$ 16: if $s'$ is a new state then 17: $H.add(s', \gamma(s'))$ 18: end if 19: end if 20: end for 21: end for	4: Initialize priority queue H
7: while H is not empty do8: $s, r = H.pop()$ 9: if isValidSQG(s) then10: return s11: end if12: for operation $op$ do13: for operate node $u \in S.V$ do14: if checkConstraint( $op,u$ ) then15: $s' = TS(s,op,u)$ 16: if s' is a new state then17: H.add(s', $\gamma(s')$ )18: end if19: end if20: end for21: end for	5: SQG $s_0 = \{V, E = \emptyset\}$
8: $s, r = H.pop()$ 9: <b>if</b> isValidSQG( $s$ ) <b>then</b> 10: <b>return</b> $s$ 11: <b>end if</b> 12: <b>for</b> operation $op$ <b>do</b> 13: <b>for</b> operate node $u \in S.V$ <b>do</b> 14: <b>if</b> checkConstraint( $op,u$ ) <b>then</b> 15: $s' = TS(s,op,u)$ 16: <b>if</b> $s'$ is a new state <b>then</b> 17: $H.add(s', \gamma(s'))$ 18: <b>end if</b> 19: <b>end if</b> 20: <b>end for</b> 21: <b>end for</b>	6: $H.add(s_0, \gamma(s_0))$
9:if isValidSQG(s) then10:return s11:end if12:for operation op do13:for operate node $u \in S.V$ do14:if checkConstraint(op,u) then15: $s' = TS(s,op,u)$ 16:if s' is a new state then17: $H.add(s', \gamma(s'))$ 18:end if19:end if20:end for21:end for	7: while <i>H</i> is not empty do
10:return $s$ 11:end if12:for operation $op$ do13:for operate node $u \in S.V$ do14:if checkConstraint( $op,u$ ) then15: $s' = TS(s,op,u)$ 16:if $s'$ is a new state then17: $H.add(s', \gamma(s'))$ 18:end if19:end if20:end for21:end for	8: $s, r = H.pop()$
11:end if12:for operation $op$ do13:for operate node $u \in S.V$ do14:if checkConstraint( $op,u$ ) then15: $s' = TS(s,op,u)$ 16:if $s'$ is a new state then17: $H.add(s', \gamma(s'))$ 18:end if19:end if20:end for21:end for	9: <b>if</b> isValidSQG( $s$ ) <b>then</b>
12:for operation $op$ do13:for operate node $u \in S.V$ do14:if checkConstraint( $op, u$ ) then15: $s' = TS(s, op, u)$ 16:if $s'$ is a new state then17: $H.add(s', \gamma(s'))$ 18:end if19:end if20:end for21:end for	10: return s
13:for operate node $u \in S.V$ do14:if checkConstraint( $op,u$ ) then15: $s' = TS(s,op,u)$ 16:if $s'$ is a new state then17: $H.add(s', \gamma(s'))$ 18:end if19:end if20:end for21:end for	11: end if
14:if checkConstraint( $op, u$ ) then15: $s' = TS(s, op, u)$ 16:if $s'$ is a new state then17: $H.add(s', \gamma(s'))$ 18:end if19:end if20:end for21:end for	12: <b>for</b> operation <i>op</i> <b>do</b>
15: $s' = TS(s, op, u)$ 16:if s' is a new state then17: $H.add(s', \gamma(s'))$ 18:end if19:end if20:end for21:end for	13: <b>for</b> operate node $u \in S.V$ <b>do</b>
16:if $s'$ is a new state then17: $H.add(s', \gamma(s'))$ 18:end if19:end if20:end for21:end for	14: <b>if</b> checkConstraint( <i>op</i> , <i>u</i> ) <b>then</b>
17: $H.add(s', \gamma(s'))$ 18: end if   19: end if   20: end for   21: end for	15: $s' = TS(s,op,u)$
18: end if   19: end if   20: end for   21: end for	16: <b>if</b> $s'$ is a new state <b>then</b>
19:   end if     20:   end for     21:   end for	17: $H.add(s', \gamma(s'))$
20:   end for     21:   end for	18: <b>end if</b>
21: end for	19: <b>end if</b>
	20: <b>end for</b>
22: end while	21: end for
	22: end while

Algorithm 1 shows the pseudo code of the SQG construction procedure. As shown in Line 1-3, we first extract relations between each pair of nodes by the relation extraction model. Each potential relation has a confidence probability which can be used in the reward function  $\gamma()$ . The initial state  $s_0$  is a semantic query graph contains all isolated nodes with no edges. We put  $s_0$  and its score  $\gamma(s_0)$  to the priority queue H (Line 4-6). During the search procedure, in each epoch we get the current best state and check whether it is a valid SQG. A valid SQG should be a connected graph with at least two nodes. It should has matches in the knowledge graph and has no subsequent SQGs with higher scores. The first valid SQG is considered as the final semantic query graph (Line 7-11). Line 12-21 are the enumeration of state transition. Specifically, for each operation op we enumerate each possible operate node u. The function check-Constraint check whether op and u satisfy the corresponding condition. Although this is a greedy search algorithm, the final SQG we generated is usually the best one, especially when the small node set.