An algorithm for the Single Source Weber Problem with Limited Distances

Combinatorial optimization

Giovanni Righini

The problem

The Single Source Weber Problem with Limited Distances (SSWPLD), also known as Facility Location Problem with Limited Distances, is a continuous optimization problem in location theory.

A set $\mathcal{N} = \{1, 2, \ldots, n\}$ of circumferences in $\Re^2$ is given.

For each circumference $i \in \mathcal{N}$ we are given

- a center in position O_i,
- a radius $r_i \geq 0$,
- a weight $w_i \geq 0$.

Objective: locate a point $X \in \Re^2$ minimizing

$$z = \min_{X \in \Re^2} \left\{ \sum_{i \in \mathcal{N}} w_i \ \min \{ d(O_i, X), r_i \} \right\},$$

where $d()$ indicates the Euclidean distance in $\Re^2$.

Decomposition

Drezner et al. (1991): solve an unrestricted single-source location problem for each region of the partition of $\Re^2$ induced by the circumferences.

A region is defined by the subset $Q \subset \mathcal{N}$ of circumferences including it.

The objective can be restated as follows:

$$z = \min_{Q \subseteq \mathcal{N}, X \in \Re^2} \left\{ \sum_{i \in Q} w_i \, d(O_i, X) + \sum_{i \notin Q} w_i r_i : d(O_i, X) \leq r_i \ \forall i \in Q \right\}.$$

The constraint $d(O_i, X) \leq r_i \ \forall i \in Q$ can be dropped, because any solution $(Q, X) : \ \exists i \in Q, d(O_i, X) > r_i$ is dominated by another solution (Q', X) with $Q' = Q \backslash \{i\}$.

Reformulation

Indicating with R the set of regions of $\Re^2$ induced by the circumferences, the SSWPLD can be reformulated as

$$z = \min_{Q \in R, X \in \Re^2} \left\{ \sum_{i \in Q} w_i \, d(O_i, X) + \sum_{i \notin Q} w_i r_i \right\}.$$

If an algorithm is available to compute the optimal location $X^*(Q)$ for each region $Q \in R$, with the corresponding optimal value $z^*(Q)$, then the problem is

$$z = \min_{Q \in R} \left\{ z^*(Q) + \sum_{i \notin Q} w_i r_i \right\}$$

and it can be solved by enumerating the regions in R.

The single-source optimal location problem, or 1-median problem, can be solved by the iterative algorithm by Weiszfeld (1937) or one of its variations.

Enumerating the regions

Theorem (Drezner, Mehrez, Wesolowsky, 1991). A set of n circumferences in $\Re^2$ induces up to $2n(n-1)$ distinct regions.

Corollary. The single-source optimal location algorithm must be executed a quadratic number of times to find the optimum of the SSWPLD.

Observation. Each intersection point between two circumferences is adjacent to four regions.

Enumeration algorithm.

- For each intersection point P between two distinct circumferences $i \in \mathcal{N}$ and $j \in \mathcal{N}$, find the set S_P of circumferences different from i and j that cover P, in $O(n)$.
- Generate $R_P = \{S_P, S_P \cup \{i\}, S_P \cup \{j\}, S_P \cup \{i, j\}\}$, in $O(1)$.
- Repeat for all intersection points ($O(n^2)$ times) to generate $R = \bigcup_P R_P$, in $O(n^3)$ time.

Pathological cases

- Circumferences completely enclosed in one another or disjoint from all the others: not revealed by any intersection point. Correction done by Aloise et al. (2012). The complexity is still $O(n^3)$.
- Degenerate intersection points, where more than two circumferences intersect. Correction proposed by Venkateshan (2019): given an intersection point P between circumferences,
  - find the subset S_P of circumferences *strictly* covering P,
  - find the subset T_P of circumferences passing through P;
  - compute the radius of a “small enough” neighborhood of P containing no intersections besides P;
  - compute the intersections of the circumferences in T_P with the frontier of the neighborhood;
  - following the frontier of the neighborhood, enumerate the set of all relevant subsets of T_P, that correspond to the regions with a vertex in P.

The construction and analysis of the neighborhood requires $O(n^4)$.

Example

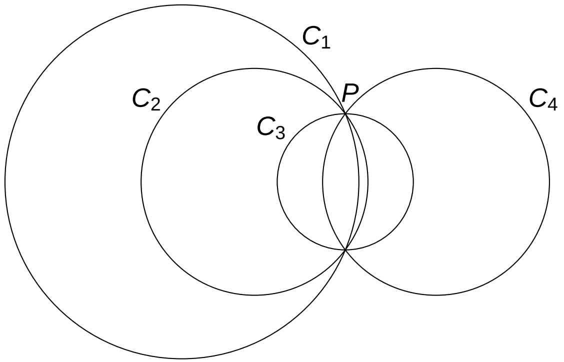


Figure: Four circumferences sharing two intersection points.

For every pair of circumferences intersecting in P, the subset S_P, as defined by Drezner et al. and Aloise et al., would include the other two circumferences; therefore the regions covered by a single circumference (C_1 or C_4) would be missed in the enumeration.

The small enough neighborhood

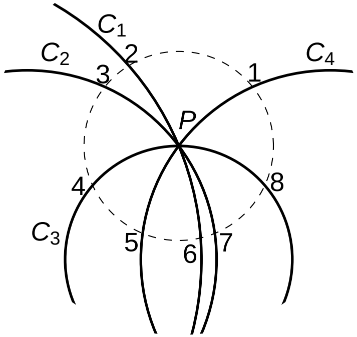


Figure: A small enough neighborhood of P, as defined in Venkateshan (2019).

Following the frontier of the neighborhood, all the 8 regions around P can be correctly enumerated.

Tangent lines

Let us call g_i the direction from P to O_i $\forall i \in T_P$.
Tangent lines: $e_i = g_i - \frac{\pi}{2}$, $l_i = g_i + \frac{\pi}{2}$.
F_P is scanned counter-clockwise, angles are computed modulo 2π.

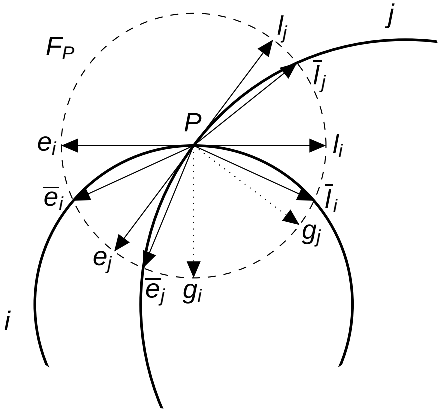


Figure: Oriented tangent lines directions.

