t-Spanners as a Data Structure for Metric Space Searching ⋆

Gonzalo Navarro[1], Rodrigo Paredes[1], and Edgar Chávez[2]

[1] Center for Web Research, Dept. of Computer Science, University of Chile. Blanco Encalada 2120, Santiago, Chile. {gnavarro,raparede}@dcc.uchile.cl

[2] Escuela de Ciencias Físico-Matemáticas, Univ. Michoacana, Morelia, Mich. México. elchavez@zeus.ccu.umich.mx.

Abstract. A *t-spanner*, a subgraph that approximates graph distances within a precision factor t, is a well known concept in graph theory. In this paper we use it in a novel way, namely as a data structure for searching metric spaces. The key idea is to consider the t-spanner as an approximation of the complete graph of distances among the objects, and use it as a compact device to simulate the large matrix of distances required by successful search algorithms like AESA [Vidal 1986]. The t-spanner provides a time-space tradeoff where full AESA is just one extreme. We show that the resulting algorithm is competitive against current approaches, e.g., 1.5 times the time cost of AESA using only 3.21% of its space requirement, in a metric space of strings; and 1.09 times the time cost of AESA using only 3.83 % of its space requirement, in a metric space of documents. We also show that t-spanners provide better space-time tradeoffs than classical alternatives such as pivot-based indexes. Furthermore, we show that the concept of t-spanners has potential for large improvements.

1 Introduction

The concept of "approximate" searching has applications in a vast number of fields. Some examples are non-traditional databases (where the concept of exact search is of no use and we search for similar objects, e.g. databases storing images, fingerprints or audio clips); machine learning and classification (where a new element must be classified according to its closest existing element); image quantization and compression (where only some vectors can be represented and those that cannot must be coded as their closest representable point); text retrieval (where we look for words in a text database allowing a small number of errors, or we look for documents which are similar to a given query or document); computational biology (where we want to find a DNA or protein sequence in a database allowing some errors due to typical variations); function prediction

⋆ This work has been supported in part by the Millenium Nucleus Center for Web Research, Grant P01-029-F, Mideplan, Chile (1st and 2nd authors), CYTED VII.19 RIBIDI Project (all authors), and AT&T LA Chile (2nd author).

(where we want to search the most similar behavior of a function in the past so as to predict its probable future behavior); etc.

All those applications have some common characteristics. There is a universe $\mathbb{X}$ of *objects*, and a nonnegative *distance function* $d : \mathbb{X} \times \mathbb{X} \longrightarrow \mathbb{R}^+$ defined among them. This distance satisfies the three axioms that make the set a *metric space*

$$\begin{aligned} & d(x,y) = 0 \quad \Leftrightarrow \quad x = y \\ & d(x,y) = d(y,x) \\ & d(x,z) \leq d(x,y) + d(y,z) \end{aligned}$$

where the last one is called the "triangle inequality" and is valid for many reasonable similarity functions. The smaller the distance between two objects, the more "similar" they are. This distance is considered expensive to compute (think, for instance, in comparing two fingerprints). We have a finite *database* $\mathbb{U} \subseteq \mathbb{X}$, which is a subset of the universe of objects and can be preprocessed (to build an index, for instance). Later, given a new object from the universe (a *query* q), we must retrieve all similar elements found in the database. There are two typical queries of this kind:

(a) Retrieve all elements which are within distance r to q.
This is, $\{x \in \mathbb{U} \; / \; d(x,q) \leq r\}$.
(b) Retrieve the k closest elements to q in $\mathbb{U}$.
This is, $A \subseteq \mathbb{U}$ such that $|A| = k$ and $\forall x \in A, y \in \mathbb{U} - A, d(x,q) \leq d(y,q)$.

Given a database of $|\mathbb{U}| = n$ objects, all those queries can be trivially answered by performing n distance evaluations. The goal is to structure the database such that we perform less distance evaluations. Since the distance is usually expensive to compute, we take the number of distance evaluations as the measure of the search complexity. This is the approach we take in this paper.

A particular case of this problem arises when the space is $\mathbb{R}^k$. There are effective methods for this case, such as kd-trees, R-trees, X-trees, etc. [6]. However, for roughly 20 dimensions or more those structures cease to work well. We focus in this paper in general metric spaces, although the solutions are well suited also for k-dimensional spaces. It is interesting to notice that the concept of "dimensionality" can be translated to metric spaces as well: the typical feature in high dimensional spaces is that the probability distribution of distances among elements has a very concentrated histogram (with larger mean as the dimension grows), difficulting the work of any similarity search algorithm [4]. We say that a general metric space is high dimensional when its histogram of distances is concentrated.

