Collective Spatial Keyword Querying

1. PROCESSING TYPE2 SPATIAL GROUP KEYWORD QUERIES

1.1 Outline of Our Solutions

We note that the optimal solution of TYPE2 problem can be limited in an area. To be specific, a circle area is reasonable because the distance of any two objects in a circle area doesn't exceed the diameter of the circle which corresponds to the second partial defination of TYPE2 problem. When such circle areas is determined the number of candidate objects can be greatly decreased which will lead to a reduced search space for further searching in these potential areas.

Based on given objects, it's possible to constuct a circle which covers all these objects while minimising the area. Actually, this problem is called Smallest Enclosing Circle which has been perfectly solved. The idea of our method is the inverse procedure of constructing smallest enclosing circle, that is, given the "approximate" enclosing circle we believe that the optimal solution must be covered by this circle. In Subsection 1.2 we introduce the relationship between the diameter of enclosing circle and the range of potential solution's cost.

To define the circle area, the diameter and position of the enclosing circle are decisive. We note that the smaller the enclosing circle is, the more objects decreased. In Subsection 1.3, we prove the lowerbound of the diameter to cover the optimal soluton. We utilize a binary search to determine the lowerbound with both correctness and efficiency. In addition to the diameter, the position is also needed to determine the circle area where we introduce a sweeping method with minimum increment.

When potential circle areas have been determined, an exhaustive search is needed to get the optimal combination of objects. In Subsection 1.4 we introduce an A-star based search with distance pruning and batched selection to approximate the optimal solution with fewer iteration.

1.2 Circle Range Limitation

Recall the defination of TYPE2 problem. The total cost is determined by two parts where the first part depends on the furthest object while the second part depends on all pairs of feasible objects. For ease of presenting our algorithm, we first consider a special case that parameter α equals 0 disregarding the first part cost. As the final algorithm for TYPE2 problem is a derivatization of the simplified problem, the algorithm framework remains similar while some details vary. To avoid being confused by the complicated details and to make the algorithm clean and comprehensible, we first introduce how to solve the simplified problem. Then we express how to generalize the algorithm to solve the original problem when parameter α considered.

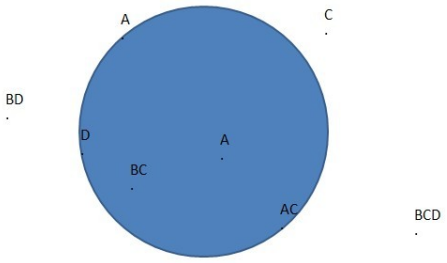

Figure 1: An example of Smallest Enclosing Circle with query {A,B,C,D}.

For some given objects, it's possible to find a Smallest Enclosing Circle which covers all the objects and has the diameter as small as possible. A circle is helpful for us to make some limitation, because it's obvious that the distance of any two objects won't exceed the diameter of this smallest enclosing circle D. Therefore, a circle limitation is helpful for us to approach the exact result of TYPE2 problem which is challenging to find the diameter of the keywords-covered objects. In order to avoid the obscurity of two diameter defination, we identify the diameter of the enclosing circle D and the diameter of the inside objects L.

For known objects it's always possible to find such a circle limitaion, nevertheless, the problem in TYPE2 is the inverse process which aims to select a subset of all objects with a circle limitaion and covers all the keywords as well. Figure 1.2 shows an example of smallest enclosing circle.

Theorem 1.1: *For a set of given points P on the 2-dimensional plane, the diameter of P is L and the diameter of P's smallest enclosing circle is D. If $|P| > 1$ we have the limition relationship:* $\frac{\sqrt{3}}{2} \cdot D \leq L \leq D$

Proof:*Some proof here* □

SIGMOD'11, June 12–16, 2011, Athens, Greece.

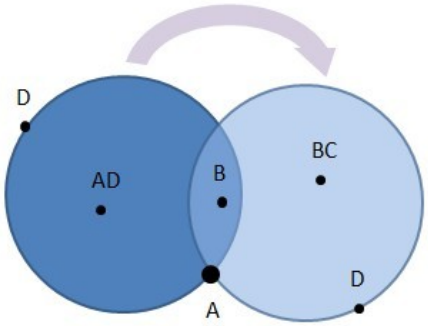

Figure 2: Sweeping Circle Method Schematic Diagram

The value of $\frac{\sqrt{3}}{2}$ will be mentional repeatedly, so we use λ to substitute this constant. According to the above property, it's more clear if we have the inequality changed into: $L \leq D \leq \frac{L}{\lambda}$. This means if we find a circle with diameter D and with feasible objects inside we can make sure that the result of diameter L must be $[\lambda \cdot D, D]$ without any calculation. Namely, the upperbound and lowerbound can be utilized to prune in the search space when even the best lowerbound value can't lead to a better solution.