Tangent lines

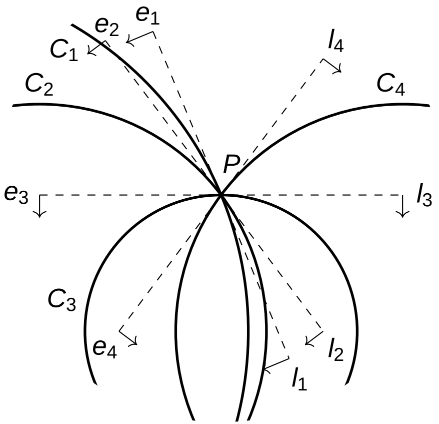


Figure: The order in which the 8 intersection points are encountered along the frontier of F_P corresponds to the order of the directions $e_1, \ldots, e_4$ and $l_1, \ldots, l_4$ sorted according to their angles.

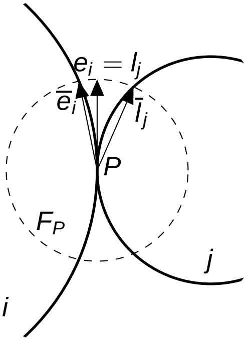
Tie-break rules

Ties can occur because distinct circumferences in T_P can have coincident tangent lines.

Tie-break rule 1. For any $i \neq j \in T_P$ such that $e_i = l_j$ and $e_j = l_i$, l_j precedes e_i and l_i precedes e_j.

Tie-break rule 2. For any $i \neq j \in T_P$ such that $l_i = l_j$ and $e_i = e_j$ with $r_i > r_j$, l_j precedes l_i and e_i precedes e_j.

Tie-break rules: examples



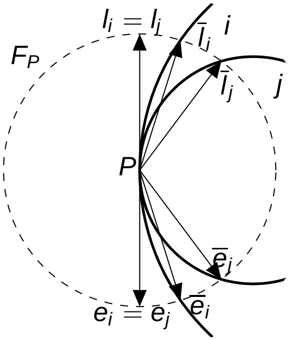


Figure: Tie-break rule 1 (left): when leaving a circumference j and entering a circumference i with $g_i = g_j \pm \pi$, direction $\overline{l}_j$ is encountered before $\overline{e}_i$.
Tie-break rule 2 (right): when leaving circumferences i and j with $g_i = g_j$ and $r_i > r_j$, direction $\overline{l}_j$ is encountered before $\overline{l}_i$; when entering them, $\overline{e}_i$ is encountered before $\overline{e}_j$.

UNIVERSITÀ DEGLI STUDI DI MILANO

Enumeration around P

```
// ProcedureScan($V_P, S_P$)
$Q \leftarrow \emptyset$
**for** $t = 1, \ldots, |V_P|$ **do**
    **if** $V_P[t] > 0$ **then**
        $Q \leftarrow Q \cup \{V_P[t]\}$
    **else**
        **if** $(-V_P[t] \in Q)$ **then**
            $Q \leftarrow Q \setminus \{-V_P[t]\}$
**for** $t = 1, \ldots, |V_P|$ **do**
    **if** $V_P[t] > 0$ **then**
        $Q \leftarrow Q \cup \{V_P[t]\}$
    **else**
        $Q \leftarrow Q \setminus \{-V_P[t]\}$
    Evaluate($Q \cup S_P$)
```

Complexity

Pre-processing.

- merge pairs of circumferences $i \in \mathcal{N}$ and $j \in \mathcal{N}$ with $O_i = O_j$ and $r_i = r_j$ in a unique circumference with the same center, the same radius and weight $w_i + w_j$;
- delete circumferences with radius $r = 0$ or weight $w = 0$.

Complexity: $O(n \log n)$ (not a bottleneck).

Definition. Multiple intersection point (m.i.p.): a point in $\Re^2$ where two or more circumferences intersect.

Complexity

1. Compute the intersection points for all pairs of distinct circumferences and detect when some of them coincide; output a list of m.i.p..
2. For each m.i.p. P, compute the set S_P of circumferences that strictly cover P.
3. Enumerate all regions with a vertex in P for each m.i.p. P and run a single-source optimal location algorithm for each detected region.

We need to analyze the worst-case time complexity of each of these three steps.

Step 1

The set of intersection points between circumferences can be computed in $O(n^2)$.

To detect coincident intersections, intersection points can be sorted so that coincident intersection points turn out to be consecutive in the ordering. Sorting a list of $O(n^2)$ elements requires $O(n^2 \log n)$ time.

All subsets T_P for each m.i.p. P are found in $O(n^2)$, by scanning the sorted list and iteratively merging consecutive elements when their positions coincide.

Each merge operation takes $O(1)$ and there are $O(n^2)$ of them.

Insertion takes $O(1)$ (representing subsets by their binary characteristic vectors) and there are $O(n^2)$ of them.

Hence, the worst-case time complexity of Step 1 is $O(n^2 \log n)$.

Step 2

For each m.i.p. P, finding all the circumferences strictly covering it requires $O(n)$.

Since there are $O(n^2)$ m.i.p., Step 2 has asymptotic worst-case time complexity $O(n^3)$.

This is the complexity of the region enumeration algorithms proposed by Drezner et al. (1991) and Aloise et al. (2012).

Step 3: rough analysis

Consider the subset T_P of c circumferences intersecting in a m.i.p. P.

Computing all directions g_i from P to O_i $\forall i \in T_P$ takes $O(c)$.

Computing all directions e_i and l_i takes $O(c)$.

Sorting the sequence L_P with $2c$ angle values takes $O(c \log c)$.

Scanning L_P to enumerate all regions around P with Scan takes $O(c)$, since insertion and deletion operations can be implemented as $O(1)$ operations on a binary array (whose initialization takes $O(c)$) and the number of iterations of the loops in Scan is bounded by $2c$.

Procedure Scan must be run for all m.i.p., i.e. $O(n^2)$ times.

Therefore, the worst-case complexity of Step 3 is not worse than $O(n^3 \log n)$.

Step 3: refined analysis

Theorem

Consider the multi-graph $\mathcal{M} = (\mathcal{V}, \mathcal{E})$, defined by n intersecting circumferences, where $\mathcal{V}$ is the set of m.i.p. and $\mathcal{E}$ is the set of circumference arcs between them. Then, $|\mathcal{E}|$ grows as $O(n^2)$.

Proof. For any given planar multi-graph $\mathcal{M} = (\mathcal{V}, \mathcal{E})$ inducing a set of regions $\mathcal{R}$ in $\Re^2$, Euler formula holds: $|\mathcal{E}| + 2 = |\mathcal{V}| + |\mathcal{R}|$. By Drezner et al. theorem $|\mathcal{R}|$ is $O(n^2)$. Since $|\mathcal{V}|$ is also $O(n^2)$, then is $|\mathcal{E}|$ is $O(n^2)$. $\square$

Corollary

The total degree of the vertices in $\mathcal{V}$ grows as $O(n^2)$.

This immediately follows from the Theorem, since the total degree is twice the number of edges.

Step 3: refined analysis

Asymptotic worst-case time complexity of Step 3: $O(\sum_{k=1}^{K} c_k \log c_k)$, where K indicates the number of m.i.p. and c_k the number of circumferences intersecting in each m.i.p. $k = 1, \ldots, K$.

