Querying Metric Spaces with Bit Operations

Richard Connor[1] and Alan Dearle[2]

[1] Department of Computing Science, University of Stirling, FK9 4LA, Scotland
[2] School of Computer Science, University of St Andrews, KY16 9SS, Scotland
richard.connor@stir.ac.uk
alan.dearle@st-andrews.ac.uk

Abstract. Metric search techniques can be usefully characterised by the time at which distance calculations are performed during a query. Most exact search mechanisms use a "just-in-time" approach where distances are calculated as part of a navigational strategy. An alternative is to use a "one-time" approach, where distances to a fixed set of reference objects are calculated at the start of each query. These distances are typically used to re-cast data and queries into a different space where querying is more efficient, allowing an approximate solution to be obtained.
In this paper we use a "one-time" approach for an exact search mechanism. A fixed set of reference objects is used to define a large set of regions within the original space, and each query is assessed with respect to the definition of these regions. Data is then accessed if, and only if, it is useful for the calculation of the query solution.
As dimensionality increases, the number of defined regions must increase, but the memory required for the exclusion calculation does not. We show that the technique gives excellent performance over the SISAP benchmark data sets, and most interestingly we show how increases in dimensionality may be countered by relatively modest increases in the number of reference objects used.

1 Context

To set a formal context, we are interested in searching a (large) finite set of objects S which is a subset of an infinite set U, where (U, d) is a metric space: that is, an ordered pair (U, d), where U is a domain of objects and d is a total distance function $d : U \times U \rightarrow \mathbb{R}$, satisfying postulates of non-negativity, identity, symmetry, and triangle inequality [20]. The general requirement is to efficiently find members of S which are similar to an arbitrary member of U given as a query, where the distance function d gives the only way by which any two objects may be compared. There are many important practical examples captured by this mathematical framework, see for example [16, 20]. The simplest type of similarity query is the *range search* query: for some threshold t, based on a query $q \in U$, the solution set is $R = \{s \in S | \, d(q, s) \leq t\}$.

The essence of metric search is to spend time pre-processing the finite set S so that solutions to queries can be efficiently calculated using only distances among objects. In all cases therefore, distances between the data and selected

reference or "pivot" objects are calculated during pre-processing, and at query time distances between the query and the same pivot objects can be used to make deductions about which data values may, or may not, be candidate solutions to the query.

Mechanisms for metric search can be divided into two main categories, which we define as using "just-in-time" and "one-time" distance calculations between the query and these pivot objects. With "just-in-time" solutions, the manner in which data is stored reflects proximity within the data set, and indexing structures attempt to allow navigation towards subsets where possible solutions may exist. As navigation occurs, distances to objects related to these local subsets are calculated. The idea is that as the computation progresses, subsets of the data which are geometrically distant from the query are never accessed.

With "one-time" solutions, a selection of pre-determined reference objects is used, and distances to all these are calculated for every element during construction. These distances are used to re-cast the original space into some other space where indexing properties are better, distance calculations are cheaper, or both. Typically the main tradeoff is that the extra query efficiency is achieved in return for a loss of semantic query effectiveness, so such mechanisms are either approximate, or produce candidate sets of results from within which the true results must be determined by re-accessing the original data.

2 Introduction

In this paper, we present a combination which we believe is completely novel, and which is robust in the face of increasing dimensionality. We use a "one-time" approach for an exact search mechanism, and thus characterise the original search space by the distances between each element and a fixed set of reference objects. However, instead of using this information to re-cast the data into some cheaper space, we instead assess queries in terms of the original metric space.

Our initial characterisation is used to define a large set of binary partitions over the original space; the data is stored as a set of bitmaps according to containment with these regions. At query time, the query is assessed against all of the regions, but without reference to the data representation. For each region, one of three possible conditions may be determined: (a) the solution set to the query must be fully contained in the region, or (b) there is no intersection between the region and the solution set, or (c) neither of these is the case. In either case (a) or (b), the containment information stored may be useful with respect to solving the query, and will be fetched from disk as part of the solution computation. In case (c) however the containment information is of no value, and is not accessed as a part of the computation. This approach maximises the effectiveness of memory used for calculation against the data representation. Indeed the amount of memory required depends on the size of the data but, critically, not its dimensionality.