There are a number of methods to preprocess the set in order to reduce the number of distance evaluations. All them work by discarding elements with the triangle inequality. See [4] for a recent survey.

By far, the most successful technique for searching metric spaces ever proposed is AESA [10]. Its main problem is that it requires precomputing and storing a matrix with all the $O(n^2)$ distances among the objects of $\mathbb{U}$. This high space requirement has prevented it from being seriously considered except in very small domains.

On the other hand, the concept of a t-spanner is well known in graph theory [9]. Let G be a connected graph $G(V, E)$ with a nonnegative cost function $d(e)$ assigned to its edges $e \in E$, and $d_G(u, v)$ be the cost of the cheapest path between $u, v \in V$. Then, a t-spanner of G is a subgraph $G'(V, E')$ where $E' \subseteq E$ and $\forall u, v \in V,\ d_{G'}(u, v) \leq t \cdot d_G(u, v)$. (It should be clear that $d_G(u, v) \leq d_{G'}(u, v)$ also holds because G' is a subgraph of G.) Several algorithms to build t-spanners are known [5, 7], and we have proposed some specific construction algorithms for our present metric space application [8] (complete G, metric costs, and $t < 2$). The naive construction algorithm is $O(n^4)$ time. On euclidean spaces, this drops to $O(n \log n)$. Our construction complexity [8] for general metric spaces is around $O(n^{2.2})$ in practice.

Our main idea is to combine both concepts so as to use the t-spanner as a controlled approximation to the full AESA distance matrix, so as to obtain a competitive space-time tradeoff. We show experimentally that t-spanners provide competitive performance as simple replacements of AESA. At the end, we argue that they give us tools for several improvements that are under study.

We apply the idea to approximate dictionary searching under the edit distance, which is a common problem in text databases. As an example, if we search permitting one error, a 1.4-spanner (needing only a 3.2% of the memory of AESA) needs only 26% distance evaluations over AESA. If we permit two errors the overhead in distance computations is 15%, and 46% for three errors. A classical pivot-based technique using the same amount of memory needs much more distance evaluations.

We also apply the idea to approximate document retrieval form textual database using the cosine distance. As an example, if we use a 2.0-spanner we need only a 3.8% of the memory of AESA for the index. With the 2.0-spanner index, we need only 9% distance evaluations over AESA in order to retrieve 1 document on average, and 8% distance evaluations over AESA to retrieve 10 documents on average.

2 Previous Work

Different data structures have been proposed to filter out elements at search time based on the triangle inequality [4]. In this paper we will focus on a particular class of algorithms called "pivot-based". These algorithms select a set of pivots $\{p_1 \ldots p_k\} \subseteq \mathbb{U}$ and store a table of kn distances $d(p_i, u),\ i \in \{1 \ldots k\},\ u \in \mathbb{U}$. To solve a range query (q, r), we measure $d(q, p_1)$ and use the fact that, because of the triangle inequality,

$$d(q, u) \quad \geq \quad |d(q, p) - d(u, p)| \ ,$$

so we can discard every $u \in \mathbb{U}$ such that $|d(q, p_1) - d(u, p_1)| > r$, as this implies $d(q, u) > r$. Once we are done with p_1 we try to discard elements from the remaining set using p_2 and so on until we use all the k pivots. The elements u that still cannot be discarded at the end are directly compared against q. Fig. 1 (left) shows the concept graphically.

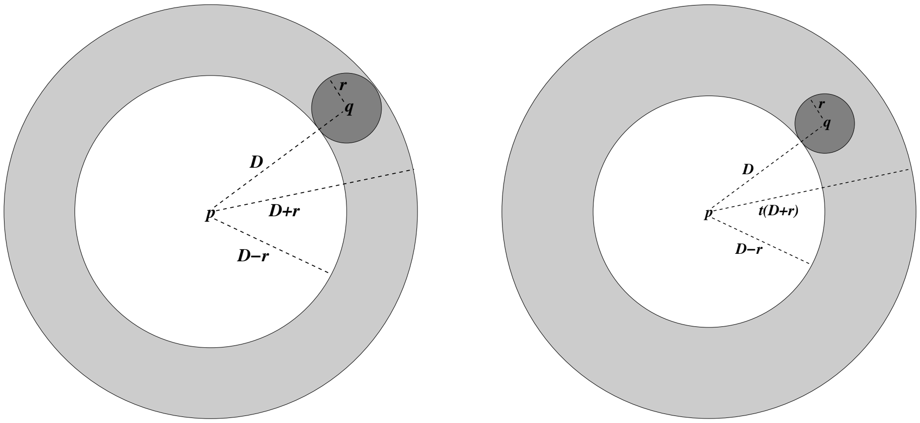

Fig. 1. On the left, the ring of elements not discarded by pivot p. On the right, the relaxed ring used when using a t-spanner. We denote as D the (real or approximated) distance between p and q.