Lemma 1.2: *The value of diameter D corresponding to the optimal solution is monotonous. That is, if the circle covering the optimal solution with a diameter D we can always find a bigger circle with diameter $D'(D' > D)$ that covers the optimal solution.*

Proof:*If set S is the optimal solution of* TYPE2 *problem, we could find it's smallest enclosing circle C with a diameter D. The circle C' with diameter D' could fulfill the condition if put outside the circle C.* □

An obvious opinion according to Lemma 1.2 is that we devote to find a circle which both covering all the query keywords and with a diameter as small as possible. But according to Theorem 1.1 the reverse process is not absolutely correct if we believe that the circle with the smallest diameter will cover the optimal feasible solution. We will discuss this problem and show how to fix it in the next part.

1.3 Sweeping Circle

We just showed that a circle would help us to limit the objects in some specific area. It's still challenging to find a concrete position of the circle. We know that three points on the 2-dimensional plane can determine a circle, but for the smallest enclosing circle problem we have an analogous conclusion.

Lemma 1.3: *For a set of given points $P(|P| > 1)$ on the 2-dimensional plane, at least two points in P are on the boundary of the smallest enclosing circle.*

Proof:*Let's utilize reduction to absurdity to prove. Suppose that only one point P_i in P is on the boundary of the circle C. If the diameter of C is D, an ϵ exists such that a circle C' with diameter $D - \epsilon$ and C' is tangent to C at point P_i. If any point is outside the circle C', we can decrease ϵ until all the points of P inside C'. Due to $D - \epsilon < D$, C' is smaller than C so C is not the smallest enclosing circle.* □

Now we can fix a point on the boundary of the circle based on Lemma 1.3. There is also an $O(nlogn)$ algorithm to calculate the diameter of given points which utilizes **Rotating Callipers in Convex Hull**. In the rotating callipers algorithm, a convex hull is computed and two horizontal lines are defined that enclose the points. Then rotate the lines together while proceeding along the convex hull. In our problem we enumerate an object to be fixed and have the diameter-determined circle rotated. If in a rotating angle all the objects inside this circle cover all the query keywords, we can make sure that at least one feasible solution exists among these objects. As we enumerate all the objects to be the fixed one, even for the best solution one object must be on the boundary of the sweeping circle according to Lemma 1.3. Figure 1.3 reflects the sweeping circle method for query {A,B,C,D}. Given the fixed object and a proper circle diameter D, the best solution must be cover by the sweeping circle at a specific angle.

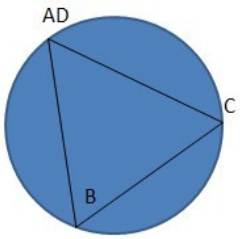

Figure 3: Extreme case of diameter lower bound.

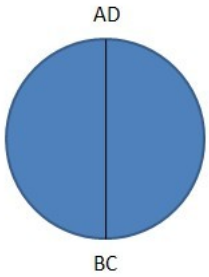

Figure 4: Extreme case of diameter upper bound.

The sweeping circle strategy is based on an assumption that the circle diameter D is given. Now we discuss how to determine the value of D to get the correct optimal results as well as avoiding useless enumeration with efficiency.

First, the correctness must be guaranteed. As we have introduced before, a smaller diameter seems always better by intuition but whether the minimum diameter would not miss the optimal objects and guarantee the minimum value L?

Example 1.1: Consider this situation as Figure 1.3 and Figure 1.3 shows: Figure 1.3 shows the furthest objects pair is exactly the diameter of enclosing circle. Figure 1.3 shows the three objects form an equilateral triangle. Here we have $L_1 = \lambda \cdot D_1$ and $L_2 = D_2$ which reflects two extreme cases of Theorem 1.1 respectively. But if $L_1 < L_2$ and $D_1 > D_2$, we get $\lambda \cdot D_1 < D_2 < D_1$ which is totally rational. If we consider D_2 to be the lower bound to find the optimal solution, we would miss L_1 which is a better result. □

Example 1.1 shows a counter-example of considering the minimum of diameter D would cover the optiaml solution.

