Approximate kNN-Graph Construction in Secondo-Pregel

Ralf Hartmut Güting, September 2020.

This script is based on ideas from the paper:

Wei Dong, Moses Charikar, Kai Li: Efficient k-nearest neighbor graph construction for generic similarity measures. WWW 2011: 577-586

1 Development on the Master

1.1 Database and Workers

In this example, we use the database germany defined in the module secondo-data at secondo-data/Databases/germany

We use the *Stadt* relation with 1456 tuples and compute an approximate nearest neighbor graph.

```
# restore database germany from '/home/ralf/secondo-data/Databases/germany'

open database germany

restore Workers from Workers6Pregel
```

1.2 Spatial Partitioning

We partition cities spatially, imposing a 2 x 3 grid. The grid coordinates are read from the Hoese viewer after displaying the *Stadt* relation.

```
let Box = [const rect value (5.4 15.5 47.0 55.0)]

let grid = [const cellgrid2d value (5.4 47.0 5.0 2.7 2)]
```

The grid partitioning can be visualized by the following queries:

```
query intstream(1, 6) transformstream extend[Cell:
  gridcell2rect(.Elem, 5.4, 47.0, 5.0, 2.7, 2)] consume

query Stadt
```

The result is shown in Figure 1.

Assign partitions to cities by the *grid*:

```
let NodesP = Stadt feed projectextend[; Id: .SNr,
  Partition: cellnumber(bbox(.Ort), grid) transformstream head[1]
    extract[Elem] - 1, Value: .Ort]
  consume

let Size = NodesP feed max[Id] + 1;
query share("Size", TRUE, Workers)
```

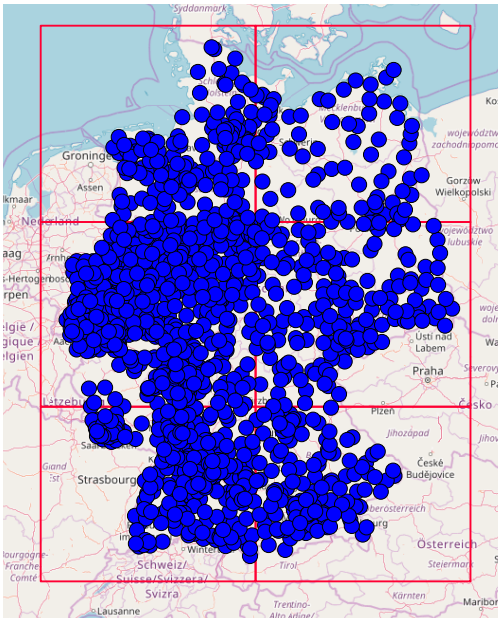

Figure 1: Partitioning of relation *Stadt*

1.3 Creating a Random Initial Graph

From each node, we create k random edges.

```
let k = 5;
query share("k", TRUE, Workers)

let wgs84 = create_geoid("WGS1984");
query share("wgs84", TRUE, Workers)

let point0 = [const point value undef];
query share("point0", TRUE, Workers)

let EdgesP = NodesP feed
  loopjoin[intstream(1, k) transformstream
    projectextend[; Tid: int2tid(randint(Size))]
    transformstream NodesP gettuples {t}]
    extend[Distance: distance(.Value, .Value_t, wgs84)]
  consume
```

1.4 Creating Main Memory Structures on the Master

We use a main memory graph of type *mgraph3* and a memory relation. The latter is indexed by a memory AVL-tree to be able to access nodes by indentifiers.

```
let G = EdgesP feed createmgraph3[Id, Id_t, Distance, Size]
```

```
let Nodes = NodesP feed extend[Active: TRUE, Iterations: 0, SumD: k * 2000000.0,
  SumDNew: k * 1800000.0] mconsume

let Nodes_Id = Nodes mcreateAVLtree[Id]
```

1.5 Defining Initial Messages

```
let InitialMessages = EdgesP feed
  projectextend[; NodeId: .Id_t, NodePartition: .Partition_t,
    Sender: .Id, SenderPartition: .Partition, Message: "m3", Value1: .Value,
    Value2: 0.0]
  consume;
query share("InitialMessages", TRUE, Workers)

let NoMessages = fun() InitialMessages feed head[0];
query share("NoMessages", TRUE, Workers)
```