As dimensionality increases, then ever more regions will fail to contribute knowledge towards the possible solution set. Since these regions are not involved

in the exclusion computation, they do not significantly impact upon the memory or time required to perform it. Thus whilst an increase of dimensionality will require a larger representation of the data in secondary storage to remain effective, it does not increase the memory requirement for performing the exclusion calculation.

2.1 Illustrated Example

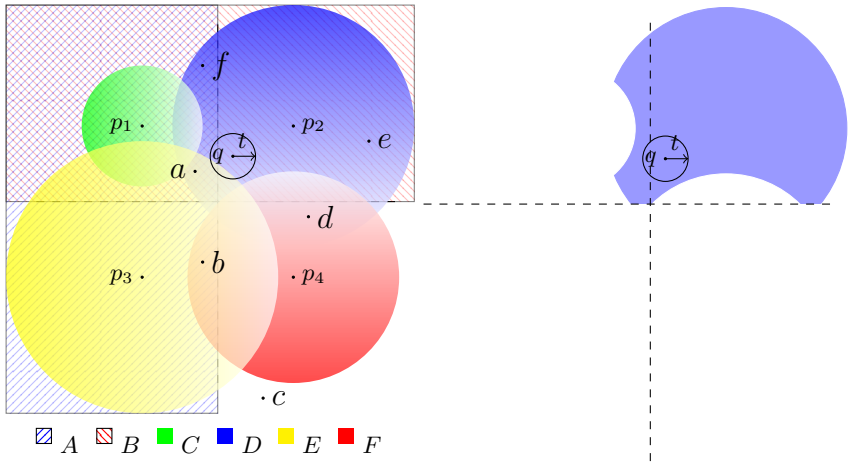

Fig. 1. Any solution to the query q with threshold t must be in the circle centred around p_2, and above the dashed horizontal line; it cannot be in the circles centred around p_1 or p_4. No information is available wrt the circle around p_3 nor the vertical line; these partitions take no part in the exclusion calculation as the region boundaries intersect the query solution boundary. The shaded area shown on the right shows the possible loci of query solutions with respect to these regions.

Figure 1 shows a simple example within the 2D plane, comprising four reference objects p_1 to p_4 and a set of six regions defined by them. The regions are respectively: A, the area to the left of the hyperplane between p_1 and p_2; B, the area above the hyperplane between p_1 and p_3; and the areas within the variously sized circles drawn around each p_1 to p_4, labelled C, D, E and F respectively. Note that each regional boundary defines a binary partition of the total space, such that each element of the space is either in, or out, of the region, and that this membership is defined only in terms of distances from defined reference objects. Thus in Figure 1, $A = \{u \in U | d(u, p_1) \geq d(u, p_2)\}$, $C = \{u \in U | d(u, p_1) \leq \mu\}$ for some value of μ, etc.

Note that the number of regions that can be defined from a fixed set of reference objects is potentially very large; for example in the experiments described

Table 1. The data representation according to containment of regions (see Figure 1). The representation of the query $\mathcal{Q}$ is equivalent to the CNF expression $B \wedge \neg C \wedge D \wedge \neg F$

Point	Regions					
	A	B	C	D	E	F
a	true	true	false	true	true	false
b	true	false	false	false	true	true
c	false	false	false	false	false	false
d	false	false	false	true	false	true
e	false	true	false	true	false	false
f	true	true	false	true	false	false
$\mathcal{Q}$	$\cap$	true	false	true	$\cap$	false

in this paper, for n reference objects, we define $\binom{n}{2} + 5n$ regions for n reference objects, by using all hyperplane boundaries and defining five hypersphere radii per object.

Figure 1 also shows a range query q drawn with a threshold t. It can be seen that all solutions to this query must lie within the area highlighted on the right hand side of the figure. The hypersphere around the query intersects with two regional boundaries, and so no information is available with respect to these; however it is completely contained within two of the defined regions, and fails to intersect with the final two. Such containment and intersection is derivable only from the measurement of distances between the query and the reference objects, the definition of the regions, and the search radius. Here for example the possible solution area shown is determined using only the four distance calculations $d(q, p_1)..d(q, p_4)$.