Lemma 1.4: *Suppose D to be the lower bound of diameter that*

could cover a feasible set of objects. Sweeping circle with a diameter of $\frac{D}{\lambda}$ could cover the optimal solution at some specific position.

Proof:*Formally, let L be the optimal distance of furthest pair objects, and L' be the second optimal one. We have $L \leq L'$.*

According to Theorem 1.1, $L \leq D \leq \frac{L}{\lambda}$ where D corresponds to the smallest enclosing circle.

Some proof here □

Second, we have known the lower bound value of diameter D that covers at least one feasible result could be utilized to ensure the correctness of covering optimal result. Now we consider to compute it more efficiently. The algorithm's idea is to perform a binary search to sovle this problem in $O(logn)$ time cost. Based on Lemma 1.2, the diameter D is monotonous that corresponds to the optimal solution. Actually, it's obvious to come up with a same monotonous property to the feasible solution. Formally,

$$minD = \min fesD_i$$

where $fesD_i$ is the minimum circle diameter to ensure a feasible solution when we fix $object_i$ and perform the sweeping method. For a given diameter D, we utilize sweeping circle method with a fixed $object_i$ by enumeration. If $D < minD$ we can't find any feasible solution, then we'll try a larger diameter.

The pesudocodo is given in Algorithm 1. In lines 1-2, we set the initial interval to be $(low, high)$ here ∞ can be set as a very large value to ensure that the initial interval would not miss the optimal solution. In line 3-13, we perform a binary search framework iteratively approach the $minD$. We check the median value D of current seach interval. By enumerating all the objects (line 6) and fixing this object as pivot we invoke the $circle\_scan()$ function to inspect whether a feasible solution exists with diameter D. If any feasible position is found we can ensure that $minD < D$, so in next iteration only consider the left interval (low, D). Conversely, the right interval $(D, high)$ needs to be searched. We set an eps to restraint the binary search steps where eps is a very small value. So the stoping criteria for the binary search is the length of current searching interval exceed eps (line 3).

When the $minD$ has been determined, we enlarge the value of D by dividing λ (line 14) according to Lemma 1.4 to ensure the correctness for the optimal solution. Then for each pivoit object we invoke $circle\_scan()$ to find specific positions and perform the exhuastive search to compute the concrete result, because the circle limitation strategy only points out the potential area but different combinations of objects may lead to different results. Actually, this is the original TYPE2 problem we want to sovle, but the number of objects has been greatly reduced.

In this binary seach framework we invoke function $circle\_scan()$ twice with the third parameter different . Both of them perform the circle sweeping with an object as the pivot and rotate the circle to find one or more specific angle where all the objects inside the circle would cover all the keywords. For the first invoking we don't need to know the concrete combination, because based on Theorem 1.1 we have L ranges in $[\lambda \cdot D, D]$. But for the second invoking, after a potential area has been found an exhausted search is invoked to compute exact result in this circle area.

Let $object_i$ to be the fixed pivot and D to be the sweeping circle diameter. If all other objects that in the circle with $2D$ as diameter and $object_i$ as the center can't cover all the query keywords, any feasible solution can't be found. It's a obvious pruning that showed in Figure 1.3 where the bigger circle with $2D$ as diameter is the total sweeping area.

Then we introduce how to perform the function $circle\_scan()$, that is, the circle sweeping process. For each object in the scan

Algorithm 1: **Framework of Binary Search (objects,eps)**

```
1  low ← 0;
2  high ← ∞;
3  while high − low > eps do
4      D ← (high + low)/2;
5      is_any_objects_possible ← false;
6      for each object in objects do
7          if circle_scan(object, D, false) then
8              is_any_objects_possible ← true;
9              break;
10     if is_all_objects_possible then
11         high ← D;
12     else
13         low ← D;
14 D ← D/λ;
15 for each object in objects do
16     circle_scan(object, D, true);
17 return D;
```

