Metrics and visualisation for crime analysis and genomics

Laros, J.F.J.

Chapter 2

Randomised Non-Linear Dimension Reduction

The field of dimension reduction has provided a set of algorithms that can be used for the visualisation of high dimensional data. In this set, some well-known instances have been studied and used to a great extent, while others have not. In this chapter, we discuss the properties of the push and pull [43] model, also known as the particle or spring model. We first analyse the basic properties of the algorithm and discuss many variants and applications. A number of natural extensions and induced models are given as well.

2.1 Introduction

The general algorithm for 2-dimensional visualisation tries to solve the following problem: We have a pairwise distance matrix as input and as output we desire a 2-dimensional picture that represents the distances in the input matrix as good as possible. In general the data in the input matrix is derived from a high dimensional input space and can not be embedded perfectly in a 2-dimensional space.

The algorithm operates by placing points (or particles) randomly on a surface, the number of points corresponding with the number of rows (and columns) of the input matrix. Now the algorithm iterates in some way over (all) pairs of points, somewhat adjusting the position of the points in question according to the distances defined in the input matrix.

The algorithm can be terminated when the points no longer move, or when sufficiently many iterations have been done. The termination criteria are specified by the user.

2.2 The surface

Since this model is searching for an optimal arrangement of particles on a given surface, we can ask ourselves if we can use different surfaces to improve the dimension reduction. Normally we use a unit square to visualise the input data, but other choices are also available. We can look at closed or even semi-closed surfaces, for example, surfaces that have two or more (simply) identified boundaries. Also infinite surfaces are a possibility [8].

We could for example identify two of the sides of a unit square to obtain a (topological) cylinder. The push and pull algorithm will work exactly the same on an object like this, the only thing that needs to be adjusted is the distance function; on a cylinder there is a maximum distance in one direction. Notice that we only make a topological 2-dimensional cylinder. There is no curvature of the space at all.

Identifying more sides of the unit square will result in a fully closed surface like a globe, torus, Klein bottle or real projective plane. Again, the distance function must be adjusted for each of these surfaces (since our picture will still be a square) but the idea stays the same. Of these closed surfaces, a torus is perhaps the most natural choice to make. Again, notice that we use a topological 2-dimensional torus, again there is no curvature as would be the case with a 3-dimensional torus.

Using a closed surface has the advantage that an embedding of non-flat data is sometimes possible. We have more freedom to place our points since we can implicitly move in a third dimension. For example, we can perfectly embed points taken from the surface of a cylinder on a 2-dimensional torus, but not on the unit square. Conversely, we can embed the unit square on a 2-dimensional torus. Therefore a closed surface is a better choice than a normal unit square, although it must be noted that the end-user might find it confusing in the beginning.

2.3 Metric algorithms

Perhaps the best way to describe this model it to view the points as particles that have two types of forces working on them, an attracting and a repulsing one. These particles are bound to a surface with (normally) two dimensions. This loose description leaves a lot of freedom. We can use different types of forces. For example, the forces do not even have to be symmetrical. An other choice can be the surface, that does not have to be a unit square, but can also be a closed surface (see Chapter 3) like a torus.

Now we shall give the metric variant of the push and pull algorithm. It is split into two parts: a part that adjusts the positions of the particles and a part that iterates over a sequence of tuples of points and calls the adjustment function in each iteration.

In Figure 2.1 we see how the adjustment of the points works. Assume that $\tilde{p}$ and $\tilde{q}$ are too far away from each other. The algorithm below describes how two

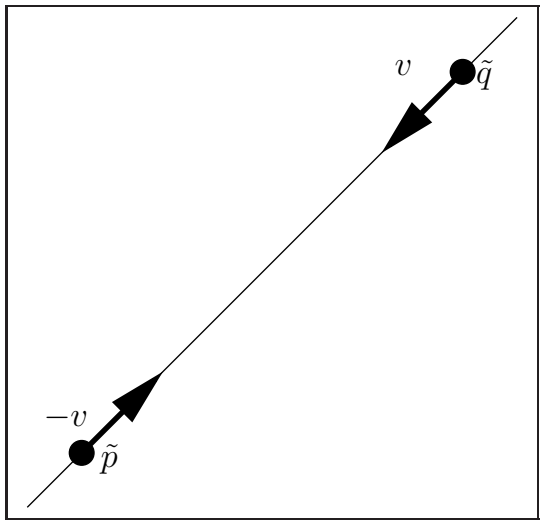

Figure 2.1: Metric correction

vectors v and $-v$ are calculated, over which the points are translated toward each other.

