HoT 2022 Round 3

General description

Proposed solution basically is a base one with some key improvements. Base solution consists of multiple random positions assignments for two unknown BS and for each case find via random iterations positions of UEs with the most optimal loss function. Loss function were calculated as sum of l_2 norms of time differences to arrival for each UE. Time differences with first BS were taken instead of just time to exclude unknown factor "UE transmission time" from the time, which is the same for for four arrival times for fixed UE.

Next improvements were added to this idea:

1. Loss function is changed to the sum of l_1 norms of times differences to arrival instead of l_2
2. For each UE loss function term is calculated in weighting form, with weights 0.5 for TDOA to the BS which were chosen randomly and weight 1 for the known BS
3. Positions of two unknown BS are being updated one by one, i.e. only one BS is updated within one iteration, and the second one is being set to the best possible position found yet
4. UE positions for checks are being chosen not fully random but in four different variants instead:
    1. As a point on hyperbola, which constructed for each UE from the known TDOA to the two known BS in case when this hyperbola can be reconstructed, i.e. difference in data is possible (it's not guaranteed due to the noises). For simple points taking, hyperbolas are parameterized with heuristically found argument substitution $t \to \log(1 + |t|)$ to have more uniform points distribution
    2. As an intersection of four hyperbolas, calculated from TDOA to the four fixed BS at the current iteration, when this intersection exists and lies inside the room (it's not guaranteed due to the noises)
    3. Random point in the room multiple times
    4. Best found position by the time

Algorithms and data structures used

1. Gauss elimination for solving SLE (for hyperbolas intersection, with linearization proposed in the first paper from the problem statement)
2. Simple linear algebra for constructing hyperbolas equations and their parametrization

