A Fast Multivariate Nearest Neighbour
Imputation Algorithm

Norman Solomon, Giles Oatley and Ken McGarry

Abstract— **Imputation of missing data is important in many areas, such as reducing non-response bias in surveys and maintaining medical documentation. Nearest neighbour (NN) imputation algorithms replace the missing values within any particular observation by taking copies of the corresponding known values from the most similar observation found in the dataset. However, when NN algorithms are executed against large multivariate datasets the poor performance (program execution speed) of these algorithms can present major practical problems. We argue that these problems have not been sufficiently addressed, and we present a fast NN imputation algorithm that can employ any method for measuring the similarity between observations. The algorithm has been designed for the imputation of missing values in large multivariate datasets that contain many different missingness patterns with large proportions of missing data. The ideas underpinning the algorithm are explained in detail, and experiments are described which show that the algorithm delivers very good performance when it is used for imputation in both segmented and non-segmented datasets containing several million rows.**

Index Terms— **Missing data, Missingness patterns, Nearest neighbour imputation, Program execution time.**

I. INTRODUCTION

A considerable amount of work has been done to evaluate and compare the results produced by various multivariate nearest neighbour (NN) imputation algorithms [1]-[4] and to analyse the functionality and properties of such algorithms [5]-[7]. In addition, several methods for measuring the similarity between dataset rows when searching for the nearest neighbour have been proposed [8]-[10]. However, the slow performance (program execution speed) and the resulting poor scalability of multivariate NN imputation algorithms has received very little attention - i.e. searching the dataset for each nearest neighbour becomes increasingly impractical as the number of rows in that dataset increases. This paper discusses this performance problem, with particular reference to NN imputation in multivariate datasets containing several million rows, with many different missingness patterns and large proportions of missing data.

Generally, when NN algorithms are executed against

complete datasets they compare a particular dataset row with every other row when searching for that rows nearest neighbour (unless the dataset is segmented by class, so that this search can be limited to the subset of rows within a single class, as discussed in section V). Consequently, it is hard to see how the execution time of this type of NN algorithm can be decreased at the macro level. However, when NN algorithms are executed against incomplete datasets this is not the case. For example, suppose the dataset has 99% missing values. In this case many of the rows would be empty or they would have very few known values. Therefore, far fewer row comparisons would be required to find any particular nearest neighbour. Extending this idea it can be seen that as the proportion of missing data increases NN algorithm execution time can, in principle, be decreased by a corresponding proportion. This can be achieved by creating an algorithm that makes the best possible use of the information content within the missingness patterns that exist within the dataset. The following sections describe just such an algorithm. and evaluate it's performance.

Section II explains how the missingness patterns within the dataset rows can be used to find each nearest neighbour. Section III presents a new NN imputation algorithm and explains the ideas underpinning it's design. Section IV evaluates the algorithm's performance, using a set of simulated missing value datasets. Section V evaluates the algorithm's performance using two survey datasets and discusses the need for segmentation when performing NN imputation in large datasets. Section VI summarises the paper and discusses the issues it raises.

II. USING THE MISSINGNESS PATTERN STRUCTURE TO FIND EACH NEAREST NEIGHBOUR

Nearest neighbour algorithms impute a missing value in a particular matrix row (dataset observation) S_m by taking a copy of the known value from the most similar donor row S_i, such that $S_{mc} = S_{ic}$, where c is the matrix column (variable) that has a missing value. And where the most similar donor row is found by comparing S_m with all of the other rows in the matrix, and using the row that returns the smallest value of $d(S_m, S_i)$ as the donor - i.e. Finding the minimum value of the similarity measure $d(S_m, S_i)$ for all $S_i \in P$. Where $P = \{S_1, \dots S_n\}$ is the set of all matrix rows and where the variables in S_m and S_i are suitably scaled, so that each variable carries the required weight in the similarity

calculations. And where $d\left(S_m, S_i\right)$ can be measured using any similarity function, such as the simple Euclidean distance, or a more complex measure, such as the Hellinger distance [10] or the Mahalanobis distance [9] and [11].

However, only some $\left(S_m, S_i\right)$ pairs can be meaningfully compared, as shown in Fig. 1 (where 1 represents a known value and 0 represents a missing value). Fig 1 shows the rows that can be considered as potential donors when imputing the missing value in column 4 of row 1. Notice that row 3 can be a potential donor for any other row, because it has a full set of known values.