PushPull_Metric (p, q) ::
var *correction*;
var v;
$correction = \alpha \cdot f(d_{\text{realised}}(\tilde{p}, \tilde{q}), \mathit{inflation} \cdot d_{\text{desired}}(p, q))$;
$v = correction \cdot (\tilde{q} - \tilde{p})$;
$\tilde{p} \leftarrow \tilde{p} - v$;
$\tilde{q} \leftarrow \tilde{q} + v$;

Adjust $\tilde{p}$ and $\tilde{q}$.

In the algorithm above, p and q are input points. By $\tilde{p}$ we denote the coordinates of a point p in the target surface; we refer to $\tilde{p}$ as the *realisation* of p. The function $d_{\text{desired}}(p, q)$ is the distance between p and q as given in the input matrix, or perhaps by some other means. The function f returns a value between -1.0 and 1.0; if the realised distance is larger than the desired distance, the function will in general return a negative value, indicating that the points must be pulled together. The choices for this function are discussed in Section 2.3.1.

The value $d_{\text{realised}}(\tilde{p}, \tilde{q})$ is the distance between points $\tilde{p}$ and $\tilde{q}$ in the target surface. Usually, the Euclidean distance is employed for this. The global parameter α is the learning rate, which may decrease over time, or may be altered by the user. This parameter is discussed in Section 2.6. The global parameter *inflation* is used to utilize the entire output space. It is often set to 1.0 and is discussed in detail in Section 2.3. Note that if $\tilde{p} = \tilde{q}$, we can not determine a di-

rection for the vector v, only the magnitude. To overcome this shortcoming, we can perhaps introduce a small distortion in the position of $\tilde{p}$ or $\tilde{q}$. Fortunately, in practice the algorithm has more than two input points, and other points will disturb the positions of $\tilde{p}$ and $\tilde{q}$, so in practice we can ignore this shortcoming.

MetricMainLoop ( ) ::
    while *NotDone* **do**
        choose some sequence $u = (u_1, u_2, \ldots, u_n)$
            of tuples of points;
        for $i \leftarrow 1$ **to** n **do**
            PushPull_Metric(u_i);

Iterate over tuples of points.

The main loop iterates over some sequence of tuples of n points. In general we iterate over all combinations of tuples exactly one time in each iteration. The order of these tuples is preferred to be random.

2.3.1 Forces

In this section we shall focus on functions that have input values between 0 and $\frac{1}{2}$ and have output values between -1 and 1 on this interval. The maximum value of $\frac{1}{2}$ is chosen because it is the maximum distance between two points on a torus. Functions for the normal bounded unit square will have input values between 0 and 1.

