Boost.Geometry R-tree
speeding up geographical computation.

Adam Wulkiewicz

FOSDEM 2020

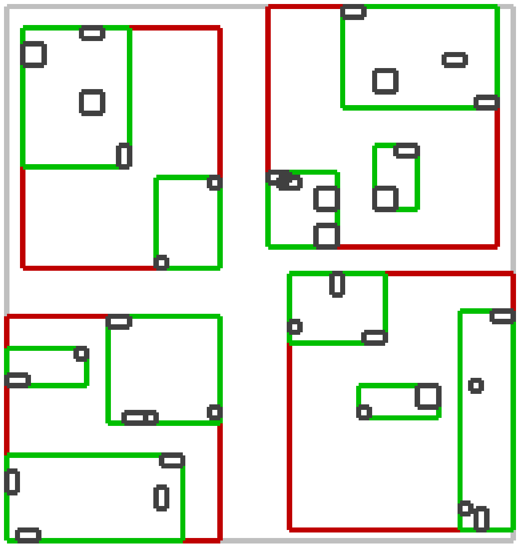
What is Boost.Geometry?

- Part of Boost C++ Libraries
- Header-only
- C++03 (conditionally C++11, C++14)
- Primitives, Algorithms, Spatial Index
- Standards: OGC SFA
- used by MySQL for GIS

Documentation: www.boost.org/libs/geometry
Mailing list: lists.boost.org/geometry
GitHub: github.com/boostorg/geometry

R-tree

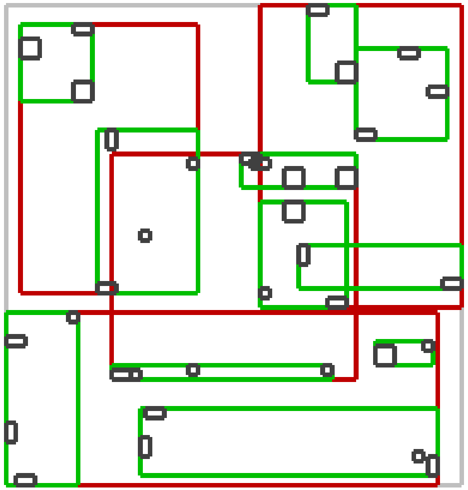
- Inspired by B-tree
- Self-balanced tree structure
- Various balancing algorithms
- Packing algorithms
- Spatial and knn searching



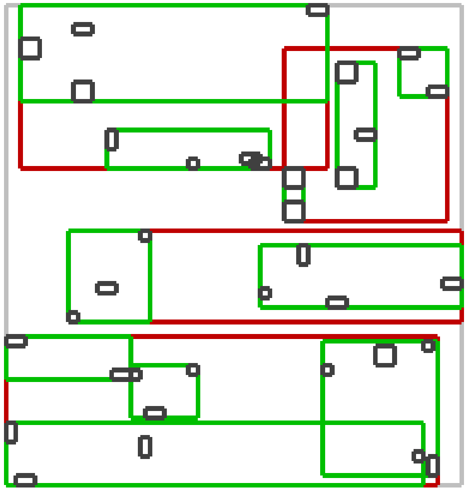
Boost.Geometry R-tree

- boost::geometry::index::rtree<…>
- User-defined element type
- Default support of Points, Boxes, Segments, std::pair, std::tuple, boost::tuple
- Three balancing algorithms and packing algorithm
- Node size defined by max/min numbers of elements
- Advanced queries
- Iterative queries

