Aufgabe 1: Weniger krumme Touren

Teilnahme-ID: ?????

Bearbeiter/-in dieser Aufgabe:
Lennart Protte

18. Februar 2023

Inhaltsverzeichnis

Anleitung: Trage oben in den Zeilen 8 bis 10 die Aufgabennummer, die Teilnahme-ID und die/den Bearbeiterin/Bearbeiter dieser Aufgabe mit Vor- und Nachnamen ein. Vergiss nicht, auch den Aufgabennamen anzupassen (statt „LaTeX-Dokument“)!

Dann kannst du dieses Dokument mit deiner LaTeX-Umgebung übersetzen.

Die Texte, die hier bereits stehen, geben ein paar Hinweise zur Einsendung. Du solltest sie aber in deiner Einsendung wieder entfernen!

1 Lösungsidee

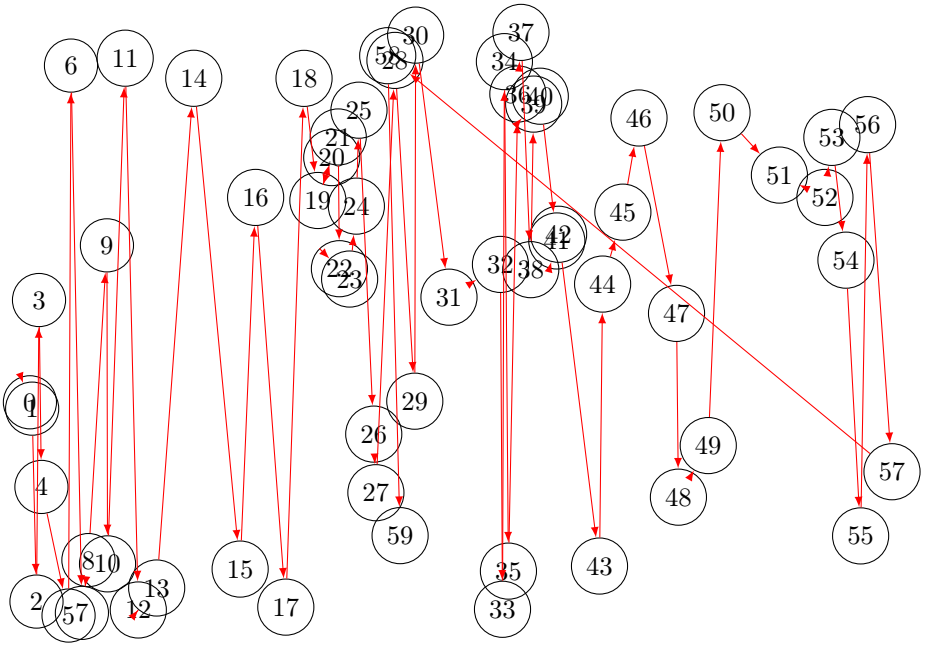
Die Idee der Lösung sollte hieraus vollkommen ersichtlich werden, ohne dass auf die eigentliche Implementierung Bezug genommen wird.