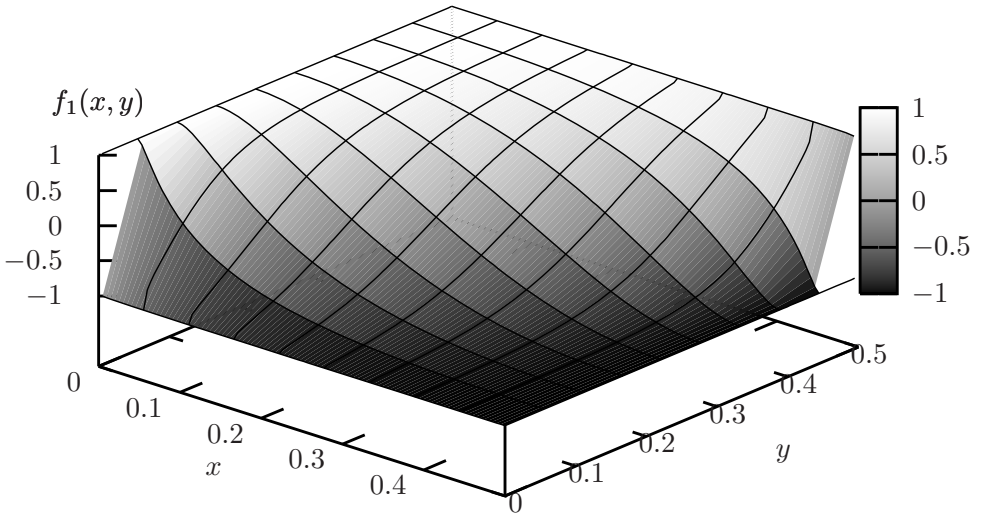

Figure 2.2: Correction function $f_1(x, y)$

The forces used for the alteration of the particles are usually symmetrical

and can be described by a correction function that yields a positive value when two particles need to be placed closer together, and a negative value when they need to be pushed apart. There are many choices for such a function, a linear one being the most widely used, but other, mostly sigmoid like functions, can perform better since these functions are "aggressive" when a particle is not in the right position.

In Figure 2.2 we see such a function. The precise definition of this function is:

$$f_1(x,y) = \begin{cases} \cos(\pi \log_t(2x(t-1)+1)) & \text{if } y \neq \frac{1}{4} \\ \cos(\pi 2x) & \text{if } y = \frac{1}{4} \end{cases}$$

where $t = (1 - 1/(2y))^2$, $0 \leq x \leq \frac{1}{2}$, $0 < y < \frac{1}{2}$. So this function will be a deformed cosine for any fixed y, and will be 0 if x equals y. Furthermore, it is 1 whenever x equals 0 and -1 if x equals $\frac{1}{2}$.

Note that $f_1(\frac{1}{4}+x, \frac{1}{4}+y) = -f_1(\frac{1}{4}-x, \frac{1}{4}-y)$. This property is indeed quite desirable, since ... We will refer to it as the *symmetry property.*

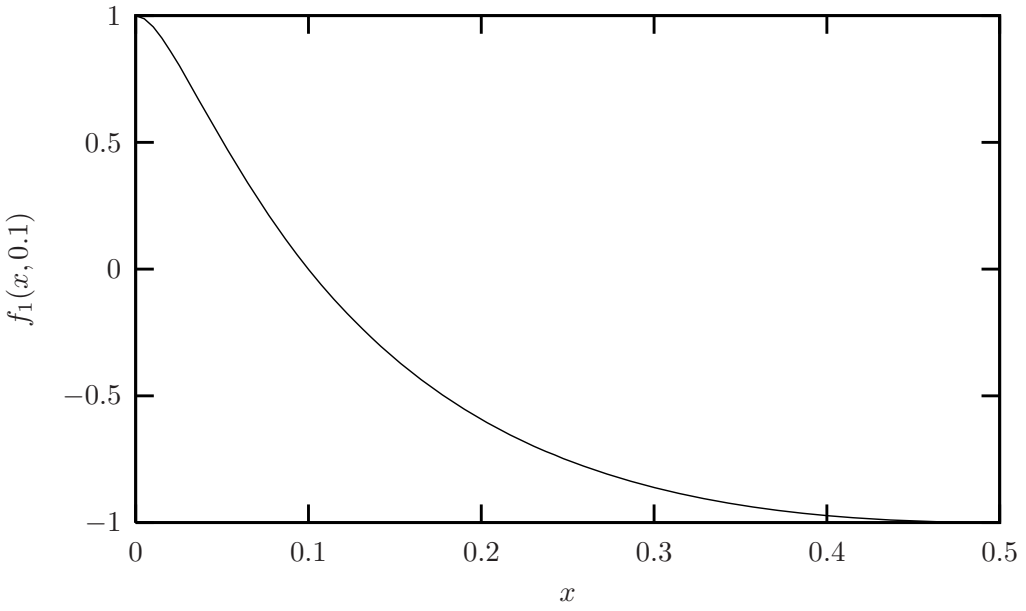

Figure 2.3: Correction function $f_1(x, 0.1)$

In the situation from Figure 2.3, the desired distance between two particles is 0.1. If the realised distance is larger than this, the function gets negative very quickly. On the other hand, the function will assume a positive value significantly larger than 0 when the realised distance is only slightly smaller than the desired distance.