The most important point is that row 2 can be considered as a potential donor, but row 5 cannot, because the similarity between rows 1 and 2 cannot be meaningfully compared with the similarity between rows 1 and 5. For example, a Euclidean distance calculation would produce a smaller value when measuring the similarity between rows 1 and 5, because only one column would be included in the calculation, whereas 3 columns would be included in the calculation when measuring the distance between rows 1 and 2.

However, row 5 could be considered as a potential donor if some form of weighting was included in the similarity calculation to compensate for the reduced number of variables used to measure that similarity. This approach has been tried by [8], who use an experimental *"pseudo-nearest-neighbours"* method to impute missing values in multivariate Gaussian datasets, where the *"pseudo-similarity"* measurement implemented *"is actually a weighted correlation value between the two vectors with partially missing element values"*. However, this approach requires each imputed row to be compared with every other row in the matrix. Therefore, the method of enhancing NN algorithm performance described in section III could not be applied (see section VI for a further discussion of this issue).
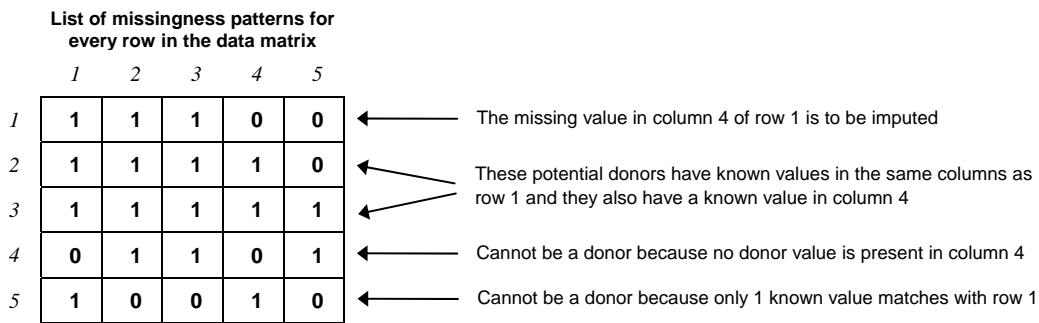
List of missingness patterns for every row in the data matrix

	1	2	3	4	5
1	1	1	1	0	0
2	1	1	1	1	0
3	1	1	1	1	1
4	0	1	1	0	1
5	1	0	0	1	0

The missing value in column 4 of row 1 is to be imputed

These potential donors have known values in the same columns as row 1 and they also have a known value in column 4

Cannot be a donor because no donor value is present in column 4

Cannot be a donor because only 1 known value matches with row 1

Fig 1 – Imputing the missing value in column 4 of row 1 in a data matrix

```
function  matrix  simple_NN_imputation_in_column ( int c,  matrix data )
    dataMatrixRow  missRow, donorRow, closestRow
    removeEmptyRowsIn ( data )

    for m = 1  to num_rows_in ( data )
        missRow  =  data ( m )
        if  ( missRow . patt ( c ) == 0 )
            minDistance  =  null
            for d = 1  to num_rows_in ( data )
                donorRow  =  donors ( d )
                if  ( donorRow . patt ( c ) == 1 )
                    match  =  true
                    j = 1
                    while ( j <= num_cols_in ( missRow )  &&  match == true )
                        if  ( missRow . patt ( j ) == 1  &&  donorRow . patt ( j ) == 0 )
                            match  =  false
                        end if
                        j ++
                    end while
                    if  ( match == true )
                        distance  =  euclideanDistanceBetween ( missRow, donorRow )
                        if  ( distance < minDistance  ||  minDistance == null )
                            minDistance  =  distance
                            closestRow  =  donorRow
                        end if
                    end if
                end if
            next d
            if  ( minDistance != null )
                missRow ( c )  =  closestRow ( c )
            end if
```

```
                          end  if
              next  m
              return  data
         end  function
```

Algorithm 1 – Simple nearest neighbour imputation algorithm

Algorithm 1 can be used to impute missing values in any multivariate dataset that has many missingness patterns, using the row comparison method shown in Fig. 1. The algorithm uses the Euclidean distance as a NN similarity measure, but any other measure could be substituted, as required.

The following parameters are passed to the algorithm; **(1)** *int c* is the *data* matrix column containing the values to be imputed. **(2)** *matrix data* is the data matrix, which is passed to the algorithm with missing values in column *c* and returned with imputed values in column *c*, where the statement *missRow (c) = closestRow (c)* imputes each missing value. And where each *dataMatrixRow* object contains a binary array *patt( )* which represents it's missingness pattern (see Fig. 1).