Since $c_k \leq n \ \forall k = 1, \ldots, K$, and hence $\log c_k \leq \log n \ \forall k = 1, \ldots, K$, a valid worst-case bound is $O(\log n \sum_{k=1}^{K} c_k)$.

The sum $\sum_{k=1}^{K} c_k$ is half the total degree of the vertices of the multi-graph $\mathcal{M}$.

For the Corollary above, such a total degree grows as $O(n^2)$.

Therefore a worst-case bound for Step 3 is $O(n^2 \log n)$.

Conclusion

Degenerate intersections in the SSWPLD can be dealt with without worsening the $O(n^3)$ worst-case time complexity of the enumeration algorithms proposed so far, that did not take degeneracy into account.

The computational complexity bottleneck in the enumeration is not due to degenerate intersections (affecting Steps 1 and 3), but rather to the need of checking whether each given circumference covers each m.i.p. in Step 2.

All algorithms proposed so far have $O(n^3)$ time complexity, because of this crucial step.

A new algorithm

The given intersecting circumferences induce one or more planar multi-graphs in $\Re^2$.

Their *vertices* are m.i.p.; their *edges* are circumference arcs between adjacent vertices.

The new algorithm runs in four steps.

1. All m.i.p. are identified (vertices of the planar multi-graphs).
2. Vertices occurring along each circumference are sorted according to a given orientation, to identify all edges of the multi-graphs, yielding the star of each vertex.
3. The edges incident to each vertex are sorted, according to a given orientation.
4. Each planar multi-graph is visited with a depth-first-search algorithm and all regions are enumerated.

Visting the regions of $\mathcal{M}$ is the same as visiting the vertices of its dual graph.

Step 1: Enumeration of vertices

1. Enumeration of intersection points
2. Ordering the intersection points
3. Enumeration of vertices

Step 1.1: Enumeration of intersection points

Algorithm Intersections computes three data-structures:

- a subset $\Omega(i)$ of enclosing circumferences $\forall i \in \mathcal{N}$;
- a flag $f(i)$ $\forall i \in \mathcal{N}$: intersecting (1) / isolated (0);
- a list Λ of all intersection points.

```
// Procedure Intersections IN: $O, r$. OUT: $\Omega, f, \Lambda$
for $i = 1, \dots, n$ do
    $f(i) \leftarrow$ false
    $\Omega(i) \leftarrow \emptyset$
$\Lambda \leftarrow \emptyset$
for $i = 1, \dots, n-1$, $j = i+1, \dots, n$ do
    if $(d(O_i, O_j) < |r_i - r_j|)$ then
        if $r_i > r_j$ then
            $\Omega(j) \leftarrow \Omega(j) \cup \{i\}$
        else
            $\Omega(i) \leftarrow \Omega(i) \cup \{j\}$
    else
        if $(d(O_i, O_j) \leq r_i + r_j)$ then
            $[P(i,j), P(j,i)] \leftarrow$ Intersect$(i,j)$
            $f(i) \leftarrow$ true
            $f(j) \leftarrow$ true
            $\Lambda \leftarrow \Lambda \cup \{[i, j, x(P(i,j)), y(P(i,j))], [j, i, x(P(j,i)), y(P(j,i))]\}$
```

Complexity: $O(n^2)$.

Intersection points

Assume all circumferences are followed counter-clockwise.

In $P(i,j)$ circumference i enters circumference j and circumference j leaves circumference i.

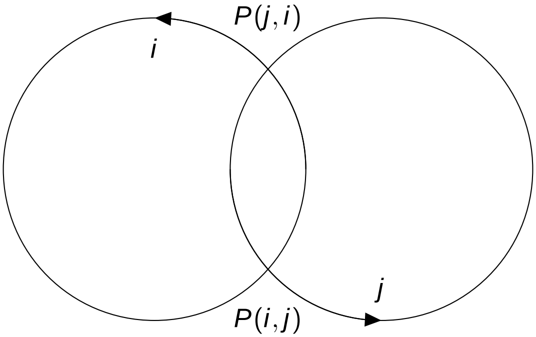


Figure: Intersection points between two circumferences.

Step 1.2: Ordering the intersection points

Sort Λ so that coincident points occur in consecutive positions.

For instance: sort Λ by non-decreasing values of x and break ties by non-decreasing values of y.

Since $|\Lambda|$ is $O(n^2)$, the complexity of sorting is $O(n^2 \log n)$.

It turns out to be a bottleneck of the whole region enumeration algorithm.

Step 1.3: Enumeration of vertices

Assume that the sorted list Λ is transformed into an array with $O(n^2)$ elements.

The sorted array Λ is scanned to find the vertices in $O(n^2)$.

t', t'': first and last position of the elements in each subset of coincident points.

v: number of vertices found.

For each vertex $k = 1, \ldots, v$,

- $x(k)$ and $y(k)$: its coordinates;
- $T(k)$: set of all circumferences passing through it.

Each set $T(k)$ can be implemented as a balanced tree to detect (and delete) duplicates. Each insertion in $T(k)$ takes $O(\log n)$.

```
// Procedure FindVertices. IN: $\Lambda$. OUT: $x, y, T, v$
$v \leftarrow 0$
$t' \leftarrow 1$
**while** $t' \leq |\Lambda|$ **do**
    $t'' \leftarrow t' + 1$
    **while** $(t'' \leq |\Lambda|) \wedge (\Lambda[t''].x = \Lambda[t'].x) \wedge (\Lambda[t''].y = \Lambda[t'].y)$ **do**
        $t'' \leftarrow t'' + 1$
    $v \leftarrow v + 1$
    $x(v) \leftarrow \Lambda[t'].x$
    $y(v) \leftarrow \Lambda[t'].y$
    $T(v) \leftarrow \emptyset$
    **for** $h = t', \ldots, t'' - 1$ **do**
        $T(v) \leftarrow T(v) \cup \{\Lambda[h].i, \Lambda[h].j\}$
    $t' \leftarrow t''$
```

Complexity: $O(n^2 \log n)$.

Step 1: Enumeration of vertices

1. Enumeration of intersection points: $O(n^2)$
2. Ordering the intersection points: $O(n^2 \log n)$
3. Enumeration of vertices: $O(n^2 \log n)$

Complexity of Step 1: $O(n^2 \log n)$.

Step 2: Enumeration of edges

Sort the vertices following each circumference counter-clockwise.

1. Enumeration of the vertices along each circumference
2. Sorting the vertices along each circumference
3. Building vertex stars

Step 2.1: Enumeration of the vertices along each circumference

A set $W(i)$ of vertices is computed $\forall i \in \mathcal{N}$.