For practical purposes, we can use any function that resembles the one given above, such as

$$f_2(x,y) = \cos(\pi(2x)^{\tan(\pi y)})$$

In order to get the symmetry property, one is lead to:

$$f_3(x,y) = \begin{cases} \cos(\pi(2x)^{4y}) & \text{if } 0 \le y \le \frac{1}{4} \\ -\cos(\pi(1-2x)^{4(\frac{1}{2}-y)}) & \text{if } \frac{1}{4} \le y \le \frac{1}{2} \end{cases}$$

In practice, this will be the kind of functions we will be using.

Another choice of function could be a function with a plateau, one that is zero on and near the desired distance.

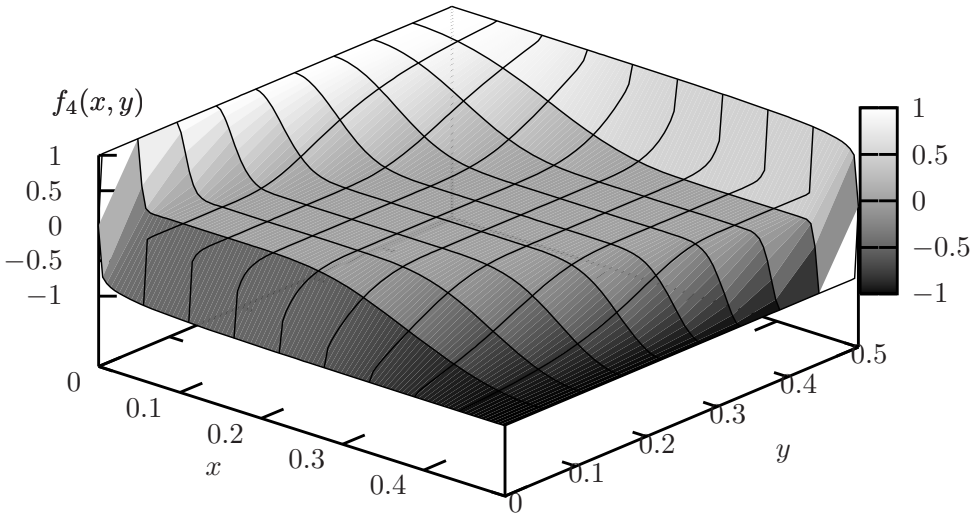

Figure 2.4: Correction function $f_4(x,y)$ with a plateau

In Figure 2.4 we see such a function, of which the exact definition is:

$$f_4(x,y) = \begin{cases} \cos^{25}(\pi \log_t(2x(t-1)+1)) & \text{if } y \ne \frac{1}{4} \\ \cos^{25}(\pi 2x) & \text{if } y = \frac{1}{4} \end{cases}$$

where $t = (1-1/(2y))^2$, $0 \le x \le \frac{1}{2}$, $0 < y < \frac{1}{2}$. Again, the symmetry property holds and in this particular case, if x roughly equals y, the function will not return a very large correction.

Also, reports have been made of even more complicated functions [12], each giving different results and emphasising different aspects of the data, or different aims, varying from creating a global picture to the desire to speed up the algorithm.

In Figure 2.5 we see an instance of a hybrid function, as used in a previous study [12]:

$$f_5(x,y) = \begin{cases} \cos^{3}(\pi \log_t(2x(t-1)+1)) & \text{if } y \ne \frac{1}{4} \\ \cos^{3}(\pi 2x) & \text{if } y = \frac{1}{4} \end{cases}$$

