Chapter 3

Algorithm SB: Selecting Backbones

3.1 Determining Edges and Weights among Query Trees

As mentioned in Section 2, given a set of query trees, we should first derive the graph G=(V, E). Intuitively, the set of vertices in the graph is referred to the set of query trees. Then, we should determine edges and the corresponding weights among these query trees. For example, given a set of query trees in Figure 3.1, the corresponding graph is shown in Figure 3.2, where each vertex denotes as a query tree and each edge represents the partial result sharing between query trees. When two query trees have some overlaps in their data sources, an edge will be added in the graph to represent the partial result sharing relationship. Specifically, suppose that the partial result on S_m of T_i is the same as the one on S_n of T_j. In other words, $D_i(m)$ is the same as $D_j(n)$ due to the same data sources. For query trees T_i, T_j, $w_{i,j}(m,n)$ denotes the number of messages reduced when T_j obtains the partial result of sensor S_n in T_i for sensor S_m in T_j. To formulate the value of $w_{i,j}(m,n)$, Figure 3.3 shows the number of messages involved, where the triangle under S_n is the subtree of S_n in T_j and $N_j(n)$ is the number of messages spent for $D_j(n)$ at S_n. Furthermore, since S_n should access the partial result $D_i(m)$ from S_m, an extra transmission cost is required. Explicitly, the transmission cost is estimated as the minimal hop count between S_m and S_n, denoted as $d_S(m,n)$. As shown in Figure 3.3, the

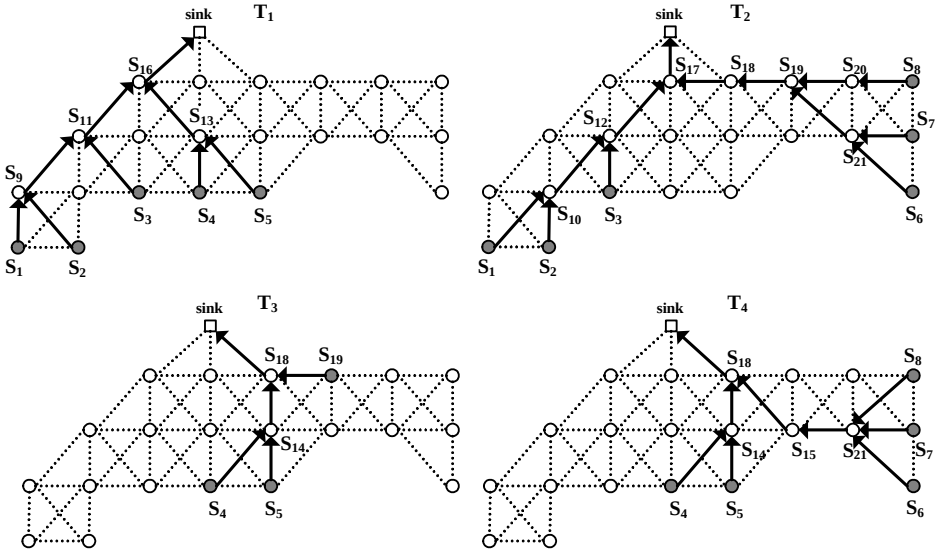


Figure 3.1: Four query trees for an illustrative example.

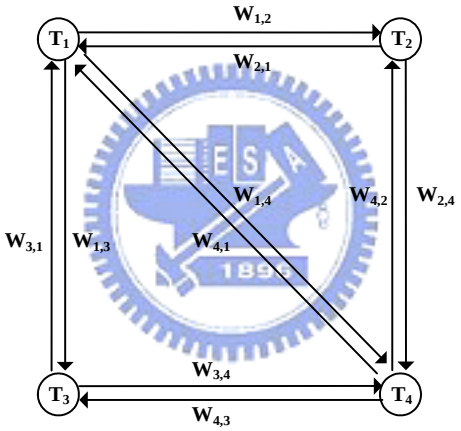


Figure 3.2: An example graph transformed from Figure 3.1.

dotted line between S_n and S_m is the communication path and thus $d_S(m,n)$ is expressed as the hop count of the communication path. Consequently, the value of $w_{i,j}(m,n)$ is formulated as $N_j(n) - d_S(m,n)$.

When two query trees have overlaps in their data sources, these two query trees are able to share partial results. Note that there are many possible sensors in query trees that could share the partial results. Consider query trees T_1 and T_2 in Figure 3.1 as an example, where both T_1 and T_2 have some overlap sensors in their data sources (i.e., S_1, S_2 and S_3). If we select T_1 as a backbone tree, there are two possible ways for T_1 to share the partial results.