Scan the vertex list: for each $i \in T(k)$, insert k in $W(i)$.

```
// Procedure EnumerateVertices. IN: $T$, $v$. OUT: $W$
for $i \in \mathcal{N}$ do
    $W(i) \leftarrow \emptyset$
for $k = 1, \dots, v$ do
    for $i \in T(k)$ do
        $W(i) \leftarrow W(i) \cup \{k\}$
```

Every time a vertex belongs to a circumference, it contributes by an amount of 2 to the total degree of the multi-graph.

The total degree of the multi-graph is $O(n^2)$; hence, there are $O(n^2)$ insertions.

Complexity: $O(n^2)$.

Step 2.2: Sorting the vertices along each circumference

Examine each circumference $i \in \mathcal{N}$ separately.

For each vertex $k \in W(i)$ compute the angle $\alpha(i, k)$ of the direction from O_i to $(x(k), y(k))$.

Sort $W(i)$ by increasing values of α.

No tie can occur, because by construction all vertices in $W(i)$ are distinct.

```
Procedure SortVertices. IN: $W$. OUT: $W$
for $i = 1, \ldots, n$ do
    for $k \in W(i)$ do
        $\alpha(i,k) \leftarrow \mathsf{arctan}(O_i, (x(k), y(k)))$
    $W(i) \leftarrow \mathrm{Sort}(W(i))$
```

Function $\mathsf{arctan}()$ is assumed to return a value in $[0, 2\pi)$ computed counter-clockwise from the positive x semiaxis.

Since the number of (i,k) pairs is $O(n^2)$, the number of calls to $\mathsf{arctan}()$ is $O(n^2)$.

Sorting the vertices takes $O(|W(i)| \log |W(i)|)$ $\forall i \in \mathcal{N}$.
$|W(i)| \leq 2(n-1)$ $\forall i \in \mathcal{N}$ and $\sum_{i=1}^{n} |W(i)| \leq 2n(n-1)$.

Complexity: $O(n^2 \log n)$ (bottleneck of the algorithm).

Step 2.3: Building vertex stars

For each vertex k each edge in its star $H(k)$ is a triplet (i, γ, h), where

- Index i: index of a circumference passing through k,
- Direction γ: edge direction "counter-clockwise" (1) / "clockwise" (0),
- Endpoint h: vertex reached from k following i in direction γ.

For each $i \in \mathcal{N}$, scan $W(i)$ as a circular list.

For each pair of consecutive vertices k' and k'' along it, insert the edge between k' and k'' in $H(k')$ as a counter-clockwise edge entering k'' and in $H(k'')$ as a clockwise edge entering k'.

Special case: if $W(i)$ contains a single vertex k, then two edges are inserted in $H(k)$ with opposite directions and endpoint k.

Step 2.3: Building vertex stars

```
Procedure BuildStar. IN: $W$. OUT: $H$
for $k = 1, \ldots, v$ do
    $H(k) \leftarrow \emptyset$
for $i \in \mathcal{N}$ do
    for $k' \in W(i)$ do
        $k'' \leftarrow \text{succ}(k')$
        $H(k') \leftarrow H(k') \cup \{(i, 1, k'')\}$
        $H(k'') \leftarrow H(k'') \cup \{(i, 0, k')\}$
```

Scanning all subsets W takes $O(n^2)$.

Each edge is inserted in two stars: the total n. of elements in subsets H is twice the total n. of edges in the multi-graph.

Complexity: $O(n^2)$.

Step 2: Enumeration of edges

1. Enumeration of the vertices along each circumference $O(n^2)$
2. Sorting the vertices along each circumference $O(n^2 \log n)$
3. Building vertex stars $O(n^2)$

Complexity of Step 2: $O(n^2 \log n)$.

Step 3: Sorting vertex stars

Sort the stars, so that consecutive edges belong to the frontier of a same region, owing to the planarity of the multi-graph.

1. Computing edge directions
2. Sorting the edges

Step 3.1: Computing edge directions

A fourth field β is added to the three-field records (i, γ, h) in $H(k)\ \forall k = 1, \ldots, v$.

It is the direction of the line tangent to i in k, oriented *from* k in direction γ.

The tangent certainly exists, because pre-processing guarantees that all circumferences have strictly positive radius.

Angles are computed counter-clockwise starting from the direction of the positive x semiaxis.

Step 3.1: Computing edge directions

```
Procedure ComputeDirections. IN: $H$, $O$, $(x, y)$. OUT: $\beta$
for $k = 1, \dots, v$ do
    for $(i, \gamma, h) \in H(k)$ do
        if $\gamma = 1$ then
            $\beta \leftarrow (\arctan(O_i, (x(k), y(k))) + \pi/2) \bmod 2\pi$
        else
            $\beta \leftarrow (\arctan(O_i, (x(k), y(k))) - \pi/2) \bmod 2\pi$
        Replace $(i, \gamma, h)$ with $(i, \gamma, h, \beta)$
```

The total number of elements in the subsets H is $O(n^2)$.

Complexity: $O(n^2)$.

Step 3.2: Sorting the edges

Sort $H(k)\ \forall k = 1, \ldots, v$ counter-clockwise, according to the values of β.

Tie-break criterion 3.
Given a tie between two edges $(i, 0, \beta)$ and $(j, 1, \beta)$, $(i, 0, \beta)$ must precede $(j, 1, \beta)$ in $H(k)$.

Tie-break criterion 4.
(a) Given a tie between two edges $(i, 0, \beta)$ and $(j, 0, \beta)$ with $r_i < r_j$, $(i, 0, \beta)$ must precede $(j, 0, \beta)$ in $H(k)$.
(b) Given a tie between two edges $(i, 1, \beta)$ and $(j, 1, \beta)$ with $r_i < r_j$, $(j, 1, \beta)$ must precede $(i, 1, \beta)$ in $H(k)$.

Step 3.2: Sorting the edges

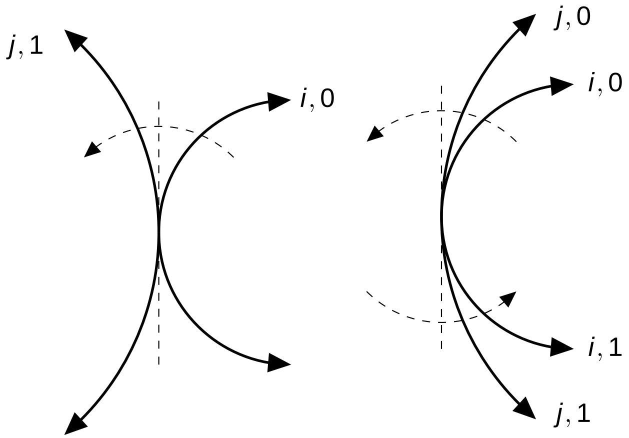


Figure: Left: Tie-break criterion 3 is applied to break ties between $(i, 0, \beta)$ and $(j, 1, \beta)$. Right: Tie-break criterion 4 is applied to break ties between $(i, 0, \beta)$ and $(j, 0, \beta)$ and between $(i, 1, \beta)$ and $(j, 1, \beta)$.