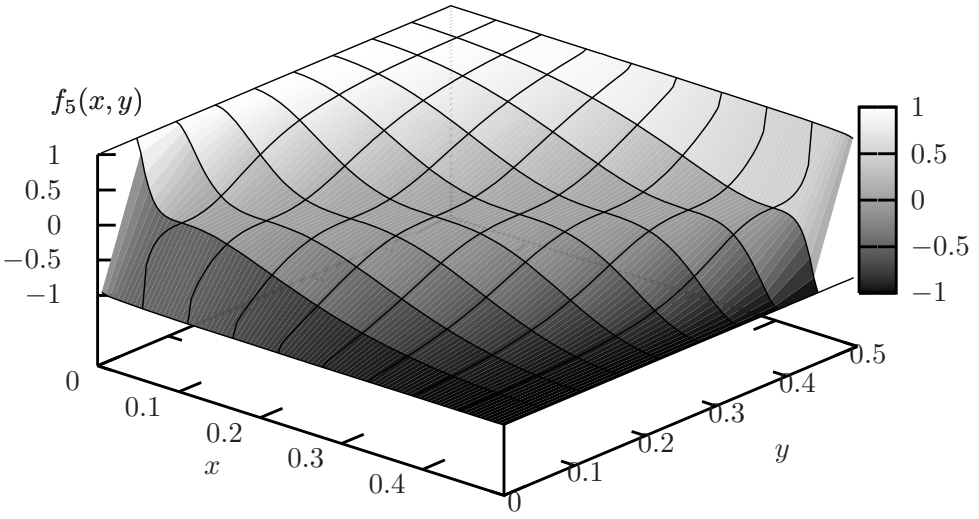

Figure 2.5: Hybrid correction function $f_5(x, y)$

where $t = (1 - 1/(2y))^2$, $0 \leq x \leq \frac{1}{2}$, $0 < y < \frac{1}{2}$. Although in the original paper a slightly different function is used, the shape is similar.

In general there is no clear way to determine which function to use. A function with a plateau will give a more global picture, since the exact distances are less important than the global placement and a sigmoid like function leads to fast convergence. It largely depends on the desired end result which function is preferred.

As mentioned before, we can also use a non-symmetrical function to describe the forces working on the particles, giving rise to a whole different set of images. Such a function would be a combination of a push force and a pull force, where the two are not each others inverse. We could for example use a logarithmic function for pushing and a linear function for pulling.

A useful example would be on a torus where we use a push function that is positive (but declining) everywhere and *no* pull function. At first glance this would seem not to work at all, but since we are working on a torus, we still get a valid dimension reduction, because the push force wraps around the torus (it is positive everywhere). So two points that are supposed to be far from each other will be pushed harder than points that are supposed to be near to each other. This results in a model where the points that are the farthest from each other have a dominance over other points and will force them into the correct position.

Using this model has the disadvantage that it is slow in comparison to the model with both forces. One big advantage, though, is that there is no need for an inflation parameter, the inflation is done automatically. For this reason, the push-only variant on a closed surface is an interesting model that should be investigated further.

The model allows for the online adding and removing of points [14], since

the algorithm can be resumed at any point. In general adding or removing one point will not result in big global differences, only local ones, so on resuming the algorithm one will quickly find a new optimum.

A consequence of this observation is that the properties of the points can be altered online to make a simulation of flowing particles, or that a single item can be traced.

A drawback of this model is that the algorithm can get stuck in a local optimum. This is almost always the case with randomised algorithms that do local optimisations. In practical situations, however, this seems to happen rarely.

Another drawback is that the result of two runs of the algorithm will produce different pictures. In most cases, this is the result of a rotation or mirroring, which can be countered by adding three reference points to the data that the algorithm will not alter. If the difference is the result of the fact that the algorithm has found a different local optimum however, using reference points will not be of any use.

The main advantage of the algorithm is that it is fast and very flexible. If, for example, no (global) optimum can be found, the parameters can be changed online to better suit the data. We can also change these parameters to get out of a local optimum or to improve the embedding by multiplying the distances by a factor to make use of a closed surface.

An other point that is worth stressing is that we only need pairwise distances to generate the dimension reduction. The original coordinates need not to be known.

Furthermore, this form of dimension reduction is non-linear, as we shall see in the next section.

2.4 Axes

Many questions arise about the meaning of the axes when using this technique. The meaning is hard to understand: in general we can only say that it is a non-linear combination of the (maybe even unknown) input dimensions, which are already "warped" by the metric used to derive the distance matrix.

To illustrate the non-linearity of the algorithm we can take some points uniformly distributed on a sphere and try to embed them onto a 2-dimensional surface. A linear dimension reduction technique will produce a picture where two halves of the sphere will be projected upon each other. The push-pull technique however will generate a different picture.