Source code

```
#include "bits/stdc++.h"

using namespace std;

const double SZ = 10.0;
const int MAX_N = 200;
const int INF = 1e+9;
int IT = 2000;
int IT_UE = 100;
```

```
const double C_NS = 0.299792458;

int n;
double bs[4][2];
double ue[MAX_N][2];
double t[MAX_N][4];
double bs_best[4][2];
double ue_best[MAX_N][2];
double toa[4];

std::random_device rd;
std::mt19937 gen(rd());
std::uniform_real_distribution < > dis(0, 1);

double get_rand(double x = 1) {
    return dis(gen) * x;
}

double norm(double x[]) {
    return sqrt(pow(x[0], 2) + pow(x[1], 2));
}

template <typename T> int sgn(T val) {
    return (T(0) < val) - (val < T(0));
}

// Gauss elimination for solving SLE
const double EPS = 0.001;
int gauss (vector < vector<double> > a, vector<double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
```

```
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }

    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}

int main() {
    cin >> n;

    cin >> bs[0][0] >> bs[0][1];
    cin >> bs[1][0] >> bs[1][1];

    // Middlepoint of the segment, center of each hyperbola
    double M[2];
    M[0] = (bs[0][0] + bs[1][0]) / 2;
    M[1] = (bs[0][1] + bs[1][1]) / 2;

    // Focus of hyperbolas after shifting to the origin
    double R[2];
    R[0] = bs[0][0] - M[0];
    R[1] = bs[0][1] - M[1];
    double Rnorm = norm(R);

    // Rotation matrix (1, 0) -> R
    double Q[2][2];
    Q[0][0] = R[0] / Rnorm; Q[0][1] = -R[1] / Rnorm;
    Q[1][0] = R[1] / Rnorm; Q[1][1] = R[0] / Rnorm;

    for (int i = 0; i < n; ++i)
        for (int j = 0; j < 4; ++j)
            cin >> t[i][j];

    // constructing hyperbolas

    // bool vector -- if hyperbola for UE reconstructed successfully
    bool total_random[n];
    // Near uniformely distributed points on the constructed hyperbolas
    vector< vector<double> > hx(n);
    vector< vector<double> > hy(n);
    // Focus distance
    double c = sqrt(pow(bs[0][0] - bs[1][0], 2) + pow(bs[0][1] - bs[1][1], 2)) /
2;
    for (int i = 0; i < n; i++) {
        // Hyperbola's difference
        double d = (t[i][0] - t[i][1]) * C_NS;
```

```
        double a = abs(d) / 2;

        if (a > c) {
            // Impossible hyperbola's difference
            total_random[i] = true;
        } else {
            total_random[i] = false;

            double b = sqrt(c * c - a * a);
            // Parametrization of points within the room
            double tmax = max(acosh(2 * SZ / a), asinh(2 * SZ / b));

            double step = 1.0 / IT_UE;
            for (double t = -1; t < 1; t+=step) {
                // parametrization adjustement to have more uniform distribution
                double ts = sgn(t) * log(1 + abs(t)) * tmax / log(2);

                double x = -sgn(d) * a * cosh(ts);
                double y = b * sinh(ts);

                // returning from the canonical form to the original coordinates
                double xx = Q[0][0] * x + Q[0][1] * y + M[0];
                double yy = Q[1][0] * x + Q[1][1] * y + M[1];

                // extra check for each parameterized point to be in the room
                if (-SZ < xx && xx < SZ && -SZ < yy && yy < SZ) {
                    hx[i].push_back(xx);
                    hy[i].push_back(yy);
                }
            }
        }
    }

    bs[2][0] = -get_rand() * SZ;
    bs[2][1] = get_rand() * SZ;
    bs[3][0] = get_rand() * SZ;
    bs[3][1] = get_rand() * SZ;

    double min_loss = INF;
    for (int i = 0; i < IT; ++i) {
        // change one BS position at the iteration
        if (i % 2 == 0) {
            bs[2][0] = -get_rand() * SZ;
            bs[2][1] = get_rand() * SZ;
            bs[3][0] = bs_best[3][0];
            bs[3][1] = bs_best[3][1];

        } else {
            bs[2][0] = bs_best[2][0];
            bs[2][1] = bs_best[2][1];
            bs[3][0] = get_rand() * SZ;
            bs[3][1] = get_rand() * SZ;
        }

        double loss_tdoa = 0;
```

```
        for (int j = 0; j < n; ++j) {
            double min_loss_ue = INF;

            for (int k = -2; k < IT_UE; ++k) {
                double ue_x, ue_y;
                // for even iteration take points on the hyperbola, if it's exist
                if (!total_random[j] && k % 2 == 0 && k >= 0) {
                    if (k >= hx[j].size()){
                        break;
                    }
                    ue_x = hx[j][k];
                    ue_y = hy[j][k];
                // for odd ones take points randomly
                } else if (k >= 0){
                    ue_x = (get_rand(2) - 1) * SZ;
                    ue_y = (get_rand(2) - 1) * SZ;
                // intersection of hyperbolas
                } else if (k == -1) {
                    vector< vector <double > > A(3, vector<double>(4));

                    for (int ii = 0; ii < 3; ii++) {
                        A[ii][0] = bs[ii+1][0] - bs[0][0];
                        A[ii][1] = bs[ii+1][1] - bs[0][1];
                        A[ii][2] = C_NS * (t[j][ii+1] - t[j][0]);
                        A[ii][3] = (pow(bs[ii+1][0], 2) + pow(bs[ii+1][1], 2)
                                    - pow(bs[0][0], 2) - pow(bs[0][1], 2) -
                                      pow(C_NS * (t[j][ii+1] - t[j][0]), 2)) / 2;
                    }

                    vector <double> ans;

                    int res = gauss(A, ans);

                    if (res != 0 && ans[0] >= -SZ && ans[0] <= SZ &&
                            ans[1] >= -SZ && ans[1] <= SZ) {
                        ue_x = ans[0];
                        ue_y = ans[1];
                    } else {
                        ue_x = (get_rand(2) - 1) * SZ;
                        ue_y = (get_rand(2) - 1) * SZ;
                    }
                // previous found best UE position
                } else if (k == -2) {
                    ue_x = ue_best[j][0];
                    ue_y = ue_best[j][1];
                }

                for (int l = 0; l < 4; ++l)
                    toa[l] = sqrt(
                        pow(ue_x - bs[l][0], 2) +
                        pow(ue_y - bs[l][1], 2)
                    ) / C_NS;

                // Improved loss function
```

```
                double loss = 0;
                double weights[4] = {1, 1, 0.5, 0.5};
                for (int l = 1; l < 4; ++l) {
                    loss += weights[l] *
                        abs((toa[l] - toa[0]) - (t[j][l] - t[j][0]));
                }

                if (loss < min_loss_ue) {
                    min_loss_ue = loss;
                    ue[j][0] = ue_x;
                    ue[j][1] = ue_y;
                }
            }

            loss_tdoa += min_loss_ue;
        }

        // Update BS and UE best found positions
        if (loss_tdoa < min_loss) {
            min_loss = loss_tdoa;

            for (int j = 0; j < 4; ++j) {
                bs_best[j][0] = bs[j][0];
                bs_best[j][1] = bs[j][1];
            }

            for (int j = 0; j < n; ++j) {
                ue_best[j][0] = ue[j][0];
                ue_best[j][1] = ue[j][1];
            }
        }
    }

    for (int i = 2; i < 4; ++i)
        printf("%0.9f %0.9f\n", bs_best[i][0], bs_best[i][1]);

    for (int i = 0; i < n; ++i)
        printf("%0.9f %0.9f\n", ue_best[i][0], ue_best[i][1]);

    return 0;
}
```

Further improvements ideas

1. Create band around calculated hyperbolas and take points randomly from thins band instead from exact hyperbola, to compensate noise
2. Optimize unknown BSs position finding by shifting them from the previous positions to randomly chosen direction and returning back in case of increasing loss
3. Loss function adjustments, for example, weight UEs contribution by their loss term