# Mama, isses noch weit? – Einzugsbereichsanalyse im Handel

**Hans Viehmann**
**ORACLE Deutschland B.V. & Co. KG**
**NL Hamburg**

**Schlüsselworte**
Oracle Spatial and Graph, Data Warehouse, Geocoding, Netzwerk-Datenmodell, Routing

## Einleitung

Die Analyse des Einzugsbereichs von Ladengeschäften spielt eine zentrale Rolle im Handel, egal, ob es um Standortplanung, zielgruppengenaues Marketing oder andere Analyseprozesse geht, wie sie im Bereich Data Warehousing oder Big Data Analytics typischerweise vorkommen. Hierbei ist die einfache Betrachtung des Abstands als Luftlinie meist nicht ausreichend, unter anderem, weil Hindernisse wie Flüsse oder Bahntrassen nicht berücksichtigt werden. Als Entscheidungsgrundlage ist die Fahrtzeit bzw. die Länge des Fußwegs zum Ziel, die sich mit Hilfe des Straßennetzes berechnen lässt, wesentlich wichtiger. Diese konkrete Kundenanforderung vor dem Hintergrund einer Marketing-aktion sollte in einem realen Szenario mit etwa 100 Filialstandorten und ca. einer Million Kunden umgesetzt werden. In der Oracle Datenbank wurde hierzu die „Spatial and Graph“ Option genutzt, die mittels eines Netzwerk-Datenmodells in der Datenbank auf Basis von Straßennetzdaten entsprechende Analysemöglichkeiten zur Verfügung stellt.

## Vorgehensweise

Zur Zuordnung jedes Kunden zu seiner zugehörigen Filiale könnte theoretisch zu jedem Kunden die Route zu jeder der etwa 100 Filialen berechnet und die kürzeste Strecke ausgewählt werden. Bei über einer Million Kunden ist aber die Antwortzeit aufgrund der komplexen Rechnung selbst bei leistungsfähiger Hardware sehr lang. Um die Auswertung abzukürzen, ließe sich natürlich eine räumliche Vorsortierung durchführen, aber auch die wäre immer entweder ungenau oder aber, wenn man sie auf Basis von Voronoi-Diagrammen exakt durchführen wollte, wiederum aufwändig in der Vorverarbeitung. Daher wurde im Falle des vorliegenden Projekts ein anderer Ansatz gewählt:
Mit Hilfe des Straßendatenbestandes wurden im ersten Schritt die Standorte aller Kunden und Filialen aus den zugehörigen Adressen ermittelt. Zu dem Standort wurde jeweils das zugehörige Straßensegment und die Position auf dem Straßensegment bestimmt. Anschließend wurde zu jeder Filiale mittels des Java API zum Network Data Model in Oracle Spatial and Graph festgestellt, welche Straßensegmente innerhalb einer vorgegebenen Zeit erreicht werden konnten. Diese Gruppen von Straßensegmenten lassen sich dann mit einer einzigen Join-Operation mit den Kundenstandorten verknüpfen und bei geschickter Nutzung der SQL Analytics Funktionalität so sortieren, dass je Kundenstandort gerade diejenige Filiale mit der kürzesten Wegstrecke gefunden wird, selbst wenn der Kunde in zwei oder mehr Einzugsbereiche fällt.
Im folgenden ist die Vorgehensweise näher erläutert. Zunächst wird das verwendete Straßennetz kurz beschrieben, dann werden die darauf aufbauende Umsetzung von Adressen in geografische Koordinaten und die Analyse auf dem Netzwerk-Datenmodell erläutert und schließlich die Funktionsweise der zugehörigen Java-Anwendung umrissen.

## Straßendatenbestand in Oracle Spatial and Graph

Grundlage für die Analyse ist die Abbildung des relevanten Straßennetzes in den entsprechenden Datenstrukturen von Oracle Spatial and Graph. Im vorliegenden Projekt wurden hierfür fertig aufbereitete Referenzdatenbestände von HERE (ehem. Nokia) eingesetzt, die als Transportable