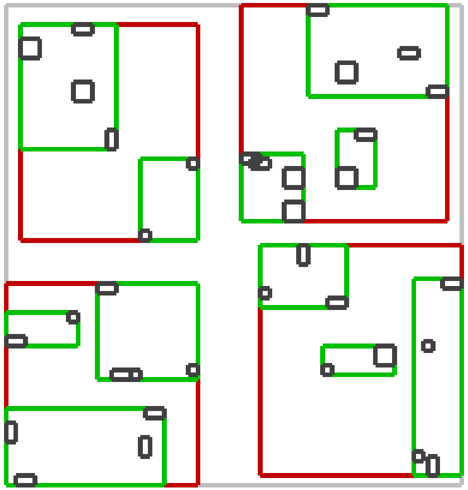
R-tree balancing and packing algorithms



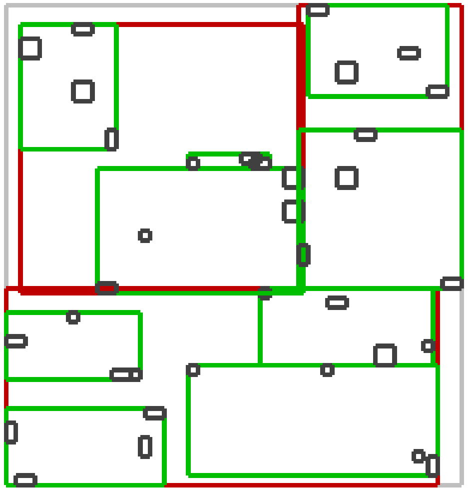
linear



quadratic



R*-tree

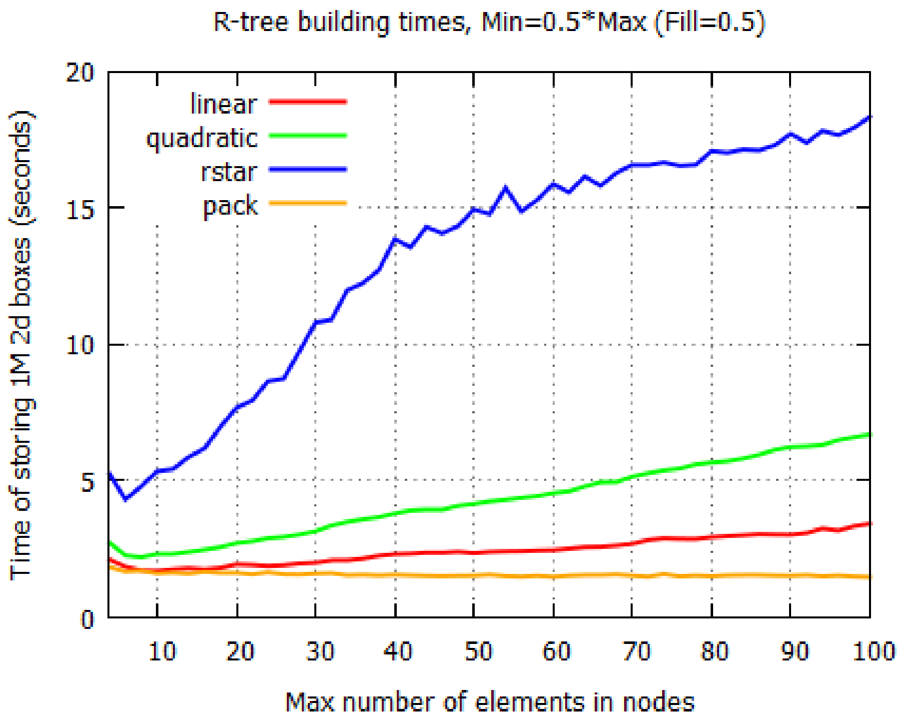


packing

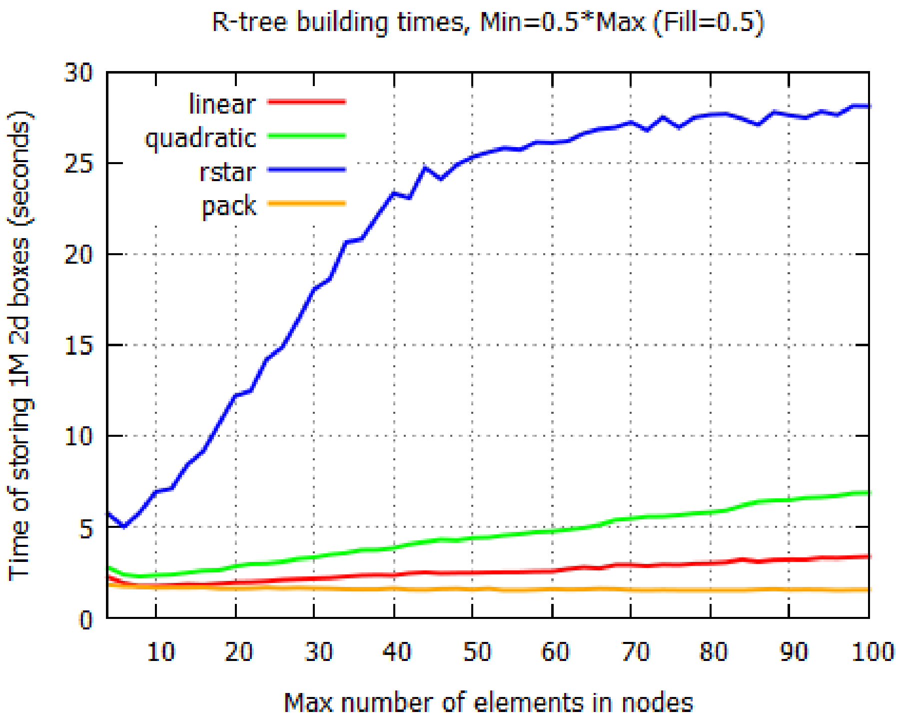
R-tree balancing and packing algorithms

	linear	quadratic	R*-tree	packing
Creation (1M boxes)	1.76s	2.47s	6.19s	0.64s
100k spatial queries	2.21s	0.51s	0.12s	0.07s
100k knn queries	6.37s	2.09s	0.64s	0.52s