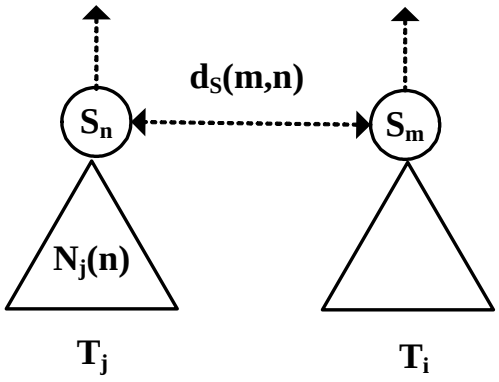


Figure 3.3: The number of messages involved for the partial result sharing between T_i and T_j.

Explicitly, as can be seen in Figure 3.1, T_1 is able to share the partial result in S_9 (respectively, S_{11}) to S_{10} (respectively, S_{12}) of T_2 due to the same data sources S_1 and S_2 (respectively, S_1, S_2 and S_3). To facilitate the presentation of all possible ways for sharing partial results from T_i and T_j, $W_{i,j}$ is used to represent the set of weights of for the edges. As mentioned above, the weight of each possible sharing scenario is in fact in the form of $w_{i,j}(S_m, S_n)$, meaning that T_i shares the partial result in S_m to sensor S_n in T_j. Therefore, given a set of query trees in Figure 3.2, we will derive the graph shown in Figure 3.1, where an auxiliary table (i.e., Table 3.1) is built up to show the detailed information of the weights.

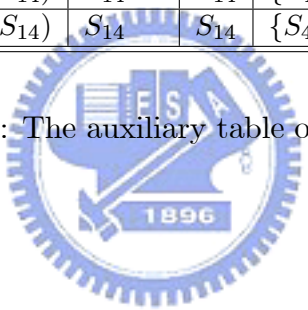
As described before, the auxiliary table is used to store all possible scenarios of partial result sharing among query trees. However, some scenarios are able to be eliminated since these scenarios will not have any impacts on the solution derived. Therefore, these unnecessary weights are removed so as to improve the performance. Consequently, in this paper, some weights should be removed in accordance with the following rule:

Rule of removing unnecessary weight values: *For the same* data source*, only the maximal weight is kept and the others are removed.*

For example, in Table 3.1, we have the same data source for $W_{2,4}$. According to the rule, we only keep the maximal weight (i.e., $w_{2,4}(S_{18}, S_{15})$). After pruning some weights, we have

	weight	source	to	data source	value
$W_{1,2}$	$w_{1,2}(S_9, S_{10})$	S_9	S_{10}	$\{S_1, S_2\}$	1
	$w_{1,2}(S_{11}, S_{12})$	S_{11}	S_{12}	$\{S_1, S_2, S_3\}$	3
$W_{2,1}$	$w_{2,1}(S_{10}, S_9)$	S_{10}	S_9	$\{S_1, S_2\}$	1
	$w_{2,1}(S_{12}, S_{11})$	S_{12}	S_{11}	$\{S_1, S_2, S_3\}$	3
$W_{1,3}$	$w_{1,3}(S_{13}, S_{14})$	S_{13}	S_{14}	$\{S_4, S_5\}$	1
$W_{3,1}$	$w_{3,1}(S_{14}, S_{13})$	S_{14}	S_{13}	$\{S_4, S_5\}$	1
$W_{1,4}$	$w_{1,4}(S_{13}, S_{14})$	S_{13}	S_{14}	$\{S_4, S_5\}$	1
$W_{4,1}$	$w_{4,1}(S_{14}, S_{13})$	S_{14}	S_{13}	$\{S_4, S_5\}$	1
$W_{2,4}$	$w_{2,4}(S_{18}, S_{15})$	S_{18}	S_{15}	$\{S_7, S_8, S_9\}$	3
	$w_{2,4}(S_{18}, S_{21})$	S_{18}	S_{21}	$\{S_7, S_8, S_9\}$	1
	$w_{2,4}(S_{19}, S_{15})$	S_{19}	S_{15}	$\{S_7, S_8, S_9\}$	3
	$w_{2,4}(S_{19}, S_{21})$	S_{19}	S_{21}	$\{S_7, S_8, S_9\}$	2
$W_{4,2}$	$w_{4,2}(S_{15}, S_{18})$	S_{15}	S_{18}	$\{S_7, S_8, S_9\}$	5
	$w_{2,4}(S_{15}, S_{19})$	S_{15}	S_{19}	$\{S_7, S_8, S_9\}$	4
	$w_{2,4}(S_{21}, S_{18})$	S_{21}	S_{18}	$\{S_7, S_8, S_9\}$	4
	$w_{2,4}(S_{21}, S_{19})$	S_{21}	S_{19}	$\{S_7, S_8, S_9\}$	4
$W_{3,4}$	$w_{3,4}(S_{14}, S_{14})$	S_{14}	S_{14}	$\{S_4, S_5\}$	2
$W_{4,3}$	$w_{4,3}(S_{14}, S_{14})$	S_{14}	S_{14}	$\{S_4, S_5\}$	2