In many cases there can be a correlation between one of the input dimensions and a direction in the resulting picture. For example, in the following picture we see the clustering of criminal "careers", where time is an implicit dimension in our input data. The data stems from the Dutch national police and because of the sensitive nature can not be disclosed. The input of the push and pull algorithm is a distance matrix, obtained by calculating the edit distance between two criminal careers. The careers themselves are defined as a string of *multisets*,

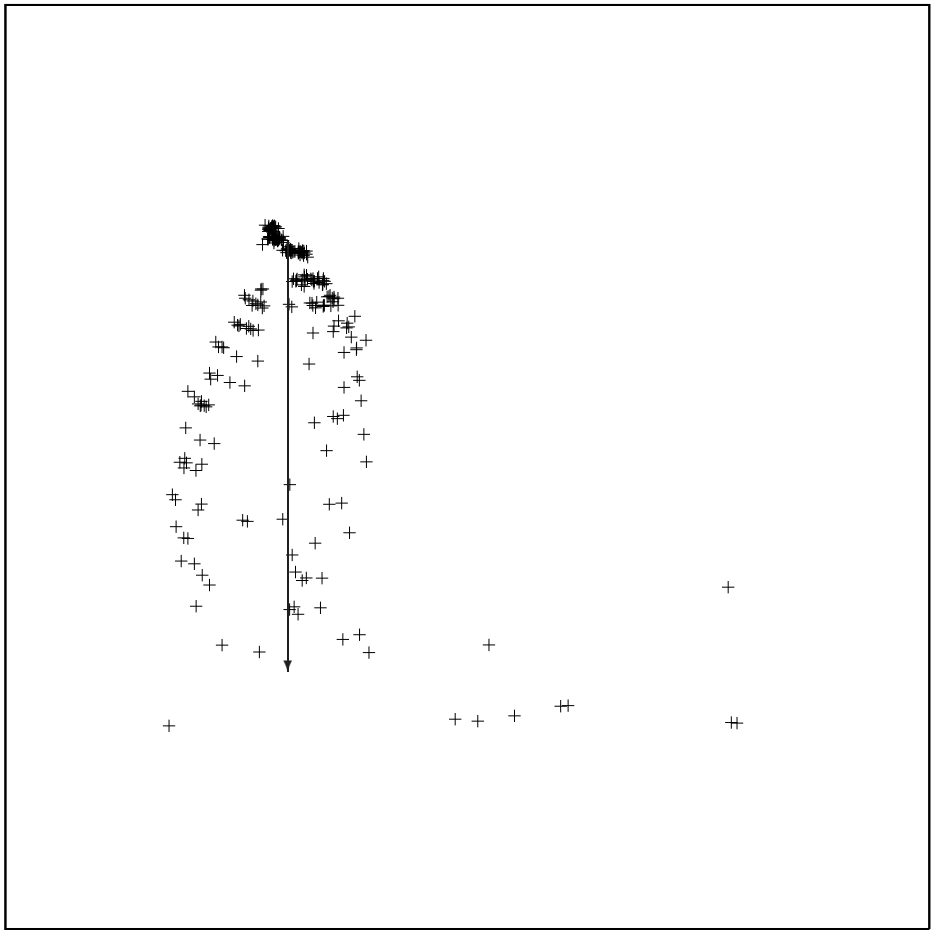

Figure 2.6: Criminal careers

where each multiset corresponds to the nature and frequency of crimes in one year.

With a metric for multisets (see Chapter 6) we can calculate a pairwise distance between two multisets and with the standard alignment algorithms [54, 66] we calculate the pairwise distance between the criminal careers.

In the resulting picture (seen in Figure 2.6) there is a correlation between time and the direction of the arrow. At the base of the arrow there is a cluster of short careers and as we proceed to the head of the arrow the length of the careers increase. Note that the same direction may correspond to a whole different combination of input dimensions at an other position in the picture.

So in general we can say that a direction has a meaning in the pictures, but that this meaning is not uniform in the whole picture. Only locally we can say something about the directions, globally this structure can be complex.

In *Principal Component Analysis* (PCA) [35,57], on the axes we always have a *linear* combination of the dimensions of the input space. For *Principal Curves and Surfaces* (PCS) [18, 29] and *Self Organising Maps* (SOM) [39] the data is projected onto a lower dimensional manifold, which does not need to be linear. It does, however, need to be continuous (smooth). In the push-pull algorithm this is not the case in general.

Both SOMs and PCA/PCS need the input points to operate. The push-pull algorithm only requires the pairwise distances, like most Multi Dimensional Scaling (MDS) [16] techniques. Unlike SOMs and PCA/PCS no parametrisation of the manifold is given as part of the output.