R-tree creation
non-overlapping vs overlapping elements

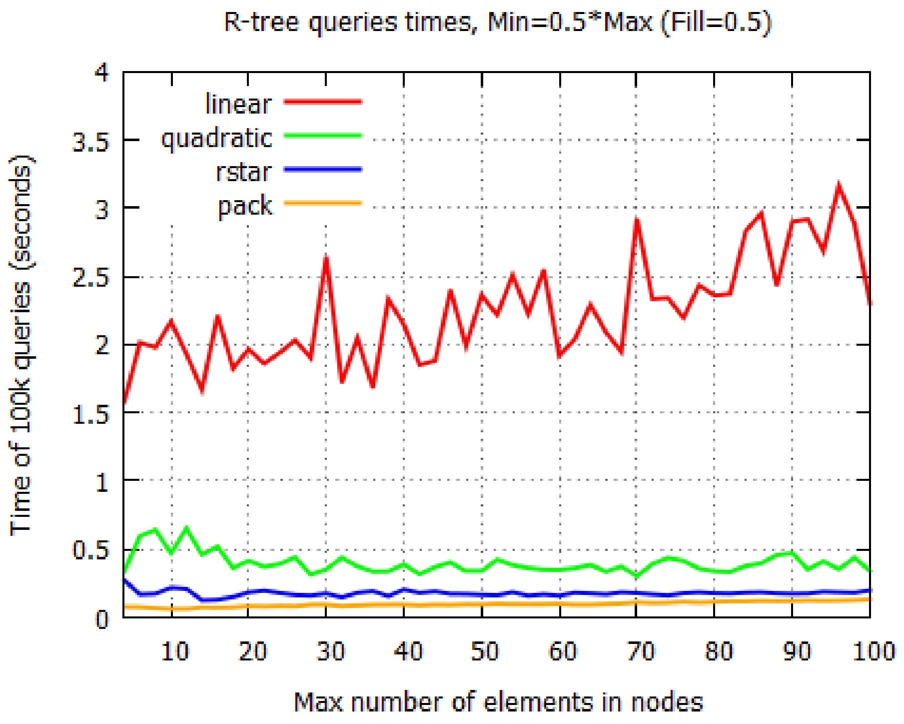


non-overlapping

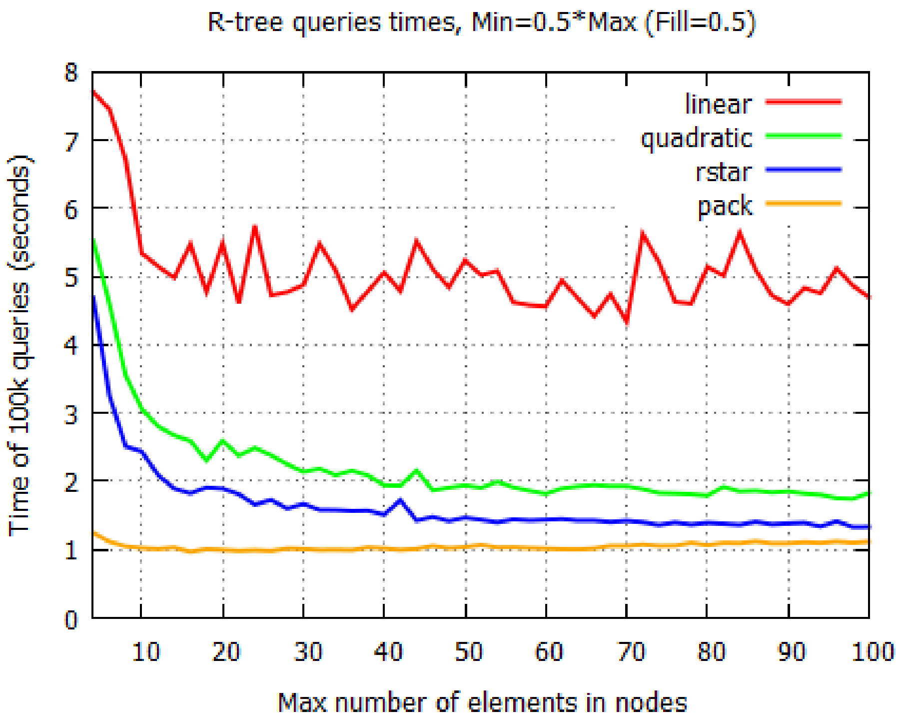


overlapping

R-tree spatial query
non-overlapping vs overlapping elements



non-overlapping



overlapping

Examples - data

- Source: www.naturalearthdata.com
- Downloads > Large scale data, 1:10m > Cultural
- Populated Places
- Airports
- Admin 0 – Countries

Examples – includes and namespaces

```cpp
#include <boost/geometry.hpp>
#include <boost/geometry/extensions/gis/io/shapefile/read.hpp> // develop
#include <boost/geometry/geometries/geometries.hpp>
#include <boost/geometry/index/rtree.hpp>

#include <boost/range/adaptors.hpp>

#include <iostream>
#include <fstream>
#include <vector>

namespace ba = boost::adaptors;
namespace bg = boost::geometry;
namespace bgi = boost::geometry::index;
namespace bgia = boost::geometry::index::adaptors;
```

Examples – basic definitions

```
using point_car = bg::model::point<double, 2,
                                   bg::cs::cartesian>;
using point_sph = bg::model::point<double, 2,
                                   bg::cs::spherical_equatorial<bg::degree>>;
using point_geo = bg::model::point<double, 2,
                                   bg::cs::geographic<bg::degree>>;

using box_geo = bg::model::box<point_geo>;
using polygon_geo = bg::model::polygon<point_geo>;
using multi_point_geo = bg::model::multi_point<point_geo>;
using multi_polygon_geo = bg::model::multi_polygon<polygon_geo>;
```

Examples – basic definitions

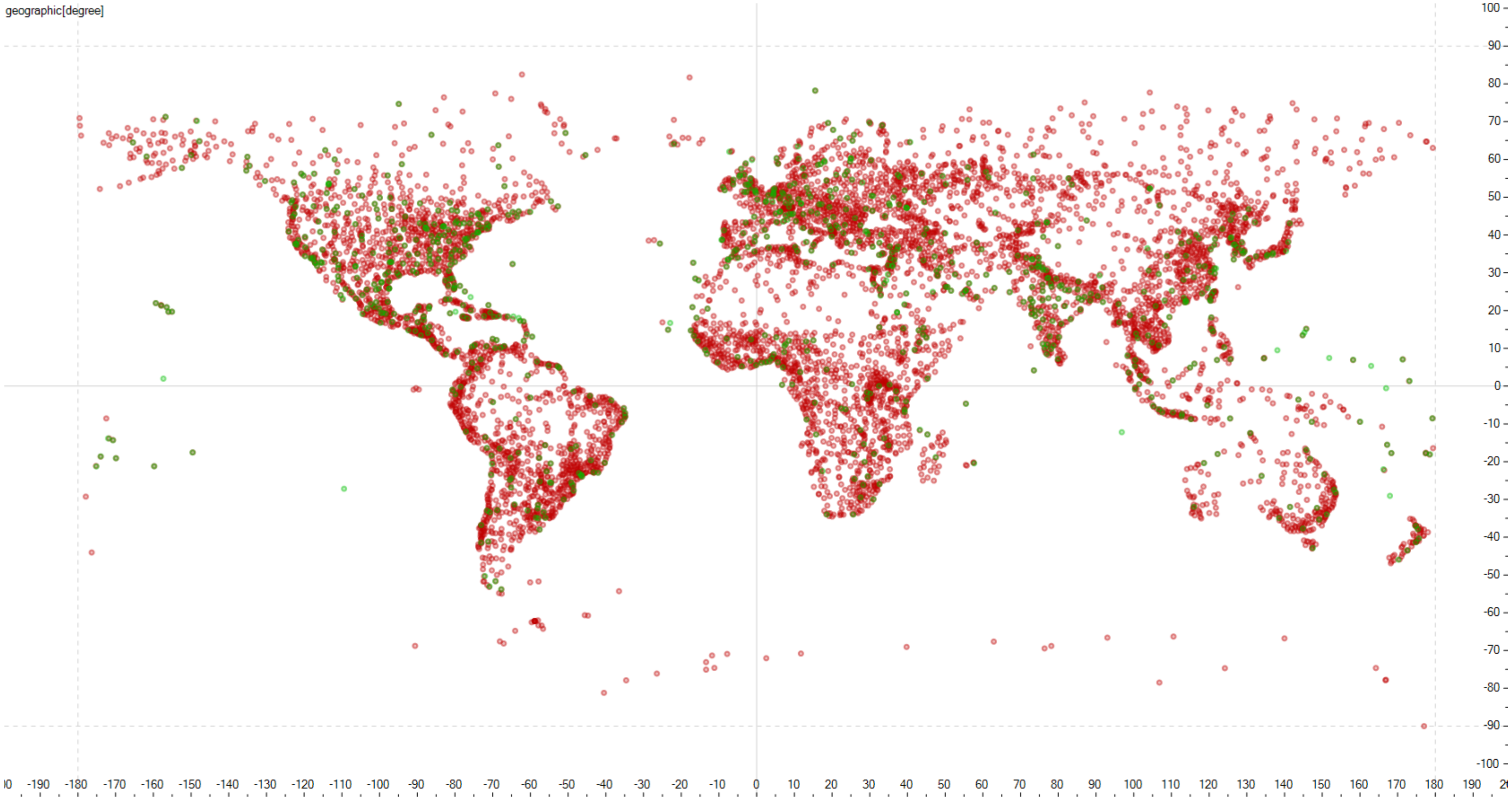
```
using point_car = bg::model::point<double, 2,
                        bg::cs::cartesian>;
using point_sph = bg::model::point<double, 2,
                        bg::cs::spherical_equatorial<bg::degree>>;
using point_geo = bg::model::point<double, 2,
                        bg::cs::geographic<bg::degree>>;

using box_geo = bg::model::box<point_geo>;
using polygon_geo = bg::model::polygon<point_geo>;
using multi_point_geo = bg::model::multi_point<point_geo>;
using multi_polygon_geo = bg::model::multi_polygon<polygon_geo>;
```

