# SOME MORE ON RESTORING DISTANCE MATRICES BETWEEN DNA CHAINS: RELIABILITY COEFFICIENTS

**Mikhail Abramyan***
Department of Computational
Mathematics and Cybernetics,
Shenzhen MSU – BIT University,
China
m-abramyan@yandex.ru

**Boris Melnikov**
Department of Computational
Mathematics and Cybernetics,
Shenzhen MSU – BIT University,
China
bormel@mail.ru

**Ye Zhang**
School of Mathematics
and Statistics,
Beijing Institute of Technology,
China
ye.zhang@bit.edu.cn

Article history:
Received 12.12.2023, Accepted 26.12.2023

**Abstract**

This article is a description of the continuation of previous research by the authors related to the restoration of distance matrices. The main difficulty that arises with such a recovery is that it is impossible to use conventional techniques such as variants of gradient descent algorithms, since too many variables would arise. In addition, in this article we also do not use algorithms for variants of the branch and boundary method, which in our previous publications were sometimes used for the problem considered in the article; the main argument for not using it is that using it would provide little information about the subtasks generated by this algorithm, in particular, it is difficult to adequately assess the real the values of the resulting boundaries. Therefore, we use the so-called step-by-step filling of the distance matrix.

In the approach considered in the paper, we consider, that the algorithm of the initial distance formation works in the best way. Therefore, we conditionally believe that the corresponding value of badness cannot be improved. This assumption really makes it possible to improve the value of badness.

Thus, in this paper we have obtained practical results of reconstructing distance matrices, which significantly improve the results given in our previous papers.

**Key words**

DNA chains, distance matrix, optimization problem, restoring algorithm, greedy algorithm, heuristics.

## 1 Introduction

In this paper, we continue to consider algorithms for restoring distance matrices between DNA chains.

Various possible uses for such matrices and related concepts can be found in the works [Needleman and Wunsch, 1970; Sykes, 2003; Lennarz and Lane (Eds), 2013; Maloy and Kelly (Eds), 2013]. It is important to note that not all of these works explicitly define distance matrices, but their implicit use is important for each of them. It is also worth mentioning works that consider algorithms for forming the distance matrices we need; in addition to the classic paper [Levenshtein, 1966], we shall provide the following links: [Needleman and Wunsch, 1970; Winkler, 1990; van der Loo, 2014].

The connection of some algorithms for studying DNA chains with various problems of experimental physics has been demonstrated in some our previous publications; to these publications it is worth adding [Sergeenko et al., 2020], which, among other things, examines the connection with Hamiltonian cycles, and, consequently, with the traveling salesman problem (TSP), also considered in some publications of the authors of this paper, see [Melnikov, Zhang, and Chaikovskii, 2022] and some others.

In the last paper, we formulated among other things another connection, i.e., a connection between two problems that at first glance seem completely different: the classical TSP mentioned before and the restoring distance matrices problem considered in this paper. Certainly, it is theoretically possible to easy formulate both these problems by the problems of minimizing the corresponding goal functions.

But, of course, solving this mathematical model by, for example, gradient descent requires in practice not only a lot of time for the programmer, but also a lot of time of working the resulting program. This is because the number of variables of the problem should just be equal to twice the number of cities-points ($2 \cdot n$, for the classical

*Corresponding author.

TSP) or to the order of the square of the dimension of the matrix ($n^2$, for the restoring problem). In both the cases, the dimensions we can process should be of the order $7-8$ only. In the full versions, however, we need to consider problems of dimension more than 100 for classical TSP (as a rule, the authors of this paper do not consider the geometric TSP, for which acceptable pseudo-optimal solutions for millions points have been found for a long time) and more than 500 for the restoring problem (see also [Melnikov, Zhang, and Chaikovskii, 2022]). Therefore, the main "theoretical aspect" of the presented paper is that we can do everything "step-by-step", i.e., if we consistently place the points, then in the limit we shall get the optimal solution to the problem. Remark also, that we are not considering possible "hits to local maxima" of the objective function: the ways to deal with this are, of course, some separate topics.

Thus, in our research, the target minimized function for matrix reconstruction is calculated based on the so-called triangular norm. The difference between the actual resulting triangles from acute-angled isosceles (and in the distance matrix of the order N×N of such triangles of the order $N^3/6$) is called "badness"; we sum up these values in the matrix for all triangles. Remark also that with real calculations, the number of "bad" triangles (i.e., having violations of the triangle inequality) is quite small, and those in which the triangle inequality is violated practically did not arise; this indirectly confirms the correctness of all such hypotheses. Also, the real calculations allowed us to compare the quality of the algorithms themselves for estimating distances between DNA chains ("heuristics for comparing heuristics").

Thus, the options for recovery algorithms were given in our previous papers [Melnikov, Pivneva, and Trifonov, 2017; Melnikov, Zhang, and Chaikovskii, 2022], see also the links from there. In [Melnikov, Zhang, and Chaikovskii, 2022], several directions of further research were given, which should lead to improved recovery algorithms, i.e., to the formation of such algorithms that would be performed in an acceptable time and at the same time would give a lower value of badness. In the current paper, we consider only one of these possible directions, that is, the considering so-called reliability coefficients.

It is important to remark, that we do not use algorithms for variants of the branch and boundary method, which in our previous publications were sometimes used for the problem considered here; the main argument for not using it is that its possible using [Melnikov and Trenina, 2018; Melnikov, Trenina, and Kochergin 2018] would provide little information about the subtasks generated by this algorithm, in particular, it is difficult to adequately assess the real the values of the resulting boundaries. This is the second argument for using so-called step-by-step methods for algorithms for filling of the distance matrix.

At the end of Introduction, let us say the following. The objects of research of these algorithms is the mitochondrial DNA s (mt DNA s), "direct female line", considered for mammals (specifically, for some species of monkeys), [Maloy and Kelly (Eds), 2013; Cibelli et al. (Eds), 2014] etc. The the length of its sequence exceeds 16 000 characters; let us remark that the total length of human DNA exceeds 3 000 000 000 characters. For comparison, we can use the following analogy. If the length of mtDNA is imagined to be equal to 1 cm (the width of a finger), then in these units of measurement, the total length of the human genome will be comparable to the length of the path around the territory of a usual university. (The real length of the entire human DNA chain, according to the American biologist G. Taylor, is about 500 – 1000 times less, but therefore the length of mtDNA is the same number of times less.)

The content of the paper is as follows.

In Section 2, we define the reliability coefficient and consider its possible using in the restoring problem.

The title of Section 3 is "The old and the modified versions of calculating the next element of the matrix". The details of the implementation of the algorithms (their old and modified versions) used to obtain the results described below should be clear based on the texts of the corresponding programs, as well as small comments given later in that section and inside in the texts of the functions.

Some results of computational experiments are given in Section 4. We can say, simplifying a little, that the topic of our works in this direction is how to obtain the missing values in the given tables with minimum badness.

Section 5 is Conclusion. We consider some possible directions for the further work on this subject. Some of them have already been mentioned in our previous publications, they are gradually being implemented in the current computer programs.

## 2 The reliability coefficient and its using

In the approach considered in this paper, we consider (in contrast to the one described in [Melnikov, Zhang, and Chaikovskii, 2022] the so-called "first direction" of the development of the work), that the algorithm of the initial distance formation itself (the Needleman – Wunsch algorithm is only one of the possible examples) works in the best way. Therefore, we conditionally believe that the corresponding value of badness cannot be improved (i.e., it is impossible to get something better than it is throughout the matrix, knowing only a certain number of its elements). Of course, such an assumption is incorrect (and this can be seen from the calculations given in the article); but, however, the assumption is exactly that, and this assumption really makes it possible to improve the value of badness.

Under this assumption, of course, the previously obtained values (and first of all, the values a priori available in the matrix) should have more influence on the final values, which we do; and we do just by introducing these coefficients.

That is why we propose an algorithm that was titled “The second direction” in [Melnikov, Zhang, and Chaikovskii, 2022]. We introduce a special “reliability coefficient”; certainly, it should be $R < 1$ (to say, $R = 0.9$ or $R = 0.7$ in the following description and in the following calculations). It means the following method.

We consider that the initial values of the matrix (in the example considered in the paper, the remaining $10\,\%$ of the elements after the removal) have a weight of $1.0$. The elements derived from only the initial ones (i.e., in the beginning of filling) have a weight of $R$.

In the general case (i.e., after filling in some elements) we proceed as follows. As in the greedy algorithm already discussed in this paper, we form all possible triangles obtained together with the element selected for filling, i.e. if the considered unfilled element of the matrix is $m_{i,j}$, then, as before, we consider all such $k$ that $m_{i,k}$ and $m_{j,k}$ are already filled. However, we calculate the obtained values *with the reliability coefficients already assigned to these values*, i.e., we minimize the general function, which includes values with these coefficients; for the reliability coefficients $R_{i,k}$ and $R_{j,k}$, we assume that the reliability coefficient of the considered triangle is

$$R_{\Delta} = \frac{R_{i,k} + R_{j,k}}{2}\,. \tag{1}$$

The resulting value obtained as a result of minimization is placed in a matrix with its new reliability coefficient equal to the a priori value of $R$ multiplied by the average reliability coefficient of all considered triangles that form the element $m_{i,j}$: using (1) and assuming we are considering $m$ triangles, this new coefficient can be written as follows:

$$\frac{\left(R_{\Delta}^{(1)} + R_{\Delta}^{(2)} + \cdots + R_{\Delta}^{(m)}\right) \cdot R}{m}\,.$$

And, of course, the best value of the reliability coefficient $R$ should be obtained as a result of some self-learning process. We continue to implement this in the current programs being developed, but in this paper, we present the results of calculations that do not take this process into account.

## 3 The old and the modified versions of calculating the next element of the matrix

The details of the implementation of the algorithms used to obtain the results described below should be clear based on the texts of the corresponding programs, as well as small comments given later in this section and inside in the texts of the functions. As one can see, we provide a detailed description of the algorithms used in the form of descriptions of the main points related to the programs developed and under development.

Now, let us give descriptions of the original algorithms.

The modification of the algorithm using the $R$ parameter is easier to describe using the example of the simplest algorithm. Its original version without this parameter is shown on Fig. 1.

*Step 1.* Among all elements of the matrix equal to $-1$, the one is selected for which the maximum number of triangles (i.e., triples of matrix elements) can be obtained, the other two sides of which are defined (not equal to $-1$); if there are several such elements, then any of them is selected. This step is provided by the `KolTrig` and `MaxTrig` functions.

The `Simplest` algorithm lives up to its name, being the simplest of the greedy ones. In it, the following actions are performed to determine the values of the missing matrix elements (initially equal to $-1$).

Next, we present a variant that uses the characteristics $R$ associated with each element of the matrix, Fig. 2.

- The initial characteristics $R$ are set in `InitR` (the `Rinit` parameter is used).
- Function `KolTrigR` does not just sum triangles, but takes into account their “weight”, obtained as an average characteristic $R$.
- `MaxTrigR` is no different from `MaxTrig`, except that function `KolTrigR` is called in it.
- In function `MakeDistantiaR`, in addition to calculating the value of a new element, i.e. a pair `I`, `J`, its characteristic $R$ is calculated; the `Rnew` parameter is used.

The `Simple` algorithm is a modification of the `Simplest` algorithm, in which only the actions performed in Step 2 are actually changed. In step 1, like the `Simplest` algorithm, the element (from the not yet defined elements of the matrix) is selected, for which the maximum number of triangles can be obtained, where the other two sides of which are defined.

Like the `Simplest` algorithm, Step 2 in the `Simple` algorithm begins by finding the `rBeg` value equal to the arithmetic mean of the maximum sides of those triangles that can be constructed for the element being determined. However, an attempt is then made to improve the average `Badness` characteristic corresponding to the `rBeg` value by performing an additional iterative process.

(Note that we shall write “badness” or `Badness`, depending on whether we are talking about the value of a mathematical quantity or a variable of a computer program.)