Table 1 shows how the example data objects a to f are stored in terms of their regional containment. The row labelled $\mathcal{Q}$ shows the containment relation between the entire query ball and each defined region. Where the boundaries intersect, a $\cap$ is shown; where they do not, a Boolean value shows whether the query ball is contained or otherwise. Therefore, the only possible solutions to the query are those which match on all non-intersecting fields; in this case, the objects a, e and f. Note that this is equivalent to the Conjunctive Normal Form (CNF) expression $B \wedge \neg C \wedge D \wedge \neg F$. This expression therefore covers the set of all possible solutions to the query.

The full set of query solutions can therefore be evaluated in three phases as follows: (a) the query is checked against the region definitions; (b) the useful information from this phase is used to conduct a series of bitwise operations to identify a set of candidate solutions for the query; and (c) the candidates are checked against the original set. This gives interesting performance tradeoffs; phase (b) should be almost constant cost, as a set of $\log n$ orthogonal, balanced partitions should exclude almost all incorrect solutions from the candidate set. The cost of phase (c) directly depends on how well this can be achieved, and will always be better with a larger number of regions which will require an increase in the cost of phase (a).

As dimensionality increases, then so will the proportion of intersections between the query and region boundaries. This can be countered by defining more regions, which will increase the storage cost of the index structure, and the cost of computing phase (a) of the query, but need *not* consequentially require any increase in memory or time for the ensuing phases of the computation.

2.2 Contribution

The contribution of this paper lies in the combination of using a fixed set of reference objects to characterise the data, followed by their use in an exact metric search algorithm which maximises the efficacy of memory use. There are many mechanisms which use similar fixed sets of reference objects (see for example Section 5); our mechanism is particularly well-suited to exact search as the dimensionality of the underlying data increases, by minimising the memory footprint required for the search algorithm. Furthermore our algorithm is inherently decomposable and parallelisable, and is well-suited to implementation on modern processors, including GPUs.

In this paper we give the basic mechanism, show its feasibility with respect to some well-known (relatively small and low-dimensional) data sets, and also show its performance as intrinsic dimensionality increases. This is an early exposition of these ideas, and there are many more aspects to study.

3 Core Mechanism

3.1 Data structures

Before describing the algorithm in more detail, we describe the data structures used in the algorithm and their initialisation. We refer to a finite space (S, d) which is a subset of an infinite metric space (U, d).

A set P of enumerated reference objects p_0 to p_m is first selected from the finite metric space S. Based on this, we define a set of surfaces within U, defined according to the distance function d, each of which divide U into two parts. Surfaces within U are either *balls*, for example $\{u \in U \mid d(u, p_i) = \mu\}$ for some values i, μ, or *sheets*, for example $\{u \in U \mid d(u, p_i)) = d(u, p_j)\}$ for some values i, j. For each such surface, it is easy to categorise any element of u_i of U as being inside or outside an associated region, according to whether it is on the same side of the surface as p_i or otherwise. Note that there are many more regions than reference objects; for example a set of m reference objects immediately defines $\binom{m}{2}$ hyperplane regions, and can be used to define many more than this. In Section 3.3 we discuss further the selection of regions from the available reference objects.

We now define the notion of an *exclusion zone* as a containment map of S based on a given region; this is the information we will use at query time to perform exclusions and derive a candidate set of solutions. We impose an ordering on S, then for each s_i map whether it is a member of the region or

otherwise. This logical containment information is best stored in a bitmap of n bits, where $n = |S|$. One such exclusion zone is generated per region and stored as the primary representation of the data set.

It is worth noting that an essential difference between our mechanism and others that use the same characterisation of the data (see Section 5) is that each of our bitmaps represents the containment of the whole data set within an individual region, rather than those regions which contain an individual object. The same information is thus divided with the opposite orientation; with reference to Table 1, we store the columns rather than the rows.

3.2 Query

The query process comprises three distinct phases as mentioned above:

Phase 1 Initially, the distance from the query q to each reference object p_i is measured. For each region, it can be established if the boundary of the solution ball intersects with the boundary of the region. For a ball region defined by reference object p_i and a radius μ, then the condition for intersection is

$$|d(p_i, q) - \mu| \leq t$$

For a sheet region, the condition depends on whether the metric d has the supermetric property (see Section 5) or otherwise: if it does, the intersection condition is

$$\frac{|d(p_i, q)^2 - d(p_j, q)^2|}{2d(p_i, p_j)} < t$$

otherwise the condition is

$$\frac{|d(p_i, q) - d(p_j, q)|}{2} < t$$

If the intersection condition holds, then the exclusion zone related to the region is not considered further; if it does not, then the exclusion zone is brought into the query calculation in one of two sets, depending on whether the query solutions are fully contained within, or without, the region in question. We will name these two sets of bitmaps B_{in} and B_{out}.