Schemas are:

```
NodesP(Id: int, Partition: int, Value: point)
EdgesP(Id: int, Partition: int, Value: point, Id_t: int, Partition_t: int,
  Value_t: point, Distance: real)

Messages(NodeId: int, NodePartition: int, Sender: int, SenderPartition: int,
  Message: string, Value1: point, Value2: real)
```

1.6 The Compute Function

```
let Compute = fun (messages: stream(tuple([NodeId: int, NodePartition: int,
    Sender: int, SenderPartition: int, Message: string, Value1: point,
    Value2: real])))
  messages mconsume
  within[fun(msgs: ANY)
    msgs mfeed extract[Message]
  switch[
    "m3",
      msgs mfeed
        loopsel[fun(m: TUPLE) G mg3successors[attr(m, NodeId)]
          projectextend[; NodeId: attr(m, Sender), NodePartition: attr(m,
            SenderPartition),
            Sender: .Id_t, SenderPartition: .Partition_t,
            Message: "m5", Value1: .Value_t, Value2:
              distance(attr(m, Value1), .Value_t, wgs84)]
        ]
    ; "m5",
      msgs mfeed
        loopjoin[Nodes_Id Nodes mexactmatch[.NodeId] addid {n}]
```

```
        filter[.Active_n]
        sortby[NodeId]
        groupby[NodeId, TID_n
        ; NInserted: group feed
            projectextend[; Id: .NodeId, Partition: .NodePartition,
              Value: .Value_n, Id_t: .Sender, Partition_t: .SenderPartition,
              Value_t: .Value1, Distance: .Value2]
            filter[.Id # .Id_t]
          G mg3successors[group feed extract[NodeId], TRUE]
          concat
          sortby[Distance] rdup head[k] mg3insert[G] count]
        Nodes mupdatedirect2[TID_n; Iterations: ..Iterations + 1,
          SumDNew: G mg3successors[.NodeId] sum[Distance],
          Active: ..SumDNew < ..SumD, SumD: ..SumDNew]
        loopsel[fun(m2: TUPLE)
          G mg3successors[attr(m2, Id)]
          projectextend[; NodeId: .Id_t, NodePartition: .Partition_t,
          Sender: .Id, SenderPartition: .Partition, Message: "m3",
          Value1: .Value, Value2: 0.0]
        ]
    ; NoMessages()
  ]]

query share("Compute", TRUE, Workers)
```

The initial messages are set up so that each node sends a message *m3* to its successors in the random graph.

On receiving *m3* messages, each node determines its own edges to successors. From successor positions it computes the distance to the node from which it received the *m3* message and sends it back to that node as an *m5* message. It also sends back successor positions so that the sender node can construct an edge from itself to the successor.

On receiving a set of *m5* messages, each node determines the k edges with the smallest distance among the existing edges and those suggested by the *m5* messages. The k edges are replaced by this set.

In addition, a node observes the sum of distances to its successors. If this distance changed in this round (i.e., the set of edges changed), then the node is set to be further active, else not active. Only active nodes process *m5* messages.

In processing *m5* messages, nodes also create new *m3* messages for the next step.

Some technical comments:

- The function needs to read the stream of incoming messages twice, once to determine the type of mesages in this round (**extract**), and then to process this set of messages. The repeated reading of the stream does not work correctly in Secondo-Pregel at the moment, because read messages are deleted and will not appear again. This behaviour may be corrected in the near future.

  At the moment, the problem is circumvented by storing the stream within the function in a main memory relation, using **mconsume** and the **within** operator. In this way the argument stream can be read repeatedly.

- The **mupdatedirect2** operator allows one to update tuples in a main memory relation using attribute values from the incoming tuple stream as well as from the stored tuple. The latter attributes are accessed by the ".." notation, as usual in Secondo for functions with two arguments (such as **symmjoin**, for example). The output from the operator is the stream of updated tuples.