Thus, we improve the average `Badness` characteristic by performing an additional iterative process. Exactly, the average `Badness` characteristics are calculated for each of the values obtained by slightly adjusting the `rBeg` value; the adjustment consists of multiplying or dividing by the following coefficients: 1.1, 1.2, 1.4, 1.7, 2.0, and 2.5.

```cpp
int Tabula::KolTrig(int I, int J) {
    int kol = 0;
    for (int k=1; k<=nDim; k++) {
        if (Get(I,k)<0) continue;
        if (Get(J,k)<0) continue;
        kol++;
    }
    return kol;
}

bool Tabula::MaxTrig(int& Imax, int& Jmax) {
    Imax = 0; Jmax = 0;
    int Kolmax = 0;
    for (int i=1; i<=nDim-1; i++) for (int j=i+1; j<=nDim; j++) {
        if (Get(i,j)>=0) continue;
            // we do not process the already installed ones
        int kol = KolTrig(i,j);
        if (kol<=Kolmax) continue;
        Kolmax = kol;
        Imax = i;
        Jmax = j;
    }
    return Imax>0;
}

void Tabula::MakeDistantia(int I, int J) {
    int kol = 0; // the number of the triangles already processed
    double sum = 0.0; // the sum of their badness values
    for (int k=1; k<=nDim; k++) {
        double r1 = Get(I,k); if (Get(I,k)<0) continue;
        double r2 = Get(J,k); if (Get(J,k)<0) continue;
        kol++;
        sum += max(r1,r2);
    }
    if (kol<=0) return;
    Set(I,J,sum/kol);
}

void Tabula::RunSimplest() {
    for (;;) {
        int Imax, Jmax;
        if (!MaxTrig(Imax,Jmax)) return;
        MakeDistantia(Imax,Jmax);
    }
}
```

Figure 1. Function `RunSimplest()` without additional heuristics and the auxiliary functions necessary for it.

```
double Tabula::KolTrigR(int I, int J) {
    double kol = 0;
    for (int k=1; k<=nDim; k++) {
        if (Get(I,k)<0) continue;
        if (Get(J,k)<0) continue;
        kol+= (GetR(I, k)+GetR(J,k))/2;
    }
    return kol;
}

bool Tabula::MaxTrigR(int& Imax, int& Jmax) {
    Imax = 0; Jmax = 0;
    double Kolmax = 0;
    for (int i=1; i<=nDim-1; i++) for (int j=i+1; j<=nDim; j++) {
        if (Get(i,j)>=0) continue;
            // we do not process the already installed ones
        double kol = KolTrigR(i,j);
        if (kol<=Kolmax) continue;
        Kolmax = kol;
        Imax = i;
        Jmax = j;
    }
    return Imax>0;
}

void Tabula::MakeDistantiaR(int I, int J, double Rnew) {
    int kol = 0; // the number of the triangles already processed
    double sum = 0.0; // the sum of their badness values
    double sumR = 0.0;
    for (int k=1; k<=nDim; k++) {
        double r1 = Get(I,k); if (Get(I,k)<0) continue;
        double r2 = Get(J,k); if (Get(J,k)<0) continue;
        kol++;
        sum += max(r1,r2);
        sumR += (GetR(I,k) + GetR(J, k))/2;
    }
    if (kol<=0) return;
    Set(I,J,sum/kol);
    SetR(I,J,sumR*Rnew/kol);
}

void Tabula::InitR(double Rinit) {
    for (int i=1; i<=nDim-1; i++) for (int j=i+1; j<=nDim; j++)
        if (GetR(i,j) >= 0) SetR(i,j,Rinit);
        else SetR(i,j,0);

}

void Tabula::RunSimplestR(double Rinit, double Rnew) {
    InitR(Rinit);
    for (;;) {
        int Imax, Jmax;
        if (!MaxTrigR(Imax,Jmax)) return;
        MakeDistantiaR(Imax,Jmax,Rnew);
    }
}
```

Figure 2. Function `RunSimplest()` with additional heuristics and the auxiliary functions necessary for it.

```
double Paria::PDP(double rDlin) {
    if (nKol<=0) return -1;
    double ret = 0.0;
    for (int i=0; i<nKol; i++) {
        Trigonum t(Maxy[i],Miny[i],rDlin);
        ret += t.Bad();
    }
    return ret / nKol;
}

double Paria::ONP(double rBeg, int nIter) {
    if (nKol<=0) return -1;
    if (nIter<=0) return rBeg;
    double rBad = 999999; // the calculated best value of badness
    double rDli; // the length corresponding to the best value
    double r;
    r = PDP(rBeg); if (r<rBad) { rBad = r; rDli = rBeg; }
    bool b = false;
        // are there any further calculations after ours ones?
    if ((r=PDP(rBeg*1.1))<rBad) { rBad=r; rDli=rBeg*1.1; b=true; }
    if ((r=PDP(rBeg/1.1))<rBad) { rBad=r; rDli=rBeg*1.1; b=true; }
    if ((r=PDP(rBeg*1.2))<rBad) { rBad=r; rDli=rBeg*1.1; b=true; }
    if ((r=PDP(rBeg/1.2))<rBad) { rBad=r; rDli=rBeg*1.1; b=true; }
    if ((r=PDP(rBeg*1.4))<rBad) { rBad=r; rDli=rBeg*1.1; b=true; }
    if ((r=PDP(rBeg/1.4))<rBad) { rBad=r; rDli=rBeg*1.1; b=true; }
    if ((r=PDP(rBeg*1.7))<rBad) { rBad=r; rDli=rBeg*1.1; b=true; }
    if ((r=PDP(rBeg/1.7))<rBad) { rBad=r; rDli=rBeg*1.1; b=true; }
    if ((r=PDP(rBeg*2.0))<rBad) { rBad=r; rDli=rBeg*1.1; b=true; }
    if ((r=PDP(rBeg/2.0))<rBad) { rBad=r; rDli=rBeg*1.1; b=true; }
    if ((r=PDP(rBeg*2.5))<rBad) { rBad=r; rDli=rBeg*1.1; b=true; }
    if ((r=PDP(rBeg/2.5))<rBad) { rBad=r; rDli=rBeg*1.1; b=true; }
    if (!b) return rBeg;
    return ONP(rDli,nIter-1);
}

bool Paria::Add(double A, double B) {
    if (nKol>=MAXKOL) return false;
    Maxy[nKol] = max(A,B);
    Miny[nKol] = min(A,B);
    nKol++;
    return true;
}
```

Figure 3. Function RunSimple() with additional heuristics and the auxiliary functions necessary for it. Part 1.

```
double Paria::Mediocris() {
    if (nKol<=0) return -1;
    double sum = 0;
    for (int i=0; i<nKol; i++) sum += Maxy[i];
    return sum / nKol;
}

void Tabula::RunSimple() {
    for (;;) {
        int Imax, Jmax;
        if (!MaxTrig(Imax,Jmax)) return;
        MakeDist(Imax,Jmax);
    }
}

void Tabula::MakeDist(int I, int J) {
    Paria p;
    for (int k=1; k<=nDim; k++) {
        double r1 = Get(I,k); if (Get(I,k)<0) continue;
        double r2 = Get(J,k); if (Get(J,k)<0) continue;
        p.Add(r1,r2);
    }
    double rBeg = p.Mediocris();
    double rDist = p.ONP(rBeg,MAXITER);
    if (rDist<0) exit(101);
    Set(I,J,rDist);
}

bool Paria::AddR(double A, double B, double r) {
    if (nKol>=MAXKOL) return false;
    Maxy[nKol] = max(A,B);
    Miny[nKol] = min(A,B);
    R[nKol] = r;
    nKol++;
    return true;
}

double Paria::MediocrisR(double *r) {
    if (nKol<=0) return -1;
    double sum = 0;
    *r = 0;
    for (int i=0; i<nKol; i++) {
        sum += Maxy[i];
        *r += R[i];
    }
    *r = *r / nKol;
    return sum / nKol;
}
```

Figure 4. Function `RunSimple()` with additional heuristics and the auxiliary functions necessary for it. Part 2.

```
void Tabula::MakeDistR(int I, int J, double Rnew) {
    Paria p;
    double avrR;
    for (int k=1; k<=nDim; k++) {
        double r1 = Get(I,k); if (Get(I,k)<0) continue;
        double r2 = Get(J,k); if (Get(J,k)<0) continue;
        p.AddR(r1,r2,(GetR(I,k) + GetR(J, k))/2);
    }
    double rBeg = p.MediocrisR(&avrR);
    double rDist = p.ONP(rBeg,MAXITER);
    if (rDist<0) exit(101);
    Set(I,J,rDist);
    SetR(I,J,avrR*Rnew);
}

void Tabula::RunSimpleR(double Rinit, double Rnew) {
    InitR(Rinit);
    for (;;) {
        int Imax, Jmax;
        if (!MaxTrigR(Imax,Jmax)) return;
        MakeDistR(Imax,Jmax,Rnew);
    }
}
```

Figure 5. Function `MakeDistR()` and the auxiliary function necessary for it.

```
490.532 0.149735 181.867 0.055515 (0.00) -- Orig

337.910 0.103147 140.148 0.042780 (0.08) -- Simplest
441.842 0.134873  97.393 0.029729 (0.09) -- SimplestR 1.00, 0.90, Ravr=0.76
317.832 0.097018  97.330 0.029710 (0.09) -- SimplestR 1.00, 0.70, Ravr=0.48

144.774 0.044192  73.155 0.022331 (0.04) -- Simple
105.817 0.032301  52.751 0.016102 (0.04) -- SimpleR 1.00, 0.90, Ravr=0.76
100.028 0.030533  50.500 0.015415 (0.04) -- SimpleR 1.00, 0.70, Ravr=0.48
```

Figure 6. The main results of the computational experiments.

(Let us also note, that such an algorithm may seem not optimal; however, the use of some other iteration process options only slows down calculations, and due to the fact that the algorithms are heuristic, the accuracy obtained is quite acceptable to us.)

If none of the corrected values has a lower badness characteristic, then the initial value of `rBeg` is assigned to the element being defined. If some of the corrected values have lower badness characteristics, then for the value with the minimum badness characteristic, the actions previously performed for the initial value of `rBeg` are repeated.

The described iterative process is completed either when it is not possible to obtain corrected values with lower Badness characteristics at some iteration, or when a predetermined number of iterations is performed.

Step 2 of the Simple algorithm is implemented in the `MakeDist` method, see also Fig. 2.

In this method, unlike the similar `MakeDistantia` method used in the `Simplest` algorithm, an additional `p` object of the `Paria` class is used, containing a set of pairs of those elements that form triangles with the element being defined.

The `Mediocris` method of the `p` object returns the arithmetic mean of the maximum sides of those triangles that can be constructed for the element being defined; in the `MakeDist` method, this value is assigned to the `rBeg` variable.

The `ONP` method provides one step of the iterative process described above with an initial value `reg` and the number of iterations `MAXITER`.

This method uses the `PDP` auxiliary method, which calculates the averaged `Badness` characteristic for the `rDlin` value (averaging is performed over all triangles containing the `rDlin` element).

In the `Bad` method, to calculate the Badness characteristic of a considered triangle, `AA`, `BB`, and `CC` determine the sides of the triangle, and `Alfa`, `Beta`, and `Gama` determine the corresponding angles; both these sets are ordered in descending order.

The main function for the `Simple` algorithm differs from the function for the `Simplest` algorithm only by calling another method to implement Step 2 (`MakeDist` instead of `MakeDistantia`).

The modified version of the `Simplest` algorithm, which uses the characteristics `R` associated with each element of the matrix, includes one additional `InitR` method and new variants of the `KolTrig`, `MaxTrig`, and `MakeDistantia` methods (titled `KolTrigR`, `MaxTrigR`, and `MakeDistantiaR` respectively). The `Init` method sets the initial characteristics `R` for all elements of the matrix (the `Rinit` parameter is used). The `KolTrigR` method calculates the "weighted" number of triangles found, and the average characteristics `R` of the existing sides of these triangles are used as weight coefficients. The `MaxTrigR` method is no different from `MaxTrig`, except that `KolTrigR` is called in it. In `MakeDistantiaR`, in addition to calculating the value of a new element (with coordinates `I` and `J`), its characteristic `R` is calculated; the `Rnew` parameter is used.

In the modification of the `Simple` algorithm using the characteristics `R`, like the modification of the `Simplest` algorithm, the new `InitR` method and new variants of the `KolTrig` and `MaxTrig` methods are used; those are `KolTrigR` and `MaxTrigR` methods described above. In addition, it was necessary to change the `Add` and `Mediocris` methods (the other methods of the `Paria` class remained unchanged). In the new version of the `Mediocris` method, the average value of the characteristic `R` is additionally calculated, which is returned as the output parameter `r`.

The main method for the modified Simple algorithm, which takes into account the characteristics `R`, differs only by calling the `InitR` method and replacing the `MaxTrig` and `MakeDist` methods for the `MaxTrigR` and `MakeDistR` methods.

Thus, the `Simple` algorithm is some more complicated, we present it immediately with the application of the characteristic R. It uses the `MakeDist` method, which calls the additional methods `Add`, `Mediocris`, and `ONP` of the `Paria` class, Fig. 3.

In the modification of the `Simple` algorithm using the characteristics of R, it was necessary to change the `Add` and `Mediocris` methods; the other methods of the `Paria` class remained unchanged, see Fig. 3 and Fig. 4.

The modified `MakeDistR` method invokes new variants of the `Add` and `Mediocris` methods, which makes it possible to determine not only the values of the new matrix elements, but also their characteristic R, see Fig. 5.

The variant of the `Simple` algorithm that takes into account the characteristics of R differs only by calling the `Init` function and calling `MaxTrigR` and `MakeDistR` instead of `MaxTrig` and `MakeDist`, see also Fig. 5.

## 4 Some results of computational experiments

First of all, let us repeat the most important (the goal) from the section with the same title of the previous article [Melnikov, Zhang, and Chaikovskii, 2022]:

> simplifying a little, *the topic of our works in this direction is how to obtain the missing values in these tables with minimum badness.*

Let us give a few comments on the given large tables of results. Both are given for the reader's possible verification of these results; at the same time, anyone can request from the authors, after which we shall send the same tables in the form of text. Having these tables, anyone can simply check their characteristics (badness, etc., according to the formulas given in the paper).