In AESA [10] this idea is taken to the extreme $k = n$, that is, every element is a potential pivot and hence we need a matrix with all the $n(n-1)/2$ distances precomputed. Since we are free to choose any pivot, the "next" pivot is chosen from the remaining set of elements, which improves locality and the search cost. Additionally, as it is well known that pivots closer to the query are much more effective, candidates to pivots u are sorted according to the sum of their lower bound distances to q up to now. That is, if we have used pivots $\{p_1 \ldots p_i\}$ and want to choose pivot p_{i+1}, we choose the element u minimizing

$$SumLB(u) \quad = \quad \sum_{j=1}^{i} |d(p_j, q) - d(p_j, u)| \qquad (1)$$

AESA is experimentally shown to have almost constant cost as a function of n. The problem is that storing $O(n^2)$ distances is unrealistic for most applications. This has restricted an excellent algorithm to the few applications where n is small. Our goal in this paper is to overcome this weakness.

3 Our Proposal

Our main idea is to use t-spanners as low memory replacement of the full distance matrix, allowing a controlled approximation to the true distances. Let us assume we have a complete graph $G(\mathbb{U}, \mathbb{U} \times \mathbb{U})$, where $d(u,v) = d_G(u,v)$ is the metric space distance between elements u and v. A t-spanner $G'(\mathbb{U}, E)$ of G would permit us estimate the distance between every pair of objects within a factor t, without the need to store $O(n^2)$ distances but only $|E|$ edges. We note that, for every $u, v \in \mathbb{U}$,

$$d(u,v) \quad \leq \quad d_{G'}(u,v) \quad \leq \quad t \cdot d(u,v) \tag{2}$$

which permits us adapting AESA to this approximated distance.

Let us return to the condition to discard an element u with a pivot p. The condition to be outside the ring can be rewritten as

$$d(p,u) \ < \ d(p,q) - r \quad \text{ or } \quad d(p,u) \ > \ d(p,q) + r \ . \tag{3}$$

If we only know $d_{G'}(p,u)$, we can use Eqs. (2) and (3) to obtain a new condition that implies Eq. (3) and hence guarantees that $d(q,u) > r$:

$$d_{G'}(p,u) \ < \ d(p,q) - r \quad \text{ or } \quad d_{G'}(p,u) \ > \ t \cdot (d(p,q) + r) \ . \tag{4}$$

Therefore, a pivot p can discard every element outside the ring $d_{G'}(p,u) \in [d(p,q) - r \ , \ t \cdot (d(p,q) + r)]$. Fig. 1 (right) illustrates.

What we have obtained is a relaxed version of AESA, which requires less memory ($O(|E|)$ instead of $O(n^2)$) and, in exchange, discards less element per pivot. As t tends to 1, our approximation becomes better but we need more and more edges. Hence we have a space-time tradeoff where the full AESA is just one extreme.

Since we have only an approximation to the distance, we cannot directly use Eq. (1). To compensate the effect of the precision factor t, we define α_t, and rewrite Eq. (1) as follows:

$$sumLB'(u) = \sum_{i=0}^{k-1} \left| d(p_i, q) - d_{G'}(p_i, u) \cdot \alpha_t \right|, \qquad \alpha_t = \frac{2/t + 1}{3} \tag{5}$$

Our search algorithm is as follows. We start with a set of candidate nodes C, which is initially $\mathbb{U}$. Then, we choose a node $p \in C$ minimizing $SumLB'$ (Eq. (5)) and remove it from C. We measure $D = d(p,q)$ and report p if $D \leq r$. Now, we run Dijkstra's shortest path algorithm in the t-spanner starting at p, until the last node v whose distance to p gets computed satisfies $d_{G'}(v,p) > t(D+r)$. (Since Dijkstra's algorithm gives the distances to p in increasing order, we know that all the remaining nodes will be farther away.) By Eq. (4), we keep from C only the nodes u such that $D - r \leq d_{G'}(p,u) \leq t(D+r)$. We repeat these steps until $C = \emptyset$. Fig. 2 depicts the algorithm.