2 Umsetzung

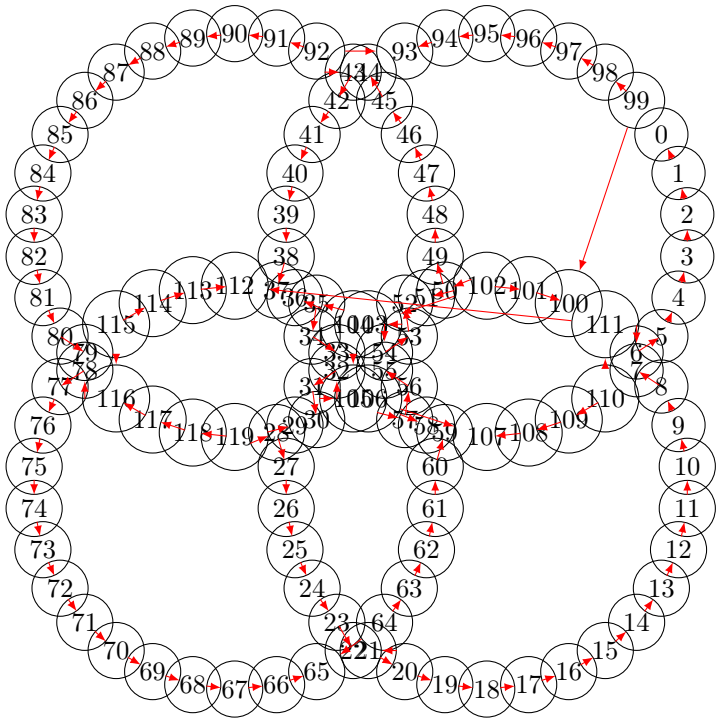
Hier wird kurz erläutert, wie die Lösungsidee im Programm tatsächlich umgesetzt wurde. Hier können auch Implementierungsdetails erwähnt werden.

```
1: function CrossAngle(from_node, over_node, to_node)
2:     p ← MakePair(over_node.first − from_node.first, over_node.second − from_node.second)
3:     q ← MakePair(to_node.first − over_node.first, to_node.second − over_node.second)
4:     angle ← acos( p.firstq.first+p.secondq.second / sqrt(p.first²+p.second²) sqrt(q.first²+q.second²) ) * 180/π
5:     if angle > 180 then
6:         angle ← 180 − angle
7:     end if
8:     return angle
9: end function
10:
11: function Solve(route, coordinates)
12:     if Size(route) = Size(coordinates) then
13:         return true
14:     end if
15:     if NOT IsEmpty(route) then
16:         p ← Back(route)
17:         Sort(coordinates, lambda(lhs, rhs) ← sqrt((p.first − lhs.first)² + (p.second − lhs.second)²) <
    sqrt((p.first − rhs.first)² + (p.second − rhs.second)²))
18:     end if
19:     for i ← 1 to Size(coordinates) do
20:         if coordinates[i] ∈ route then
21:             continue
22:         end if
23:         angle ← −1
24:         if Size(route) ≥ 2 then
25:         end if
26:         if IsEmpty(route) OR ((NOT coordinates[i] ∈ route) AND (Size(route) < 2 OR angle ≥ 90
    OR angle = 0)) then
27:             PushBack(route, coordinates[i])
28:             if Solve(route, coordinates) then
29:                 return true
30:             else
31:                 PopBack(route)
32:             end if
33:         end if
34:     end for
35:     return false
36: end function
```