Table 1. The initial matrix obtained by applying the algorithm to 28 species of monkeys (no more than one species from each genus)

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 299 | 258 | 269 | 315 | 324 | 285 | 295 | 503 | 327 | 268 | 271 | 302 | 293 | 305 | 283 | 262 | 261 | 302 | 317 | 266 | 961 | 272 | 328 | 266 | 242 | 268 | 274 |
| 299 | 0 | 369 | 298 | 240 | 209 | 301 | 222 | 505 | 327 | 303 | 302 | 321 | 307 | 229 | 296 | 297 | 287 | 265 | 298 | 292 | 961 | 316 | 269 | 298 | 288 | 296 | 299 |
| 258 | 369 | 0 | 292 | 343 | 339 | 358 | 372 | 584 | 398 | 358 | 372 | 376 | 373 | 378 | 311 | 344 | 299 | 341 | 386 | 344 | 997 | 356 | 401 | 347 | 273 | 238 | 304 |
| 269 | 298 | 292 | 0 | 293 | 348 | 292 | 298 | 500 | 306 | 273 | 279 | 283 | 278 | 297 | 271 | 275 | 253 | 327 | 309 | 257 | 961 | 289 | 314 | 270 | 260 | 311 | 264 |
| 315 | 240 | 343 | 293 | 0 | 217 | 318 | 236 | 506 | 316 | 318 | 309 | 314 | 312 | 249 | 312 | 302 | 307 | 289 | 304 | 297 | 961 | 335 | 272 | 311 | 309 | 316 | 312 |
| 324 | 209 | 339 | 348 | 217 | 0 | 341 | 198 | 510 | 358 | 351 | 337 | 367 | 357 | 267 | 334 | 346 | 332 | 252 | 320 | 337 | 961 | 354 | 224 | 346 | 336 | 355 | 350 |
| 285 | 301 | 358 | 292 | 318 | 341 | 0 | 304 | 507 | 318 | 292 | 290 | 309 | 292 | 312 | 275 | 288 | 279 | 320 | 317 | 270 | 961 | 310 | 333 | 279 | 280 | 285 | 276 |
| 295 | 222 | 372 | 298 | 236 | 198 | 304 | 0 | 505 | 323 | 314 | 297 | 324 | 316 | 222 | 301 | 297 | 294 | 261 | 301 | 295 | 999 | 319 | 268 | 304 | 298 | 331 | 310 |
| 503 | 505 | 584 | 500 | 506 | 510 | 507 | 505 | 0 | 511 | 502 | 505 | 506 | 502 | 501 | 502 | 506 | 503 | 504 | 510 | 501 | 999 | 504 | 508 | 500 | 501 | 504 | 503 |
| 327 | 327 | 398 | 306 | 316 | 358 | 318 | 323 | 511 | 0 | 326 | 322 | 302 | 311 | 330 | 311 | 310 | 318 | 364 | 329 | 304 | 961 | 343 | 329 | 320 | 301 | 354 | 311 |
| 268 | 303 | 358 | 273 | 318 | 351 | 292 | 314 | 502 | 326 | 0 | 282 | 295 | 287 | 308 | 281 | 284 | 273 | 330 | 326 | 267 | 961 | 275 | 332 | 272 | 267 | 312 | 268 |
| 271 | 302 | 372 | 279 | 309 | 337 | 290 | 297 | 505 | 322 | 282 | 0 | 294 | 303 | 306 | 285 | 283 | 274 | 323 | 315 | 274 | 999 | 289 | 335 | 290 | 277 | 271 | 290 |
| 302 | 321 | 376 | 283 | 314 | 367 | 309 | 324 | 506 | 302 | 295 | 294 | 0 | 300 | 318 | 296 | 297 | 298 | 256 | 320 | 281 | 999 | 313 | 321 | 297 | 288 | 300 | 287 |
| 293 | 307 | 373 | 278 | 312 | 357 | 292 | 316 | 502 | 311 | 287 | 303 | 300 | 0 | 311 | 277 | 294 | 285 | 335 | 323 | 264 | 999 | 313 | 328 | 264 | 274 | 296 | 243 |
| 305 | 229 | 378 | 297 | 249 | 267 | 312 | 222 | 501 | 330 | 308 | 306 | 318 | 311 | 0 | 301 | 305 | 297 | 264 | 301 | 301 | 961 | 313 | 266 | 300 | 295 | 302 | 299 |
| 283 | 296 | 311 | 271 | 312 | 334 | 275 | 301 | 502 | 311 | 281 | 285 | 296 | 277 | 301 | 0 | 285 | 267 | 321 | 315 | 255 | 999 | 294 | 327 | 256 | 263 | 312 | 259 |
| 262 | 297 | 344 | 275 | 302 | 346 | 288 | 297 | 506 | 310 | 284 | 283 | 297 | 294 | 305 | 285 | 0 | 272 | 326 | 312 | 273 | 961 | 294 | 327 | 278 | 257 | 297 | 286 |
| 261 | 287 | 299 | 253 | 307 | 332 | 279 | 294 | 503 | 318 | 273 | 274 | 298 | 285 | 297 | 267 | 272 | 0 | 307 | 318 | 261 | 961 | 287 | 326 | 265 | 257 | 298 | 273 |
| 302 | 265 | 341 | 327 | 289 | 252 | 320 | 261 | 504 | 364 | 330 | 323 | 356 | 335 | 264 | 321 | 326 | 307 | 0 | 326 | 317 | 998 | 327 | 297 | 322 | 315 | 333 | 326 |
| 317 | 298 | 386 | 309 | 304 | 320 | 317 | 301 | 510 | 329 | 326 | 315 | 320 | 323 | 301 | 315 | 312 | 318 | 326 | 0 | 303 | 961 | 334 | 322 | 318 | 313 | 318 | 316 |
| 266 | 292 | 344 | 257 | 297 | 337 | 270 | 295 | 501 | 304 | 267 | 274 | 281 | 264 | 301 | 255 | 273 | 261 | 317 | 303 | 0 | 961 | 288 | 312 | 259 | 252 | 306 | 240 |
| 961 | 961 | 997 | 961 | 961 | 961 | 961 | 999 | 999 | 961 | 961 | 999 | 999 | 999 | 961 | 999 | 961 | 961 | 998 | 961 | 961 | 0 | 995 | 961 | 961 | 999 | 989 | 961 |
| 272 | 316 | 356 | 289 | 335 | 354 | 310 | 319 | 504 | 343 | 275 | 289 | 313 | 313 | 313 | 294 | 294 | 287 | 327 | 334 | 288 | 995 | 0 | 338 | 287 | 281 | 296 | 289 |
| 328 | 269 | 401 | 314 | 272 | 224 | 333 | 268 | 508 | 329 | 332 | 335 | 321 | 328 | 266 | 327 | 327 | 326 | 297 | 322 | 312 | 961 | 338 | 0 | 326 | 322 | 364 | 322 |
| 266 | 298 | 347 | 270 | 311 | 346 | 279 | 304 | 500 | 320 | 272 | 290 | 297 | 264 | 300 | 256 | 278 | 265 | 322 | 318 | 259 | 961 | 287 | 326 | 0 | 252 | 308 | 242 |
| 242 | 288 | 273 | 260 | 309 | 336 | 280 | 298 | 501 | 301 | 267 | 277 | 288 | 274 | 295 | 263 | 257 | 257 | 315 | 313 | 252 | 999 | 281 | 322 | 252 | 0 | 244 | 250 |
| 268 | 296 | 238 | 311 | 316 | 355 | 285 | 331 | 504 | 354 | 312 | 271 | 300 | 296 | 302 | 312 | 297 | 298 | 333 | 318 | 306 | 989 | 296 | 364 | 308 | 244 | 0 | 276 |
| 274 | 299 | 304 | 264 | 312 | 350 | 276 | 310 | 503 | 311 | 268 | 290 | 287 | 243 | 299 | 259 | 286 | 273 | 326 | 316 | 240 | 961 | 289 | 322 | 242 | 250 | 276 | 0 |

$$\text{badness } \delta = 0.056$$

Table 2. The initial matrix to carry out all further calculations

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | –1 | 258 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 262 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 |
| –1 | 0 | –1 | –1 | –1 | –1 | –1 | 222 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 297 | –1 | –1 | –1 | 292 | –1 | –1 | 269 | –1 | –1 | –1 | –1 |
| 258 | –1 | 0 | –1 | 343 | 339 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 273 | –1 | –1 |
| –1 | –1 | –1 | 0 | –1 | –1 | 292 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 257 | –1 | –1 | –1 | –1 | –1 | –1 | –1 |
| –1 | –1 | 343 | –1 | 0 | –1 | –1 | 236 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 304 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 |
| –1 | –1 | 339 | –1 | –1 | 0 | –1 | –1 | –1 | –1 | 351 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 337 | –1 | –1 | –1 | –1 | –1 | –1 | –1 |
| –1 | –1 | –1 | 292 | –1 | –1 | 0 | –1 | –1 | 318 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 279 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 |
| –1 | 222 | –1 | –1 | 236 | –1 | –1 | 0 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 298 | –1 | –1 |
| –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 0 | –1 | –1 | 505 | 506 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 |
| –1 | –1 | –1 | –1 | –1 | –1 | 318 | –1 | –1 | 0 | –1 | –1 | 302 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 |
| –1 | –1 | –1 | –1 | –1 | 351 | –1 | –1 | –1 | –1 | 0 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 326 | –1 | –1 | –1 | 332 | –1 | –1 | –1 | 268 |
| –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 505 | –1 | –1 | 0 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 315 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 |
| –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 506 | 302 | –1 | –1 | 0 | –1 | –1 | –1 | 297 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 |
| –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 0 | –1 | –1 | –1 | –1 | –1 | –1 | 264 | 999 | –1 | –1 | –1 | –1 | –1 | –1 |
| –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 0 | 301 | –1 | –1 | 264 | –1 | –1 | –1 | –1 | –1 | 30 0 | –1 | –1 | –1 |
| –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 301 | 0 | –1 | –1 | 321 | –1 | –1 | –1 | –1 | –1 | 256 | –1 | –1 | –1 |
| 262 | 297 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 297 | –1 | –1 | –1 | 0 | –1 | 326 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 |
| –1 | –1 | –1 | –1 | –1 | –1 | 279 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 0 | –1 | –1 | –1 | –1 | 287 | –1 | 265 | –1 | –1 | –1 |
| –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 264 | 321 | 326 | –1 | 0 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 |
| –1 | –1 | –1 | –1 | 304 | –1 | –1 | –1 | –1 | –1 | 326 | 315 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 0 | 303 | –1 | –1 | –1 | –1 | –1 | –1 | –1 |
| –1 | 292 | –1 | 257 | –1 | 337 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 264 | –1 | –1 | –1 | –1 | –1 | 303 | 0 | –1 | –1 | –1 | –1 | –1 | –1 | –1 |
| –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 999 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 0 | 995 | –1 | –1 | –1 | –1 | –1 |
| –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 287 | –1 | –1 | –1 | 995 | 0 | –1 | –1 | –1 | –1 | –1 |
| –1 | 269 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 332 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 0 | –1 | –1 | –1 | –1 |
| –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 30 0 | 256 | –1 | 265 | –1 | –1 | –1 | –1 | –1 | –1 | 0 | –1 | –1 | –1 |
| –1 | –1 | 273 | –1 | –1 | –1 | –1 | 298 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 0 | 244 | –1 |
| –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 244 | 0 | –1 |
| –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 268 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | –1 | 0 |