III. A FAST NEAREST NEIGHBOUR IMPUTATION ALGORITHM

The functionality of all NN imputation algorithms that process datasets with multivariate missingness patterns must be based on (be some variant of) the simple algorithm given above, since it will generally be necessary to test each row in the dataset to discover whether it can be meaningfully compared with the imputed row before measuring the similarity (where meaningful comparisons are defined as shown in Fig. 1 for the algorithms given here, but they could be defined otherwise, as required). More precisely, we can say that NN algorithms generally have a time complexity of $O(n^2)$, where n is the number of rows in the data matrix [12] and [13], i.e. the

algorithm execution time $T(n)$ is proportional to the square of the number of rows in the data matrix, such that $T(n) = cn^2$, where c is the constant of proportionality. However, the algorithm given below reduces the size of the constant of proportionality for all missing value datasets, which in turn reduces $T(n)$. An explanation of the ideas underpinning the algorithm's functionality is given below.

The donor matrix shown in Fig. 2 would be constructed by the algorithm when imputing the missing values in column 3 of every data matrix row that has missingness pattern 1, where 1 represents a known value and 0 represents a missing value. Only those data matrix rows that have the pattern shown in rows 2 and 4 can be added to this donor matrix, because these are the only rows with a known value in column 3 and with the same set of known values as the rows that have pattern 1.

It can be seen that pattern 5 also has missing values in column 3. However, the rows with pattern 5 cannot use the same donor matrix as the rows with pattern 1, since only those rows with pattern 4 can be used to construct the donor matrix in this case. Notice also that the rows with pattern 4 can be added to every donor matrix constructed by the algorithm, because pattern 4 has a full set of known values.

This method is faster than the simple algorithm given in the previous section because the search for each nearest neighbour is carried out within the subset of data matrix rows added to each donor matrix (rather than searching all of the rows in the data matrix every time).

List of missingness patterns in the data matrix

Donor matrix constructed by the NN algorithm

Each missingness pattern contains a set of matrix rows

	1	2	3	4
1	1	0	0	1
2	1	0	1	1
3	0	1	1	1
4	1	1	1	1
5	1	1	0	0

All data matrix rows that have missingness pattern 2

All data matrix rows that have missingness pattern 4

This donor matrix would be constructed by the NN algorithm when imputing the missing values in column 3 of missingness pattern 1.
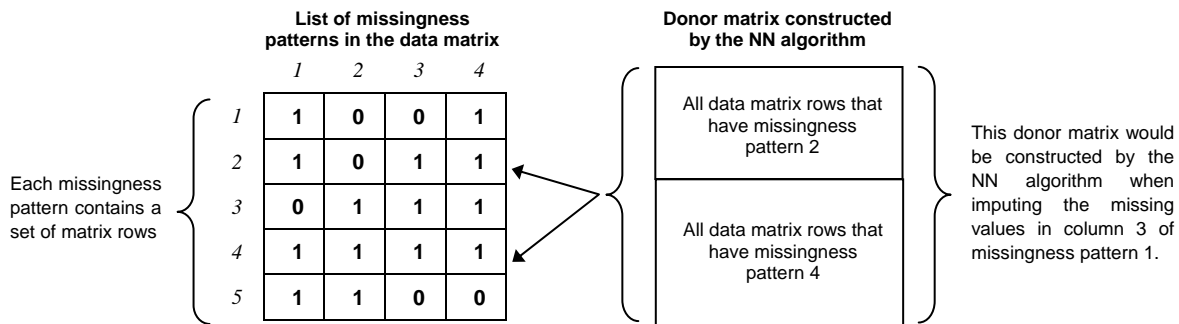
Fig 2 – Constructing a donor matrix for multivariate NN imputation

```
function  matrix fast_NN_imputation_in_column ( int c,  matrix data )
     missPatternRow  missPatt,  matchPatt
     dataMatrixRow  missRow,  donorRow,  closestRow
     removeEmptyRowsIn ( data )
     vector  patterns  =  missPatternListFor ( data )

     for  i  =  1  to  num_rows_in ( patterns )
          missPatt  =  patterns ( i )
          if  ( missPatt . patt ( c )  ==  0 )
               donors  =  new  vector ( )
               for  p  =  1  to  num_rows_in ( patterns )
                    matchPatt  =  patterns ( p )
                    if  ( matchPatt . patt ( c )  ==  1 )
                         match  =  true
                         j = 1
                         while ( j  <=  num_cols_in ( missPatt . patt )  &&  match  ==  true )
                              if  ( missPatt . patt ( j )  ==  1  &&  matchPatt . patt ( j )  ==  0 )
                                   match  =  false
                              end if
                              j ++
                         end while
                         if  ( match  ==  true )
                              for r  =  matchPatt . pattStartRow  to  matchPatt . pattEndRow
                                   donors . add_to_end ( data ( r ) )
                              next r
                         end if
                    end if
               next p

               for m  =  missPatt . pattStartRow  to  missPatt . pattEndRow
                    missRow  =  data ( m )
                    minDistance  =  null
                    for  d  =  1  to  num_rows_in ( donors )
                         donorRow  =  donors ( d )
                         distance  =  euclideanDistanceBetween ( missRow, donorRow )
                         if  ( distance  <  minDistance  ||  minDistance  ==  null )
                              minDistance  =  distance
                              closestRow  =  donorRow
                         end if
                    next d
                    if  ( minDistance  !=  null )
                         missRow ( c )  =  closestRow ( c )
                    end if
               next m
          end if
     next  i
     return  data
end  function
```