1.7 Experiments

1.7.1 Observe One Node

This should be run in the Javagui, to see edges.

```
let Messages = InitialMessages

query G mg3successors[1]
extend[Curve: create_sline(.Value, .Value_t)] extend[SourcePos: .Value,
TargetPos: .Value_t] remove[Value, Value_t] consume
```

In a loop (manually):

```
update Messages := Compute(Compute(Messages feed)) consume

query G mg3successors[1]
extend[Curve: create_sline(.Value, .Value_t)] extend[SourcePos: .Value,
TargetPos: .Value_t] remove[Value, Value_t] consume
```

1.7.2 Graph Construction with Termination on Master

```
let Messages = InitialMessages

let NActive = Nodes mfeed filter[.Active] count

let SumNActive = NActive

let Rounds = 0

while NActive > 0 do
{
update Messages := Compute(Compute(Messages feed)) consume
|
update NActive := Nodes mfeed filter[.Active] count
|
query NActive
|
update SumNActive := SumNActive + NActive
|
update Rounds := Rounds + 1
|
query Rounds
}
endwhile
```

SumNActive = 9904. Each active node sends for 25 distance evaluations = 247600. We have 1456 Nodes, hence 247600 / 1456 ≈ 170 distance evaluations per node. Note that this number will be similar also for larger sets of nodes.

Total number of rounds is 20.

Start at 16:29:02, finish at 16:29:58, hence about one minute. Protocol is

---

```
(resultsequence
    (
        (resultsequence
            (
                ()
                ()
                (int 1456)
                ()
                ()
                (int 1)))
        (resultsequence
            (
                ()
                ()
                (int 1456)
                ()
                ()
                (int 2)))
        (resultsequence
            (
                ()
                ()
                (int 1456)
                ()
                ()
                (int 3)))
        (resultsequence
            (
                ()
                ()
                (int 1455)
                ()
                ()
                (int 4)))
        (resultsequence
            (
                ()
                ()
                (int 1409)
                ()
                ()
```

```
        (int 5)))
(resultsequence
    (
        ()
        ()
        (int 866)
        ()
        ()
        (int 6)))
(resultsequence
    (
        ()
        ()
        (int 255)
        ()
        ()
        (int 7)))
(resultsequence
    (
        ()
        ()
        (int 57)
        ()
        ()
        (int 8)))
(resultsequence
    (
        ()
        ()
        (int 17)
        ()
        ()
        (int 9)))
(resultsequence
    (
        ()
        ()
        (int 5)
        ()
        ()
        (int 10)))
(resultsequence
    (
        ()
        ()
        (int 3)
        ()
        ()
        (int 11)))
```

```
(resultsequence
    (
        ()
        ()
        (int 2)
        ()
        ()
        (int 12)))
(resultsequence
    (
        ()
        ()
        (int 2)
        ()
        ()
        (int 13)))
(resultsequence
    (
        ()
        ()
        (int 2)
        ()
        ()
        (int 14)))
(resultsequence
    (
        ()
        ()
        (int 2)
        ()
        ()
        (int 15)))
(resultsequence
    (
        ()
        ()
        (int 2)
        ()
        ()
        (int 16)))
(resultsequence
    (
        ()
        ()
        (int 1)
        ()
        ()
        (int 17)))
(resultsequence
```

```
        (
            ()
            ()
            (int 1)
            ()
            ()
            (int 18)))
    (resultsequence
        (
            ()
            ()
            (int 1)
            ()
            ()
            (int 19)))
    (resultsequence
        (
            ()
            ()
            (int 0)
            ()
            ()
            (int 20)))))
```

---

2 Data Distribution

```
let NodesSD = NodesP feed ddistribute2["", Partition, 6, Workers]
  makeSimple[FALSE, "NodesPersistent"]

let EdgesSD = EdgesP feed ddistribute2["", Partition, 6, Workers]
  makeSimple[FALSE, "EdgesPersistent"]
```

3 Running the Algorithm in Pregel

This part can be run repeatedly, each time starting a new session.

```
query setupPregel(Workers);
```

Create main memory structures:

```
query remotePregelCommand('let G = EdgesPersistent feed
  createmgraph3[Id, Id_t, Distance, Size]')

query remotePregelCommand('let Nodes = NodesPersistent feed
  extend[Active: TRUE, Iterations: 0, SumD: k * 2000000.0,
  SumDNew: k * 1800000.0] mconsume')
```

```
query remotePregelCommand('let Nodes_Id = Nodes mcreateAVLtree[Id]')
```

Create the same structures on the master, to be able to run **setPregelFunction** which does type checking. These structures can be made small; only their types are evaluted.

```
let G = EdgesP feed head[1] createmgraph3[Id, Id_t, Distance, Size]

let Nodes = NodesP feed head[1] extend[Active: TRUE, Iterations: 0,
  SumD: k * 2000000.0, SumDNew: k * 1800000.0] mconsume

let Nodes_Id = Nodes mcreateAVLtree[Id]
```

Set function and run.

```
query setPregelFunction(Compute, NodePartition)

query InitialMessages feed initPregelMessages;

query startPregel(-1)

query getPregelMessages() consume

query pregelStatus2() consume
```

Look at results. Can be done in the Javagui.