Table 3. The result of the “simplest” algorithm without additional heuristics

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | 3139 | 2580 | 3340 | 3285 | 3415 | 3340 | 3261 | 5058 | 3248 | 3449 | 4218 | 3155 | 3340 | 3353 | 3353 | 2620 | 3340 | 3322 | 3379 | 3340 | 9970 | 5064 | 3294 | 3384 | 3253 | 3253 | 3449 |
| 3139 | 0000 | 3309 | 3130 | 3039 | 3423 | 3235 | 2220 | 5058 | 3204 | 3402 | 4201 | 3129 | 3130 | 3345 | 3345 | 2970 | 3288 | 3316 | 3327 | 2920 | 9970 | 5047 | 2690 | 3367 | 3165 | 3209 | 3426 |
| 2580 | 3309 | 0000 | 3354 | 3430 | 3390 | 3334 | 3205 | 5058 | 3271 | 3473 | 4228 | 3192 | 3354 | 3348 | 3348 | 2964 | 3328 | 3319 | 3435 | 3413 | 9970 | 5057 | 3358 | 3377 | 2730 | 3097 | 3449 |
| 3340 | 3130 | 3354 | 0000 | 3293 | 3406 | 2920 | 3305 | 5058 | 3272 | 3444 | 4209 | 3282 | 3116 | 3349 | 3349 | 3325 | 3225 | 3338 | 3340 | 2570 | 9970 | 5036 | 3330 | 3356 | 3335 | 3309 | 3442 |
| 3285 | 3039 | 3430 | 3293 | 0000 | 3431 | 3324 | 2360 | 5058 | 3304 | 3411 | 4177 | 3285 | 3293 | 3367 | 3367 | 3251 | 3335 | 3342 | 3040 | 3271 | 9970 | 5055 | 3294 | 3383 | 3205 | 3288 | 3435 |
| 3415 | 3423 | 3390 | 3406 | 3431 | 0000 | 3413 | 3415 | 5058 | 3413 | 3510 | 4219 | 3413 | 3406 | 3413 | 3413 | 3415 | 3413 | 3414 | 3440 | 3370 | 9970 | 5066 | 3434 | 3415 | 3415 | 3413 | 3452 |
| 3340 | 3235 | 3334 | 2920 | 3324 | 3413 | 0000 | 3326 | 5057 | 3180 | 3448 | 4211 | 3262 | 3299 | 3354 | 3354 | 3344 | 2790 | 3342 | 3378 | 3314 | 9970 | 4998 | 3348 | 3312 | 3347 | 3329 | 3443 |
| 3261 | 2220 | 3205 | 3305 | 2360 | 3415 | 3326 | 0000 | 5058 | 3294 | 3449 | 4206 | 3268 | 3308 | 3360 | 3360 | 3220 | 3334 | 3336 | 3338 | 3301 | 9970 | 5050 | 3253 | 3374 | 2980 | 3249 | 3443 |
| 5058 | 5058 | 5058 | 5058 | 5058 | 5058 | 5057 | 5058 | 0000 | 5058 | 5058 | 5050 | 5060 | 5058 | 5058 | 5058 | 5058 | 5058 | 5058 | 5055 | 5058 | 9970 | 5232 | 5058 | 5058 | 5058 | 5058 | 5058 |
| 3248 | 3204 | 3271 | 3272 | 3304 | 3413 | 3180 | 3294 | 5058 | 0000 | 3448 | 4210 | 3020 | 3489 | 3545 | 3545 | 3287 | 3505 | 3511 | 3376 | 3336 | 9970 | 5082 | 3335 | 3542 | 3323 | 3476 | 3604 |
| 3449 | 3402 | 3473 | 3444 | 3411 | 3510 | 3448 | 3449 | 5058 | 3448 | 0000 | 4200 | 3448 | 3595 | 3616 | 3616 | 3449 | 3615 | 3601 | 3260 | 3414 | 9970 | 5091 | 3320 | 3604 | 3449 | 3597 | 2680 |
| 4218 | 4201 | 4228 | 4209 | 4177 | 4219 | 4211 | 4206 | 5050 | 4210 | 4200 | 0000 | 4209 | 4277 | 4284 | 4284 | 4209 | 4285 | 4279 | 3150 | 4193 | 9970 | 5142 | 4207 | 4279 | 4208 | 4279 | 4285 |
| 3155 | 3129 | 3192 | 3282 | 3285 | 3413 | 3262 | 3268 | 5060 | 3020 | 3448 | 4209 | 0000 | 3551 | 3613 | 3613 | 2970 | 3587 | 3575 | 3375 | 3329 | 9970 | 5093 | 3326 | 3605 | 3310 | 3537 | 3661 |
| 3340 | 3130 | 3354 | 3116 | 3293 | 3406 | 3299 | 3308 | 5058 | 3489 | 3595 | 4277 | 3551 | 0000 | 3615 | 3615 | 3557 | 3588 | 3598 | 3566 | 2640 | 9990 | 6789 | 3559 | 3613 | 3562 | 3578 | 3672 |
| 3353 | 3345 | 3348 | 3349 | 3367 | 3413 | 3354 | 3360 | 5058 | 3545 | 3616 | 4284 | 3613 | 3615 | 0000 | 3010 | 3582 | 3576 | 2640 | 3622 | 3615 | 9971 | 5347 | 4018 | 3000 | 4016 | 3643 | 3674 |
| 3353 | 3345 | 3348 | 3349 | 3367 | 3413 | 3354 | 3360 | 5058 | 3545 | 3616 | 4284 | 3613 | 3615 | 3010 | 0000 | 3582 | 3556 | 3210 | 3622 | 3615 | 9971 | 5346 | 4018 | 2560 | 4016 | 3643 | 3674 |
| 2620 | 2970 | 2964 | 3325 | 3251 | 3415 | 3344 | 3220 | 5058 | 3287 | 3449 | 4209 | 2970 | 3557 | 3582 | 3582 | 0000 | 3596 | 3260 | 3374 | 3317 | 9970 | 5093 | 3305 | 3428 | 3276 | 3545 | 3664 |
| 3340 | 3288 | 3328 | 3225 | 3335 | 3413 | 2790 | 3334 | 5058 | 3505 | 3615 | 4285 | 3587 | 3588 | 3576 | 3556 | 3596 | 0000 | 3569 | 3614 | 3594 | 9970 | 2870 | 3600 | 2650 | 3598 | 3600 | 3673 |
| 3322 | 3316 | 3319 | 3338 | 3342 | 3414 | 3342 | 3336 | 5058 | 3511 | 3601 | 4279 | 3575 | 3598 | 2640 | 3210 | 3260 | 3569 | 0000 | 3600 | 3588 | 9971 | 5347 | 3992 | 3105 | 3988 | 3619 | 3673 |
| 3379 | 3327 | 3435 | 3340 | 3040 | 3440 | 3378 | 3338 | 5055 | 3376 | 3260 | 3150 | 3375 | 3566 | 3622 | 3622 | 3374 | 3614 | 3600 | 0000 | 3030 | 9970 | 5094 | 3362 | 3616 | 3366 | 3594 | 3655 |
| 3340 | 2920 | 3413 | 2570 | 3271 | 3370 | 3314 | 3301 | 5058 | 3336 | 3414 | 4193 | 3329 | 2640 | 3615 | 3615 | 3317 | 3594 | 3588 | 3030 | 0000 | 9970 | 5093 | 3310 | 3609 | 3340 | 3577 | 3662 |
| 9970 | 9970 | 9970 | 9970 | 9970 | 9970 | 9970 | 9970 | 9970 | 9970 | 9970 | 9970 | 9970 | 9990 | 9971 | 9971 | 9970 | 9970 | 9971 | 9970 | 9970 | 0000 | 9950 | 9970 | 9971 | 9970 | 9971 | 9971 |
| 5064 | 5047 | 5057 | 5036 | 5055 | 5066 | 4998 | 5050 | 5232 | 5082 | 5091 | 5142 | 5093 | 6789 | 5347 | 5346 | 5093 | 2870 | 5347 | 5094 | 5093 | 9950 | 0000 | 5093 | 5312 | 5093 | 5339 | 5353 |
| 3294 | 2690 | 3358 | 3330 | 3294 | 3434 | 3348 | 3253 | 5058 | 3335 | 3320 | 4207 | 3326 | 3559 | 4018 | 4018 | 3305 | 3600 | 3992 | 3362 | 3310 | 9970 | 5093 | 0000 | 4013 | 3328 | 3959 | 4050 |
| 3384 | 3367 | 3377 | 3356 | 3383 | 3415 | 3312 | 3374 | 5058 | 3542 | 3604 | 4279 | 3605 | 3613 | 3000 | 2560 | 3428 | 2650 | 3105 | 3616 | 3609 | 9971 | 5312 | 4013 | 0000 | 4012 | 3972 | 4020 |
| 3253 | 3165 | 2730 | 3335 | 3205 | 3415 | 3347 | 2980 | 5058 | 3323 | 3449 | 4208 | 3310 | 3562 | 4016 | 4016 | 3276 | 3598 | 3988 | 3366 | 3340 | 9970 | 5093 | 3328 | 4012 | 0000 | 2440 | 4053 |
| 3253 | 3209 | 3097 | 3309 | 3288 | 3413 | 3329 | 3249 | 5058 | 3476 | 3597 | 4279 | 3537 | 3578 | 3643 | 3643 | 3545 | 3600 | 3619 | 3594 | 3577 | 9971 | 5339 | 3959 | 3972 | 2440 | 0000 | 4021 |
| 3449 | 3426 | 3449 | 3442 | 3435 | 3452 | 3443 | 3443 | 5058 | 3604 | 2680 | 4285 | 3661 | 3672 | 3674 | 3674 | 3664 | 3673 | 3673 | 3655 | 3662 | 9971 | 5353 | 4050 | 4020 | 4053 | 4021 | 0000 |

badness $\delta = 0.043$

Table 4. The result of the “simplest” algorithm with additional heuristics, $R = 0.9$