Algorithm 2 – Fast nearest neighbour imputation algorithm

The pseudo-code implementation of the method is given in Algorithm 2. The algorithm uses the Euclidean distance as a NN similarity measure, but any other measure could be substituted The parameters passed to the algorithm are the same as those passed to the simple algorithm, as described in section II.

The function *missPatternListFor (data)* creates and returns the *patterns* vector, which contains a list of *missPatternRow* objects, which represent the missingness patterns in the data matrix. Where each *missPatternRow* object has the following attributes; **(1)** A binary array *patt()* representing the missingness pattern, as shown in Fig. 2. **(2)** *pattStartRow* which gives the first row number of the pattern in the *data* matrix. **(3)** *pattEndRow* which gives the last row number of the pattern in the *data* matrix.

Note that the algorithm requires the data matrix to be sorted into missingness pattern order - i.e. all rows with the same missingness pattern must be adjacent. Further, this algorithm and the simple algorithm will produce the same set of imputed values when they are executed against the same data matrix, provided that the rows in this matrix are sorted into the same order (matrix row order affects the values imputed by NN algorithms). However, the simple algorithm does not require the matrix to be in any particular order when it is executed independently.

IV. PERFORMANCE EVALUATION USING SIMULATED MISSING VALUE DATASETS

The pseudo-code versions of the two NN algorithms described in the preceding sections have been implemented using the Microsoft Visual C# programming language. This section evaluates the performance of these algorithms when they are used to impute missing values in the simulated datasets described below.

The experiments were performed to try to answer the following questions; **(1)** Would algorithm execution time decrease as the proportion of missing data in the matrix was increased? **(2)** Would the method of creating donor matrices (as implemented in the algorithm above) decrease algorithm execution time?

Table I – Comparison of NN algorithm execution times.

Number of rows in the data matrix	% of missing values in the data matrix	Simple algorithm execution time (seconds)	Fast algorithm execution time (seconds)
10,000	25	7	5
	50	5	2
	75	4	1
20,000	25	26	21
	50	20	9
	75	16	3
30,000	25	58	50
	50	46	22
	75	37	7
40,000	25	104	87
	50	78	38
	75	62	11
50,000	25	159	136
	50	125	58
	75	102	18
60,000	25	233	193
	50	181	94
	75	140	29
70,000	25	315	275
	50	238	127
	75	195	34
80,000	25	417	353
	50	314	159
	75	241	45
90,000	25	529	445
	50	421	195
	75	300	57
100,000	25	649	534
	50	515	247
	75	379	72
110,000	25	787	655
	50	603	300
	75	464	95
120,000	25	918	765
	50	720	357
	75	559	118