Loading data

```
multi_point_geo populated_places, airports;
std::vector<multi_polygon_geo> countries;

std::ifstream ifs_places("ne_10m_populated_places.shp",
                         std::ifstream::binary);
std::ifstream ifs_airports("ne_10m_airports.shp",
                           std::ifstream::binary);
std::ifstream ifs_countries("ne_10m_admin_0_countries.shp",
                            std::ifstream::binary);

// only with develop branch
bg::read_shapefile(ifs_places, populated_places);
bg::read_shapefile(ifs_airports, airports);
bg::read_shapefile(ifs_countries, countries);
```

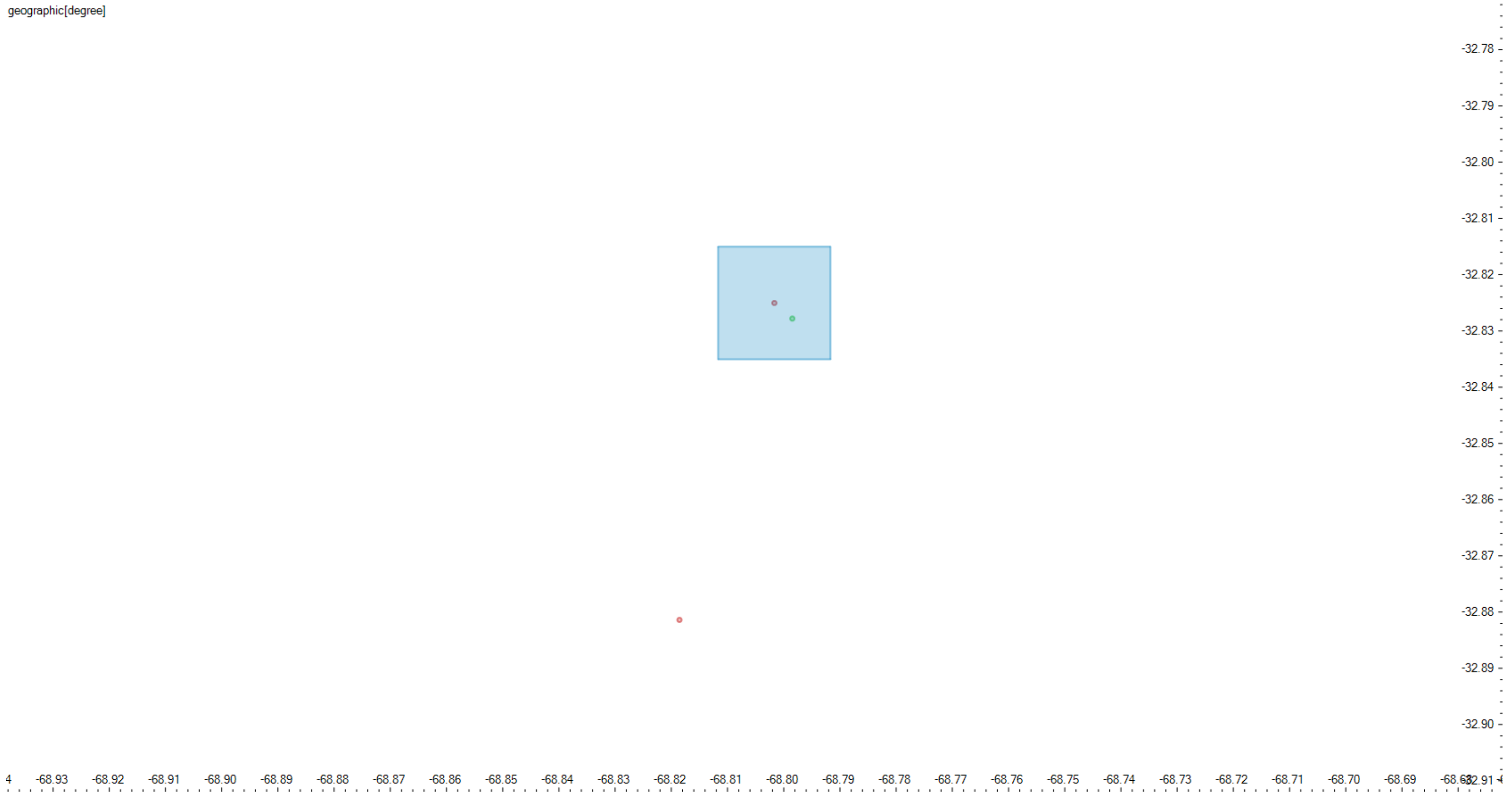
Example 1

Find populated places with airport in a small area around.



Example 1 - problem

Find populated places with airport in a small area around.