Tablespaces direkt in die Datenbank eingebunden wurden und die dann sowohl für die Umsetzung von Adressen in geografische Koordinaten (Geocoding), als auch für die Routenberechnung zur Verfügung standen. HERE stellt entsprechende Datensätze für alle gängigen Länder weltweit zur Verfügung. Sie umfassen die Konnektivität und Geometrie des Straßennetzes, zahlreiche Attribute zum Geocoding, Randbedingungen zum Routing (Höchstgeschwindigkeiten, Beschränkungen für LKW, traffic patterns, ...), aber auch weitere geografische Daten, die vor allem für die Visualisierung nützlich sind (Wasser- und Grünflächen, usw.). Ähnliche Datenbestände lassen sich auch von anderen Anbietern wie Tomtom oder ADCI beschaffen. Das Datenmodell ist vollständig offen und dokumentiert, sodass auf jedes einzelne Element, wie etwa jedes Straßensegment samt seiner Attribute, zugegriffen werden kann. Auch eigene Datenbestände oder Änderungen am Referenzdatenbestand können eingepflegt werden, um etwa das Straßennetz um Strecken auf dem eigenen Firmengelände zu ergänzen. Für Demozwecke steht ein Test-Datensatz von HERE mit dem Straßennetz von San Francisco auf dem Oracle Technology Network bereit, zu dem auch ein Tutorial verfügbar ist (s. Abb. 1).

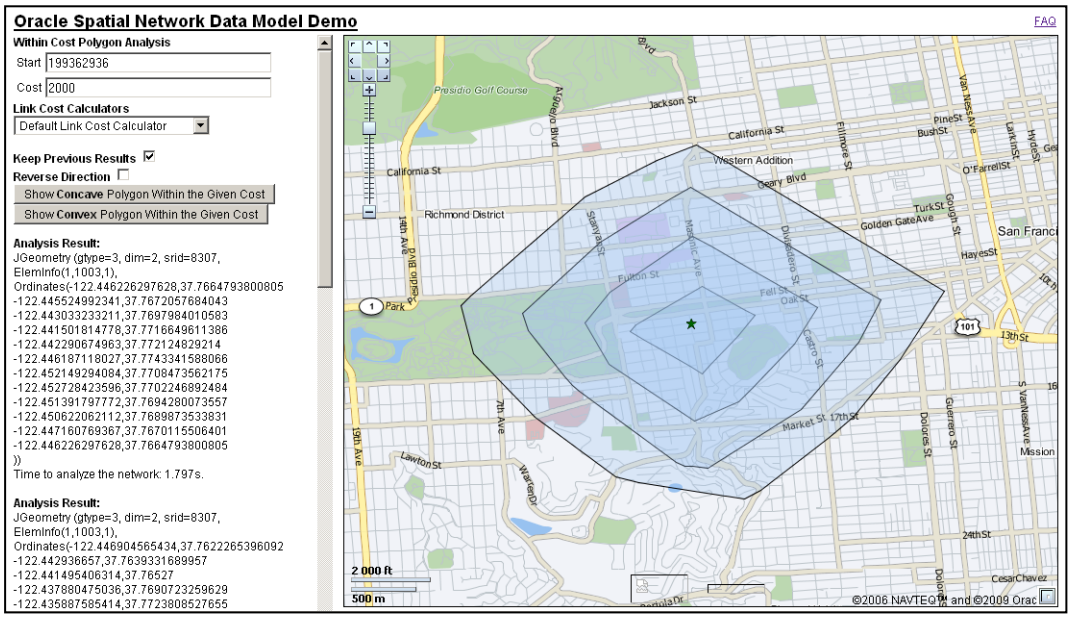


*Abb. 1: Einzugsbereichsanalyse auf dem HERE Demo-Datenbestand von San Francisco (NDM Tutorial)*

## Geocoding

Die Geocoding Engine in der Datenbank dient zur Umsetzung entweder von strukturierten oder von unstrukturierten Adressen in eine Position auf dem Referenz-Straßennetz, oder umgekehrt zur Bestimmung der nächstgelegenen Adresse zu gegebenen geografischen Koordinaten. Häufig wird dazu die Position der Hausadresse entlang des zugehörigen Straßenabschnitts durch Interpolation ermittelt. Zunehmend werden aber auch umfangreichere Datenbestände genutzt, die zu jeder Hausnummer die Gebäudekoordinaten enthalten und entsprechend die exakte Position liefern. Ist die Adresse unvollständig oder nicht völlig korrekt, versucht der Geocoder die bestmögliche Lösung zu finden und liefert ggf. eine Straße mit ähnlichem Namen, den Mittelpunkt des PLZ-Gebiets oder den Ortsmittelpunkt gemeinsam mit einem Matchcode, der darauf schließen lässt, welche Teile der Adresse nicht erkannt wurden. Neben Koordinaten und Matchcode enthält das Ergebnis auch die bereinigte Adresse, die Nummer des Straßensegments (EDGE_ID) samt Position der Adresse entlang des Segments als Zahl zwischen Null und Eins, Straßenseite, Postleitzahl, Gemeinde und Bundesland. Eine unstrukturierte Adresse aus einzelnen Keywords wird beispielsweise in der folgenden Abfrage verwendet:

```
select sdo_gcdr.geocode(
  'ODF_EU_Q312',          -- Schema des Referenzdatenbestands
  sdo_keywordarray('Riesstr. 25','München'),
  'DE',                   -- Land
  'DEFAULT'
) as geocode from dual;
```