Step 3.2: Sorting the edges

Each sorted list $H(k)$ is managed as a circular array.

The edges in $H(k)$ are sorted in the same order as are encountered when moving counter-clockwise along the frontier of a small enough neighborhood of vertex k.

Property. Consecutive edges in $H(k)$ belong to the contour of a same region.

Complexity. There are $O(n^2)$ vertices and there are $|H(k)|$ edges in each vertex star. The complexity for sorting all vertex stars is $O(\sum_{k=1}^{v} |H(k)| \log |H(k)|)$. $|H(k)| \leq 2n \; \forall k = 1, \ldots, v$ and $\sum_{k=1}^{v} |H(k)| \leq 4n(n-1)$.

Complexity: $O(n^2 \log n)$ (bottleneck of the algorithm).

Step 3: Sorting vertex stars

1. Computing edge directions: $O(n^2)$
2. Sorting the edges: $O(n^2 \log n)$

Complexity of Step 3: $O(n^2 \log n)$.

Step 4: Connected components

We define a *connected component* to be a set of circumferences, such that it possible to move from any of them to any other along their arcs.

Each circumference belongs to exactly one connected component.

The connected component $\phi(i) \subseteq \mathcal{N}$ of circumference i exists and is unique for each $i \in \mathcal{N}$.

An isolated circumference is just a special case of a connected component.

Theorem

If circumference $i \in \mathcal{N}$ is a rightmost circumference in its connected component $\phi(i)$, then $\Omega(i)$ is the set of circumferences that strictly enclose $\phi(i)$.

Internal regions

Observation. The set of points of $\Re^2$ not enclosed in any connected component is of no interest, because it cannot contain the optimal solution. It is the set of the worst solutions of the SSWPLD, where $z = \sum_{i \in \mathcal{N}} w_i r_i$.

Observation. Distinct connected components induce disjoint sets of regions, that can be enumerated independently. Their union is the whole set of regions that must be enumerated.

Step 4 visits the whole set of input circumferences, one connected component at a time.

If a connected component is a multi-graph, then it is completely visited and all its internal regions are enumerated.

Data-structures

N: set of unvisited circumferences.

Circumferences in N are sorted

1. by non-increasing x value of their rightmost point;
2. by decreasing values of their radius;
3. at random.

A rightmost unvisited circumference is iteratively selected and its connected component is visited.

$\mu(k)$: sequential number of a visited vertex $k = 1, \ldots, v$.

$\overline{\mu}$: counter of visited vertices.

The visit of a multi-graph with DFS starts from vertex $W(i^*)_1$. By construction, this is first vertex in $W(i)$ that is encountered moving along circumference i counter-clockwise starting from its rightmost point.

```
Procedure Components
$N \leftarrow \text{SortCircles}(\mathcal{N})$
$\overline{\mu} \leftarrow 0$
for $k \in 1, \ldots, v$ do
    $\mu(k) \leftarrow 0$
while $N \neq \emptyset$ do
    $i^* \leftarrow \text{Rightmost}(N)$
    $Q \leftarrow \Omega(i^*)$
    if $f(i^*)$ then
        /* Multi-graph */
        $k \leftarrow W(i^*)_1$
        $\overline{\mu} \leftarrow \overline{\mu} + 1$
        $\mu(k) \leftarrow \overline{\mu}$
        $\text{ScanStar}(k, i^*, 1)$
    else
        /* Isolated circumference */
        $\text{Flip}(i^*)$
        $\text{Evaluate}(Q)$
        $N \leftarrow N \setminus \{i^*\}$
```

DFS: forward edges and backtrack edges

In $H(k)$ (circular array) each edge incident to k has a successor.

Every time a vertex k is reached for the first time, its star $H(k)$ is scanned counter-clockwise, starting from the successor in $H(k)$ of the last traversed edge.

When traversing an edge, two cases can occur:

- if $\mu(h) = 0$, then vertex h has not yet been visited (forward edge): ScanStar is recursively called to scan $H(h)$;
- otherwise (backtrack edge), the search backtracks from h to k.

Property. Each star is scanned once; each forward edge is traversed once; each backtrack edge is traversed twice.

Data-structures

Initialization. A procedure (FindEdge) computes the position t in $H(k)$ of the edge that has been traversed to reach vertex k, i.e. the edge leaving k along circumference i in direction opposite to γ.

Such an edge certainly exists and is unique, owing to the following property.

Property. By construction, $H(k)$ contains exactly two records $(i, 0, *)$ and $(i, 1, *)$ for each circumference $i \in \mathcal{N}$ passing through vertex $k = 1, \ldots, v$.

Iteration. A call to ScanStar$(\mathrm{k}, \mathrm{i}, \gamma)$ has three parameters:

- k: the vertex whose star must be scanned;
- i: the circumference of the edge traversed to reach k;
- γ: the direction in which the edge has been traversed.

They are passed by value: a local copy is created for each call.

```
Procedure ScanStar($k, i, \gamma$)
$t \leftarrow$ FindEdge($k, i, 1 - \gamma$)
**for** $p = 1, \ldots, |H(k)| - 1$ **do**
    $t \leftarrow t \bmod |H(k)| + 1$
    $(j, \gamma', h) \leftarrow H(k)[t]$
    $N \leftarrow N \setminus \{j\}$
    **if** $\mu(h) = 0$ **then**
        /* Forward edge */
        $\overline{\mu} \leftarrow \overline{\mu} + 1$
        $\mu(h) \leftarrow \overline{\mu}$
        ScanStar($h, j, \gamma'$)
    **else**
        /* Backtrack edge */
        Flip(j)
        **if** $(\mu(h) < \mu(k)) \vee ((\mu(h) = \mu(k)) \wedge (\gamma' = 1))$ **then**
            /* First traversal */
            Evaluate($Q$)
```

UNIVERSITÀ DEGLI STUDI DI MILANO

Properties of the search

Property. Since all vertex stars are completely scanned in each connected component, then all vertices are visited and all edges are traversed.

Property. DFS defines an orientation of the edges, i.e. the direction in which they are traversed the first time. Since all edges are traversed, all edges are oriented.

Property. Since all vertices are reached for the first time once, then each vertex has one forward edge entering it, but the start vertex of each connected component.

Property. Forward edges cannot form directed circuits, since $\mu(h) > \mu(k)$ for all forward edges from k to h.

Theorem. The set of forward edges is a spanning arborescence rooted at the starting vertex for each connected component.

Detecting first and second traversals

No attempt is made to traverse forward edges for the second time.

On the contrary, backtrack occurs twice on each backtrack edge.

Property. The current vertex has maximum value of μ among all open vertices.

Theorem. When a backtrack edge is traversed the first time and its orientation is from vertex k to vertex h, then $\mu(h) \leq \mu(k)$.