The experiments were performed against 12 randomly generated data matrices containing between 10,000 and 120,000 rows, as shown in column 1 of Table I. The data values inserted into these matrices were randomly generated integers in the range 1 to 100, where 25%, 50% and 75% of these values were randomly deleted in three stages. This process generated
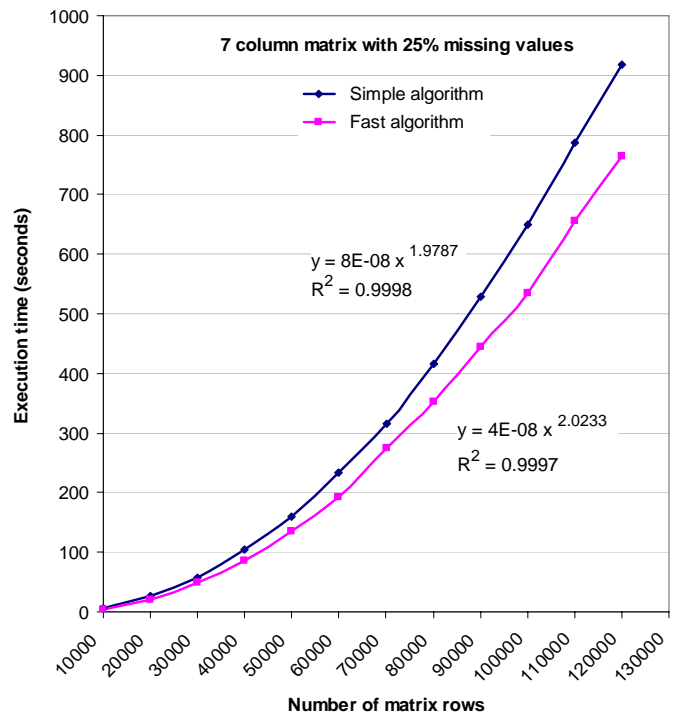
36 different matrices, as shown in column 2 of Table I. Each of these matrices contained 7 columns, where the missing values in column 1 were imputed for every experiment.
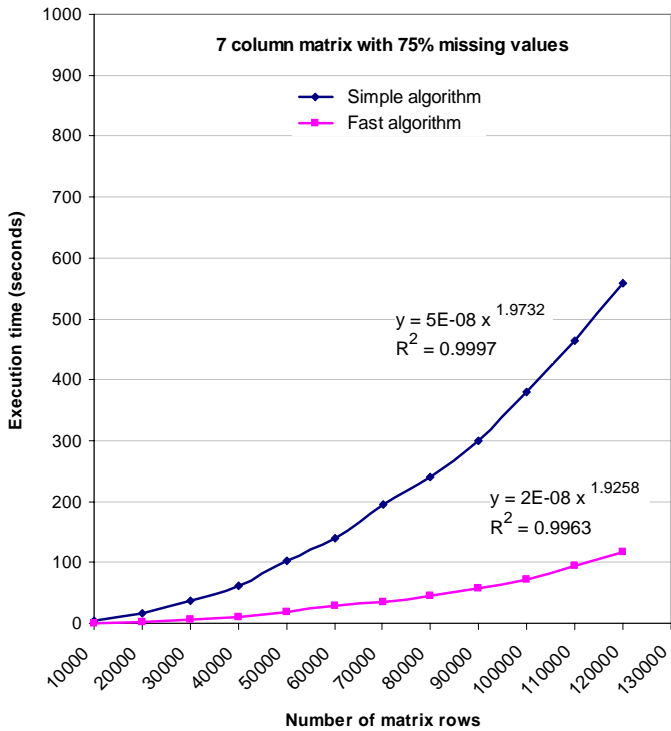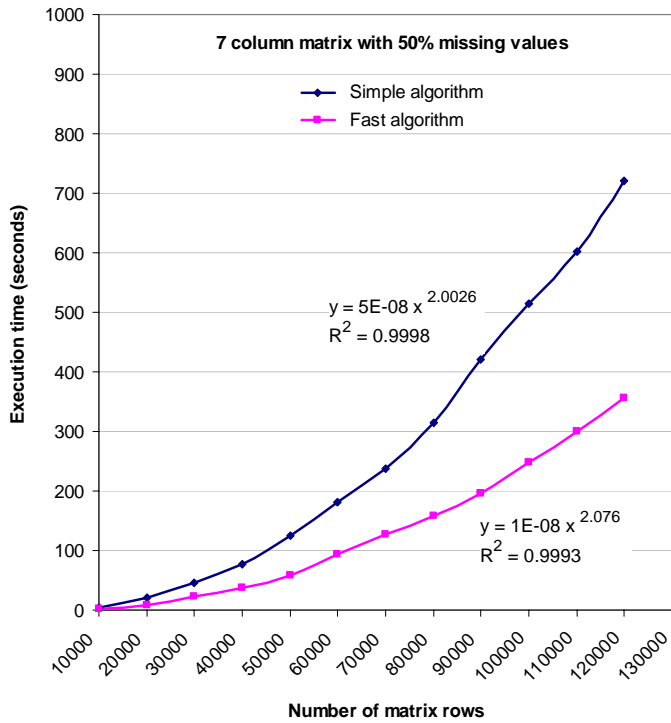
Each matrix contained the maximum possible number of missingness patterns. Generally, the maximum number of missingness patterns in any matrix $= 2^n$, where n is the number of columns in the matrix. In this case each matrix contained 7 columns, therefore $2^7 = 128$ patterns were added to each matrix. Where these patterns were balanced and evenly distributed across the rows in each matrix, because the missing values were deleted completely at random.