2.5 The non-metric variant

Since the metric used for the calculation of pairwise distances is a parameter in the dimension reduction, we can also use the topological ordering of distances to derive a projection onto a low-dimensional surface. The idea behind this is to make an embedding in such a way that the relative order of distances is preserved, but in general not the distances themselves. This means that the objective is a picture where the distance between two points is smaller than the distance between two other points if and only if this is also true in the input data.

Next we discuss the non-metric variant of the algorithms that do the dimension reduction. This variant is also split into two parts; one part is responsible for the adjustment of two tuples of particles, and the other part iterates over a sequence of tuples of tuples and calls the adjustment function in each iteration.

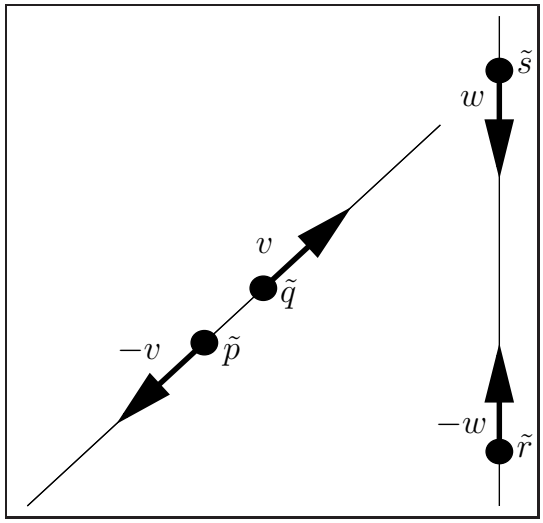

Figure 2.7: Non-metric correction

In Figure 2.7 we see how the adjustment of the points works. Assume that

$\tilde{p}$ and $\tilde{q}$ are too close to each other with respect to $\tilde{r}$ and $\tilde{s}$. The algorithm below describes how four vectors v, $-v$, w and $-w$ are calculated, over which the points are translated respectively.

PushPull_NonMetric $((p, q), (r, s))$ ::
  var *correction* $\leftarrow 0.0$;
  var v, w;
  if $d_{\text{realised}}(\tilde{p}, \tilde{q}) < d_{\text{realised}}(\tilde{r}, \tilde{s})$ **and** $d_{\text{desired}}(\tilde{p}, \tilde{q}) > d_{\text{desired}}(\tilde{r}, \tilde{s})$ **then**
    correction $\leftarrow \alpha \cdot \epsilon$;
  if $d_{\text{realised}}(\tilde{p}, \tilde{q}) > d_{\text{realised}}(\tilde{r}, \tilde{s})$ **and** $d_{\text{desired}}(\tilde{p}, \tilde{q}) < d_{\text{desired}}(\tilde{r}, \tilde{s})$ **then**
    correction $\leftarrow -\alpha \cdot \epsilon$;
  $v \leftarrow$ *correction* $\cdot (\tilde{q} - \tilde{p})$;
  $w \leftarrow -$*correction* $\cdot (\tilde{s} - \tilde{r})$;
  $\tilde{p} \leftarrow \tilde{p} - v$;
  $\tilde{q} \leftarrow \tilde{q} + v$;
  $\tilde{r} \leftarrow \tilde{r} - w$;
  $\tilde{s} \leftarrow \tilde{s} + w$;

Adjust $(\tilde{p}, \tilde{q})$ and $(\tilde{r}, \tilde{s})$.

The non-metric variant of the algorithm adjusts four points each iteration by a small amount $\alpha\ \epsilon$, if the realised distance of one tuple is smaller than the other one, but the desired distance is larger, the points in the first tuple are pushed apart and the ones in the second tuple are pulled togeter. This variant is covered in Section 2.5.

NonMetricMainLoop ( ) ::
  while *NotDone* **do**
    choose some sequence $u = (u_1, u_2, \ldots, u_n)$
      of tuples of tuples of points;
    for $i \leftarrow 1$ **to** n **do**
      PushPull_NonMetric(u_i);

Iterate over tuples of tuples of points.

The main loop iterates over some sequence of tuples of tuples of points. In general we iterate over all combinations of two tuples exactly one time in each iteration. The order of these tuples is preferred to be random.