Phase 2 The second phase comprises the manipulation of the bitmaps deriving from the first phase to identify a set of candidate solutions. This may be efficiently achieved by a series of bitwise operations over these bitmaps. The solution to the query is guaranteed to lie within the intersection of the inclusion sets (derived by bitwise and operations) and not in the union of the exclusion set (derived by bitwise or). Thus any solution is guaranteed to be identified by the bitmap deriving from the following logical expression:

$$\left(\bigwedge_{b \in B_{in}} b \right) \wedge \left(\neg \left(\bigvee_{b \in B_{out}} b \right) \right)$$

Phase 3 The last phase consists of filtering the result sets derived in phase 2 against the original space and distance metric in order to produce an exact solution to the query.

3.3 Efficiency considerations

No data is considered during Phase 1, and n distance calculations can be used to generate a great many judgements. With increasing dimensionality, a larger number of reference objects will be required to usefully characterise the space, as more queries will intersect with each region, and therefore a greater number of regions is required to maintain the size of $B_{in} \cup B_{out}$. However this adds no further cost to the second and third phases of the query.

If each bitmap used in the Phase 2 calculation is balanced, i.e. it contains the same number of 0s and 1s, and orthogonal, i.e. there is no logical dependency among them, then only $\log_2 n$ bitmaps (where $n = |S|$) are required to exclude almost all non-relevant data. As each bitmap is n bits long, this gives a space requirement of $O(n \log n)$, and a time requirement of $O(\log n)$. In this sense, the solution can not be said to be scalable. However it is important to look more deeply than this: the space requirement is literally $n \log_2 n$ bits, which cannot cause any real problem for any context where any n objects are being stored - even if n is huge, $\log_2 n$ bits is unlikely to approach the size of a single data object. If the bitmaps are huge, as there is no logical internal dependency they can be partitioned and accessed in parallel. Furthermore the time requirement on modern hardware is likely to approximate to a small constant time even with relatively large values of $log_2 n$.

Phase 3 is essentially optional, and required only for exact search. Alternatively, the first two phases can be considered as an approximate search technique. Whether this is desirable or not depends on how well the data is characterised according to the selected regions; in Section 4 we show an example where the number of false positives is so small this phase is hardly required.

Finally it can be noticed that each of the three phases of the computation are inherently parallelisable, and in particular Phase 2 should be extremely efficient on modern hardware, comprising as it does only parallelisable bitwise operations.

Balancing In our initial experiments, we have tried both balanced and unbalanced bitsets. Balancing can be achieved by selecting a set of *witness* objects from the finite space S and finding a median distance or offset for these, so that the regional boundary divides the finite set into two equal parts. A large enough set of witness objects will give a good statistical approximation to the distribution of S. For ball partitions, the median distance to the centre is used; for sheet partitions, an offset can be selected left or right of the central hyperplane [5, 11]. Furthermore, for supermetric spaces, the XY plane can also be rotated to maximise the spread of values as described in [8].

We still have much to investigate in terms of finding the best parameters for a given data set. In the meantime we note that balancing does increase performance with lower-dimensional spaces and smaller query thresholds, but that it starts to have a detrimental effect as either of these increases - the tradeoff being that balancing will increase the effectiveness of the second phase algorithm, but decrease the effectiveness of the first phase. In general, as query radius or

dimensionality increases, we perceive that this is best offset by an increase in reference objects but also some controlled rebalancing of regional offsets.

4 Experiments and Results

Experiment 1 In the first experiment we investigate the efficacy of the algorithm by running queries against the SISAP *nasa* and *colors* data sets [9]. The metric used for both datasets is Euclidean distance. The number of reference objects is varied from 10 to 60 in steps of 5 and queries comprising 10% of the dataset are made against the remaining 90%. Thresholds of $t_0 = 0.12, t_1 = 0.285$, and $t_2 = 0.53$ are used for *nasa* and $t_0 = 0.052, t_1 = 0.083, t_2 = 0.13$ for *colors*. In this set of experiments the number of ball radii is set to 5 with the radii being set to a mean radius of 1.81[3] and mean ± 0.3 and mean ± 0.6. We report residual distance calculations, which are the number of calculations made in phase 3 of the algorithm, excluding reference object distance calculations. The results of this experiment are shown in Figure 2, with the figures for 60 reference objects (the right hand side of the graph) given in Table 2. To put these figures into the metric indexing context, the top two rows of the table give the number of distance calculations per query reported in [8] for the Distal SAT operating with both normal metric and supermetric exclusion mechanisms.