Both of the algorithms described in the two preceding sections were executed against all 36 of the matrices described above, thus creating 72 sets of experimental results, as shown in columns 3 and 4 of Table I. The results given in Table I are also presented in line chart form below. Algorithm execution times are shown on the y axis of each chart. The number of rows in the data matrices are shown on the x axis.

The execution time increases for both algorithms as the number of matrix rows increases, as expected. However, execution time decreases as the proportion of missing data in each matrix increases, which confirms the central hypothesis - i.e. the fastest execution times are shown on the final chart, where each matrix has 75% missing values.

The charts show that the fast algorithm executed more quickly than the simple algorithm for every experiment - i.e. the method of creating donor matrices implemented as part of the fast algorithm did in fact decrease the execution times. This performance advantage increases as the proportion of missing data increases, with the largest difference between the algorithm execution times occurring for the largest input matrix shown on the final chart.

7 column matrix with 25% missing values

$y = 8\text{E-}08 \, x^{1.9787}$
$R^2 = 0.9998$

$y = 4\text{E-}08 \, x^{2.0233}$
$R^2 = 0.9997$

7 column matrix with 50% missing values

$y = 5\text{E-}08\ x^{2.0026}$
$R^2 = 0.9998$

$y = 1\text{E-}08\ x^{2.076}$
$R^2 = 0.9993$

7 column matrix with 75% missing values

$y = 5\text{E-}08\ x^{1.9732}$
$R^2 = 0.9997$

$y = 2\text{E-}08\ x^{1.9258}$
$R^2 = 0.9963$

The regression equations shown on the charts are very close to the expected $T(n) = cn^2$ results, with the very small differences in the expected exponents perhaps explained by the small samples used for the regression calculations. The high R^2 values suggest that these equations can be used to predict algorithm execution times for larger matrices, as shown in Table II.

Table II – Predicted execution times (given to the nearest hour) for the algorithms described above.

Number of rows in the data matrix	% of missing values in the data matrix	Simple algorithm execution time (hours)	Fast algorithm execution time (hours)
1,000,000	25	17	15
	50	14	8
	75	10	2
2,000,000	25	65	62
	50	58	33
	75	38	8
3,000,000	25	146	142
	50	130	78
	75	84	17
4,000,000	25	257	253
	50	231	141
	75	148	29
5,000,000	25	400	398
	50	361	224
	75	230	44

V. PERFORMANCE EVALUATION USING TWO SEGMENTED SURVEY DATASETS

The experiments described in the previous section were performed against simulated datasets. However, it is also important to evaluate the algorithm's performance using real datasets. For example, the simulated datasets all contained a full set of well balanced and evenly distributed missingness patterns, but this can hardly be expected to occur in practice. This point is important for the current investigation, since the missingness pattern structure is the most crucial factor affecting algorithm performance.

The experiments described below evaluate the algorithm's performance when it is executed against segmented datasets. This is an important consideration, because in practice datasets are often segmented by class, so that the search for each nearest neighbour can be limited to the subset of rows within a single class [5]. The required classes can be created by segmenting the dataset using a fully observed categorical variable (as for the experiments described below) or by dividing the dataset into numeric class intervals using a fully observed numeric variable. This approach can improve the quality of the imputed values *and* reduce the execution time of the NN algorithm. However, if the segmentation process is carried out *only* to reduce algorithm execution time it can reduce the quality of the imputed values, for example;

- In some cases segmenting the dataset may not produce the best imputation results - i.e. searching the entire dataset for donors may be the only sensible way to find the best estimates for the missing values.
- Segmenting the dataset by category can produce poorly categorised observations when the categories are created *only* to improve performance. This can result in donor rows being selected from the wrong category.
- The boundaries selected for numeric class intervals can be

somewhat arbitrary when they are chosen *only* to improve performance. This can result in the selection of sub-optimal donors, taken from the wrong class.