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | 3177 | 2580 | 3328 | 3266 | 3420 | 3253 | 3191 | 5056 | 3212 | 3399 | 4028 | 2970 | 3325 | 3251 | 3251 | 2620 | 3251 | 3260 | 3336 | 3293 | 9976 | 6099 | 3314 | 3251 | 3121 | 3274 | 3425 |
| 3177 | 0000 | 3384 | 2920 | 3138 | 3440 | 3112 | 2220 | 5057 | 3185 | 3353 | 4121 | 3074 | 2920 | 3228 | 3225 | 2970 | 3223 | 3260 | 3274 | 2920 | 9972 | 5536 | 2690 | 3223 | 3296 | 3339 | 3419 |
| 2580 | 3384 | 0000 | 3410 | 3430 | 3390 | 3300 | 3205 | 5058 | 3276 | 3429 | 4205 | 3140 | 3410 | 3279 | 3279 | 3002 | 3278 | 3285 | 3435 | 3367 | 9974 | 5822 | 3384 | 3278 | 2730 | 2730 | 3435 |
| 3328 | 2920 | 3410 | 0000 | 3292 | 3440 | 2920 | 3271 | 5056 | 3247 | 3353 | 4029 | 3314 | 2780 | 3309 | 3307 | 3303 | 3295 | 3333 | 3279 | 2570 | 9972 | 5554 | 3036 | 3303 | 3345 | 3358 | 3417 |
| 3266 | 3138 | 3430 | 3292 | 0000 | 3435 | 3258 | 2360 | 5058 | 3261 | 3385 | 4134 | 3240 | 3292 | 3274 | 3273 | 3239 | 3273 | 3289 | 3040 | 3229 | 9976 | 6168 | 3282 | 3273 | 3205 | 3317 | 3422 |
| 3420 | 3440 | 3390 | 3440 | 3435 | 0000 | 3418 | 3422 | 5056 | 3418 | 3510 | 3994 | 3420 | 3440 | 3416 | 3416 | 3420 | 3416 | 3420 | 3440 | 3370 | 9976 | 6220 | 3440 | 3416 | 3416 | 3416 | 3510 |
| 3253 | 3112 | 3300 | 2920 | 3258 | 3418 | 0000 | 3217 | 5057 | 3180 | 3391 | 4096 | 3247 | 3430 | 3153 | 3124 | 3223 | 2790 | 3308 | 3309 | 3096 | 9977 | 6221 | 3274 | 3049 | 3301 | 3472 | 3548 |
| 3191 | 2220 | 3205 | 3271 | 2360 | 3422 | 3217 | 0000 | 5056 | 3203 | 3386 | 4007 | 3129 | 3271 | 3241 | 3241 | 3122 | 3240 | 3260 | 3297 | 3213 | 9974 | 5819 | 3205 | 3240 | 2980 | 3168 | 3424 |
| 5056 | 5057 | 5058 | 5056 | 5058 | 5056 | 5057 | 5056 | 0000 | 5056 | 5056 | 5050 | 5060 | 5057 | 5057 | 5057 | 5057 | 5056 | 5057 | 5055 | 5057 | 9974 | 5975 | 5056 | 5057 | 5057 | 5056 | 5056 |
| 3212 | 3185 | 3276 | 3247 | 3261 | 3418 | 3180 | 3203 | 5056 | 0000 | 3392 | 4036 | 3020 | 3462 | 3242 | 3242 | 3192 | 3242 | 3306 | 3321 | 3233 | 9974 | 5871 | 3300 | 3242 | 3287 | 3468 | 3547 |
| 3399 | 3353 | 3429 | 3353 | 3385 | 3510 | 3391 | 3386 | 5056 | 3392 | 0000 | 3985 | 3398 | 3353 | 3397 | 3397 | 3398 | 3397 | 3399 | 3260 | 3385 | 9976 | 6217 | 3320 | 3397 | 3407 | 3410 | 2680 |
| 4028 | 4121 | 4205 | 4029 | 4134 | 3994 | 4096 | 4007 | 5050 | 4036 | 3985 | 0000 | 4195 | 4094 | 4090 | 4090 | 4107 | 4037 | 4102 | 3150 | 4123 | 9974 | 5924 | 4018 | 4090 | 4116 | 4039 | 4040 |
| 2970 | 3074 | 3140 | 3314 | 3240 | 3420 | 3247 | 3129 | 5060 | 3020 | 3398 | 4195 | 0000 | 3313 | 3249 | 3249 | 2970 | 3249 | 3260 | 3330 | 3273 | 9976 | 6099 | 3302 | 3249 | 3180 | 3308 | 3425 |
| 3325 | 2920 | 3410 | 2780 | 3292 | 3440 | 3430 | 3271 | 5057 | 3462 | 3353 | 4094 | 3313 | 0000 | 3469 | 3469 | 3300 | 3468 | 3333 | 3279 | 2640 | 9990 | 6729 | 3036 | 3469 | 3345 | 3358 | 3421 |
| 3251 | 3228 | 3279 | 3309 | 3274 | 3416 | 3153 | 3241 | 5057 | 3242 | 3397 | 4090 | 3249 | 3469 | 0000 | 3010 | 3241 | 3103 | 2640 | 3330 | 3267 | 9974 | 5881 | 3314 | 3000 | 3269 | 3472 | 3550 |
| 3251 | 3225 | 3279 | 3307 | 3273 | 3416 | 3124 | 3241 | 5057 | 3242 | 3397 | 4090 | 3249 | 3469 | 3010 | 0000 | 3241 | 2988 | 3210 | 3330 | 3265 | 9974 | 5876 | 3314 | 2560 | 3269 | 3472 | 3550 |
| 2620 | 2970 | 3002 | 3303 | 3239 | 3420 | 3223 | 3122 | 5057 | 3192 | 3398 | 4107 | 2970 | 3300 | 3241 | 3241 | 0000 | 3241 | 3260 | 3328 | 3259 | 9974 | 5840 | 3289 | 3241 | 3156 | 3296 | 3425 |
| 3251 | 3223 | 3278 | 3295 | 3273 | 3416 | 2790 | 3240 | 5056 | 3242 | 3397 | 4037 | 3249 | 3468 | 3103 | 2988 | 3241 | 0000 | 3206 | 3330 | 3262 | 9970 | 2870 | 3313 | 2650 | 3269 | 3469 | 3548 |
| 3260 | 3260 | 3285 | 3333 | 3289 | 3420 | 3308 | 3260 | 5057 | 3306 | 3399 | 4102 | 3260 | 3333 | 2640 | 3210 | 3260 | 3206 | 0000 | 3338 | 3304 | 9976 | 6085 | 3326 | 3105 | 3273 | 3330 | 3425 |
| 3336 | 3274 | 3435 | 3279 | 3040 | 3440 | 3309 | 3297 | 5055 | 3321 | 3260 | 3150 | 3330 | 3279 | 3330 | 3330 | 3328 | 3330 | 3338 | 0000 | 3030 | 9974 | 5793 | 3297 | 3330 | 3344 | 3367 | 3385 |
| 3293 | 2920 | 3367 | 2570 | 3229 | 3370 | 3096 | 3213 | 5057 | 3233 | 3385 | 4123 | 3273 | 2640 | 3267 | 3265 | 3259 | 3262 | 3304 | 3030 | 0000 | 9980 | 6657 | 3153 | 3262 | 3314 | 3332 | 3427 |
| 9976 | 9972 | 9974 | 9972 | 9976 | 9976 | 9977 | 9974 | 9974 | 9974 | 9976 | 9974 | 9976 | 9990 | 9974 | 9974 | 9974 | 9970 | 9976 | 9974 | 9980 | 0000 | 9950 | 9973 | 9974 | 9976 | 9975 | 9975 |
| 6099 | 5536 | 5822 | 5554 | 6168 | 6220 | 6221 | 5819 | 5975 | 5871 | 6217 | 5924 | 6099 | 6729 | 5881 | 5876 | 5840 | 2870 | 6085 | 5793 | 6657 | 9950 | 0000 | 5668 | 5871 | 6118 | 6060 | 6063 |
| 3314 | 2690 | 3384 | 3036 | 3282 | 3440 | 3274 | 3205 | 5056 | 3300 | 3320 | 4018 | 3302 | 3036 | 3314 | 3314 | 3289 | 3313 | 3326 | 3297 | 3153 | 9973 | 5668 | 0000 | 3313 | 3334 | 3348 | 3412 |
| 3251 | 3223 | 3278 | 3303 | 3273 | 3416 | 3049 | 3240 | 5057 | 3242 | 3397 | 4090 | 3249 | 3469 | 3000 | 2560 | 3241 | 2650 | 3105 | 3330 | 3262 | 9974 | 5871 | 3313 | 0000 | 3269 | 3472 | 3550 |
| 3121 | 3296 | 2730 | 3345 | 3205 | 3416 | 3301 | 2980 | 5057 | 3287 | 3407 | 4116 | 3180 | 3345 | 3269 | 3269 | 3156 | 3269 | 3273 | 3344 | 3314 | 9976 | 6118 | 3334 | 3269 | 0000 | 2440 | 3426 |
| 3274 | 3339 | 2730 | 3358 | 3317 | 3416 | 3472 | 3168 | 5056 | 3468 | 3410 | 4039 | 3308 | 3358 | 3472 | 3472 | 3296 | 3469 | 3330 | 3367 | 3332 | 9975 | 6060 | 3348 | 3472 | 2440 | 0000 | 3426 |
| 3425 | 3419 | 3435 | 3417 | 3422 | 3510 | 3548 | 3424 | 5056 | 3547 | 2680 | 4040 | 3425 | 3421 | 3550 | 3550 | 3425 | 3548 | 3425 | 3385 | 3427 | 9975 | 6063 | 3412 | 3550 | 3426 | 3426 | 0000 |

badness $\delta = 0.030$

Table 5. The result of the “simplest” algorithm with additional heuristics, R = 0.7

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | 3177 | 2580 | 3328 | 3266 | 3420 | 3247 | 3191 | 5056 | 3205 | 3399 | 4028 | 2970 | 3325 | 3251 | 3251 | 2620 | 3251 | 3260 | 3335 | 3293 | 9976 | 6156 | 3314 | 3251 | 3121 | 3274 | 3426 |
| 3177 | 0000 | 3384 | 2920 | 3138 | 3440 | 3112 | 2220 | 5057 | 3185 | 3353 | 4121 | 3074 | 2920 | 3233 | 3229 | 2970 | 3227 | 3260 | 3274 | 2920 | 9972 | 5538 | 2690 | 3227 | 3296 | 3339 | 3419 |
| 2580 | 3384 | 0000 | 3410 | 3430 | 3390 | 3297 | 3205 | 5058 | 3273 | 3429 | 4205 | 3140 | 3410 | 3279 | 3279 | 3002 | 3279 | 3285 | 3435 | 3367 | 9974 | 5883 | 3384 | 3279 | 2730 | 2730 | 3437 |
| 3328 | 2920 | 3410 | 0000 | 3292 | 3440 | 2920 | 3271 | 5057 | 3247 | 3353 | 4093 | 3314 | 2780 | 3310 | 3308 | 3303 | 3296 | 3333 | 3279 | 2570 | 9978 | 6440 | 3036 | 3303 | 3345 | 3358 | 3422 |
| 3266 | 3138 | 3430 | 3292 | 0000 | 3435 | 3258 | 2360 | 5058 | 3261 | 3385 | 4134 | 3240 | 3292 | 3275 | 3275 | 3239 | 3274 | 3289 | 3040 | 3229 | 9977 | 6224 | 3282 | 3274 | 3205 | 3317 | 3423 |
| 3420 | 3440 | 3390 | 3440 | 3435 | 0000 | 3418 | 3422 | 5056 | 3418 | 3510 | 3994 | 3420 | 3440 | 3416 | 3416 | 3420 | 3416 | 3420 | 3440 | 3370 | 9974 | 5822 | 3440 | 3416 | 3416 | 3416 | 3510 |
| 3247 | 3112 | 3297 | 2920 | 3258 | 3418 | 0000 | 3220 | 5057 | 3180 | 3391 | 4091 | 3247 | 3428 | 3153 | 3124 | 3230 | 2790 | 3308 | 3307 | 3096 | 9973 | 5647 | 3276 | 3049 | 3302 | 3472 | 3549 |
| 3191 | 2220 | 3205 | 3271 | 2360 | 3422 | 3220 | 0000 | 5056 | 3206 | 3386 | 4007 | 3129 | 3271 | 3243 | 3243 | 3122 | 3243 | 3260 | 3297 | 3213 | 9974 | 5880 | 3205 | 3243 | 2980 | 3168 | 3425 |
| 5056 | 5057 | 5058 | 5057 | 5058 | 5056 | 5057 | 5056 | 0000 | 5056 | 5056 | 5050 | 5060 | 5056 | 5056 | 5056 | 5057 | 5057 | 5056 | 5055 | 5057 | 9975 | 6036 | 5056 | 5056 | 5057 | 5057 | 5057 |
| 3205 | 3185 | 3273 | 3247 | 3261 | 3418 | 3180 | 3206 | 5056 | 0000 | 3392 | 4032 | 3020 | 3457 | 3246 | 3246 | 3199 | 3246 | 3306 | 3320 | 3233 | 9976 | 6143 | 3302 | 3246 | 3289 | 3471 | 3551 |
| 3399 | 3353 | 3429 | 3353 | 3385 | 3510 | 3391 | 3386 | 5056 | 3392 | 0000 | 3985 | 3398 | 3353 | 3397 | 3397 | 3398 | 3397 | 3399 | 3260 | 3385 | 9974 | 5820 | 3320 | 3397 | 3407 | 3410 | 2680 |
| 4028 | 4121 | 4205 | 4093 | 4134 | 3994 | 4091 | 4007 | 5050 | 4032 | 3985 | 0000 | 4195 | 4029 | 4039 | 4039 | 4107 | 4092 | 4028 | 3150 | 4122 | 9975 | 5988 | 4018 | 4039 | 4116 | 4094 | 4095 |
| 2970 | 3074 | 3140 | 3314 | 3240 | 3420 | 3247 | 3129 | 5060 | 3020 | 3398 | 4195 | 0000 | 3313 | 3251 | 3251 | 2970 | 3251 | 3260 | 3329 | 3273 | 9975 | 5917 | 3302 | 3251 | 3180 | 3308 | 3426 |
| 3325 | 2920 | 3410 | 2780 | 3292 | 3440 | 3428 | 3271 | 5056 | 3457 | 3353 | 4029 | 3313 | 0000 | 3466 | 3466 | 3300 | 3468 | 3333 | 3279 | 2640 | 9990 | 6729 | 3036 | 3466 | 3345 | 3358 | 3422 |
| 3251 | 3233 | 3279 | 3310 | 3275 | 3416 | 3153 | 3243 | 5056 | 3246 | 3397 | 4039 | 3251 | 3466 | 0000 | 3010 | 3245 | 3103 | 2640 | 3330 | 3268 | 9976 | 6123 | 3315 | 3000 | 3270 | 3472 | 3551 |
| 3251 | 3229 | 3279 | 3308 | 3275 | 3416 | 3124 | 3243 | 5056 | 3246 | 3397 | 4039 | 3251 | 3466 | 3010 | 0000 | 3245 | 2988 | 3210 | 3330 | 3266 | 9976 | 6119 | 3314 | 2560 | 3270 | 3472 | 3551 |
| 2620 | 2970 | 3002 | 3303 | 3239 | 3420 | 3230 | 3122 | 5057 | 3199 | 3398 | 4107 | 2970 | 3300 | 3245 | 3245 | 0000 | 3245 | 3260 | 3327 | 3259 | 9974 | 5901 | 3289 | 3245 | 3156 | 3296 | 3426 |
| 3251 | 3227 | 3279 | 3296 | 3274 | 3416 | 2790 | 3243 | 5057 | 3246 | 3397 | 4092 | 3251 | 3468 | 3103 | 2988 | 3245 | 0000 | 3206 | 3330 | 3263 | 9970 | 2870 | 3314 | 2650 | 3270 | 3472 | 3551 |
| 3260 | 3260 | 3285 | 3333 | 3289 | 3420 | 3308 | 3260 | 5056 | 3306 | 3399 | 4028 | 3260 | 3333 | 2640 | 3210 | 3260 | 3206 | 0000 | 3338 | 3304 | 9976 | 6141 | 3326 | 3105 | 3273 | 3330 | 3426 |
| 3335 | 3274 | 3435 | 3279 | 3040 | 3440 | 3307 | 3297 | 5055 | 3320 | 3260 | 3150 | 3329 | 3279 | 3330 | 3330 | 3327 | 3330 | 3338 | 0000 | 3030 | 9977 | 6229 | 3297 | 3330 | 3344 | 3367 | 3385 |
| 3293 | 2920 | 3367 | 2570 | 3229 | 3370 | 3096 | 3213 | 5057 | 3233 | 3385 | 4122 | 3273 | 2640 | 3268 | 3266 | 3259 | 3263 | 3304 | 3030 | 0000 | 9980 | 6657 | 3153 | 3263 | 3314 | 3332 | 3427 |
| 9976 | 9972 | 9974 | 9978 | 9977 | 9974 | 9973 | 9974 | 9975 | 9976 | 9974 | 9975 | 9975 | 9990 | 9976 | 9976 | 9974 | 9970 | 9976 | 9977 | 9980 | 0000 | 9950 | 9977 | 9976 | 9976 | 9975 | 9975 |
| 6156 | 5538 | 5883 | 6440 | 6224 | 5822 | 5647 | 5880 | 6036 | 6143 | 5820 | 5988 | 5917 | 6729 | 6123 | 6119 | 5901 | 2870 | 6141 | 6229 | 6657 | 9950 | 0000 | 6329 | 6114 | 6174 | 5978 | 5982 |
| 3314 | 2690 | 3384 | 3036 | 3282 | 3440 | 3276 | 3205 | 5056 | 3302 | 3320 | 4018 | 3302 | 3036 | 3315 | 3314 | 3289 | 3314 | 3326 | 3297 | 3153 | 9977 | 6329 | 0000 | 3314 | 3334 | 3348 | 3412 |
| 3251 | 3227 | 3279 | 3303 | 3274 | 3416 | 3049 | 3243 | 5056 | 3246 | 3397 | 4039 | 3251 | 3466 | 3000 | 2560 | 3245 | 2650 | 3105 | 3330 | 3263 | 9976 | 6114 | 3314 | 0000 | 3270 | 3472 | 3551 |
| 3121 | 3296 | 2730 | 3345 | 3205 | 3416 | 3302 | 2980 | 5057 | 3289 | 3407 | 4116 | 3180 | 3345 | 3270 | 3270 | 3156 | 3270 | 3273 | 3344 | 3314 | 9976 | 6174 | 3334 | 3270 | 0000 | 2440 | 3427 |
| 3274 | 3339 | 2730 | 3358 | 3317 | 3416 | 3472 | 3168 | 5057 | 3471 | 3410 | 4094 | 3308 | 3358 | 3472 | 3472 | 3296 | 3472 | 3330 | 3367 | 3332 | 9975 | 5978 | 3348 | 3472 | 2440 | 0000 | 3427 |
| 3426 | 3419 | 3437 | 3422 | 3423 | 3510 | 3549 | 3425 | 5057 | 3551 | 2680 | 4095 | 3426 | 3422 | 3551 | 3551 | 3426 | 3551 | 3426 | 3385 | 3427 | 9975 | 5982 | 3412 | 3551 | 3427 | 3427 | 0000 |