Table 3.1: The auxiliary table of weights



	Weights	From	To	Data Source	Value
$W_{1,2}$	$w_{1,2}(S_9, S_{10})$	S_9	S_{10}	$\{S_1, S_2\}$	1
	$w_{1,2}(S_{11}, S_{12})$	S_{11}	S_{12}	$\{S_1, S_2, S_3\}$	3
$W_{2,1}$	$w_{2,1}(S_{12}, S_{11})$	S_{12}	S_{11}	$\{S_1, S_2, S_3\}$	3
$W_{1,3}$	$w_{1,3}(S_{13}, S_{14})$	S_{13}	S_{14}	$\{S_4, S_5\}$	1
$W_{3,1}$	$w_{3,1}(S_{14}, S_{13})$	S_{14}	S_{13}	$\{S_4, S_5\}$	1
$W_{1,4}$	$w_{1,4}(S_{13}, S_{14})$	S_{13}	S_{14}	$\{S_4, S_5\}$	1
$W_{4,1}$	$w_{4,1}(S_{14}, S_{13})$	S_{14}	S_{13}	$\{S_4, S_5\}$	1
$W_{2,4}$	$w_{2,4}(S_{18}, S_{15})$	S_{18}	S_{15}	$\{S_7, S_8, S_9\}$	3
$W_{4,2}$	$w_{4,2}(S_{15}, S_{18})$	S_{15}	S_{18}	$\{S_7, S_8, S_9\}$	5
$W_{3,4}$	$w_{3,4}(S_{14}, S_{14})$	S_{14}	S_{14}	$\{S_4, S_5\}$	2
$W_{4,3}$	$w_{4,3}(S_{14}, S_{14})$	S_{14}	S_{14}	$\{S_4, S_5\}$	2

Table 3.2: The auxiliary table after removing unnecessary weight values

the final auxiliary table shown in Table 3.2.

3.2 Design of Algorithm SB

With the graph and the corresponding weights devised above, we develop algorithm SB (standing for Selecting Backbones) to determine which query tree should be included in the backbone set to minimize the total number of messages involved. As mentioned before, minimizing the total number of messages spent for all queries is equal to maximizing the number of messages reduced through sharing the partial results of backbones. More specifically, the objective of algorithm SB is to maximize $\sum_{T_j \in NB} R(T_j, B)$. Consequently, algorithm SB is greedy in nature and selects the backbone with the maximal number of messages reduced for those non-backbones each time until no any message could be reduced when additional backbone is selected.

In essence, the value of $R(T_j, B)$ is related to how the sensor nodes of T_j access partial results from backbones. Since there are many possible scenarios for T_j to get partial results from backbones, we should avoid redundant message transmission when formulating the value of $R(T_j, B)$. Thus, we have the following property.

Property 1: *If non-backbone T_j gets a partial result for a node S_x on T_j, it is unnecessary to further access partial results for the ancestors or descendants of S_x on T_j.*

Figure 3.4 clearly illustrates Property 1, where nodes S_y and S_z are the child nodes of node S_x. Assume that nodes S_y and S_z access partial results from backbones. Obviously, since the partial results of S_y and S_z is used to aggregate the result on S_x, node S_x should not access the partial result from backbones to reduce the redundant message transmission. Similarly, It is also unnecessary to take partial results for the descendants of S_y or S_z. Consequently, if the reduced number of messages by taking partial result for S_n on T_j is already counted in $R(T_j, B)$, we should avoid overcounting the number of messages reduced by taking partial

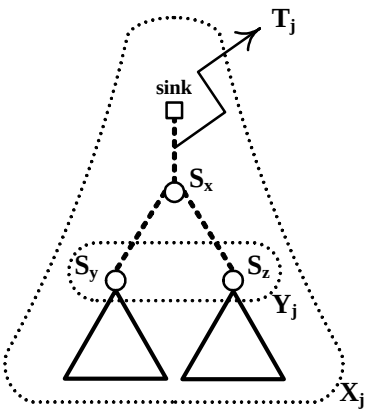


Figure 3.4: Formulation of $R(T_j, B)$

result for the ancestors or descendants of S_n.