und liefert als Ergebnis die geografischen Koordinaten (11.573° Ost, 48.180° Nord), die Nummer des Straßenabschnitts (52901891) und die Position darauf (0.5, d.h. auf halber Strecke zwischen Anfang und Ende) samt Straßenseite (L):

```
GEOCODE
--------------------------------------------------------------------------
SDO_GEO_ADDR(0, SDO_KEYWORDARRAY(), NULL, 'Riesstrasse', NULL, NULL,
'München', 'München', 'BAYERN', 'DE', NULL, NULL, NULL, NULL, '25',
'RIES', 'STRASSE', 'F', 'F', NULL, NULL, 'L', .5, 52901891, '????#EN
UT?B281CP?', 1, 'DEFAULT', 11.536734, 48.1800773, '???10101010??401?')
```

Als Schnittstelle steht entweder eine PL/SQL API in der Datenbank oder eine Web Service API zur Verfügung, die in einem J2EE Container läuft. Für Batch-Verarbeitung ist die PL/SQL-Schnittstelle insbsondere dann gut geeignet, wenn große Datenmengen parallel verarbeitet werden sollen. Hierfür lassen sich sehr elegant Pipelined Table Functions nutzen. Man verwendet dafür einen Cursor, der die Adressen aus einer Tabelle auswählt und der an eine entsprechend definierte Parallel Pipelined Table Function übergeben wird. Ergebnis ist eine nicht-persistente Tabelle („collection of rows“), aus der mittels TABLE() selektiert werden kann:

```
SELECT /*+ parallel (16) */
       a.id customer_id, a.longitude, a.latitude,
       a.edgeid link_id, a.percent percentage
FROM TABLE(geocode_utils.geocode_parsed(
           CURSOR(SELECT in_customer_id,
                    in_housenumber, in_streetname, in_city, in_state, in_zip
                  FROM customer_addresses),
     'HERE_SF')) a;
```

Durch die Verkettung des Select Statements im Cursor mit der Funktion `geocode_parsed` zu einer Pipeline werden die Funktionsaufrufe jeweils in unterschiedlichen Prozessen ausgeführt. Entsprechend verteilt sich die Last auf die vorgegebene Anzahl von Datenbank-Kernen. In einem Test mit amerikanischen Adressen auf einem Exadata X4-2 Half Rack konnten auf den vorhandenen 96 cores mit diesem Mechanismus über 77000 Adressen in gut 3.3s verarbeitet werden. Das entspricht immerhin über 23000 Adressen je Sekunde. Die Vorgehensweise skaliert ebenfalls sehr gut auf konventioneller Hardware.
Für das vorliegende Projekt wurden auf diesem Wege zu allen Kundenadressen und allen Filialstandorten jeweils das zugehörige Straßensegment (EDGE_ID) und die Position darauf (PERCENT) bestimmt.

**Netzwerk-Datenmodell**
Für die Analyse von Netzwerken (Graphen), wie sie beispielsweise bei Versorgungsunternehmen oder eben in Straßennetzen vorkommen, bietet die Oracle Spatial and Graph Option ein entsprechendes Datenmodell an, für das eine umfangreiche Palette von Analysefunktionen vorhanden ist. Das Modell beruht darauf, dass alle Knoten im Netzwerk über Kanten miteinander verbunden sind. Jede Kante besitzt einen Knoten am Anfang und einen Knoten am Ende und kann mit Kosten versehen sein, die zum Beispiel die Durchschnittsgeschwindigkeit wiedergeben. Sie können unidirektional (gerichteter

Graph) oder bidirektional (ungerichteter Graph) definiert sein und können wahlweise auch eine Geometrie besitzen, die für die grafische Darstellung verwendet werden kann. Letztere ist für die Berechnungen aber nicht notwendig.