3 Beispiele

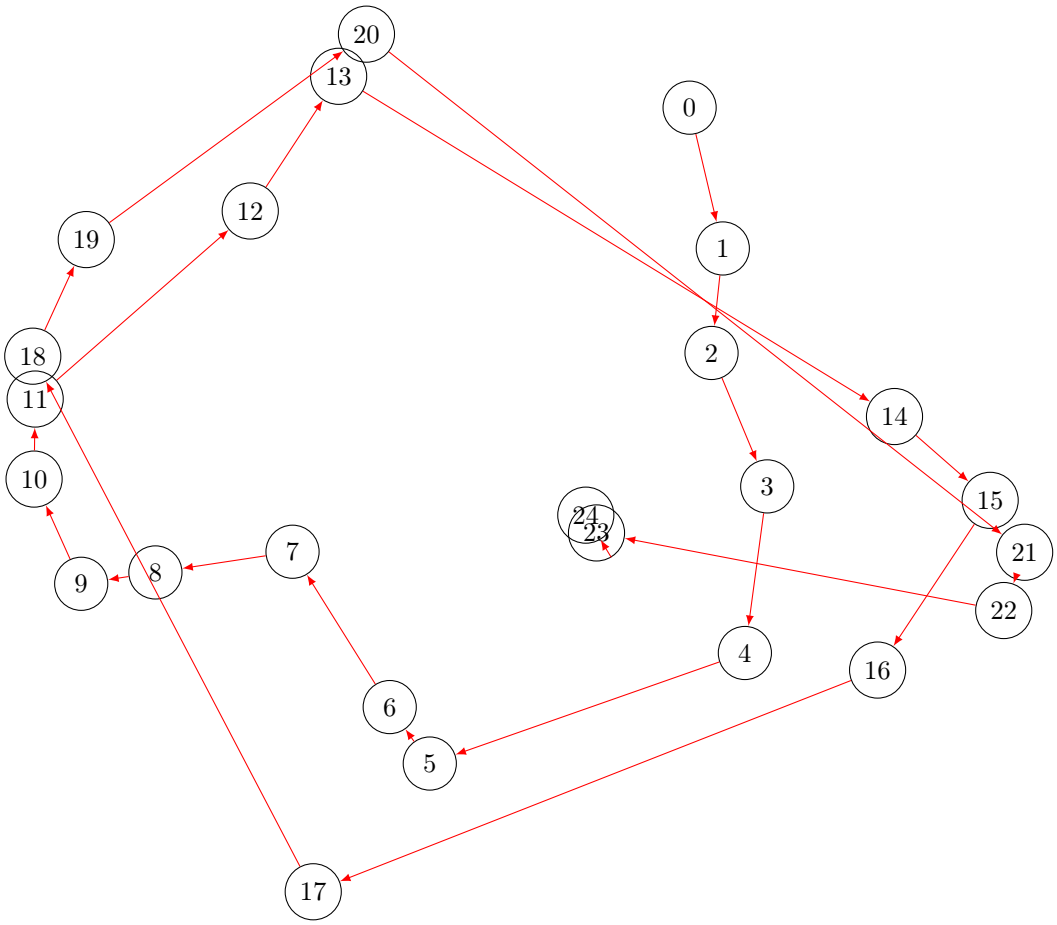


Figur: wenigerkrumm

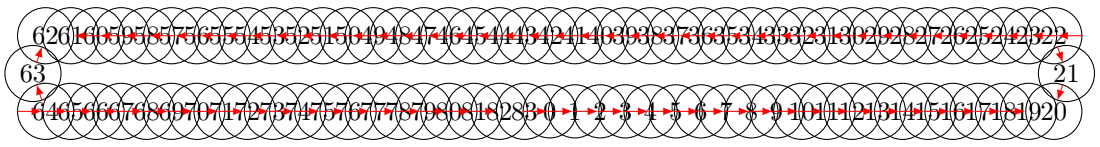


Figur: wenigerkrumm3

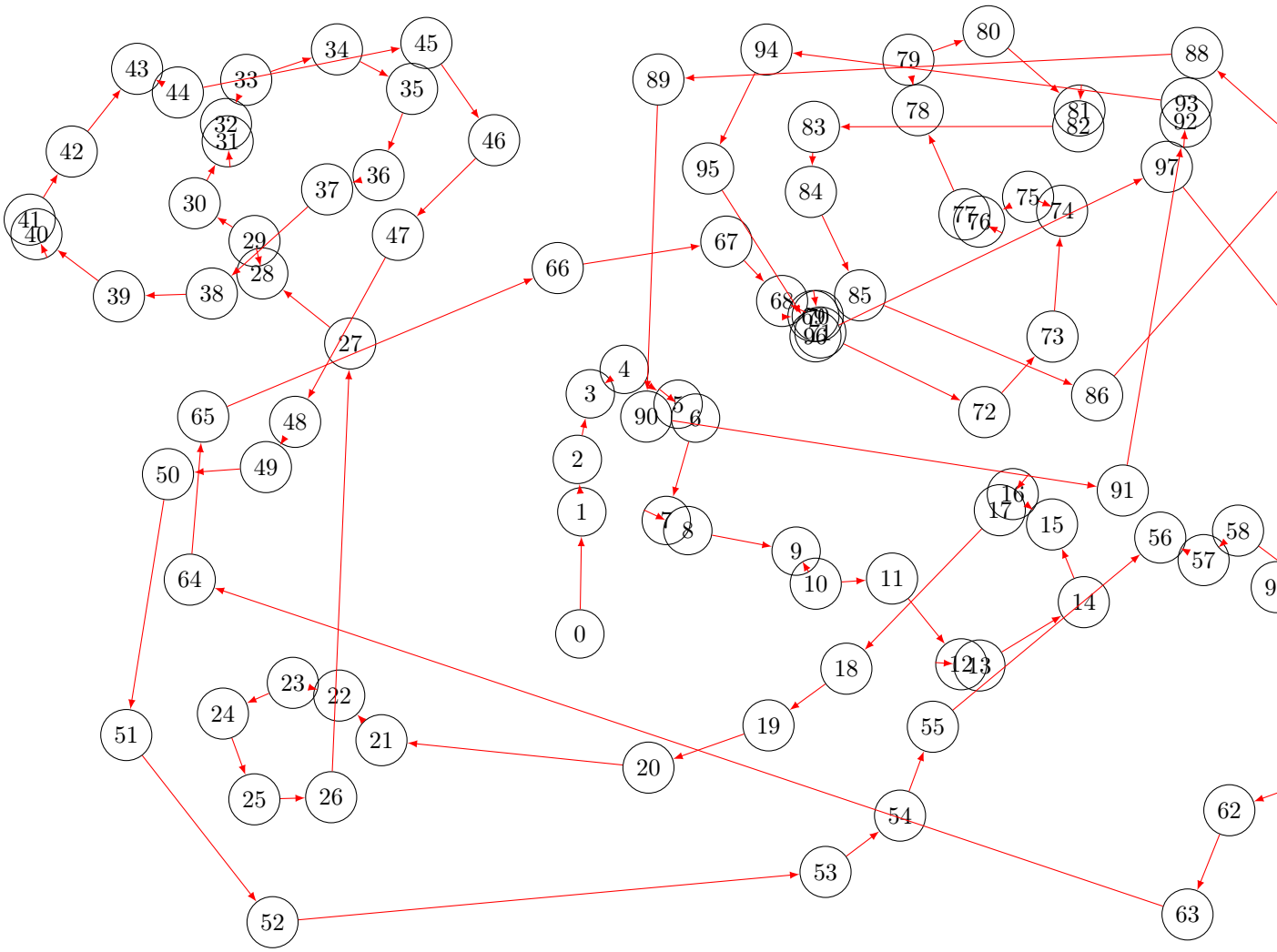
4 Quellcode



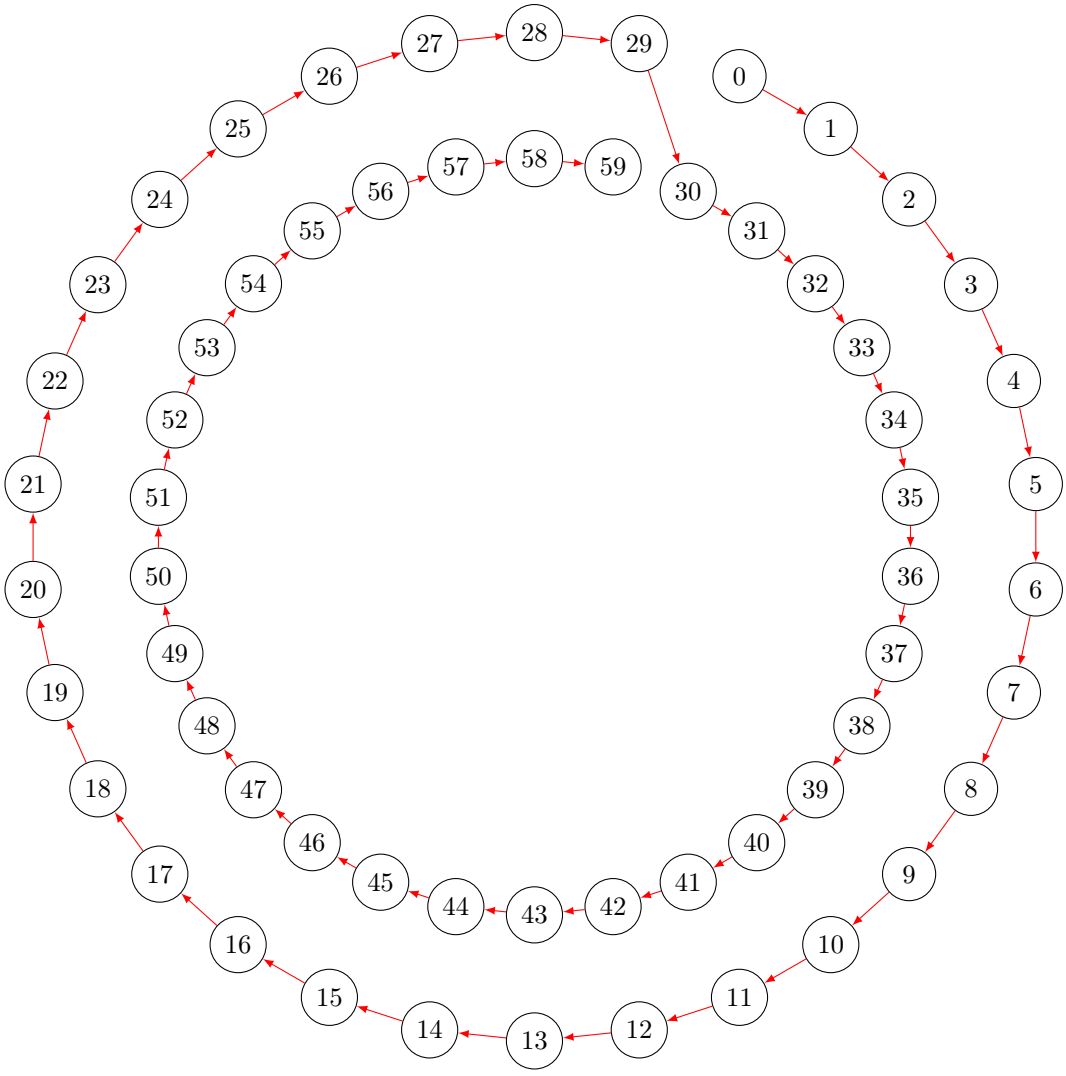
Figur: wenigerkrumm4



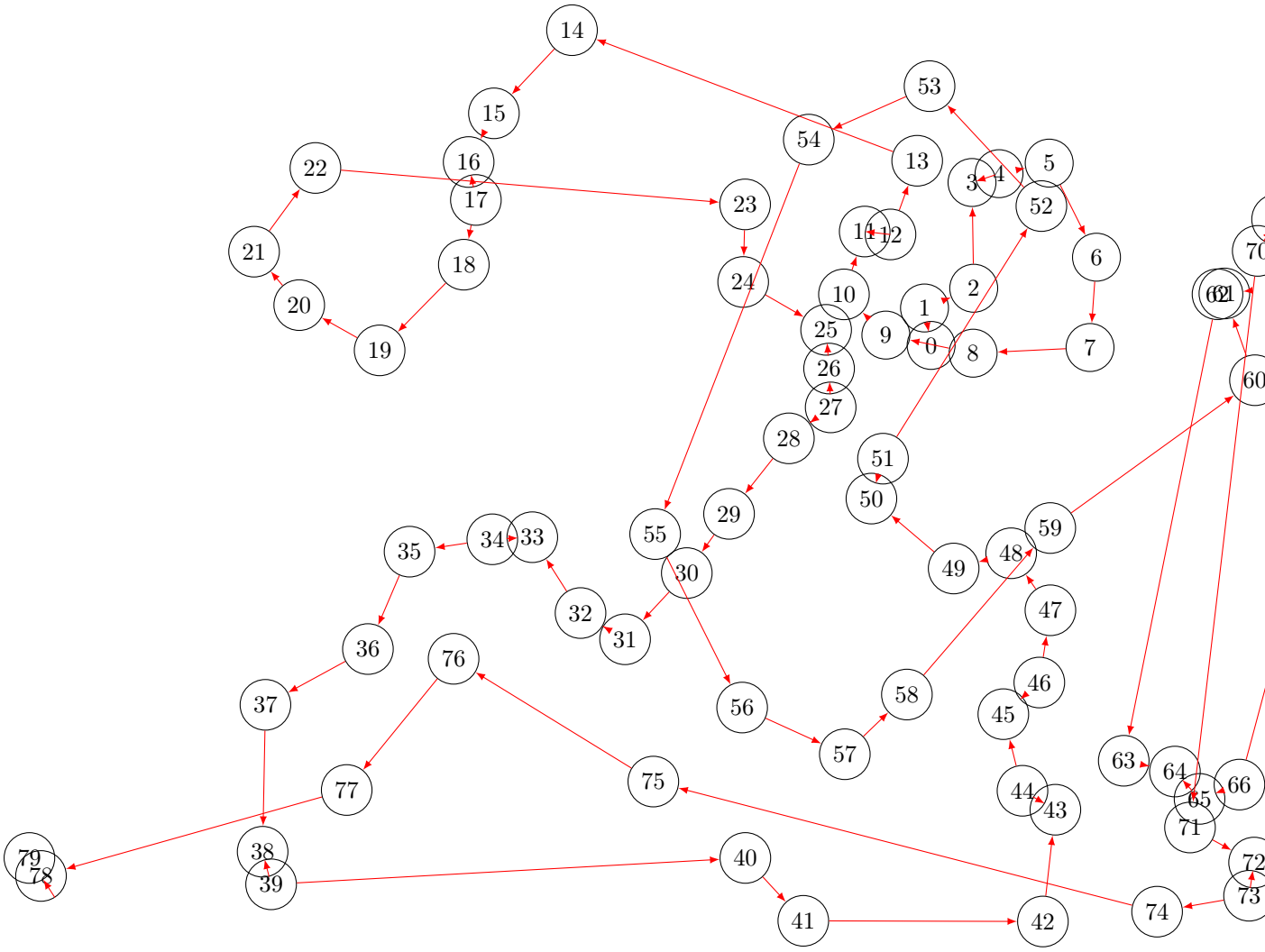
Figur: wenigerkrumm1



Figur: wenigerkrumm7



Figur: wenigerkrumm2



Figur: wenigerkrumm6

```
    /**
 * Liest die Eingabedateien ein
 * und versucht für jede Datei eine Lösung entsprechend der Aufgabenstellung zu finden
 * Die Lösung wird anschließend in die entsprechende Ausgabedatei geschrieben