Proof. When an edge from k to h is traversed the first time, k is the current vertex. Since the edge belongs also to $H(h)$ and it has not yet been traversed from h to k, then h is open. By the Property above, $\mu(k) \geq \mu(h)$. $\square$

Self-loops

When $\mu(h) = \mu(k)$, the edge is a self-loop.

Property. The unique vertex k of a self-loop on a circumference $i \in \mathcal{N}$ cannot be reached from any forward edge along or within circumference i.

Proof. The rightmost point of its connected component cannot be inside circumference $i \in N$.

Property. When $H(k)$ is scanned counter-clockwise, the edge corresponding to traversing the self-loop counter-clockwise is encountered before the edge corresponding to traversing the self-loop clock-wise.

Test for detecting when a self-loop is traversed the first time is $\mu(h) = \mu(k) \ \wedge \ \gamma' = 1$.

From visiting edges to enumerating regions

Exploiting the planarity property of the multi-graph, the depth-first-search algorithm transforms the guarantee of traversing all edges into the guarantee of enumerating all regions within them.

However, we want to evaluate each region only once.

A global variable $Q \subseteq \mathcal{N}$ represents the *current region*.

Data-structure: binary vector. Inserting or deleting an element is done in $O(1)$ by flipping the corresponding bit (procedure Flip).

The current region Q is iteratively updated during the DFS visit.

Under suitable conditions the current region Q is evaluated (a single-source optimal location algorithm is run).

Flip rule and Evaluation rule

Observation. Two adjacent regions separated by an edge belonging to circumference j correspond to subsets that differ only by component j.

Hence, flipping $Q[j]$ corresponds to moving from the region on one side of an edge belonging to circumference $j \in \mathcal{N}$ to the region on the other side.

Rule 1 (Flip rule). *$Q[j]$ is flipped iff a backtrack occurs on an edge along circumference $j \in \mathcal{N}$.*

Rule 2 (Evaluation rule). *The current region Q is evaluated iff it is on the left side of a backtrack edge traversed the first time.*

Moves

Definition.
Forward move: DFS traverses an edge.
Backward move: DFS backtracks along an edge.

Definition. ν is a natural number enumerating the moves in the order they are done by DFS.

Definition. $e(\nu)$ the edge along which move ν occurs.

Definition. right(e), left(e): regions on the right/left side of edge e, according to its orientation.

Definition. right(ν), left(ν): regions on the right/left side with respect to move ν.

Observation. right(ν) $=$ right($e(\nu)$) and left(ν) $=$ left($e(\nu)$) iff $e(\nu)$ is traversed according to its orientation.
right(ν) $=$ left($e(\nu)$) and left(ν) $=$ right($e(\nu)$) iff $e(\nu)$ is traversed opposite to its orientation.

Enumerated regions

Definition.
$R(\nu)$: the set of regions enumerated by DFS up to move ν.
$Q(\nu)$: the current region when move ν is done.

Definition.
$R(0)$: the region surrounding the current connected component.

Observation. $R(\nu') \subseteq R(\nu'') \; \forall \nu' < \nu''$, since R is only subject to insertions, not to deletions.

Theorem (Left-right property).
For each forward move ν, $Q(\nu) = \text{right}(\nu) \in R(\nu - 1)$ (right property).
For each backward move ν, $Q(\nu) = \text{left}(\nu) \in R(\nu)$ (left property).

Proof. The proof is by induction:
(i) prove that the properties hold for $\nu = 1$;
(ii) assume that the two properties hold for all moves up to move $\nu - 1$ and prove that they hold for move ν.

Basis of the induction

By the initialization of Q, $Q(1) = \Omega(i)$.

By construction, $\Omega(i) = \text{right}(e(1))$.

The edge traversed by move $\nu = 1$ is certainly traversed for the first time; hence $\text{right}(1) = \text{right}(e(1))$.

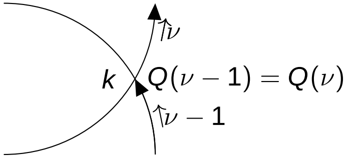
By the initialization, $R(0) = \Omega(i)$.

Hence the right property $Q(1) = \text{right}(1) \in R(0)$ holds for $\nu = 1$.

To prove the induction step we distinguish four cases, depending on $\nu - 1$ and ν being forward or backward moves.

Case I: move $\nu - 1$ is forward and move ν is forward

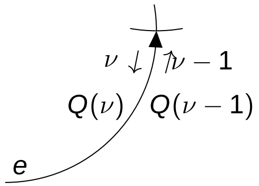
In this case $e(\nu - 1)$ and $e(\nu)$ belong to $H(k)$ for some k: $e(\nu - 1)$ is the forward edge entering k and k is reached for the first time along it.
Then, $e(\nu)$ is not traversed by any move $\nu' < \nu - 1$.
Then, $e(\nu - 1)$ and $e(\nu)$ are traversed according to their orientations: $\text{right}(\nu - 1) = \text{right}(e(\nu - 1))$ and $\text{right}(\nu) = \text{right}(e(\nu))$.
Edge $e(\nu)$ is the edge next to $e(\nu - 1)$ in $H(k)$ counter-clockwise.
Hence $\text{right}(e(\nu)) = \text{right}(e(\nu - 1))$.
By the induction hypothesis, $Q(\nu - 1) = \text{right}(\nu - 1) \in R(\nu - 2)$.
By Rule 1, $Q(\nu) = Q(\nu - 1)$.
By construction, $R(\nu - 2) \subseteq R(\nu - 1)$.
Therefore $Q(\nu) = \text{right}(\nu) \in R(\nu - 1)$: the right property holds for the forward move ν.



Case II: move $\nu - 1$ is forward and move ν is backward

In this case $e = e(\nu - 1) = e(\nu)$ is a backtrack edge.

Case IIa. If e is traversed for the first time, $\text{right}(\nu - 1) = \text{right}(e)$ and $\text{left}(\nu) = \text{left}(e)$.
By the induction hypothesis, $Q(\nu - 1) = \text{right}(\nu - 1)$.
By Rule 1 $Q(\nu - 1) = \text{right}(e)$ implies $Q(\nu) = \text{left}(e)$ (e is flipped).
Then, $Q(\nu) = \text{left}(\nu)$.
By Rule 2, if e is traversed for the first time, $Q(\nu)$ is inserted in $R(\nu)$.
Hence $Q(\nu) = \text{left}(\nu) \in R(\nu)$.



Case II: move $\nu - 1$ is forward and move ν is backward

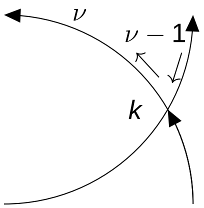
Case IIb. If e is traversed for the second time, then
right$(\nu - 1) =$ left(e), left$(\nu) =$ right(e).
By the induction hypothesis, the right property holds up to $\nu - 1$, i.e.
$Q(\nu - 1) =$ right$(\nu - 1) \in R(\nu - 2)$.
By Rule 1, $Q(\nu - 1) =$ left(e) implies $Q(\nu) =$ right(e) (e is flipped).
Then, $Q(\nu) =$ left(ν).
If $e = e(\nu - 1)$ is visited for the second time, there exists a forward move $\nu' < \nu - 1$ such that $e = e(\nu')$.
By the induction hypothesis, $Q(\nu') =$ right$(\nu') \in R(\nu' - 1)$; moreover right$(\nu') =$ right(e), because ν' is a forward move along e.
By Rule 2, $R(\nu) = R(\nu - 1)$.
Therefore $Q(\nu) =$ left$(\nu) \in R(\nu)$.