badness $\delta = 0.030$

Table 6. The result of the “simple” algorithm without additional heuristics

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | 3139 | 2580 | 3328 | 3285 | 3415 | 3328 | 3261 | 5058 | 3174 | 3449 | 3179 | 2863 | 3328 | 3309 | 3309 | 2620 | 3328 | 3293 | 3335 | 3328 | 9970 | 3328 | 3294 | 3325 | 2705 | 2705 | 3449 |
| 3139 | 0000 | 3309 | 2840 | 2453 | 3423 | 2840 | 2220 | 5058 | 3156 | 3402 | 3163 | 3113 | 2840 | 3176 | 2856 | 2970 | 2804 | 3219 | 3024 | 2920 | 9970 | 2739 | 2690 | 2864 | 3165 | 3152 | 3426 |
| 2580 | 3309 | 0000 | 3350 | 3430 | 3390 | 3329 | 3205 | 5058 | 3208 | 3473 | 3405 | 2834 | 3350 | 3308 | 3308 | 2695 | 3322 | 3295 | 3435 | 3413 | 9970 | 3327 | 3358 | 3320 | 2730 | 2650 | 3449 |
| 3328 | 2840 | 3350 | 0000 | 2852 | 3406 | 2920 | 2832 | 5058 | 3255 | 3444 | 3291 | 3260 | 2533 | 3222 | 2858 | 3281 | 2827 | 3257 | 2933 | 2570 | 9970 | 2767 | 2835 | 2853 | 3272 | 3275 | 3442 |
| 3285 | 2453 | 3430 | 2852 | 0000 | 3431 | 2845 | 2360 | 5058 | 3260 | 3411 | 3239 | 3257 | 2800 | 3233 | 2826 | 3228 | 2774 | 3261 | 3040 | 2952 | 9970 | 2735 | 2672 | 2813 | 3205 | 3269 | 3435 |
| 3415 | 3423 | 3390 | 3406 | 3431 | 0000 | 3413 | 3415 | 5058 | 3413 | 3510 | 3322 | 3413 | 3406 | 3413 | 3413 | 3415 | 3413 | 3413 | 3440 | 3370 | 9970 | 3412 | 3434 | 3413 | 3415 | 3413 | 3452 |
| 3328 | 2840 | 3329 | 2920 | 2845 | 3413 | 0000 | 2830 | 5057 | 3180 | 3448 | 3187 | 3228 | 2801 | 3247 | 2831 | 3277 | 2790 | 3274 | 2923 | 2875 | 9970 | 2770 | 2818 | 2819 | 3270 | 3291 | 3443 |
| 3261 | 2220 | 3205 | 2832 | 2360 | 3415 | 2830 | 0000 | 5058 | 3247 | 3449 | 3148 | 3244 | 2761 | 3242 | 2790 | 3216 | 2767 | 3264 | 2980 | 2906 | 9970 | 2718 | 2625 | 2779 | 2980 | 3231 | 3443 |
| 5058 | 5058 | 5058 | 5058 | 5058 | 5058 | 5057 | 5058 | 0000 | 5058 | 5058 | 5050 | 5060 | 5058 | 5058 | 5058 | 5058 | 5058 | 5058 | 5055 | 5058 | 9970 | 5058 | 5058 | 5058 | 5058 | 5058 | 5058 |
| 3174 | 3156 | 3208 | 3255 | 3260 | 3413 | 3180 | 3247 | 5058 | 0000 | 3448 | 3137 | 3020 | 3141 | 3099 | 3076 | 3249 | 3153 | 3116 | 3306 | 3300 | 9970 | 3101 | 3293 | 3115 | 3259 | 3123 | 3277 |
| 3449 | 3402 | 3473 | 3444 | 3411 | 3510 | 3448 | 3449 | 5058 | 3448 | 0000 | 3252 | 3448 | 3265 | 3604 | 3604 | 3449 | 3281 | 3592 | 3260 | 3414 | 9970 | 3553 | 3320 | 3568 | 3449 | 3595 | 2680 |
| 3179 | 3163 | 3405 | 3291 | 3239 | 3322 | 3187 | 3148 | 5050 | 3137 | 3252 | 0000 | 3116 | 3085 | 3101 | 3076 | 3102 | 3113 | 3117 | 3150 | 3094 | 9970 | 3085 | 3070 | 3103 | 3063 | 3113 | 3233 |
| 2863 | 3113 | 2834 | 3260 | 3257 | 3413 | 3228 | 3244 | 5060 | 3020 | 3448 | 3116 | 0000 | 3095 | 3069 | 3051 | 2970 | 3102 | 3086 | 3306 | 3300 | 9970 | 3065 | 3294 | 3081 | 2912 | 2766 | 3546 |
| 3328 | 2840 | 3350 | 2533 | 2800 | 3406 | 2801 | 2761 | 5058 | 3141 | 3265 | 3085 | 3095 | 0000 | 3048 | 2665 | 3379 | 2682 | 3082 | 3006 | 2640 | 9990 | 2829 | 2926 | 2680 | 3361 | 3096 | 3532 |
| 3309 | 3176 | 3308 | 3222 | 3233 | 3413 | 3247 | 3242 | 5058 | 3099 | 3604 | 3101 | 3069 | 3048 | 0000 | 3010 | 3084 | 3033 | 2640 | 3041 | 3019 | 9971 | 3003 | 2992 | 3000 | 3319 | 3071 | 3556 |
| 3309 | 2856 | 3308 | 2858 | 2826 | 3413 | 2831 | 2790 | 5058 | 3076 | 3604 | 3076 | 3051 | 2665 | 3010 | 0000 | 3360 | 2649 | 3210 | 2974 | 2918 | 9971 | 2891 | 2888 | 2560 | 3304 | 3354 | 3556 |
| 2620 | 2970 | 2695 | 3281 | 3228 | 3415 | 3277 | 3216 | 5058 | 3249 | 3449 | 3102 | 2970 | 3379 | 3084 | 3360 | 0000 | 3385 | 3260 | 3304 | 3290 | 9970 | 3389 | 3283 | 3322 | 2649 | 2720 | 3547 |
| 3328 | 2804 | 3322 | 2827 | 2774 | 3413 | 2790 | 2767 | 5058 | 3153 | 3281 | 3113 | 3102 | 2682 | 3033 | 2649 | 3385 | 0000 | 3064 | 3006 | 2950 | 9970 | 2870 | 2912 | 2650 | 3361 | 3365 | 3537 |
| 3293 | 3219 | 3295 | 3257 | 3261 | 3413 | 3274 | 3264 | 5058 | 3116 | 3592 | 3117 | 3086 | 3082 | 2640 | 3210 | 3260 | 3064 | 0000 | 3072 | 3051 | 9971 | 3044 | 3031 | 3105 | 3342 | 3052 | 3554 |
| 3335 | 3024 | 3435 | 2933 | 3040 | 3440 | 2923 | 2980 | 5055 | 3306 | 3260 | 3150 | 3306 | 3006 | 3041 | 2974 | 3304 | 3006 | 3072 | 0000 | 3030 | 9970 | 3005 | 2932 | 3002 | 3279 | 3369 | 3538 |
| 3328 | 2920 | 3413 | 2570 | 2952 | 3370 | 2875 | 2906 | 5058 | 3300 | 3414 | 3094 | 3300 | 2640 | 3019 | 2918 | 3290 | 2950 | 3051 | 3030 | 0000 | 9970 | 2945 | 2886 | 2940 | 3272 | 3362 | 3546 |
| 9970 | 9970 | 9970 | 9970 | 9970 | 9970 | 9970 | 9970 | 9970 | 9970 | 9970 | 9970 | 9970 | 9990 | 9971 | 9971 | 9970 | 9970 | 9971 | 9970 | 9970 | 0000 | 9950 | 9970 | 9971 | 9970 | 9971 | 9971 |
| 3328 | 2739 | 3327 | 2767 | 2735 | 3412 | 2770 | 2718 | 5058 | 3101 | 3553 | 3085 | 3065 | 2829 | 3003 | 2891 | 3389 | 2870 | 3044 | 3005 | 2945 | 9950 | 0000 | 2659 | 2944 | 3364 | 3306 | 3494 |
| 3294 | 2690 | 3358 | 2835 | 2672 | 3434 | 2818 | 2625 | 5058 | 3293 | 3320 | 3070 | 3294 | 2926 | 2992 | 2888 | 3283 | 2912 | 3031 | 2932 | 2886 | 9970 | 2659 | 0000 | 2941 | 3271 | 3301 | 3471 |
| 3325 | 2864 | 3320 | 2853 | 2813 | 3413 | 2819 | 2779 | 5058 | 3115 | 3568 | 3103 | 3081 | 2680 | 3000 | 2560 | 3322 | 2650 | 3105 | 3002 | 2940 | 9971 | 2944 | 2941 | 0000 | 3343 | 3280 | 3467 |
| 2705 | 3165 | 2730 | 3272 | 3205 | 3415 | 3270 | 2980 | 5058 | 3259 | 3449 | 3063 | 2912 | 3361 | 3319 | 3304 | 2649 | 3361 | 3342 | 3279 | 3272 | 9970 | 3364 | 3271 | 3343 | 0000 | 2440 | 3450 |
| 2705 | 3152 | 2650 | 3275 | 3269 | 3413 | 3291 | 3231 | 5058 | 3123 | 3595 | 3113 | 2766 | 3096 | 3071 | 3354 | 2720 | 3365 | 3052 | 3369 | 3362 | 9971 | 3306 | 3301 | 3280 | 2440 | 0000 | 3443 |
| 3449 | 3426 | 3449 | 3442 | 3435 | 3452 | 3443 | 3443 | 5058 | 3277 | 2680 | 3233 | 3546 | 3532 | 3556 | 3556 | 3547 | 3537 | 3554 | 3538 | 3546 | 9971 | 3494 | 3471 | 3467 | 3450 | 3443 | 0000 |

badness $\delta = 0.022$

Table 7. The result of the "simple" algorithm with additional heuristics, $R = 0.9$