Example 1 – solution
Find populated places with airport in a small area around

```
multi_point_geo populated_places, airports;

// load data

for (auto const& p : populated_places) {
    box_geo b = small_area_around(p);
    for (auto const& a : airports) {
        if (bg::covered_by(a, b)) {
            std::cout << bg::wkt(p) << " - "
                      << bg::wkt(a) << std::endl;
        }
    }
}
```

Example 1 – solution
Find populated places with airport in a small area around

```
multi_point_geo populated_places, airports;

// load data

for (auto const& p : populated_places) {
    box_geo b = small_area_around(p);
    for (auto const& a : airports) {
        if (bg::covered_by(a, b)) {
            std::cout << bg::wkt(p) << " - "
                      << bg::wkt(a) << std::endl;
        }
    }
}
```

Example 1 – R-tree 1

Find populated places with airport in a small area around

```
bgi::rtree<point_geo, bgi::rstar<4>> rtree;
for (auto const& a : airports) {
    rtree.insert(a); // use balancing algorithm
}

for (auto const& p : populated_places) {
    box_geo b = small_area_around(p);
    std::vector<point_geo> result;
    rtree.query(bgi::covered_by(b), std::back_inserter(result));
    for (auto const& a : result) {
        std::cout << bg::wkt(p) << " - "
                  << bg::wkt(a) << std::endl;
    }
}
```

Example 1 – R-tree 1

Find populated places with airport in a small area around

```
bgi::rtree<point_geo, bgi::rstar<4>> rtree;
for (auto const& a : airports) {
    rtree.insert(a); // use balancing algorithm
}

for (auto const& p : populated_places) {
    box_geo b = small_area_around(p);
    std::vector<point_geo> result;
    rtree.query(bgi::covered_by(b), std::back_inserter(result));
    for (auto const& a : result) {
        std::cout << bg::wkt(p) << " - "
                  << bg::wkt(a) << std::endl;
    }
}
```

Example 1 – R-tree 1

Find populated places with airport in a small area around

```
bgi::rtree<point_geo, bgi::rstar<4>> rtree;
for (auto const& a : airports) {
    rtree.insert(a); // use balancing algorithm
}

for (auto const& p : populated_places) {
    box_geo b = small_area_around(p);
    std::vector<point_geo> result;
    rtree.query(bgi::covered_by(b), std::back_inserter(result));
    for (auto const& a : result) {
        std::cout << bg::wkt(p) << " - "
                  << bg::wkt(a) << std::endl;
    }
}
```

Example 1 – R-tree 1
Find populated places with airport in a small area around

```
bgi::rtree<point_geo, bgi::rstar<4>> rtree;
for (auto const& a : airports) {
    rtree.insert(a); // use balancing algorithm
}

for (auto const& p : populated_places) {
    box_geo b = small_area_around(p);
    std::vector<point_geo> result;
    rtree.query(bgi::covered_by(b), std::back_inserter(result));
    for (auto const& a : result) {
        std::cout << bg::wkt(p) << " - "
                  << bg::wkt(a) << std::endl;
    }
}
```

Example 1 – R-tree 2

Find populated places with airport in a small area around

```
// use packing algorithm
bgi::rtree<point_geo, bgi::rstar<4>> rtree(airports.begin(),
                                            airports.end());

for (auto const& p : populated_places) {
    box_geo b = small_area_around(p);
    std::for_each(rtree.qbegin(bgi::covered_by(b)),
                  rtree.qend(),
                  [&](auto const& a) {
                      std::cout << bg::wkt(p) << " - "
                                << bg::wkt(a) << std::endl;
                  });
}
```

Example 1 – R-tree 2

Find populated places with airport in a small area around

```
// use packing algorithm
bgi::rtree<point_geo, bgi::rstar<4>> rtree(airports.begin(),
                                            airports.end());

for (auto const& p : populated_places) {
    box_geo b = small_area_around(p);
    std::for_each(rtree.qbegin(bgi::covered_by(b)),
                  rtree.qend(),
                  [&](auto const& a) {
                      std::cout << bg::wkt(p) << " - "
                                << bg::wkt(a) << std::endl;
                  });
}
```

Example 1 – R-tree 2
Find populated places with airport in a small area around

```
// use packing algorithm
bgi::rtree<point_geo, bgi::rstar<4>> rtree(airports.begin(),
                                           airports.end());

for (auto const& p : populated_places) {
    box_geo b = small_area_around(p);
    std::for_each(rtree.qbegin(bgi::covered_by(b)),
                  rtree.qend(),
                  [&](auto const& a) {
                      std::cout << bg::wkt(p) << " - "
                                << bg::wkt(a) << std::endl;
                  });
}
```

Example 1 – R-tree 3

Find populated places with airport in a small area around

```
// use packing algorithm
bgi::rtree<point_geo, bgi::rstar<4>> rtree(airports);

for (auto const& p : populated_places) {
    box_geo b = small_area_around(p);
    for (auto const& a : rtree
                       | bgia::queried(bgi::covered_by(b))) {
        std::cout << bg::wkt(p) << " - "
                  << bg::wkt(a) << std::endl;
    }
}
```

Example 1 – R-tree 3

Find populated places with airport in a small area around

```
// use packing algorithm
bgi::rtree<point_geo, bgi::rstar<4>> rtree(airports);

for (auto const& p : populated_places) {
    box_geo b = small_area_around(p);
    for (auto const& a : rtree
                         | bgia::queried(bgi::covered_by(b))) {
        std::cout << bg::wkt(p) << " - "
                  << bg::wkt(a) << std::endl;
    }
}
```