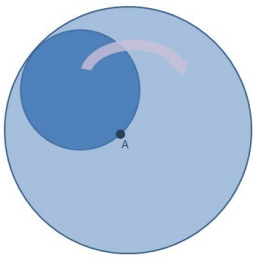

Figure 5: Sweeping Area

area, if we sweep the circle by 360 degree it will be covered by the circle once. Because each object is scaned ouside-in or inside-out by the circle exactly once. The shape of the circle is given and one pivot on the boundary is fixed, so for each $object_i$ it's possible to compute the specific angle $angle\_out_i$ when be scaned outside-in and the angle $angle\_in_i$ when be scaned inside-out. Actually, clockwise or counter-clockwise sweep makes no difference, so we might as well take the clockwise. For each object we have computered the two angles respectively, then sort all these angles. We use minimum increment method to sweep the cirle, that is, for the current sweeping angle we choose whether to sweep in a new object or sweep out an old object that coverd by current circle.

Figure 1.3 shows the minimum incrememt sweeping of the case in Figure 1.3. We map the position where an object swept out of the circle and the position where an object swept into the circle to polar angles of red and blue color respectively. In each step we only move the polar angle to the next position with minimum increment, either cover a new object into the circle or pop-up the last object out of the circle. The polar angles need to be sorted initially and be scanned one by one. Sorting these angles costs $O(nlogn)$ and the $angle\_in$ and $angle\_out$ are exactly scanned once during the sweeping which costs $O(n)$, so the time complexity for a circle sweeping is $O(nlogn)$.

The TYPE2 problem is made up of two parts, and we have disregarded the first part. When consider the distance between the furthest object and the query location, we need to adjust the algorithm to fit the differences. The furthest object to the query actually is on the boundary of the circle, so when we enumerate which object to be the rotating pivot we can treat this object to be the furthest ob-

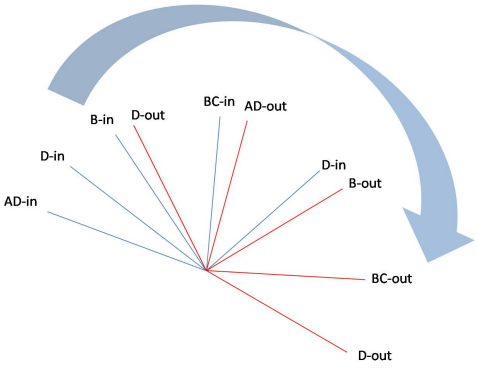

Figure 6: Circle Scan with Minimum Incrememt

ject. So all the other objects swept should be nearer than the pivot. When we perform the binary search we also consider the distance between the pivot and the query location.

Algorithm 2: **Framework of Binary Search for TYPE2 problem (objects,q,eps)**

```
1  low ← 0;
2  high ← ∞;
3  while high − low > eps do
4      totcost ← (high + low)/2;
5      is_any_objects_possible ← false;
6      for each object in objects do
7          D ← totcost − Dist(object, q);
8          if circle_scan(object, D, false) then
9              is_any_objects_possible ← true;
10             break;
11     if is_all_objects_possible then
12         high ← totcost;
13     else
14         low ← totcost;
15 for each object in objects do
16     D ← totcost − Dist(object, q);
17     circle_scan(object, D/λ, true);
18 return totcost;
```

The presudocode is described in Algorithm 2. One of the differences is we utilize binary search for $totcost$ instead of D which is the sum of D and $Dist(object, q)$ (lines 7-8). Another difference is when the lower bound of $totcost$ has been found we enlarge the diameter by dividing λ (line 17) but not the $totcost$.

1.4 Enumerate the best group containing an object

When we have found several circle positions that covers all the query keywords during the sweeping, an exhuastive search getCandidiateSet() is invoked to compute the combination of objects. Unfortunately, if the number of candidiate objects is too large the time cost increases exponentially. To facilitate more efficient checking, we use some pruning strategies based on textual and geometric properties. Suppose $curCost$ to be the optimal result for MAX+MAX SKG problem, the value of $curCost$ is updated during the exhaustive search.

A reasonable search approach is to append a new object to the current selected object set and perform the search iteratively. This new object must cover a new keyword which has not been covered by the selected set. The deepest level of this depth-first-search is at most the number of query keywords because each object contains at least one new keyword. Let $selectedCost$ be the furthest distance of any selected object pair and $furthestDist$ to be the distance between the pivot and query location, the goal is to make $selectedCost + furthestDist < curCost$ and update $curCost$ while covering all query keywords.