Table 2. Residual distance calculations required when 60 reference objects are used (the numbers reported do not include the 60 distance calculations required for these.) The top two rows give comparable figures for the state-of-the-art Distal SAT.

	colors			nasa		
	t_0	t_1	t_2	t_0	t_1	t_2
DiSAT: metric	4049	9112	19745	554	2176	6448
DiSAT: supermetric	2015	5737	16199	320	1300	5444
metric, unbalanced	4207	14930	42139	14	1120	9202
metric, balanced	2246	9610	30250	11	822	7671
supermetric, unbalanced	544	**3114**	**13045**	3	296	4122
supermetric, balanced	**518**	3259	17512	**1**	**91**	**2204**

The three sets of graphs shown in Figure 2 illustrate a number of interesting facets of the algorithm. As would be expected, increasing the number of reference objects has a significant effect of the number of distance calculations performed. However the number of reference objects used in these experiments is really quite small, much smaller than has been reported for other regional approaches eg [13, 1] given the accuracy shown by the relatively small number of residual distance calculations required. In particular we have the quite stunning result

[3] For balanced versions (see Section 3.3) the central radius is reset, but the same increments are applied

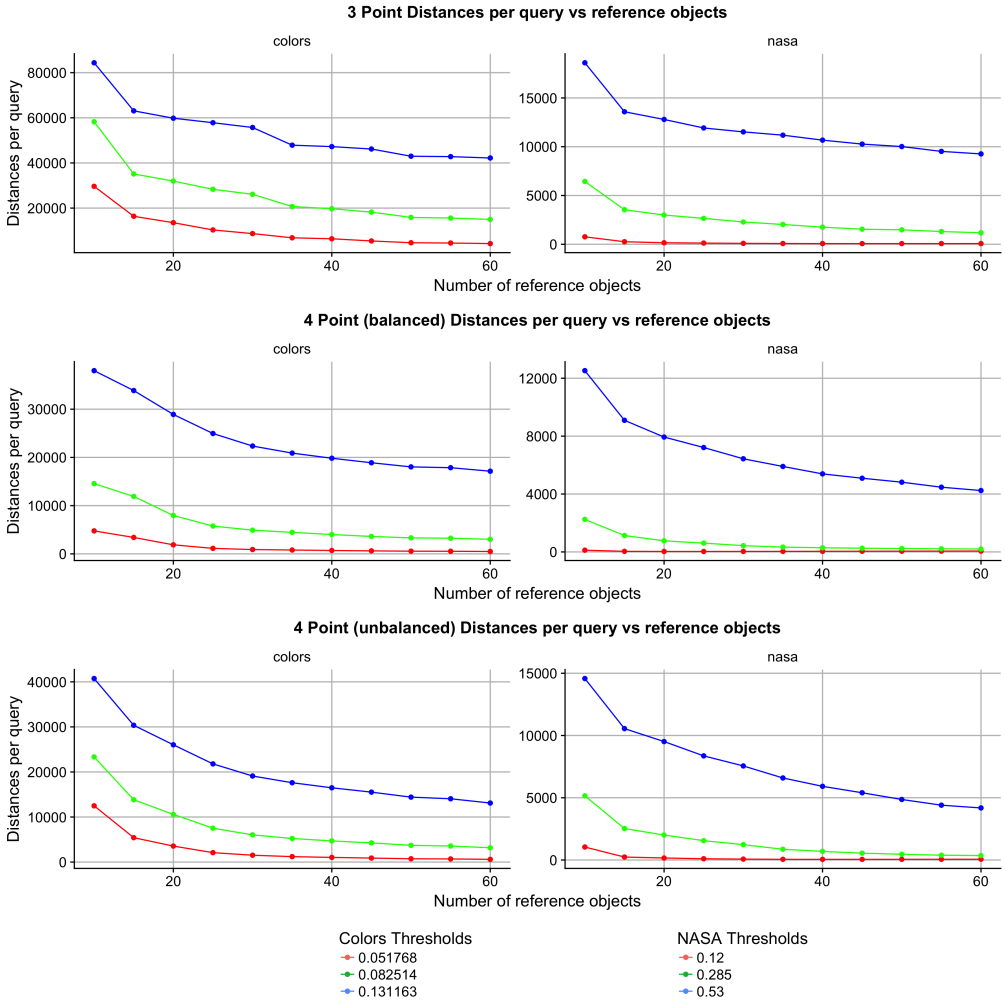

Fig. 2. Queries against SISAP colors and NASA datasets

that, using 60 reference objects, the supermetric property, and balancing the data structures, we have an almost perfect characterisation of the *nasa* data set, with less that one false positive result per query.

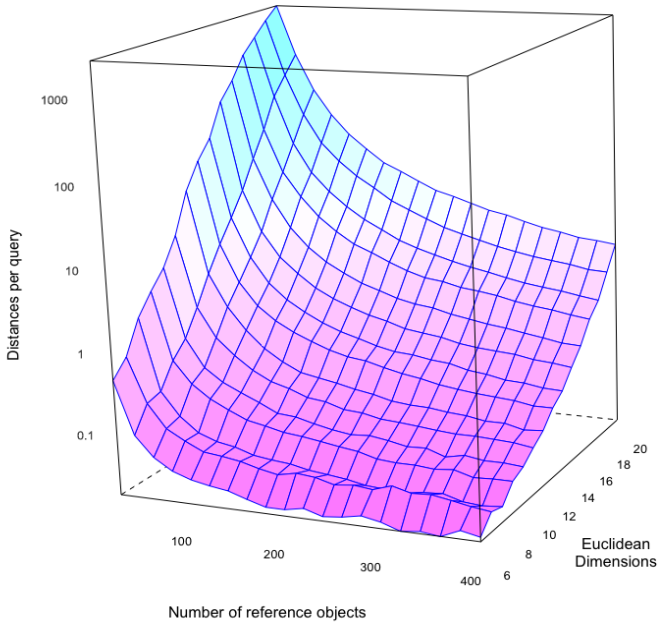

Fig. 3. Residual distances per query vs Euclidean dimensions vs number of reference objects: 1000 random queries against 10000 objects; all queries callibrated to return one-millionth of the space

Experiment 2 The second experiment is designed to show how the algorithm can combat increasing dimensionality by using increased numbers of reference objects. In this experiment we use evenly-distributed generated Euclidean spaces of increasing dimensions, ranging from 6 to 20. The intention is to increase the intrinsic dimensionality [4] of the data being manipulated, up to 27.5 in the last case. The query threshold at each dimension represents the radius of a hypersphere that contains one-millionth of the space in which the data is generated[4]. In all cases the number of points in the space is ten thousand and