A. Performing the experiments and interpreting the results

This section evaluates the performance of the new algorithm when it is used to impute missing values in the survey datasets described below. The experiments were performed to try to answer the following questions; **(1)** Would dividing the matrix into an increasingly large number of segments steadily decrease algorithm execution time? **(2)** Would the method of creating donor matrices decrease algorithm execution time for large segmented datasets?

The experiments described below were performed against two similar survey datasets describing various different Firms (business enterprises) within the UK. The first survey dataset described 1,128,463 MICRO Firms, which are defined as those Firms with less than 10 employees. The second dataset described 271,955 SMALL Firms, which have between 10 and 49 employees. Both datasets contained the 10 variables described below. The first 4 variables listed were fully observed, but the other 6 variables all had large proportions of missing data.

- The *UKSIC Category* (United Kingdom Standard Industrial Classification Category) is a categorical variable which defines the commercial activities carried out by each Firm, such as *"Publishing of software"* etc.
- The *OS Easting* and *OS Northing* variables specify the geographical location of each Firm, using UK Ordnance Survey mapping co-ordinates.
- The *Number of Employees* variable specifies the number of people employed by each Firm.
- The *Payroll, Sales, Net Worth, PBT, Director Pay* and *Depreciation* variables describe each Firm's financial situation. These six variables all had large proportions of missing data.

The *UKSIC Category* was used to segment both datasets at 4 different levels of granularity (but it was not used to measure the similarity between Firms, as the other 9 variables were), where the number of segments created increased as the level of granularity was increased. This process created 8 different data matrices, as shown in column 5 of Table III, below.

The algorithms described in sections II and III were executed against each of the 8 matrices created, where the missing *Payroll* values were imputed for every experiment. This process created 16 sets of experimental results, as shown in the 2 rightmost columns of Table III.

The algorithms described in sections II and III were amended so that the search for each nearest neighbour could be limited to the subset of rows within a single *UKSIC Category*. The amendments made to the algorithms were quite simple, but they are not shown in the pseudo-code versions of the algorithms for reasons of clarity. However, it should be noted that each donor matrix created by the fast algorithm (see Fig. 2) was segmented at the required level of granularity immediately after it was created (while the algorithm was running). For optimum performance a lookup table (a vector) was then created which stored the first row number of each category within each donor matrix, where each donor matrix was sorted by *UKSIC Category* so that this could be achieved.

The figures given in the 3 rightmost columns of Table III show that dividing the matrix into an increasingly large number of segments steadily decreased the execution time required for both algorithms. This occurred because as the number of segments was increased the search for each nearest neighbour took place within a smaller number of rows.

The figures in brackets given in the rightmost column of Table III show the improvement in performance offered by the new algorithm for both datasets. For example, the first row of figures show that fast algorithm executed in just under one seventh of the time (given to the nearest minute) taken by the simple algorithm. It can be seen that these performance improvements are similar for both datasets, regardless of the number of segments created.

Table III – Comparison of algorithm execution times using two survey datasets which were segmented into an increasingly large number of categories.

Dataset description (number of employees)	Number of rows in the data matrix	% of missing values in the data matrix	Number of missingness patterns	Number of category segments	Simple algorithm execution time (minutes)	Fast algorithm execution time (minutes)	
MICRO Firms (with less than 10 employees)	1,128,463	61.72%	28	58	312	44	(0.14)
				203	119	16	(0.13)
				412	53	9	(0.17)
				488	47	8	(0.17)
SMALL Firms (with 10 to 49 employees)	271,955	54.97%	27	57	17	6	(0.35)
				200	6	2	(0.33)
				409	3	1	(0.33)
				485	3	1	(0.33)

VI. SUMMARY AND DISCUSSION

The slow performance and the resulting poor scalability of multivariate NN imputation algorithms is a major problem that has not been sufficiently addressed. Consequently, we have presented a new, fast, NN imputation algorithm that can be used to impute missing values in large multivariate datasets containing many different missingness patterns. We have also described a set of experiments which have confirmed the hypothesis that NN imputation algorithm execution times would decrease as the proportion of missing values in the dataset was increased. The experiments have also shown that the method of creating donor matrices (implemented as part of the new algorithm) significantly decreases NN algorithm execution time for both segmented and non-segmented datasets containing several million rows.