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | 3177 | 2580 | 3301 | 3266 | 3420 | 3231 | 3191 | 5056 | 3196 | 3390 | 3291 | 2970 | 3297 | 3235 | 3235 | 2620 | 3235 | 3260 | 3293 | 3278 | 9976 | 3367 | 3294 | 3235 | 2832 | 2697 | 3424 |
| 3177 | 0000 | 3384 | 2920 | 2294 | 3440 | 2783 | 2220 | 5057 | 3172 | 3353 | 3207 | 3073 | 2920 | 3149 | 2780 | 2970 | 2861 | 3260 | 2977 | 2920 | 9972 | 2903 | 2690 | 2817 | 3273 | 3327 | 3419 |
| 2580 | 3384 | 0000 | 3409 | 3430 | 3390 | 3285 | 3205 | 5058 | 3268 | 3427 | 3453 | 3132 | 3409 | 3264 | 3264 | 2729 | 3264 | 3285 | 3435 | 3365 | 9974 | 3393 | 3384 | 3264 | 2730 | 2730 | 3435 |
| 3301 | 2920 | 3409 | 0000 | 2803 | 3440 | 2920 | 2797 | 5056 | 3226 | 3353 | 3293 | 3272 | 2527 | 3225 | 2804 | 3257 | 2842 | 3313 | 2855 | 2570 | 9972 | 2877 | 2893 | 2821 | 3276 | 3329 | 3417 |
| 3266 | 2294 | 3430 | 2803 | 0000 | 3435 | 2794 | 2360 | 5058 | 3239 | 3383 | 3188 | 3219 | 2766 | 3185 | 2782 | 3197 | 2845 | 3289 | 3040 | 2823 | 9976 | 2765 | 2758 | 2811 | 3205 | 3317 | 3422 |
| 3420 | 3440 | 3390 | 3440 | 3435 | 0000 | 3418 | 3422 | 5056 | 3418 | 3510 | 3437 | 3420 | 3440 | 3416 | 3416 | 3420 | 3416 | 3420 | 3440 | 3370 | 9976 | 3468 | 3440 | 3416 | 3416 | 3416 | 3510 |
| 3231 | 2783 | 3285 | 2920 | 2794 | 3418 | 0000 | 2774 | 5057 | 3180 | 3383 | 3086 | 3226 | 2927 | 3133 | 2725 | 3172 | 2790 | 3299 | 2879 | 2802 | 9977 | 2914 | 2821 | 2768 | 3248 | 3383 | 3509 |
| 3191 | 2220 | 3205 | 2797 | 2360 | 3422 | 2774 | 0000 | 5056 | 3185 | 3377 | 3296 | 3129 | 2765 | 3156 | 2763 | 3122 | 2834 | 3260 | 2930 | 2840 | 9974 | 2744 | 2782 | 2795 | 2980 | 3168 | 3423 |
| 5056 | 5057 | 5058 | 5056 | 5058 | 5056 | 5057 | 5056 | 0000 | 5056 | 5056 | 5050 | 5060 | 5057 | 5057 | 5057 | 5057 | 5056 | 5057 | 5055 | 5057 | 9974 | 5056 | 5056 | 5057 | 5057 | 5056 | 5056 |
| 3196 | 3172 | 3268 | 3226 | 3239 | 3418 | 3180 | 3185 | 5056 | 0000 | 3384 | 3288 | 3020 | 3079 | 3223 | 3223 | 3169 | 3223 | 3299 | 3271 | 3217 | 9974 | 3271 | 3270 | 3223 | 3254 | 3073 | 3509 |
| 3390 | 3353 | 3427 | 3353 | 3383 | 3510 | 3383 | 3377 | 5056 | 3384 | 0000 | 3396 | 3386 | 3353 | 3386 | 3386 | 3386 | 3386 | 3390 | 3260 | 3385 | 9976 | 3490 | 3320 | 3386 | 3391 | 3400 | 2680 |
| 3291 | 3207 | 3453 | 3293 | 3188 | 3437 | 3086 | 3296 | 5050 | 3288 | 3396 | 0000 | 3269 | 3094 | 3059 | 3362 | 3128 | 3268 | 3129 | 3150 | 3266 | 9974 | 3278 | 3294 | 3358 | 3174 | 3314 | 3425 |
| 2970 | 3073 | 3132 | 3272 | 3219 | 3420 | 3226 | 3129 | 5060 | 3020 | 3386 | 3269 | 0000 | 3271 | 3234 | 3234 | 2970 | 3234 | 3260 | 3260 | 3245 | 9976 | 3330 | 3268 | 3234 | 3115 | 2950 | 3423 |
| 3297 | 2920 | 3409 | 2527 | 2766 | 3440 | 2927 | 2765 | 5057 | 3079 | 3353 | 3094 | 3271 | 0000 | 3316 | 2907 | 3253 | 2927 | 3313 | 2811 | 2640 | 9990 | 2842 | 2893 | 2904 | 3276 | 3329 | 3421 |
| 3235 | 3149 | 3264 | 3225 | 3185 | 3416 | 3133 | 3156 | 5057 | 3223 | 3386 | 3059 | 3234 | 3316 | 0000 | 3010 | 3216 | 3101 | 2640 | 3228 | 3187 | 9974 | 3335 | 3236 | 3000 | 3233 | 3386 | 3509 |
| 3235 | 2780 | 3264 | 2804 | 2782 | 3416 | 2725 | 2763 | 5057 | 3223 | 3386 | 3362 | 3234 | 2907 | 3010 | 0000 | 3216 | 2715 | 3210 | 2862 | 2791 | 9974 | 2902 | 2814 | 2560 | 3219 | 3387 | 3509 |
| 2620 | 2970 | 2729 | 3257 | 3197 | 3420 | 3172 | 3122 | 5057 | 3169 | 3386 | 3128 | 2970 | 3253 | 3216 | 3216 | 0000 | 3216 | 3260 | 3245 | 3228 | 9974 | 3242 | 3249 | 3216 | 2757 | 2656 | 3423 |
| 3235 | 2861 | 3264 | 2842 | 2845 | 3416 | 2790 | 2834 | 5056 | 3223 | 3386 | 3268 | 3234 | 2927 | 3101 | 2715 | 3216 | 0000 | 3202 | 2893 | 2834 | 9970 | 2870 | 2849 | 2650 | 3218 | 3385 | 3509 |
| 3260 | 3260 | 3285 | 3313 | 3289 | 3420 | 3299 | 3260 | 5057 | 3299 | 3390 | 3129 | 3260 | 3313 | 2640 | 3210 | 3260 | 3202 | 0000 | 3306 | 3296 | 9976 | 3335 | 3310 | 3105 | 3270 | 3312 | 3424 |
| 3293 | 2977 | 3435 | 2855 | 3040 | 3440 | 2879 | 2930 | 5055 | 3271 | 3260 | 3150 | 3260 | 2811 | 3228 | 2862 | 3245 | 2893 | 3306 | 0000 | 3030 | 9974 | 2767 | 2901 | 2874 | 3265 | 3325 | 3385 |
| 3278 | 2920 | 3365 | 2570 | 2823 | 3370 | 2802 | 2840 | 5057 | 3217 | 3385 | 3266 | 3245 | 2640 | 3187 | 2791 | 3228 | 2834 | 3296 | 3030 | 0000 | 9980 | 2877 | 2866 | 2811 | 3246 | 3306 | 3427 |
| 9976 | 9972 | 9974 | 9972 | 9976 | 9976 | 9977 | 9974 | 9974 | 9974 | 9976 | 9974 | 9976 | 9990 | 9974 | 9974 | 9974 | 9970 | 9976 | 9974 | 9980 | 0000 | 9950 | 9973 | 9974 | 9976 | 9975 | 9975 |
| 3367 | 2903 | 3393 | 2877 | 2765 | 3468 | 2914 | 2744 | 5056 | 3271 | 3490 | 3278 | 3330 | 2842 | 3335 | 2902 | 3242 | 2870 | 3335 | 2767 | 2877 | 9950 | 0000 | 2892 | 2911 | 3130 | 3336 | 3420 |
| 3294 | 2690 | 3384 | 2893 | 2758 | 3440 | 2821 | 2782 | 5056 | 3270 | 3320 | 3294 | 3268 | 2893 | 3236 | 2814 | 3249 | 2849 | 3310 | 2901 | 2866 | 9973 | 2892 | 0000 | 2826 | 3273 | 3327 | 3412 |
| 3235 | 2817 | 3264 | 2821 | 2811 | 3416 | 2768 | 2795 | 5057 | 3223 | 3386 | 3358 | 3234 | 2904 | 3000 | 2560 | 3216 | 2650 | 3105 | 2874 | 2811 | 9974 | 2911 | 2826 | 0000 | 3218 | 3387 | 3509 |
| 2832 | 3273 | 2730 | 3276 | 3205 | 3416 | 3248 | 2980 | 5057 | 3254 | 3391 | 3174 | 3115 | 3276 | 3233 | 3219 | 2757 | 3218 | 3270 | 3265 | 3246 | 9976 | 3130 | 3273 | 3218 | 0000 | 2440 | 3424 |
| 2697 | 3327 | 2730 | 3329 | 3317 | 3416 | 3383 | 3168 | 5056 | 3073 | 3400 | 3314 | 2950 | 3329 | 3386 | 3387 | 2656 | 3385 | 3312 | 3325 | 3306 | 9975 | 3336 | 3327 | 3387 | 2440 | 0000 | 3425 |
| 3424 | 3419 | 3435 | 3417 | 3422 | 3510 | 3509 | 3423 | 5056 | 3509 | 2680 | 3425 | 3423 | 3421 | 3509 | 3509 | 3423 | 3509 | 3424 | 3385 | 3427 | 9975 | 3420 | 3412 | 3509 | 3424 | 3425 | 0000 |

badness $\delta = 0.016$

Table 8. The result of the "simple" algorithm with additional heuristics, $R = 0.7$