[4] For dimension n, radius $r_n = \frac{\Gamma(\frac{n}{2}+1)}{\pi^{\frac{n}{2}}}$, where Γ is Euler's gamma function

one thousand random queries are made against this data set. The number of reference objects ranges from 20 to 400. Results are shown in Figure 3.

The surface shown in the image reports the number of residual distance calculations required per query plotted on a logarithmic scale, as both dimensions and number of reference points increase. Notice that the mid-point of the Z-axis (the number of residual calculations) represents accessing only 0.1% of the data, thus the figure demonstrates highly tractable search all the way to 20 dimensions. We do not believe that this has been demonstrated previously for any exact search mechanism.

Furthermore it can be seen that much of the surface has has a z-value of substantially less than 1; in these cases, the space is effectively being perfectly characterised by the regions derived from the reference points. For example this occurs with less than 200 reference points for 10 dimensions, a space normally considered to be at the bounds of tractability for most exact search mechanisms.

5 Related Work

5.1 Search using Fixed Reference Objects

There are a number of well-known mechanisms in which the distances between a query and a fixed set of reference objects are used to guide a first phase of search. All such mechanisms may either be regarded as approximate, or subsequently checked against the original data set for accuracy.

LAESA [14] has typically been used for metric filtering, rather than approximate search. For each element of the data, distances to a fixed set of reference points are recorded in a table. At query time, the distances between the query and each reference point are calculated; the table can then be scanned row at a time, and each distance compared; if, for any reference object p_i and data object s_j the absolute difference $|d(q, p_i) - d(s_j, p_i)| > t$, then it is impossible for s_j to be within distance t of the query, and the distance calculation can be avoided. LAESA can be used as an efficient pre-filter for exact search when memory size is limited. The same data can be re-cast into a metric space using the Chebyshev metric but this does not typically result in any significant performance increase.