In light of Property 1, we have develop a procedure to determine how many messages could be reduced through the partial result sharing. The algorithmic form of the proposed procedure is given below:

Procedure $R(T_j, B)$:

1. set $Y = \cup_{i \in B} \mathrm{W}_{i,j}$, to determine the union set of sensors from the auxiliary table ;

2. Generate the power set of Y, denoted as $P(Y)$, is the set of all subsets of Y;

3. $\forall\ X \in P(Y)$, if there exists any ancestor or descendant relationship in X, prune X from $P(Y)$;

4. return $\max_{\forall X \in P(Y)}(\sum_{n \in X \text{ and } i \in B} w_{i,j}(m,n))$

In the beginning, we will determine the set of Y from the auxiliary table. As described above, the auxiliary table will contain all the detailed information related to the partial result sharing. Thus, given the backbone set, we could easily decide the set of Y. In fact, Y contains all the sensors in T_j that could access the partial results from backbones. In order to enumerate all the possible scenarios, we should generate the power set of Y, denoted as $P(Y)$. As pointed out earlier, we should avoid the redundant over-estimation. Thus, for each set in $P(Y)$, we should check whether there is any ancestor and descendant relationship or not. Note that one could refer to query tree T_j to verify any ancestor and descendant relationship. As such, the set of $P(Y)$ has all the possible scenarios of partial result sharing for T_j. Consequently, the number of messages reduced for T_j is able to be the maximal value among these possible

Sets in $P(Y)$	$R(T_2, B)$
$\{S_{10}, S_{12}, S_{18}\}$	N/A
$\{S_{10}, S_{12}\}$	N/A
$\{S_{12}, S_{18}\}$	$w_{1,2}(S_{11}, S_{12}) + w_{4,2}(S_{15}, S_{18}) = 8^*$
$\{S_{10}, S_{18}\}$	$w_{1,2}(S_9, S_{10}) + w_{4,2}(S_{15}, S_{18}) = 6$
$\{S_{10}\}$	$w_{1,2}(S_9, S_{10}) = 1$
$\{S_{12}\}$	$w_{1,2}(S_{11}, S_{12}) = 3$
$\{S_{18}\}$	$w_{4,2}(S_{15}, S_{18}) = 5$

Table 3.3: The final result of $R(T_2, B)$, where $B = \{T_1, T_4\}$.

scenarios.

Consider an example in Figure 3.1, where the auxiliary table is shown in Table 3.2. Assume that the backbone set contains T_1 and T_4. We will illustrate how to derive the number of messages reduced for T_2 (i.e., $R(T_2, B)$). From Table 3.2, both $W_{1,2}$ and $W_{4,2}$ should be considered and we could verify that the set of Y should be $\{S_{10}, S_{12}, S_{18}\}$. Then, we will generate the power set of Y (i.e., P(Y)) and prune some sets in P(Y) if there exists any ancestor and descendant relationship. Note that since S_{12} is the ancestor of S_{10} in query tree T_2, both $\{S_{10}, S_{12}, S_{18}\}$ and $\{S_{10}, S_{12}\}$ will be eliminated from P(Y). For each set of P(Y), we will derive the number of reduced messages through partial result sharing of backbones. For example, for set $\{S_{12}, S_{18}\}$, we could derive that the value of $R(T_2, B)$ will be 8 (i.e., $w_{1,2}(S_{11}, S_{12}) + w_{4,2}(S_{15}, S_{18}) = 8$). Following the same operation, Table 3.3 shows the set of P(Y) and the corresponding values. Among these scenarios, $\{S_{12}, S_{18}\}$ should be selected since this scenario is able to reduce as many number of messages as possible through partial result sharing of backbones.

By exploring the formulation of $R(T_j, B)$, we could estimate how many benefits (i.e., number of messages reduced) when one query tree is selected as a backbone. In order to evaluate the benefits of selecting query tree Ti as a backbone, we have the following definition.

Definition 1: The backbone gain achieved by selecting T_i as a backbone, denoted by $\delta(T_i)$, can be formulated as $\delta(T_i) = \sum_{T_j \in (NB - T_i)} R(T_j, B \cup T_i) - \sum_{T_j \in NB} R(T_j, B)$.