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | 3177 | 2580 | 3300 | 3266 | 3420 | 3223 | 3191 | 5056 | 3189 | 3389 | 3290 | 2970 | 3296 | 3233 | 3233 | 2620 | 3233 | 3260 | 3286 | 3278 | 9976 | 3094 | 3293 | 3233 | 2832 | 2697 | 3425 |
| 3177 | 0000 | 3384 | 2920 | 2294 | 3440 | 2783 | 2220 | 5057 | 3172 | 3353 | 3206 | 3073 | 2920 | 3203 | 3164 | 2970 | 3161 | 3260 | 2977 | 2920 | 9972 | 3243 | 2690 | 3160 | 3273 | 3327 | 3419 |
| 2580 | 3384 | 0000 | 3409 | 3430 | 3390 | 3281 | 3205 | 5058 | 3265 | 3427 | 3452 | 3132 | 3409 | 3263 | 3263 | 2729 | 3263 | 3285 | 3435 | 3365 | 9974 | 3384 | 3384 | 3263 | 2730 | 2730 | 3437 |
| 3300 | 2920 | 3409 | 0000 | 2803 | 3440 | 2920 | 2797 | 5057 | 3226 | 3353 | 3101 | 3272 | 2527 | 3273 | 3252 | 3256 | 3250 | 3313 | 2855 | 2570 | 9978 | 3291 | 2893 | 3250 | 3276 | 3329 | 3422 |
| 3266 | 2294 | 3430 | 2803 | 0000 | 3435 | 2772 | 2360 | 5058 | 3232 | 3383 | 3188 | 3219 | 2766 | 3232 | 3196 | 3197 | 3197 | 3289 | 3040 | 2823 | 9977 | 3250 | 2758 | 3195 | 3205 | 3317 | 3423 |
| 3420 | 3440 | 3390 | 3440 | 3435 | 0000 | 3418 | 3422 | 5056 | 3418 | 3510 | 3437 | 3420 | 3440 | 3416 | 3416 | 3420 | 3416 | 3420 | 3440 | 3370 | 9974 | 3434 | 3440 | 3416 | 3416 | 3416 | 3510 |
| 3223 | 2783 | 3281 | 2920 | 2772 | 3418 | 0000 | 2752 | 5057 | 3180 | 3383 | 3071 | 3226 | 2937 | 3133 | 2725 | 3170 | 2790 | 3298 | 2869 | 2802 | 9973 | 3195 | 2831 | 2767 | 3246 | 3381 | 3515 |
| 3191 | 2220 | 3205 | 2797 | 2360 | 3422 | 2752 | 0000 | 5056 | 3184 | 3377 | 3295 | 3129 | 2765 | 3198 | 3156 | 3122 | 3161 | 3260 | 2930 | 2840 | 9974 | 3293 | 2782 | 3158 | 2980 | 3168 | 3424 |
| 5056 | 5057 | 5058 | 5057 | 5058 | 5056 | 5057 | 5056 | 0000 | 5056 | 5056 | 5050 | 5060 | 5056 | 5056 | 5056 | 5057 | 5057 | 5056 | 5055 | 5057 | 9975 | 5057 | 5056 | 5056 | 5057 | 5057 | 5057 |
| 3189 | 3172 | 3265 | 3226 | 3232 | 3418 | 3180 | 3184 | 5056 | 0000 | 3383 | 3291 | 3020 | 3067 | 3232 | 3232 | 3171 | 3232 | 3298 | 3261 | 3217 | 9976 | 3327 | 3275 | 3232 | 3252 | 3379 | 3515 |
| 3389 | 3353 | 3427 | 3353 | 3383 | 3510 | 3383 | 3377 | 5056 | 3383 | 0000 | 3396 | 3386 | 3353 | 3386 | 3386 | 3386 | 3386 | 3390 | 3260 | 3385 | 9974 | 3362 | 3320 | 3386 | 3391 | 3400 | 2680 |
| 3290 | 3206 | 3452 | 3101 | 3188 | 3437 | 3071 | 3295 | 5050 | 3291 | 3396 | 0000 | 3268 | 3280 | 3276 | 3267 | 3127 | 3069 | 3318 | 3150 | 3266 | 9975 | 3072 | 3293 | 3266 | 3174 | 3106 | 3516 |
| 2970 | 3073 | 3132 | 3272 | 3219 | 3420 | 3226 | 3129 | 5060 | 3020 | 3386 | 3268 | 0000 | 3271 | 3234 | 3234 | 2970 | 3234 | 3260 | 3259 | 3245 | 9975 | 3279 | 3268 | 3234 | 3115 | 2950 | 3424 |
| 3296 | 2920 | 3409 | 2527 | 2766 | 3440 | 2937 | 2765 | 5056 | 3067 | 3353 | 3280 | 3271 | 0000 | 3353 | 3043 | 3252 | 3333 | 3313 | 2811 | 2640 | 9990 | 3331 | 2893 | 3333 | 3276 | 3329 | 3422 |
| 3233 | 3203 | 3263 | 3273 | 3232 | 3416 | 3133 | 3198 | 5056 | 3232 | 3386 | 3276 | 3234 | 3353 | 0000 | 3010 | 3215 | 3101 | 2640 | 3264 | 3235 | 9976 | 3312 | 3271 | 3000 | 3240 | 3394 | 3515 |
| 3233 | 3164 | 3263 | 3252 | 3196 | 3416 | 2725 | 3156 | 5056 | 3232 | 3386 | 3267 | 3234 | 3043 | 3010 | 0000 | 3215 | 2715 | 3210 | 3238 | 3203 | 9976 | 3327 | 3244 | 2560 | 3235 | 3391 | 3515 |
| 2620 | 2970 | 2729 | 3256 | 3197 | 3420 | 3170 | 3122 | 5057 | 3171 | 3386 | 3127 | 2970 | 3252 | 3215 | 3215 | 0000 | 3215 | 3260 | 3238 | 3228 | 9974 | 3288 | 3249 | 3215 | 2757 | 2656 | 3424 |
| 3233 | 3161 | 3263 | 3250 | 3197 | 3416 | 2790 | 3161 | 5057 | 3232 | 3386 | 3069 | 3234 | 3333 | 3101 | 2715 | 3215 | 0000 | 3202 | 3237 | 3201 | 9970 | 2870 | 3244 | 2650 | 3235 | 3384 | 3515 |
| 3260 | 3260 | 3285 | 3313 | 3289 | 3420 | 3298 | 3260 | 5056 | 3298 | 3390 | 3318 | 3260 | 3313 | 2640 | 3210 | 3260 | 3202 | 0000 | 3306 | 3296 | 9976 | 3326 | 3310 | 3105 | 3270 | 3312 | 3425 |
| 3286 | 2977 | 3435 | 2855 | 3040 | 3440 | 2869 | 2930 | 5055 | 3261 | 3260 | 3150 | 3259 | 2811 | 3264 | 3238 | 3238 | 3237 | 3306 | 0000 | 3030 | 9977 | 3302 | 2901 | 3236 | 3265 | 3325 | 3385 |
| 3278 | 2920 | 3365 | 2570 | 2823 | 3370 | 2802 | 2840 | 5057 | 3217 | 3385 | 3266 | 3245 | 2640 | 3235 | 3203 | 3228 | 3201 | 3296 | 3030 | 0000 | 9980 | 3238 | 2866 | 3200 | 3246 | 3306 | 3427 |
| 9976 | 9972 | 9974 | 9978 | 9977 | 9974 | 9973 | 9974 | 9975 | 9976 | 9974 | 9975 | 9975 | 9990 | 9976 | 9976 | 9974 | 9970 | 9976 | 9977 | 9980 | 0000 | 9950 | 9977 | 9976 | 9976 | 9975 | 9975 |
| 3094 | 3243 | 3384 | 3291 | 3250 | 3434 | 3195 | 3293 | 5057 | 3327 | 3362 | 3072 | 3279 | 3331 | 3312 | 3327 | 3288 | 2870 | 3326 | 3302 | 3238 | 9950 | 0000 | 3313 | 3314 | 3153 | 3392 | 3514 |
| 3293 | 2690 | 3384 | 2893 | 2758 | 3440 | 2831 | 2782 | 5056 | 3275 | 3320 | 3293 | 3268 | 2893 | 3271 | 3244 | 3249 | 3244 | 3310 | 2901 | 2866 | 9977 | 3313 | 0000 | 3244 | 3273 | 3327 | 3412 |
| 3233 | 3160 | 3263 | 3250 | 3195 | 3416 | 2767 | 3158 | 5056 | 3232 | 3386 | 3266 | 3234 | 3333 | 3000 | 2560 | 3215 | 2650 | 3105 | 3236 | 3200 | 9976 | 3314 | 3244 | 0000 | 3235 | 3391 | 3515 |
| 2832 | 3273 | 2730 | 3276 | 3205 | 3416 | 3246 | 2980 | 5057 | 3252 | 3391 | 3174 | 3115 | 3276 | 3240 | 3235 | 2757 | 3235 | 3270 | 3265 | 3246 | 9976 | 3153 | 3273 | 3235 | 0000 | 2440 | 3425 |
| 2697 | 3327 | 2730 | 3329 | 3317 | 3416 | 3381 | 3168 | 5057 | 3379 | 3400 | 3106 | 2950 | 3329 | 3394 | 3391 | 2656 | 3384 | 3312 | 3325 | 3306 | 9975 | 3392 | 3327 | 3391 | 2440 | 0000 | 3426 |
| 3425 | 3419 | 3437 | 3422 | 3423 | 3510 | 3515 | 3424 | 5057 | 3515 | 2680 | 3516 | 3424 | 3422 | 3515 | 3515 | 3424 | 3515 | 3425 | 3385 | 3427 | 9975 | 3514 | 3412 | 3515 | 3425 | 3426 | 0000 |

badness $\delta = 0.015$

We present this paragraph and the following one almost unchanged based on the text of [Melnikov, Zhang, and Chaikovskii, 2022]. As we have already noted, we evaluate the results of computational experiments well. Exactly, an improvement in the performance of the algorithm was obtained (according to both criteria given in the previous section), compared with the simplest variant of the branch and boundary method, see [Melnikov and Trenina, 2018; Melnikov, Trenina, and Kochergin 2018] etc.; let us add, that by the comparison with [Melnikov, Zhang, and Chaikovskii, 2022], a significant improvement was also obtained. More precisely, by the words “simplest variant”, we mean that the simplest greedy heuristic used to select the next separating element. Here we apply a more complex greedy heuristic, while abandoning the method of branch and boundary. It is clear that even more successful results (from the point of view of the quantitative criteria formulated above) we would have obtained by using both the branch and boundary method and a more complex greedy heuristic at the same time; however, it seems that we shall not satisfy acceptable time constraints. Though, we did not conduct detailed computational experiments for this case.

To perform all the computational experiments described in the article, we used a computer with the following characteristics:

```
Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
```

In all our computational experiments, the total time of the computer was extremely short, and we did not record it, since it is significantly less than the time required to output the results of the algorithm (especially in comparison with the time necessary for the initial filling of only one cell of the matrix).

Let us add to the above that for this work, the absolute speed of program execution is hardly of interest. The relative speed is a little more important (in comparison with other programs that calculate the same values, but use different algorithms); however, we also do not give this relative speed in this paper. The explanation for this thing is simple: significantly more than relative speed, the relative results of calculations are important, which we are considering here.

Let us go directly to the description of the results obtained. For them, we immediately note that all the numerical values given here can be very easily verified: exactly, the obtained pseudo-optimal matrices can easily be copied from the presented pdf-file by the ordinary copy-pasting.

The most important results are shown in the Fig. 6. It clearly shows the gradual decrease in the average value of badness when using newer algorithms, and, in particular, when decreasing the value of $R$; this average value of badness is given in the 4th column (i.e., the value $0.055515$ in the 1st line and the value $0.015415$ in the last line).

Now let us move on to *the description of the tables of results given before*.

The Table 1 is the initial table of distances between DNA sequences taken for $28$ species of monkeys belonging to different genera. The table is taken from our previous publications. The Needleman – Wunsch algorithm has been applied, but, of course, similar constructions are possible for any other algorithm for calculating distances between pairs of sequences. We note once again that the construction of such a table on an average modern personal computer takes about one day – which indirectly indicates the need to develop and apply algorithms for restoring partially filled matrices.

The Table 2 can also be called the initial one; it contains about $10\,\%$ of non-removable elements. It is fed to the input of all the algorithms we are considering.

For the Tables 3 – 8, we marked the algorithms used for their constructing in their captions. After each of these tables, as well as after the original Table 1, the value of badness is given; moreover, according to the paper [Melnikov, Zhang, and Chaikovskii, 2022], the badness $\delta$ is used, not $\sigma$.

Specially note the results related to the genome located in the 7th row from the bottom (in the 7th column on the right). There is a clear error here! However, we note the following two things:

- firstly, this is not our error, but either the error of the genome available in the database, or the Needleman – Wunsch algorithm itself;
- secondly, even with such data, our algorithms cope well, and the badness turns out to be quite acceptable.

It can also be noted that the results obtained without the use of additional heuristics and with its application differ very much. At the same time, according to our terminology, $\delta$, not $\sigma$ is of great importance. Therefore, we shall not give the values of $\sigma$ (to obtain them, we need to compare with the original matrix), and we believe that such a small change indicates the correctness of the approach we describe. In any case, we repeat that the resulting value of badness is significantly less. what is the value for the original matrix, calculated exactly according to the Needleman – Wunsch algorithm.

At the end of the direct description of the results of the work, let us say a few words about significant decimal digits:

- we did not specifically monitor this in the computer output (Fig. 6);
- when describing brief results of computational experiments, we usually used 3 decimal digits, the same in the original Table 1 (meaning that 3 digits are the signs after the decimal point);
- but in obtained Tables 3 – 8, we used 4 decimal digits.

## 5 Conclusion. Some possible directions for further work on this subject

Some of the directions described below have already been mentioned in our previous publications, they are gradually being implemented in the current computer programs. However, it is necessary to specifically focus on some of the areas noted in our previous paper [Melnikov, Zhang, and Chaikovskii, 2022, Sect. 5]. At the same time, it is clear that the current paper is entirely devoted to the direction titled in the previous article as the second one.

The first direction stands apart from the rest: it is devoted not to improving the algorithms for forming a partially filled matrix, but to the algorithms necessary for its initial formation.

Apparently, along with the second direction considered here, the third one is the most interesting. Let us say, slightly reformulating the text of the previous article, that it is supposed to compare:

- very complicated greedy algorithms without using other heuristics;
- and the branch and boundary method with simple greedy algorithms as heuristics for selecting separating elements: as a rule, such very complicated greedy algorithms cannot be used as auxiliary due to time constraints.

It is important to note that approximately the same problems (greedy heuristic vs branch and boundary method) arise in many other discrete optimization problems considered by the authors, we are preparing several publications in which, among other things, we shall present such comparisons.

Some other areas are primarily related not to algorithms for solving the formulated problem, but to modifying algorithms for generating source data; we shall return to these areas in future publications.

## 6 Acknowledgement

This work was supported by the following grants.

- Grant of the Scientific program of Chinese universities “Higher Education Stability Support Program” (chapter “Shenzhen 2022 – Science, Technology and Innovation Commission of Shenzhen Municipality”);
- Grant of the Natural Science Foundation of Guangdong Province (No. 2019A1515110971);
- Grant of the Shenzhen National Science Foundation No. 20200827173701001).

## References

Cibelli, J., Wilmut, I., Jaenisch, R. et al. (Eds) (2014). *Principles of Cloning*. Academic Press, New York.

Lennarz, J. and Lane, M. (Eds) (2013). *Encyclopedia of Biological Chemistry*. Elsevier Publ., Amsterdam.

Levenshtein, V. (1966). Binary codes capable of correcting. Deletions, insertions, and reversals. *Soviet Physics. Doklady*, **10**, pp. 707.

Maloy, S. and Hughes, K. (Eds) (2013). *Brenner's Encyclopedia of Genetics*. Elsevier Publ., Amsterdam.

Melnikov, B., Pivneva, S., and Trifonov, M. (2017). Various algorithms, calculating distances of DNA sequences, and some computational recommendations for use such algorithms. *In: CEUR Workshop Proceedings, 1902*, pp. 43–50.

Melnikov, B. and Trenina, M. (2018). On possible methods for solving the problem of reconstructing the matrix of distances between DNA strings. *In: CEUR Workshop Proceedings, 2258*, pp. 11–20.

Melnikov, B., Trenina, M., and Kochergin, A. (2018). On one problem of reconstructing matrix distances between chains of DNA. *In: IFAC-Papers OnLine*, **51**(32), pp. 378–383.

Melnikov, B., Zhang, Y., and Chaikovskii, D. (2022). An inverse problem for matrix processing: an improved algorithm for restoring the distance matrix for DNA chains. *Cybernetics and Physics*, **11**(4), pp. 217–226.

Needleman, S. and Wunsch, Ch. (1970). A general method is applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, **48** (3), pp. 443–453.

Sergeenko A., Granichin O., and Yakunina M. (2020). Hamiltonian path problem: the time consumption comparison of DNA computing and branch and boundary method. *Cybernetics and Physics*, **9**(1), pp. 121–127.

Sykes, B. (2003). *Adam's Curse: The Science That Reveals Our Genetic Destiny*. W. W. Norton and Company, New York.

van der Loo, M. (2014). The stringdist package for approximate string matching. *The R Journal*, **6**, pp. 111–122.

Winkler, W. (1990). String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage. *In: Proceedings of the Survey Research Methods Sections, American Statistical Association*, pp. 354–359.