We argue that the following three questions are the most appropriate ones to ask when evaluating the quality of any multivariate NN imputation process. **(1)** Is the similarity measure used suitable for the missing value dataset? - i.e. does this measure find the best possible donor rows for the imputation of missing values? **(2)** Is the method used to decide which dataset rows should be considered as potential donors appropriate for the dataset? - e.g. if the dataset has been segmented, has this been done in such a way that the search for donors takes place within the best possible subset of rows? **(3)** Is the method used to decide which rows can be meaningfully compared logical, given the nature of the data? This question is discussed further, below.

The first two questions posed above need not be asked for the algorithm presented here, because it is generic by design - i.e. It can employ *any method* for measuring the similarity between observations and it can be executed against *any sort* of segmented dataset, regardless of the classification scheme used. On the other hand, the algorithm does specify which rows can be meaningfully compared, as described in section II. We argue that this method is the most logical approach for most missing value datasets, for the following reasons.

In surveys, respondents often fail to answer one or more of the questions put to them - i.e. It is often the case that every variable in a survey dataset will have a certain proportion of missing values [14]. The method we propose can be applied to all such datasets (from surveys or otherwise), regardless of the structure and distribution of the missingness patterns they contain.

Employing any method that uses *only* the fully observed variables to measure the similarity between observations *does not* make full use of every known value in the dataset. We argue that the method we propose is more logical, since it attempts to make the best possible use of the information content within *all of the known values*, so as to generate the best possible estimates for the missing values.

Employing any method that involves comparing rows that do *not have* a common set of known values would require the use of a similarity measure that returns *different values* depending on the number of common values in the rows compared. We

argue that this would build an unnecessary level of uncertainty into the NN imputation process.

REFERENCES

[1] Wasito, I. and Mirkin, B., (2005), Nearest neighbour approach in the least squares data imputation algorithms, *Information Sciences, 169 (1)*, pp. 1-25.

[2] Wasito, I. and Mirkin, B., (2006), Nearest neighbours in least-squares data imputation algorithms with different missing patterns, *Computational Statistics & Data Analysis, 50 (4)*, pp. 926-949.

[3] Durrant, G. B., (2005), Imputation methods for handling item-nonresponse in the social sciences: A methodological review, *ESRC National Centre for Research Methods: Methods Review Working Paper No. NCRM/002*. Available; http://www.ncrm.ac.uk/publications/index.php Accessed 26th March 2007.

[4] Kalton, G., *Compensating for Missing Survey Data*, Ann Arbor, MI: Survey Research Center, University of Michigan, 1983.

[5] Chen, J. and Shao, J., (2000), Nearest neighbour imputation for survey data, *Journal of Official Statistics, 16 (2)*, pp. 113-131.

[6] Fay, R. E., (1999), Theory and applications of nearest neighbor imputation in census 2000, *Proceedings of the section on survey research methods, American Statistical Association 1999*, pp. 112-121.

[7] Rancourt, E., Sarndal, C.E. and Lee H., (1994), Estimation of the variance in the presence of nearest neighbor imputation, *Proceedings of the section on survey research methods, American Statistical Association 1994*, pp. 883-893.

[8] Huang, X. and Zhu, Q., (2002), A pseudo-nearest-neighbor approach for missing data recovery on Gaussian random data sets, *Pattern Recognition Letters, 23 (13)*, pp. 1613 – 1622.

[9] Stage, A.R. and Crookston, N.L., (2002), Measuring similarity in nearest neighbor imputation: Some new alternatives, *In Proceedings of the Symposium on Statistics and Information Technology in Forestry, Blacksburg, Virginia*, pp. 91-96.

[10] Lee, C.H. and Shin, D.G., (1999), Using Hellinger distance in a nearest neighbour classifier for relational databases, *Knowledge-Based Systems, 12 (7)*, pp. 363-370.

[11] Mahalanobis, P.C., (1936), On the generalised distance in statistics, *Proceedings of the National Institute of Science of India*, pp. 49-55.

[12] Dunham, M. H., *Data Mining Introductory and Advanced Topics*, Prentice-Hall, New Jersey, 2003, pp. 125-142

[13] Aho, A.V., Hopcroft, J.E. and Ullman, J.D., *Data Structures and Algorithms*, Addison Wesley, MA, 1983, pp. 16-24

[14] Allison, P. D., *Missing Data (Quantitative Applications in the Social Sciences, series no. 07-136)*, Sage Publications, California, 2001, pp. 1-12.