In light of Definition 1, we propose a heuristic algorithm SB that iteratively select backbones according to *backbone gains* of query trees. Initially, the backbone set is empty and the non-backbone set is the set of query trees given. For each query tree in the non-backbone set, we will calculate the corresponding backbone gain. Then, the query tree with the maximal backbone gain is selected in the backbone set. Once one query tree is selected as a backbone, we should update the backbone gains for query trees in the non-backbone set. Similarly, according to the backbone gains of query trees in the non-backbone set, we will select the one with the maximal backbone gain as a backbone. Algorithm SB selects the query trees in the non-backbone set iteratively until no additional query tree is selected in the backbone set. When query trees in the non-backbone set have their corresponding backbone gains smaller than zero, no query tree will be selected in the backbone set since no more benefit will be earned. As such, a set of query trees is divided into two sets: the backbone set and the non-backbone set, which is akin to the Max-Cut problem with the objective of maximizing the cut, meaning that the number of messages reduced is maximized.

Algorithm 1 SB: Selecting Backbone

Input: A set of query trees $AT = \{T_1, ..., T_n\}$
Output: A set of backbones B

1: $NB \Leftarrow AT$
2: $B \Leftarrow \phi$
3: **while** it exists some query tree whose $BR_b > 0$ **do**
4: Select T_b as backbone whose $BR(T_b)$ is maximal among all query trees in NB
5: $B \Leftarrow B \cup T_b$
6: $NB \Leftarrow NB - T_b$
7: **end while**
8: Arrange for each non-backbones in NB to take the partial results of backbones in B

Algorithm 1 shows the algorithm SB for selecting backbones by using backbone gain. the initial setting is performed from line 1 to line 2. The selection of backbones is shown from line 3 to line 7. Specifically, if query tree T_i in the non-backbone set has $\delta(T_i)$ larger than zero, algorithm SB will iteratively select query trees as backbones until no query tree has its backbone gain larger than zero. In line 4, algorithm SB selects query tree T_i with the

Selection Iteration	Backbone and Non-backbone set	Backbone Rank			
		T_1	T_2	T_3	T_4
1	$B = \{\}, NB = \{T_1, T_2, T_3, T_4\}$	5	6	3	8
2	$B = \{T_4\}, NB = \{T_1, T_2, T_3\}$	2	-2	-1	-
3	$B = \{T_1, T_4\}, NB = \{T_2, T_3\}$	-	-7	-2	-

Table 3.4: An execution scenario of algorithm SB.

maximal backbone gain so as to guarantee that the benefit of selecting this query tree as a backbone will be most worthy. Once one query tree is selected as a backbone, we shall update backbone gains of query trees in the non-backbone set. Finally, we will have the backbone set and the non-backbone set. In light of the graph derived in Section 3.1 and the auxiliary table, query trees in the non-backbone set will adjust their tree structures to access partial results of backbones.

Consider four query trees in Figure 3.1, where the corresponding graph in Figure 3.2 with the auxiliary table in Table . 3.4 shows the execution scenario of algorithm SB. In the beginning, backbone gains of these four query trees will be calculated. For example, $\delta(T_4) = \sum_{T_j \in (NB - T_4)} R(T_j, B \cup T_4) - \sum_{T_j \in NB} R(T_j, B) = R(T_1, T_4) + R(T_2, T_4) + R(T_3, T_4) - R(T1, \phi) - R(T2, \phi) - R(T3, \phi)$ -$R(T4, \phi) = 1 + 5 + 2 - 0 - 0 - 0 = 8$. Note that the maximal value of backbone gain is marked with an '*'. It can be seen in the first iteration, query tree T_4 is selected as a backbone since query tree T_4 will bring the maximal benefit to other query trees in the non-backbone set. After selecting query tree T_4 as a backbone, we should update backbone gains of non-backbone query trees. Following the same procedure, we could obtain the up-to-dated backbone gains and then select the maximal value of backbone gain among query trees in the non-backbone set. In the second iteration, T_1 will be included in the backbone set. Similarly, all query trees in the non-backbone set should update their corresponding backbone gains. In the third iteration, since all query trees in the non-backbone set have their backbone gains smaller than zero, algorithm SB will stop selecting backbones. As a result,

we have the backbone set B={ T_1, T_4) and the non-backbone set NB={T_2, T_3}. Therefore, it can be seen that the number of messages reduced is formulated as $\sum_{T_j \in NB} R(T_j, B)$. In this case, the number of messages reduced is $R(T_2, B) + R(T_3, B) = 8 + 2 = 10$.

The time complexity of algorithm SB is analyzed as follows. For each iteration of backbone selection, the backbone gains of all non-backbones should be updated. Suppose there are i non-backbones currently exist, then that update costs $O(i * O(R))$. After this update, a non-backbone with maximal backbone gain is selected as a backbone. This selection costs $O(i * log(i))$. For the worst case that the iterations of backbone selection execute $|Q|$ times. From above, the time complexity of SB is $\sum_{i=1}^{|Q|}(O(i * O(R)) + O(i * log(i)))$.