 * Sollte es keine Lösung geben, wird dies ebenfalls in die Ausgabedatei geschrieben
 * @return 0, wenn es zu keinem RuntimeError oder keiner RuntimeException gekommen ist
 */
int main() {
    string input_dir = "../LennartProtte/Aufgabe1-Implementierung/Eingabedateien";
    string output_dir = "../LennartProtte/Aufgabe1-Implementierung/Ausgabedateien";

    //Durchläuft alle Dateien im Eingabeordner
    for (const std::filesystem::directory_entry &entry: filesystem::directory_iterator(
    input_dir)) {

        //Liest den Dateinamen aus
        string input_file = entry.path();
        string output_file = output_dir + "/" + entry.path().filename().string();

        //Öffnet die Eingabedatei
        ifstream fin(input_file);

        //Öffnet die Ausgabedatei
        ofstream fout(output_file);

        //Liest die Eingabedatei ein
        vector<pair<double, double> > coordinates;
        double x, y;
        while (fin >> x >> y) {
            coordinates.emplace_back(x, y);
        }

        //Berechnet die Lösung
        vector<pair<double, double> > result;
        if (solve(result, coordinates)) {
            fout << "Es␣konnte␣eine␣Flugstrecke␣durch␣alle␣Außenposten␣ermittelt␣werden"
    << endl;
            for (int i = 0; i < result.size(); i++) {
                if (i != 0 && i != result.size() - 1) {
                    fout << cross_angle(result[i - 1], result[i], result[i + 1]) << "°␣";
                }
                fout << "[(" << result[i].first << ",␣" << result[i].second << ")]␣->␣"
    << endl;
            }
        } else {
            fout << "Es␣konnte␣keine␣Flugstrecke␣durch␣alle␣Außenposten␣ermittelt␣werde"
    << endl;
        }
    }
    return 0;
}
```