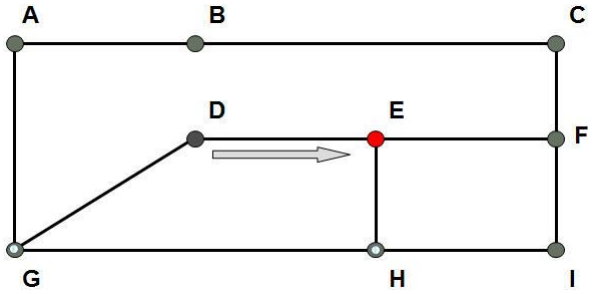


*Abb. 2:Ungerichteter Graph (nächstgelegener Knoten zu D ist E, obgleich B räumlich näher liegt)*

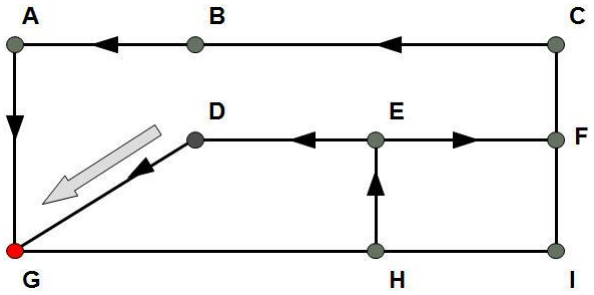


*Abb. 3:Gerichteter Graph (nächstgelegener Knoten zu D ist G)*

Zu dem Netzwerk-Datenmodell gibt es ein zugehöriges Java Interface dessen `NetworkAnalyst` Klasse (`oracle.spatial.network.lod.NetworkAnalyst`) folgende Funktionen umfasst:

- Shortest Path: Finde zu einem gegebenen Anfangs- und Endpunkte die kürzeste Strecke entlang des Netzwerks
- Reachability: Finde zu einem gegebenen Anfangspunkt alle Knoten, die erreicht werden können, oder umgekehrt: Finde alle Knoten, von denen aus ein gegebener Endpunkt erreicht werden kann
- Within-cost Analysis: Finde zu einem gegebenen Endpunkt und vorgegebenen Kosten alle Knoten, die das Ziel erreichen können, ohne das Kostenlimit zu überschreiten
- Nearest-neighbour Analysis: Finde zu einem gegebenen Endpunkt und einer vorgegebenen Anzahl n die n nächstgelegenen Knoten und die Kosten, um sie zu erreichen
- Traveling Salesman Analysis: Finde zu einer Liste von Punkten die günstigste Strecke, die alle Punkte verbindet
- Dynamic Data Input: Dient zur Erzeugung eines Network Update Objekts, um das bestehende Netzwerk zu verändern
- User-defined Link and Node Cost Calculators: Dienen zur Definition einer Kostenfunktion für Knoten bzw. Kanten

Die Berechnungen erfolgen im Hauptspeicher auf der Anwendungsseite. Dafür ist es aber nicht notwendig, das gesamte Netzwerk im Memory vorzuhalten. Große Netzwerke können hierarchisch gruppiert (etwa in Autobahnen, Bundesstraßen, usw.) und außerdem räumlich in Netzwerk-Partitionen aufgeteilt werden. Die Partitionen werden in einem Cache verwaltet und bei Bedarf automatisch geladen bzw. nachgeladen (das „load-on-demand“ API existiert seit Oracle 11gR1). Die

Partitionen werden im Voraus berechnet und können binär gespeichert werden, sodass sie von der Anwendung ohne weitere Prozessierung in den Hauptspeicher kopiert werden können.

**Routing-Anwendung**
Die Programmlogik der Routenberechnung für das Projekt befand sich in einer einfachen Java-Anwendung, die auf dem oben beschriebenen Java API aufsetzt. Im wesentlichen wird die erforderliche Konfiguration geladen, eine `NetworkAnalyst` Instanz erzeugt und dann die eigentliche Analyse als Reverse Network Buffer je Filiale unter Verwendung von EDGE_ID (=LinkID) und PERCENT(=percentage) auf dem Netzwerk durchgeführt:

```
// Erzeugt reverse network buffer, cost ist Zeitlimit in Sekunden
 PointOnNet   startPoint  = new PointOnNet(startLinkId, percentage);
 PointOnNet[] startPoints = {startPoint};
 NetworkBuffer buffer = analyst.reachingNetworkBuffer(startPoints, cost,
null);
```