Example 1 – R-tree 3

Find populated places with airport in a small area around

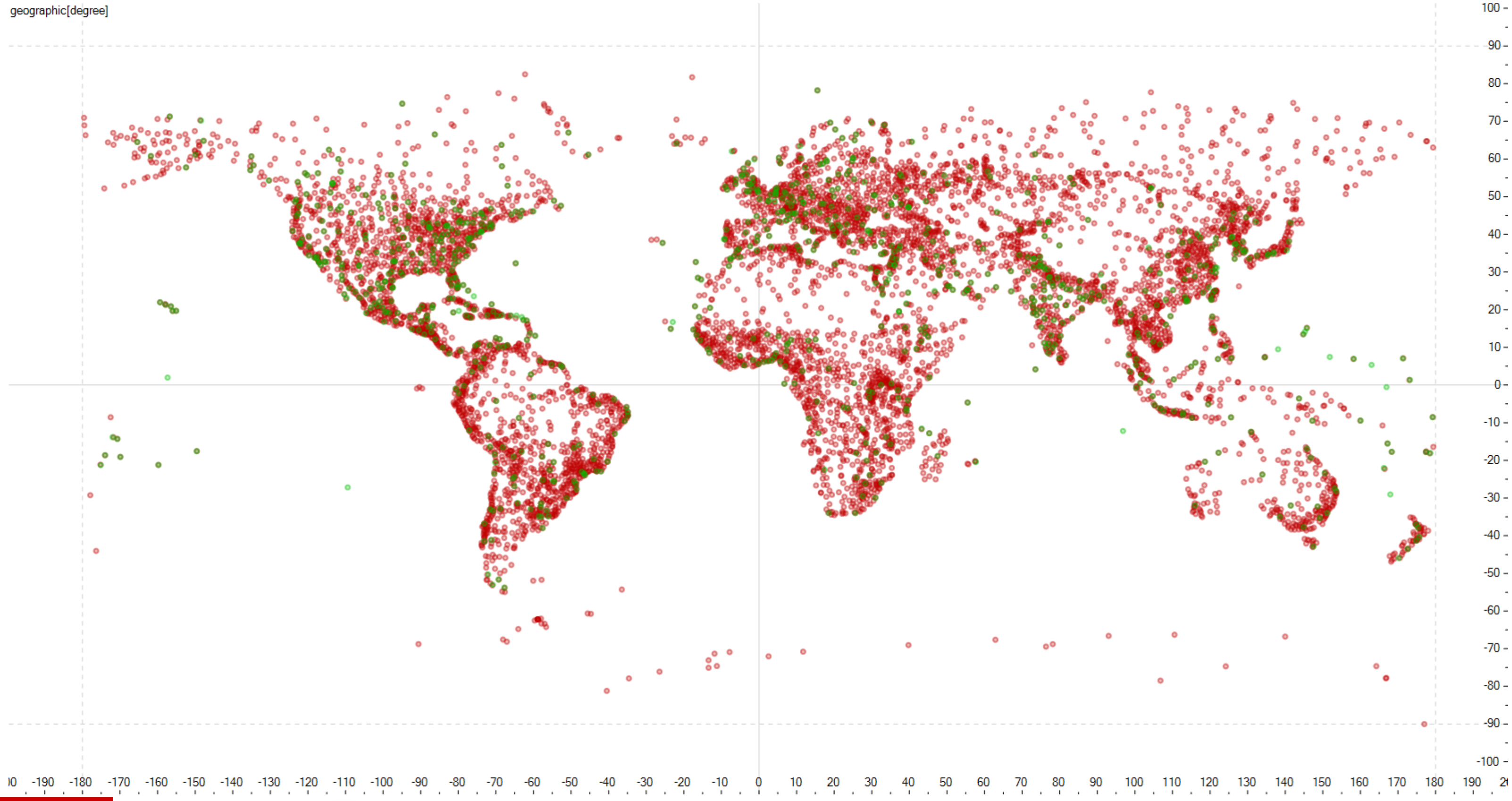
```
// use packing algorithm
bgi::rtree<point_geo, bgi::rstar<4>> rtree(airports);

for (auto const& p : populated_places) {
    box_geo b = small_area_around(p);
    for (auto const& a : rtree
                         | bgia::queried(bgi::covered_by(b))) {
        std::cout << bg::wkt(p) << " - "
                  << bg::wkt(a) << std::endl;
    }
}
```

Example 1 – results

Find populated places with airport in a small area around

```
POINT(-68.8017 -32.825) - POINT(-68.7985 -32.8278)
POINT(-74.17 40.7004) - POINT(-74.1771 40.6905)
POINT(19.9147 39.6154) - POINT(19.9148 39.6068)
POINT(-45.4166 61.1666) - POINT(-45.4164 61.1626)
POINT(122.231 -17.9618) - POINT(122.234 -17.9526)
POINT(-62.717 17.302) - POINT(-62.7142 17.3111)
POINT(146.77 -19.25) - POINT(146.771 -19.2562)
```

Example 2 – travelling salesman problem
Find shortest route that visits each airport



Example 2 – solution, greedy heuristic
Find shortest route that visits each airport

```
multi_point_geo airports;

// load data

linestring_geo route;
route.reserve(airports.size());

std::vector<bool> visited(airports.size(), false);

route.push_back(airports[0]);
visited[0] = true;
```

Example 2 – solution, greedy heuristic
Find shortest route that visits each airport

```
multi_point_geo airports;

// load data

linestring_geo route;
route.reserve(airports.size());

std::vector<bool> visited(airports.size(), false);

route.push_back(airports[0]);
visited[0] = true;
```

Example 2 – greedy heuristic
Find shortest route that visits each airport

```
while (route.size() < airports.size()) {
    double min_dist = std::numeric_limits<double>::max();
    size_t k = airports.size();
    for (size_t i = 0; i < airports.size(); ++i) {
        if (!visited[i]) {
            double dist = bg::comparable_distance(route.back(),
                                                   airports[i]);
             if (dist < min_dist) {
                min_dist = dist;
                k = i;
            }
        }
    }

    route.push_back(airports[k]);
    visited[k] = true;
}
```

Example 2 – greedy heuristic
Find shortest route that visits each airport

```
while (route.size() < airports.size()) {
    double min_dist = std::numeric_limits<double>::max();
    size_t k = airports.size();
    for (size_t i = 0; i < airports.size(); ++i) {
        if (!visited[i]) {
            double dist = bg::comparable_distance(route.back(),
                                                  airports[i]);
             if (dist < min_dist) {
                min_dist = dist;
                k = i;
            }
        }
    }

    route.push_back(airports[k]);
    visited[k] = true;
}
```

Example 2 – greedy heuristic
Find shortest route that visits each airport

```
while (route.size() < airports.size()) {
    double min_dist = std::numeric_limits<double>::max();
    size_t k = airports.size();
    for (size_t i = 0; i < airports.size(); ++i) {
        if (!visited[i]) {
            double dist = bg::comparable_distance(route.back(),
                                                  airports[i]);
             if (dist < min_dist) {
                min_dist = dist;
                k = i;
            }
        }
    }

    route.push_back(airports[k]);
    visited[k] = true;
}
```

Example 2 – greedy heuristic
Find shortest route that visits each airport

```
while (route.size() < airports.size()) {
    double min_dist = std::numeric_limits<double>::max();
    size_t k = airports.size();
    for (size_t i = 0; i < airports.size(); ++i) {
        if (!visited[i]) {
            double dist = bg::comparable_distance(route.back(),
                                                  airports[i]);
             if (dist < min_dist) {
                min_dist = dist;
                k = i;
            }
        }
    }

    route.push_back(airports[k]);
    visited[k] = true;
}
```