Methode graphFromLines

```
    /**
 * Versucht rekursiv mit backtracking eine möglichst kurze Route durch den Graphen zu finden,
 * welche die Kriterien der Aufgabenstellung erfüllt.
 * @param route die aktuelle Route
 * @param coordinates eine Menge aller eingelesenen Koordinaten
 * @return true, wenn alle Knoten in der Lösungsmenge (route) enthalten sind, sonst false
 */
bool solve(vector<pair<double, double> > &route, vector<pair<double, double> > &coordinates) {
    //Wenn alle Knoten in der Lösungsmenge sind
    if (route.size() == coordinates.size()) {
        return true;
    }
    //Sortiere nach dem nächsten Knoten
    if (!route.empty()) {
        const auto &p = route.back();
        sort(coordinates.begin(), coordinates.end(),
             [p](const auto &lhs, const auto &rhs) {
                 return sqrt(pow((p.first - lhs.first), 2.0) + (pow((p.second - lhs.second), 2.0)))
                 < sqrt(pow((p.first - rhs.first), 2.0) + (pow((p.second - rhs.second), 2.0)));
             });
    }
    //Für jeden Knoten
    for (int i = 0; i < coordinates.size(); i++) {
        //Wenn dieser Knoten bereits in der Lösungsmenge existiert, überspringe diesen
        if (std::find(route.begin(), route.end(), coordinates[i]) != route.end()) {
            continue;
        }
        double angle = -1;
        if(route.size() >= 2) {
            angle = cross_angle(route[route.size() - 2], route.back(), coordinates[i]);
        }
        if (route.empty() ||
            (std::find(route.begin(), route.end(), coordinates[i]) == route.end() &&
             (route.size() < 2 || angle >= 90 || angle == 0))
                ) {
            //Füge den Knoten hinzu
            route.push_back(coordinates[i]);
            //Wenn es eine Lösung mit der aktuellen Route gibt
            if (solve(route, coordinates)) {
                return true;
            } else {
                route.pop_back();
            }
        }
    }
//Wenn es mit der aktuellen Route keine Lösung geben kann
return false;
}
```

Methode graphFromLines

```
    /**
 * Berechnet den Winkel zwischen den Vektoren von from_node nach over_node und over_node
    nach to_node
 * @param over_node der zweite Knoten
 * @param to_node der dritte Knoten (Zielknoten)
 * @param from_node der erste Knoten
 * @return false, wenn der Winkel der Kanten kleiner als 90° beträgt, sonst true
 */
double cross_angle(const pair<double, double> &from_node,
                   const pair<double, double> &over_node,
                   const pair<double, double> &to_node) {
    pair<double, double> p, q;
    p = make_pair(over_node.first - from_node.first,
                  over_node.second - from_node.second);
    q = make_pair(to_node.first - over_node.first,
                  to_node.second - over_node.second);
    double angle = acos(
            (p.first * q.first + p.second * q.second) / (
                    sqrt(pow(p.first, 2.0) + (pow(p.second, 2.0))) *
                    sqrt(pow(q.first, 2.0) + (pow(q.second, 2.0))
                    )
            )
    ) * 180 / M_PI; //Umrechnung von Radian nach Grad
    if(angle > 180) {
        angle = 180 - angle;
    }
    return angle;
}

```

Methode graphFromLines