When using the following commands for the first time in a database, one needs to replace *update* by *let*.

```
query remotePregelCommand('update NodeResults := Nodes mfeed consume');
update NodeResults := createSDArray("NodeResults", Workers) dsummarize consume

query remotePregelCommand('let GraphResult = G mg3feed consume');
let GraphResult = createSDArray("GraphResult", Workers) dsummarize consume
```

We study the number of iterations required per node:

```
Secondo => query NodeResults feed sortby[Iterations]
  groupby[Iterations; Cnt: group count] consume
Secondo ->
command
'query NodeResults feed sortby[Iterations]
  groupby[Iterations; Cnt: group count] consume'
started at: Wed Sep 30 19:48:25 2020

noMemoryOperators = 2
perOperator = 8000
Total runtime ...   Times (elapsed / cpu): 0.438087sec / 0.05sec = 8.76174


Iterations : 4
```

```
       Cnt : 1

Iterations : 5
       Cnt : 46

Iterations : 6
       Cnt : 543

Iterations : 7
       Cnt : 611

Iterations : 8
       Cnt : 198

Iterations : 9
       Cnt : 40

Iterations : 10
       Cnt : 12

Iterations : 11
       Cnt : 2

Iterations : 12
       Cnt : 1

Iterations : 17
       Cnt : 1

Iterations : 20
       Cnt : 1

Secondo => query NodeResults feed sum[Iterations]
Secondo ->
command
'query NodeResults feed sum[Iterations]'
started at: Wed Sep 30 19:49:30 2020

noMemoryOperators = 0
perOperator = 0
Total runtime ...   Times (elapsed / cpu): 0.075601sec / 0.02sec = 3.78005

9904
Secondo => query NodeResults feed avg[Iterations]
Secondo ->
command
'query NodeResults feed avg[Iterations]'
started at: Wed Sep 30 19:51:16 2020
```

```
noMemoryOperators = 0
perOperator = 0
Total runtime ...   Times (elapsed / cpu): 0.056272sec / 0.03sec = 1.87573

6.8021978022
```

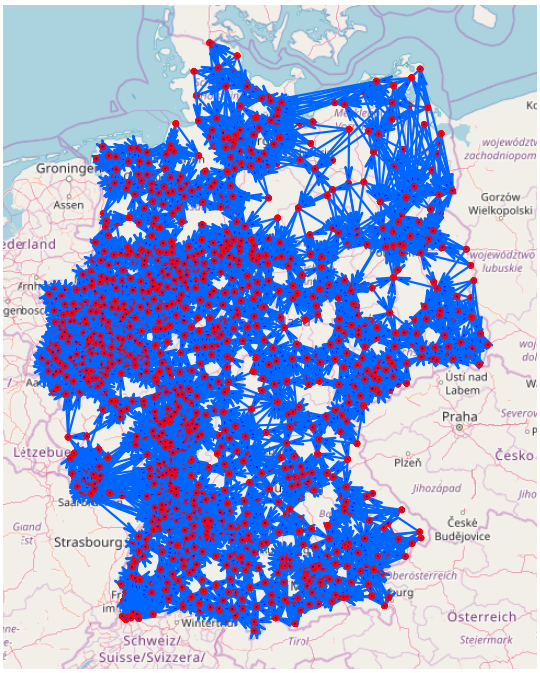

Figure 2: Approximate kNN-graph

The kNN-graph constructed in *GraphResult* is shown in Figure 2. The visualization in the Javagui is done with the query

```
query GraphResult feed extend[Curve: create_sline(.Value, .Value_t)]
  extend[SourcePos: .Value, TargetPos: .Value_t] remove[Value, Value_t] consume
```

In the Javagui, use settings:

- HoeseViewer
- File→Load categories BerlinU.cat

- Settings→Projections→OSMMercator
- Settings→Category by name
- Settings→Background→Tiled Map (OSM, ...)

To export the picture, use

- File→Export Graphic as PS

Put the resulting .eps file into the Scripts/Figures folder. The eps file may be adjusted to a desired size using the command *epsffit*, for example

```
<prompt>epsffit -c 0 0 200 250 kNNGraph.eps kNNGraphB.eps
```

providing a target bounding box.