The best known mechanisms which use a fixed partition of the original data for approximate search are based on *permutation orderings* [1, 10, 15]. There is a significant variety of techniques, but the essence of the approach is to characterise each element of the data in terms of its distances to a set of pre-selected reference objects, and then to compare elements using various aspects of similarity in this ordering. The approach is similar in that this effectively creates a large number of regions within the universal space, one for each ordering. However in all cases that we know of, an approximate search mechanism is created based on some cost function over the resulting orderings; there does not seem to be any clear mechanism for using the regions thus defined as an exact search mechanism. Especially in higher dimensional spaces, two very similar objects in the original space may lie on opposing sides of a number of the important boundaries, which then appear as distant objects and fail to appear in the search results.

The approach with *Sketches* [13] is more like our own, and the characterisation of the data as a set of bitmaps is almost identical. The approach differs however as it proceeds to use each bitmap as an object proxy with other search techniques, the underlying notion being the probabilistic mechanism that two close objects in the original space will result in two very similar bitmaps; thus the Hamming distance over these bitmaps should give a good proxy to the distance in the original space. The technique suffers from the same probabilistic issues as permutation orderings, in that for any two very similar objects there is a finite probability that they will appear very different in the proxy space.

Other probabilistic mechanisms have also been proposed, for example [2, 17, 18]. These are all quite similar in outline but with different ways of defining regional boundaries and treating the proxy space. Our work differs significantly in that we describe a mechanism which is guaranteed to give all correct results from the original space, and in the way that the increased cost of increasing dimensionality or query threshold can be controlled.

5.2 Metrics and Supermetrics

Much work on finite isometric embeddings was conducted in the 20^{th} century, by e.g. Blumenthal [3], Wilson [19] and Menger [12]. Blumenthal uses the phrase *four-point property* to mean a space that is 4-embeddable in 3-dimensional Euclidean space: that is, that for any four objects in the original space it is possible to construct a distance-preserving tetrahedron.

This simple property has been shown to have profound consequences in metric search. Connor and Vadicamo have applied these results in theoretical mathematics to this more practical domain [6–8] and the term *supermetric* is now used to refer to metrics with the property. For this context, the important result is that the four-point property applies to many commonly-used distance metrics, including Euclidean, Cosine[5], Jensen-Shannon, Triangular and Quadratic Form distances, all of which can be safely used in conjunction with the mechanisms described here.

Use of the supermetric property gives better exclusion conditions, as detailed in Section 3.2. The concepts given in this paper are equally applicable to both metric and supermetric spaces; as would be expected from a space with tighter geometric properties, however the results we show using the four-point property are significantly better than results relying only on the three-point property.

6 Conclusions and further work

This paper presents a novel exact search mechanism which gives excellent performance compared with other exact methods. It is especially well suited to higher dimensional data, in cases where indexing methods become intractable. The first phase of the algorithm is evaluated independently of the data set, and its cost is

[5] for the correct formulation, see [6].

approximately $\mathcal{O}(m^2)$ where m is the number of reference objects selected; however this cost has a low constant factor and the cost of the phase is dominated by the m distance calculations.

The actual selection computation requires as few as $\log_2 n$ bits per datum, in which context the space required, although proportional to $\mathcal{O}(n \log n)$, is unlikely to cause a practical problem. Even when n is 10^9 the computation still fits comfortably within the memory of a modern laptop computer.

The algorithm is inherently decomposable and parallelisable. Every phase of the computation is parallelisable: the distances between queries and reference objects, the assessment of queries against regions, the bitwise operations over the data representation, and finally the filtering of the candidate results. Amdhal's law applies to the end to end pipeline but we believe that great speedups are possible by exploiting parallel hardware resources due to the nature of the algorithm. We have begun experimenting with GPUs to perform the bit operations but it is premature to report any results at this time.

The algorithm exhibits many opportunities for tuning which are yet to be explored. We have already demonstrated the effect of changing the number of reference objects and how this may help in combating increasing dimensionality. However it may be possible to employ techniques to choose pivots which could enable queries to perform a number of distance calculations that are closer to the theoretical minimum. We have already experimented with choosing pivots to optimise sheet exclusions but reporting on this work here would also be premature. The size, number and uniformity of the radii used for ball exclusions is also worthy of exploration.