The correction function f used to alter the realisation of the points is highly configurable. If f is very small for all input values, the algorithm will reach a stable state if one exists with high probability. The only provision is that the total amount of distances is preserved. The reason that this will lead to a good embedding in general is because distances can be divided infinitely many times (in principle), so a valid ordering will nearly always be found if f is small enough

for all input values.

This, however, is not very practical when the objective is a fast algorithm. To increase the effectiveness of f, several methods can be used like letting f depend on the number of input points, or on the amount of iterations (see Section 2.6). Another interesting method would be to let f depend upon the relative amount of points that already have a good position; this idea is discussed further in Section 2.6.

Since only relative distances have meaning in a non-metric dimension reduction, the size of a picture is irrelevant for correctness. However, for insight we want to have it as large as the space permits. If we use a closed or semi-closed surface, this is even more preferable, because then the properties of that particular space can be exploited. In the first case, we can do the dimension reduction and afterwards inflate the picture (zoom in) to make use of the entire space. In the second case however, we need to have a kind of entropy law to make the points want to float away from each other. We can not simply use a zoom function, since some points will be put closer together because of the nature of the torus; it has a maximum distance.

Apart from an inflation function, we want the diversity of distances to be as large as possible. Since the dimension reduction is non-metric there is some freedom in general. There are several ways to utilize this freedom in order to make the picture more insightful. First we can use the freedom to make the diversity of distances as large as possible without compromising the topological order. This will in general emphasize the difference in distances between two pairs of pairs of points. Another way to utilise this freedom is to make the distances resemble the real distances without compromising the topological order. This will result in a dimension reduction that is non-metric, but tries to be "as metric as possible".

2.6 Simulated annealing

In both the metric and non-metric variant, simulated annealing [61] can be used to speed up the algorithm and to force convergence. The general idea of simulated annealing is to use large alterations at the beginning of the algorithm and small ones near the end. In this particular case, the strength of the correction function can be subject to such a technique.

2.7 Comparison with other methods

In this sections we compare with PCA, PCS, SOM and MDS.

Principal component analysis (PCA) is a linear technique that requires the input points. As output a (hyper) plane is given with the projected points on it. This is a deterministic algorithm, and thus always produces the same image when given identical input data. The fact that it is linear results in a

linear combination of the input dimensions on the axes of the output picture. This might not give the best result though.

Principal Curves and Surfaces (PCS) is a non-linear technique that requires the input points. As part of the output, a parametrised manifold is given, onto which the points are projected. Although this dimension reduction technique is non-linear, the manifold is continuous (or smooth), which needs not be the case in the push-pull algorithm. PCS is, like PCA, also a deterministic technique.

In a **Self Organising Map** (SOM), a field of vectors is initialised randomly and then trained with input examples. The vector that looks most like the example, is altered in such a way that it looks even more like the example and further more, its neighbours are also altered, but to a lesser extent.

This results in a non-linear output manifold, similar to one used in PCS. The technique itself is non-deterministic though. Because of the non-determinism, the non-linear output manifold and the training component where mostly local changes are made, there is a strong relation with the push and pull model.

Push-pull has much resemblance with **Classical MDS**. First of all, they both are non-linear, only require the pairwise distances and minimise a stress function. A difference is that the stress function in Classical MDS is explicitly defined, where the stress function in Push-pull is not. The correction function can in a way be seen as part of the stress function and summation over the correction of all pairs of input points would result in a stress function. Like MDS, emphasis can be given to small distances (through the correction function). An other difference is that the correction is not a global, but a local one (hence the correction function as opposite to the stress function). MDS uses *gradient descent* to alter the position of the projected points, whereas push-pull makes local changes. The non-metric variant of MDS has a strong resemblance to non-metric push-pull for the same reasons.

Stochastic Neighbour Embedding [30] is a probabilistic dimension reduction technique where neighbourhood identities are preserved. The neighbours of each object in high-dimensional space are defined using probability techniques. A noticeable difference with other techniques is that not necessarily every high-dimensional object is associated with a single one in the low-dimensional space.

2.8 Conclusions and further research

In this chapter we have given an overview of the push and pull model. We have shown the flexibility of the model and we have given guidelines on how to interpret and use the parameters of this model.

Further studies is required for the push-only variant on a closed surface. The advantages are clear: the input values are automatically scaled in such a way that the output space is optimally used.