So, in both cases the left property holds for the backward move ν.

Case III: move $\nu - 1$ is backward and move ν is forward

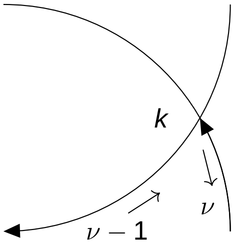
In this case $e(\nu-1)$ and $e(\nu)$ belong to the star of a same vertex k and $e(\nu)$ is next to $e(\nu-1)$ in $H(k)$ counter-clockwise.
Hence $\text{left}(\nu-1) = \text{right}(\nu)$.
For the induction hypothesis $Q(\nu-1) = \text{left}(\nu-1)$ and by Rule 1 $Q(\nu) = Q(\nu-1)$.
Hence $Q(\nu) = Q(\nu-1) = \text{left}(\nu-1) = \text{right}(\nu)$.
For the induction hypothesis $Q(\nu-1) \in R(\nu-1)$.
Hence, $Q(\nu) = \text{right}(\nu) \in R(\nu-1)$.
So, the right property holds for the forward move ν.

$Q(\nu-1) = Q(\nu)$



Case IV: move $\nu - 1$ is backward and move ν is backward

In this case $e(\nu - 1)$ and $e(\nu)$ belong to the star of a same vertex k, $e(\nu)$ is the forward edge entering k and it is next to $e(\nu - 1)$ in $H(k)$ counter-clockwise.
Hence $\text{left}(\nu - 1) = \text{left}(\nu)$.
For the induction hypothesis $Q(\nu - 1) = \text{left}(\nu - 1) \in R(\nu - 1)$.
By Rule 1, $Q(\nu) = Q(\nu - 1)$ and hence $Q(\nu) = \text{left}(\nu)$.
By Rule 2, $R(\nu) = R(\nu - 1)$ and hence $Q(\nu) \in R(\nu)$.
Hence, $Q(\nu) = \text{left}(\nu) \in \mathcal{R}(\nu)$. So, the left property holds for the backward move ν. $\square$



$Q(\nu - 1) = Q(\nu)$

Completeness of the enumeration

Observation. Since forward edges do not form circuits, every region must have at least one backtrack edge along its contour.

Theorem.
All regions are enumerated.

Proof. If a region is on the left side of a backtrack edge, then it is enumerated when the backtrack edge is traversed for the first time. If a region is on the right hand side of a backtrack edge e, then it must also be on the left side of another backtrack edge e' traversed for the first time before e, because, for Left-Right Property, when e is traversed for the first time the region right(e) must have been already enumerated.
Since all backtrack edges are traversed, all regions are guaranteed to be enumerated. □

Non-redundant enumeration

Lemma.
The number of backtrack edges is equal to the number of internal regions of the multi-graph.

Proof. Indicate by E^{fw} the number of forward edges, by E^{bt} the number of backtrack edges, by $|R|$ the number of regions and by v the number of vertices of a directed planar multi-graph.
By Euler formula, $E + 2 = v + |R|$, where $E = E^{fw} + E^{bt}$.
Since forward edges form a spanning arborescence, $E^{fw} = v - 1$.
Therefore $E^{bt} = |R| - 1$.
Since R includes the external region which is unique, then $|R| - 1$ is the number of internal regions of the multi-graph. $\square$

Non-redundant enumeration

Theorem.
Each region is enumerated once.

Proof. By Rule 2, no more than one internal region can be enumerated for each backtrack edge.
By the Lemma above, there are as many backtrack edges as the number of internal regions.
Therefore, no internal region can be enumerated more than once.
By the previous theorem, all internal regions are enumerated at least once.
Then, all internal regions are enumerated exactly once. □

Complexity of Components

Sorting the n circumferences with SortCircles takes $O(n \log n)$.

Initializing μ takes $O(1)$ for each vertex, i.e. $O(n^2)$.

The while loop is executed $O(n)$ times, since at least one circumference is deleted from N at each iteration: all $O(1)$ operations in the loop yield an overall $O(n)$ contribution.

The total contribution of the executions of Rightmost is $O(n^2)$, since a sorted list of cardinality $O(n)$, produced by SortCircles must be scanned $O(n)$ times.

Initializing Q takes $O(n)$ for each connected component ϕ; the overall contribution is $O(n^2)$.

The overall contribution of all $O(1)$ flip operations is bounded by twice the number of edges in the multi-graph, i.e. $O(n^2)$.

The overall contribution of deletions from N is $O(n)$, each deletion takes $O(1)$, when N is implemented as a binary array.

Complexity of ScanStar

FindEdge is executed once for each vertex. Therefore the total number of steps required by FindEdge is bounded by the total degree of the multi-graphs $O(\sum_{k=1}^{v} |H(k)|)$, i.e. $O(n^2)$.

The total number of iterations of the for loop in all executions of ScanStar is also bounded by the total degree of the multi-graphs, i.e. $O(n^2)$ and the loop includes only $O(1)$ operations.

The number of calls to ScanStar is v, i.e. $O(n^2)$.

Therefore the overall worst-case time complexity of Step 4 is $O(n^2)$.

The overall complexity of the region enumeration algorithm is $O(n^2 \log n)$. The bottlenecks are the three sorting procedures in Step 1.2, Step 2.2 and Step 3.2.

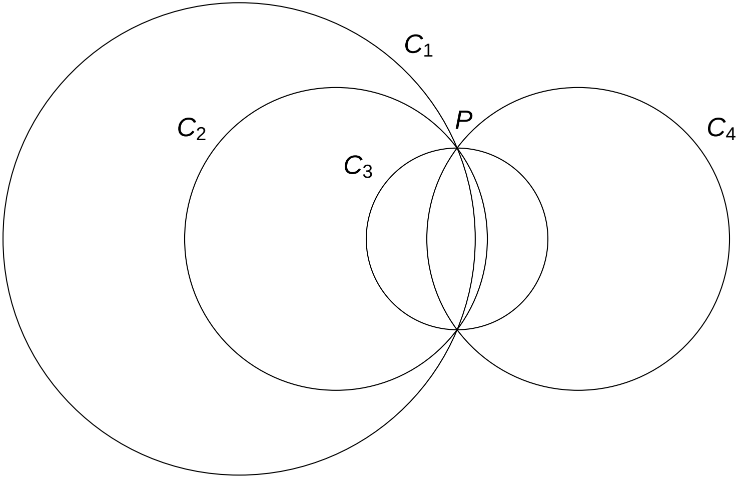
Implementation

No implementation exists so far.