Example 2 – greedy heuristic
Find shortest route that visits each airport

```
while (route.size() < airports.size()) {
    double min_dist = std::numeric_limits<double>::max();
    size_t k = airports.size();
    for (size_t i = 0; i < airports.size(); ++i) {
        if (!visited[i]) {
            double dist = bg::comparable_distance(route.back(),
                                                   airports[i]);
             if (dist < min_dist) {
                min_dist = dist;
                k = i;
            }
        }
    }

    route.push_back(airports[k]);
    visited[k] = true;
}
```

Example 2 – greedy heuristic, R-tree
Find shortest route that visits each airport

```
bgi::rtree<std::pair<point_geo, size_t>, bgi::rstar<4>>
    rtree(airports | ba::indexed(size_t(0))
                   | ba::transformed([](auto const& iv) {
                         return std::make_pair(iv.value(), iv.index());
                     }));

while (route.size() < airports.size()) {
    std::pair<point_geo, size_t> result;
    rtree.query(bgi::nearest(route.back(), 1)
                && bgi::satisfies([&](auto const& v) {
                       return !visited[v.second];
                   }),
                &result);

    route.push_back(result.first);
    visited[result.second] = true;
}
```

Example 2 – greedy heuristic, R-tree
Find shortest route that visits each airport

```
bgi::rtree<std::pair<point_geo, size_t>, bgi::rstar<4>>
    rtree(airports | ba::indexed(size_t(0))
                   | ba::transformed([](auto const& iv) {
                         return std::make_pair(iv.value(), iv.index());
                     }));

while (route.size() < airports.size()) {
    std::pair<point_geo, size_t> result;
    rtree.query(bgi::nearest(route.back(), 1)
                && bgi::satisfies([&](auto const& v) {
                       return !visited[v.second];
                   }),
                &result);

    route.push_back(result.first);
    visited[result.second] = true;
}
```

Example 2 – greedy heuristic, R-tree
Find shortest route that visits each airport

```
bgi::rtree<std::pair<point_geo, size_t>, bgi::rstar<4>>
    rtree(airports | ba::indexed(size_t(0))
                   | ba::transformed([](auto const& iv) {
                         return std::make_pair(iv.value(), iv.index());
                     }));

while (route.size() < airports.size()) {
    std::pair<point_geo, size_t> result;
    rtree.query(bgi::nearest(route.back(), 1)
                && bgi::satisfies([&](auto const& v) {
                       return !visited[v.second];
                   }),
                &result);

    route.push_back(result.first);
    visited[result.second] = true;
}
```

Example 2 – greedy heuristic, R-tree
Find shortest route that visits each airport

```
bgi::rtree<std::pair<point_geo, size_t>, bgi::rstar<4>>
    rtree(airports | ba::indexed(size_t(0))
                   | ba::transformed([](auto const& iv) {
                         return std::make_pair(iv.value(), iv.index());
                     }));

while (route.size() < airports.size()) {
    std::pair<point_geo, size_t> result;
    rtree.query(bgi::nearest(route.back(), 1)
                && bgi::satisfies([&](auto const& v) {
                       return !visited[v.second];
                   }),
                &result);

    route.push_back(result.first);
    visited[result.second] = true;
}
```

Example 2 – greedy heuristic, R-tree
Find shortest route that visits each airport

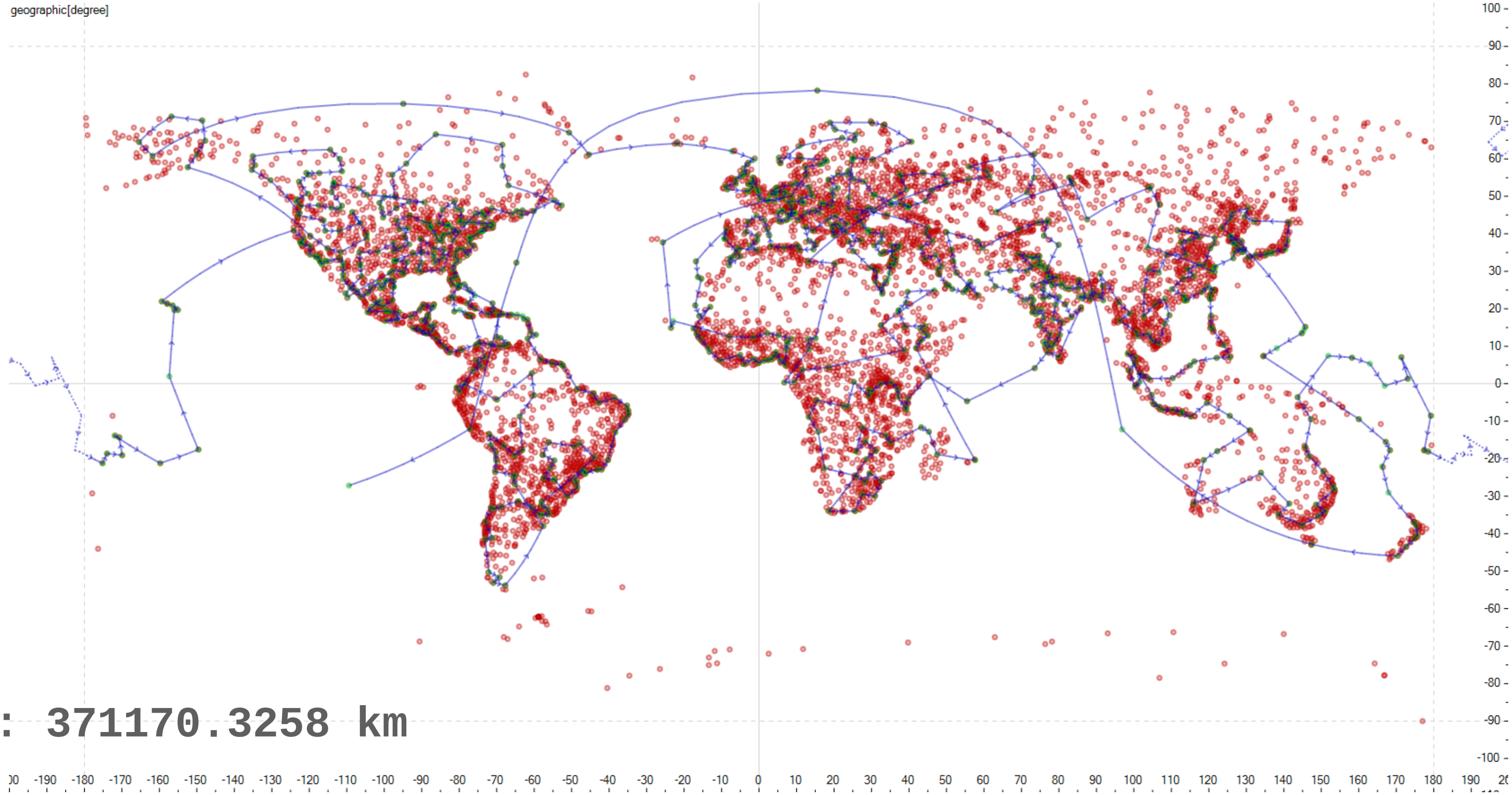
```
bgi::rtree<std::pair<point_geo, size_t>, bgi::rstar<4>>
    rtree(airports | ba::indexed(size_t(0))
                   | ba::transformed([](auto const& iv) {
                         return std::make_pair(iv.value(), iv.index());
                     }));

while (route.size() < airports.size()) {
    std::pair<point_geo, size_t> result;
    rtree.query(bgi::nearest(route.back(), 1)
                && bgi::satisfies([&](auto const& v) {
                       return !visited[v.second];
                   }),
                &result);

    route.push_back(result.first);
    visited[result.second] = true;
}
```