Die Liste aller Straßensegmente, die sich im Einzugsbereich der jeweiligen Filiale befinden, wurden in einer Tabelle abgelegt, wo sie für die weitere Auswertung oder andere Analysen bereitstanden.
Für die Analyse war es vorher noch notwendig, eine Kostenfunktion zu definieren, die statt der Länge der Strecke (default) die Fahrtzeit zu Grunde legt. Hierzu wird neben der Länge des Links die (Höchst-)Geschwindigkeit genutzt, die je Link als Attribut verfügbar ist:

```
public class LinkTravelTimeCalculator implements LinkCostCalculator {
       int [] defaultUserDataCategories =
              {UserDataMetadata.DEFAULT_USER_DATA_CATEGORY};
       public LinkTravelTimeCalculator () {}
       public double getLinkCost(LODAnalysisInfo analysisInfo)  {
              LogicalLink link = analysisInfo.getNextLink();
                                  // speed in Meter/Sekunde
              double speed =
                 ((Double)link.getUserData(0).get
                           (RouterPartitionBlobTranslator11gR2.
                            USER_DATA_INDEX_SPEED_LIMIT)).doubleValue();
       return (link.getCost()/speed);      // distance/speed ist Fahrtzeit
}
```

Aus den geocodierten Kundenadressen und der Liste der jeweiligen Straßensegmente im Einzugsbereich einer jeden Filiale lässt sich dann in einem einzigen SQL Statement zu jedem Kunden die nächstgelegene Filiale berechnen:

```
CREATE TABLE results as
WITH
  part1 AS (SELECT c.customer_id c_customer_id,
                   c.link_id     c_link_id,
                   c.percentage  c_percentage,
                   b.buffer_id   b_buffer_id,
                   CASE WHEN b.link_id < 0 THEN -b.link_id ELSE b.link_id
END as b_link_id,
                   CASE WHEN b.link_id < 0 THEN 1-start_percentage ELSE
start_percentage END as b_start_percentage,
                   CASE WHEN b.link_id < 0 THEN 1-end_percentage ELSE
end_percentage END as b_end_percentage,
                   b.start_cost       b_start_cost,
                   b.end_cost         b_end_cost
```

```
            FROM customers c,
                 sf_nbl$ b
            WHERE c.link_id = b.link_id OR
                  c.link_id = -b.link_id),
  part2 AS (SELECT c_customer_id,
                   b_buffer_id,
                   b_start_cost +(c_percentage-
b_start_percentage)*(b_end_cost-b_start_cost)/
                                  (b_end_percentage-b_start_percentage) cost
            FROM part1
            WHERE c_percentage BETWEEN b_end_percentage AND
b_start_percentage),
  part3 AS (SELECT c_customer_id,
                   b_buffer_id,
                   cost,
                   row_number() over (partition by c_customer_id order by
cost) rn
            FROM part2)
SELECT c_customer_id customer_id, b_buffer_id store_id, cost/60 cost_in_min
FROM part3
WHERE rn = 1;
```

**Schlussbemerkung**
Durch geschickte Wahl der Vorgehensweise konnte die ursprüngliche Fragestellung von einem n × m Problem mit jeweils einer komplexen Routenberechnung so weit vereinfacht werden, dass ohne weiteres über eine Million Kunden mit den etwa 100 Filialen verknüpft werden konnten. Die hier beschriebenen Technologien – Straßennetz in der Datenbank, Geocoding, Netzwerk-Datenmodell samt Java API, usw. – sind universell einsetzbar und bieten viel Potenzial zur räumlichen Analyse sowohl im Umfeld Data Warehousing, als auch für operationale Systeme.
Die hier vorgestellte Lösung steht als Demo-Umgebung in Form eines Virtual Box Image auf dem Oracle Technology Network zum Download bereit (http://www.oracle.com/technetwork/database/options/spatialandgraph/community/sagsummit-2014-2196705.html). Das Image auf Basis Oracle Enterprise Linux 6.4 enthält neben Datenbank 12c, SQL Developer und Application Server auch einen Demo-Datensatz mit dem Straßennetz von San Francisco, der von HERE bereitgestellt wurde. Alle Komponenten sind fertig konfiguriert und die Skripte und Java-Anwendung direkt lauffähig. Zur Visualisierung der Ergebnisse ist zusätzlich der Oracle Fusion Middleware MapViewer enthalten.

Mein besonderer Dank gilt Dan Abugov von HERE, sowie Dan Geringer und dem gesamten Team in Nashua, NH, ohne die dieser Beitrag nicht möglich gewesen wäre.

**Kontaktadresse:**
Hans Viehmann
ORACLE Deutschland B.V. & Co. KG
Kühnehöfe 5
D-22761 Hamburg

| | |
|---|---|
| Telefon: | +49 (0) 40-89091-173 |
| Fax: | +49 (0) 40-89091-250 |
| E-Mail | hans.viehmann@oracle.com |
| Internet: | www.oracle.com/goto/spatial |