7 Acknowledgements

This work was supported by ESRC grant ES/L007487/1 "Administrative Data Research Centre—Scotland". We would like to thank Tom Dalton for his help with preparation of the data and creating R scripts for rendering results, and Peter Christen along with the anonymous reviewers for helpful comments on earlier drafts.

References

1. Giuseppe Amato, Claudio Gennaro, and Pasquale Savino. Mi-file: using inverted files for scalable approximate similarity search. *Multimedia Tools and Applications*, 71(3):1333–1362, Aug 2014.
2. José María Andrade, César A. Astudillo, and Rodrigo Paredes. Metric space searching based on random bisectors and binary fingerprints. In Agma Juci Machado Traina, Caetano Traina, and Robson Leonardo Ferreira Cordeiro, editors, *Similarity Search and Applications*, pages 50–57, Cham, 2014. Springer International Publishing.
3. L. M. Blumenthal. A note on the four-point property. *Bull. Amer. Math. Soc.*, 39(6):423–426, 06 1933.

4. Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, September 2001.
5. Richard Connor. Reference point hyperplane trees. In Laurent Amsaleg, Michael E. Houle, and Erich Schubert, editors, *Similarity Search and Applications*, pages 65–78, Cham, 2016. Springer International Publishing.
6. Richard Connor, Franco Alberto Cardillo, Lucia Vadicamo, and Fausto Rabitti. Hilbert Exclusion: Improved metric search through finite isometric embeddings. *ACM Transactions on Information Systems*, 35(3):17:1–17:27, December 2016.
7. Richard Connor, Lucia Vadicamo, Franco Alberto Cardillo, and Fausto Rabitti. *Supermetric Search with the Four-Point Property*, pages 51–64. Springer International Publishing, Cham, 2016.
8. Richard Connor, Lucia Vadicamo, Franco Alberto Cardillo, and Fausto Rabitti. Supermetric search. *Information Systems*, 2018.
9. Karina Figueroa, Gonzalo Navarro, and Edgar Chávez. Metric spaces library. *Online http://www. sisap. org*, 2007.
10. E. Chavez Gonzalez, K. Figueroa, and G. Navarro. Effective proximity retrieval by ordering permutations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(9):1647–1658, Sept 2008.
11. Jakub Lokoč and Tomáš Skopal. On applications of parameterized hyperplane partitioning. In *Proceedings of the Third International Conference on SImilarity Search and APplications*, SISAP '10, pages 131–132, New York, NY, USA, 2010. ACM.
12. Karl Menger. New foundation of euclidean geometry. *American Journal of Mathematics*, 53(4):721–745, 1931.
13. Vladimir Mic, David Novak, and Pavel Zezula. Improving sketches for similarity search. In *Proceedings of MEMICS 2015*, pages 45–57, 2015.
14. María Luisa Micó, José Oncina, and Enrique Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. *Pattern Recogn. Lett.*, 15(1):9–17, January 1994.
15. Hisham Mohamed and Stéphane Marchand-Maillet. Quantized ranking for permutation-based indexing. *Information Systems*, 52:163 – 175, 2015. Special Issue on Selected Papers from SISAP 2013.
16. Laura C. Rivero, Jorge Horacio Doorn, and Viviana E. Ferraggine, editors. *Encyclopedia of Database Technologies and Applications*. Idea Group, 2005.
17. Eliezer Silva, Thiago Teixeira, George Teodoro, and Eduardo Valle. Large-scale distributed locality-sensitive hashing for general metric data. In Agma Juci Machado Traina, Caetano Traina, and Robson Leonardo Ferreira Cordeiro, editors, *Similarity Search and Applications*, pages 82–93, Cham, 2014. Springer International Publishing.
18. Eric Sadit Tellez and Edgar Chavez. On locality sensitive hashing in metric spaces. In *Proceedings of the Third International Conference on SImilarity Search and APplications*, SISAP '10, pages 67–74, New York, NY, USA, 2010. ACM.
19. Wallace A Wilson. A relation between metric and euclidean spaces. *American Journal of Mathematics*, 54(3):505–517, 1932.
20. Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. Similarity search - the metric space approach. In *Advances in Database Systems*, 2006.