According to the cost function definition, the first MAX is related to only the furthest object to the query. Based on this observation, we first choose an object to be the furthest object, so the search space is greatly reduced. Formally, let o_f to be the furthest object we enumerated, an object o should not be taken into consideration iff $\mathsf{Dist}(o, query) > \mathsf{Dist}(o_f, query)$.

Algorithm 3: **enumerateBestGroup (objectSet, pivot, diameter)**

```
1  sort objectSet by Dist(query, ∗);
2  candidateSet ← ∅;
3  keywords ← pivot.ψ;
4  for each object in objectSet do
5      selectedSet ← {pivot, object};
6      candidateSet ← candidateSet + object;
7      pairDist ← Dist(pivot, object);
8      furthestDist ← Dist(query, object);
9      keywords ← keywords ∪ object.ψ;
10     if keywords = query.ψ then
11         localSearch(selectedSet, candidateSet, pairDist,
           furthestDist, pivot.ψ ∪ object.ψ, 0);
```

The presudocode is described in Algorithm 3. The first step is to determine the furthest object. We sort all the objects according to their ascending distance to the query location (Line 1). In line 6, each object is added to the candidateSet one by one to make sure that the last added object must be the furthest to the query location. The selectedSet initially contains only $pivot$ and the furthest object we choosed, so the distance and textual value are initialized respectively (Line 7-9). Finally, when the $candidateSet$ could cover all query keywords we call the function $localSearch()$ to search the optimal solution iteratively(Line 10-11).

Distance Pruning When the first MAX is fixed, the cost of any feasible solution is only related to the second MAX. We use a pruning strategy based on this property: if an object added to the selected object set whose cost is larger than the $curCost$ then the search process can be terminated. This strategy reduces the search space because some objects can be removed from the candidate set for any further search step.

Keywords Pruning A new object is added to the selected object set in each search step, so the $|selectedSet.\psi|$ would increase until it covers all the query keywords. According to this property, we use a keywords pruning strategy that an object o can be pruned iff $o.\psi \subset selectedSet.\psi$. This approach implies an object can be removed from the candidate set if it would not contribute any new keyword.

Terminate Condition The size of candidate set decreases based on the two pruning strategies. We utilize the textual information again to decide whether the search can be terminated. The stop condition is iff $selectedSet.\psi \cup candidateSet.\psi! = query.\psi$. This approach implies that even we selcet all the left objects a feasible solution can not be guaranteed, so the further search is meaningless.

Permutation Order We notice that a set may be enumerated repeatly because different permutations order exist. For example o_1, o_2, o_3 can be generated in another order o_3, o_1, o_2. The number

of ways a set can be generated is factorial, so a lot of time is wasted. In our approach, the set is generated uniquely if the id of each object in ascending order. To be more specific, a candidate object o can be added to the selected set iff $o.id > \max_{s \in selectedSet} s.id$.

Algorithm 4: **localSearch (selectedSet, candidateSet, pairDist, furthestDist, keywords, startId)**

```
1  if keywords = query.ψ then
2      if furthestDist + pairDist < curCost then
3          curCost ← furthestDist + pairDist;
4          curGroup ← selectedSet;
5      return curCost;
6  if furthest + pairDist > curCost then
7      return curCost;
8  nextSet ← ∅;
9  leftKeyowrds ← ∅;
10 for each candidate in candidateSet do
11     if (query.ψ − keywords) ∩ candidate.ψ = ∅ then
12         continue;
13     if candidate.Id < startId then
14         continue;
15     candidate.dist ← 0;
16     for each selectedObject in selectedSet do
17         candidate.dist =
           max(candidate.dist, Dist(selectedObject, candidate));
18     if candidate.dist + furthestDist > curCost then
19         continue;
20     nextSet ← nextSet + candidate;
21     leftKeywords ← leftKeywords + candidate.ψ;
22 if leftKeywords ∪ keywords! = query.ψ then
23     return curCost;
24 for each object in nextSet do
25     selectedSet ← selectedSet ∪ object;
26     localSearch(selectedSet, nextSet, max(pairDist, candidate.dist), furthestDist, keywords∪
       object.ψ, object.Id);
27     selectedSet ← selectedSet − object;
```