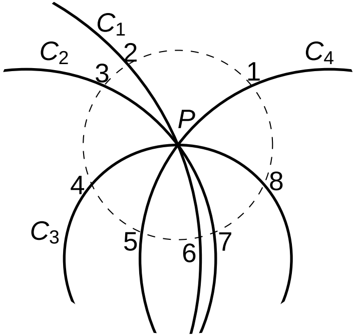
Some ideas:

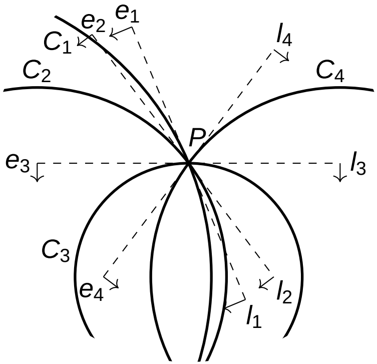
- evaluate the regions in the reverse order, i.e. from the innermost to the outermost regions: the order in which regions are enumerated in each component corresponds to a closed walk in the dual multi-graph;
- early termination of the single-source optimal location algorithm, when the current point leaves the region to be evaluated;
- skip some regions, by computing a lower bound based on centers, radii and weights, without running the single-source optimal location algorithm.

Example 1

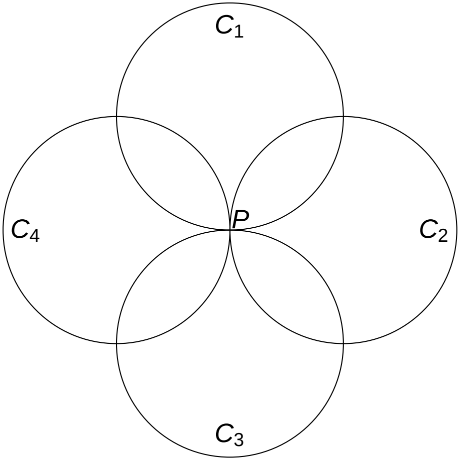


Example 1

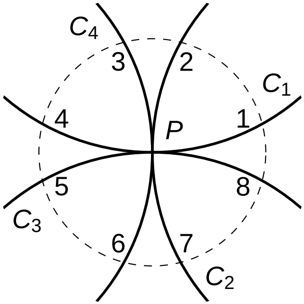


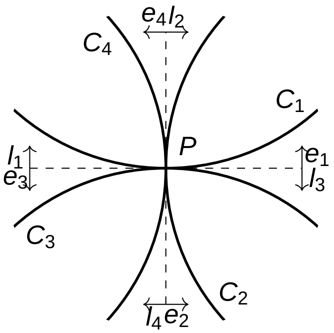


Example 2

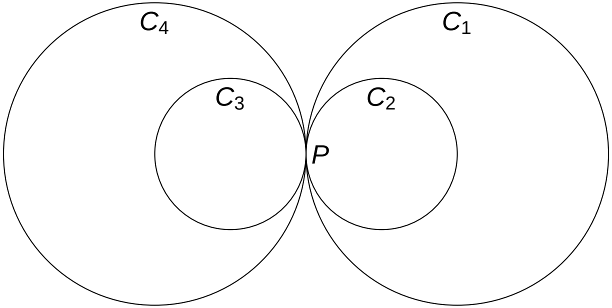


Example 2

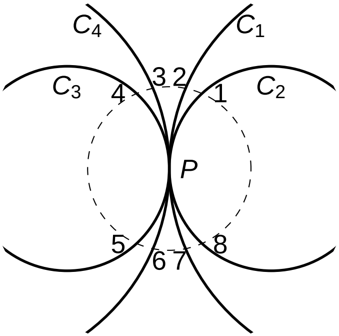


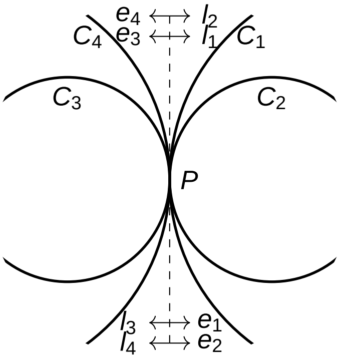


Example 3

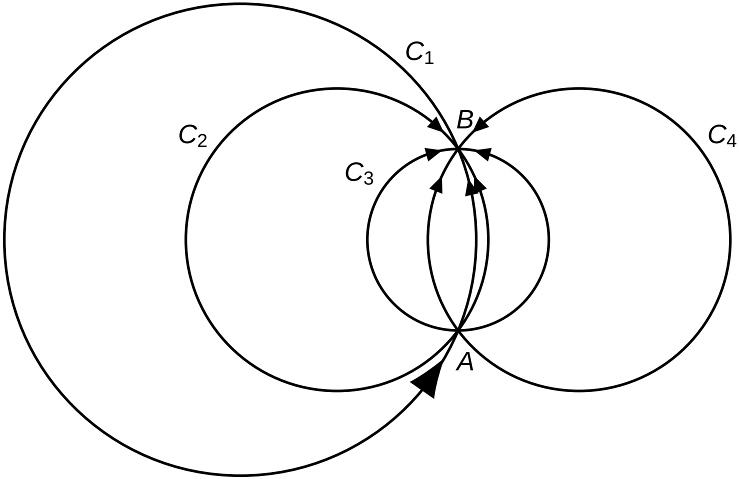


Example 3

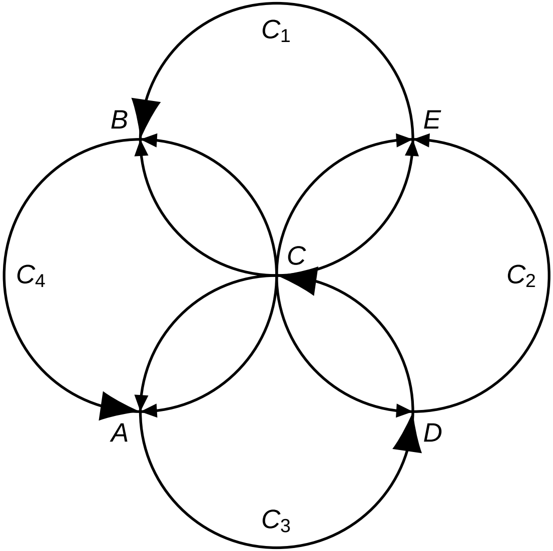

C_4
C_1
C_3
C_2
3
2
4
1
P
5
8
6
7


$e_4 \longleftrightarrow l_2$
$e_3 \longleftrightarrow l_1$
C_4
C_1
C_3
C_2
P
$l_3 \longleftrightarrow e_1$
$l_4 \longleftrightarrow e_2$

Example 1 revisited



Example 2 revisited



Example 3 revisited