Example 2 – greedy heuristic, R-tree
Find shortest route that visits each airport

```
bgi::rtree<std::pair<point_geo, size_t>, bgi::rstar<4>>
    rtree(airports | ba::indexed(size_t(0))
                   | ba::transformed([](auto const& iv) {
                         return std::make_pair(iv.value(), iv.index());
                     }));

while (route.size() < airports.size()) {
    std::pair<point_geo, size_t> result;
    rtree.query(bgi::nearest(route.back(), 1)
                && bgi::satisfies([&](auto const& v) {
                       return !visited[v.second];
                   }),
                &result);

    route.push_back(result.first);
    visited[result.second] = true;
}
```

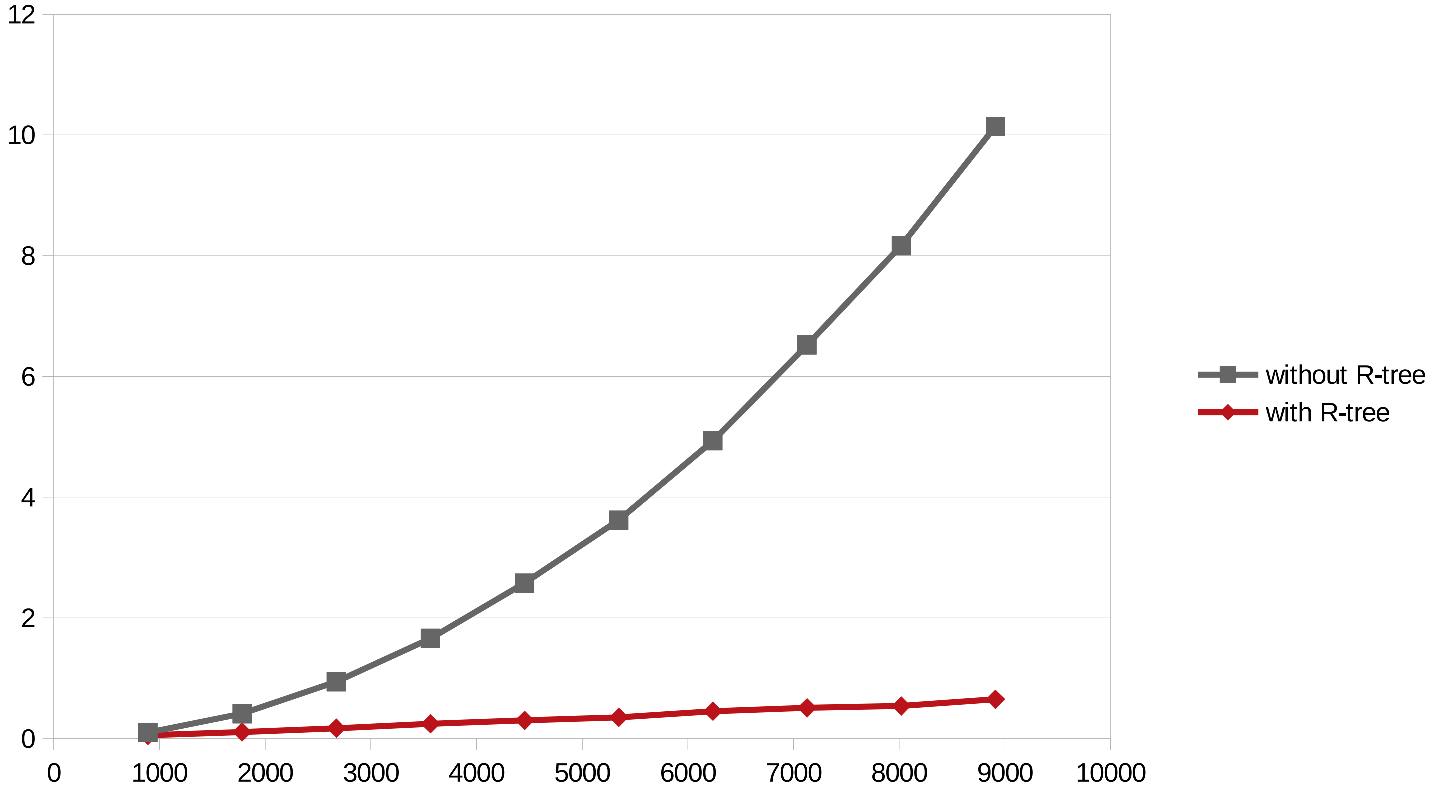
ORACLE MySQL boost GEOMETRY

Example 2 – result
Find shortest route that visits each airport



Length: 371170.3258 km

Example 2 – benchmark time(#points)
Find shortest route that visits each airport



Thanks!