The analysis is similar to that of AESA. Let n_i be the number of pivots we have to consider before we can remove node u_i from C (it may be necessary to finally compare q against u_i directly). Then the number of distance computations made by AESA is $\max_{i=1...n} n_i$ and its extra CPU cost is $\sum_{i=1...n} n_i$ (which is between $O(n)$ and $O(n^2)$). In practice it is shown that the number of distance evaluations is close to $O(1)$ and the extra CPU time to $O(n)$ [10].

```
Search (Query $q$, Radius $r$, $t$-Spanner $G'$)

$C \leftarrow \mathbb{U}$
$\alpha_t \leftarrow (2/t+1)/3$
for $p \in C$ do $SumLB(p) \leftarrow 0$
while $C \neq \emptyset$ do
    $p \leftarrow \text{argmin}_{c \in C} SumLB'(c)$
    $C \leftarrow C - \{p\}$
    $D \leftarrow d(q,p)$
    if $D \leq r$ then Report $p$
    $d_{G'} \leftarrow$ Dijkstra$(G', p, t(D+r))$
    for $u \in C$ do
        if $d_{G'}(p,u) \notin [D-r, t(D+r)]$ then
            $C \leftarrow C - \{u\}$
        else $SumLB'(u) \leftarrow SumLB'(u) + |D - d_{G'}(p,v) \cdot \alpha_t|$
```

Fig. 2. Search algorithm. `Dijkstra`(G', p, x) computes the distances from p in G' for all nodes up to distance x, and marks the remaining ones as "farther away".

In our case, however, we have the additional cost of running Dijkstra. Albeit we are interested only in the nodes belonging to C, we need to compute the distances to all the others to obtain the ones we need. We remark that this algorithm works only up to the point where the next closest element it extracts is far enough. Overall, this can be as bad as $O(n|E|\log n)$ or $O(n^3)$ depending on the version of Dijkstra we use. On the other hand, if we assume that we work to obtain little more than the distances we preserve in C, the overall cost is only that of AESA multiplied by $O(\log n)$. In any case, we remind that we are focusing on applications where the cost to compute d dominates even heavy extra CPU costs.

4 Experimental Results

We have tested our *t-spanner* on two real-world metric spaces. The first is a string metric space using the edit distance (a discrete function that measures the minimum number of character insertions, deletions and replacements needed to make them equal). The strings form an English dictionary, where we index a subset of n = 23,023 words. The second is a space of 1,215 documents under the Cosine distance, which is used to retrieve documents with higher rank with respect to a query (i.e., closer to the query point under Cosine distance) [1]. Both spaces are of interest to Information Retrieval applications.

As our index data structure we use t-spanners with precision factors $t \in [1.4, 2.0]$, and compare them against AESA. Since t-spanners offer a time-space tradeoff and AESA does not, we consider also pivot-based indexes with varying number of pivots. For every t value, we measure the size of the resulting t-spanner and build a pivot-based index using the same amount of memory (pivots are chosen at random). This way we compare t-spanners against the classical space-time alternative tradeoff. Note that AESA needs more than 250 millions of cells (1 gigabytes of memory) even for the relatively small example of strings.

Since in some cases the pivots were too many compared to the average number of candidates to eliminate, we decided to stop using the pivots when the remaining set of candidates was smaller than the remaining set of pivots to use. This way we never pay more for having more pivots available than necessary. Also, it turns out that even the smallest number of pivots shown is beyond the optimal sometimes. In these cases we show also the result with less pivots until we reach the optimum.

4.1 Strings under Edit Distance

In the space of strings, we select 100 queries randomly from dictionary words not included in the index, and search with radii $r = 1, 2, 3$, which return 0.0041%, 0.036% and 0.29% of the database, respectively. Tables 1, 2 and 3 show the size of the index structures tested, as well as the distance evaluations required for searching.

t	$\|E'\|$	$r = 1$	$r = 2$	$r = 3$
1.4	8,507,720	27.66	98.54	723.20
1.5	3,740,705	34.55	135.59	944.13
1.6	2,658,556	39.00	167.36	1188.44
1.7	1,861,260	42.53	185.24	1205.32
1.8	1,249,313	56.15	267.22	1581.68
1.9	901,577	62.79	293.80	1763.81
2.0	626,266	96.25	471.35	2306.07

Table 1. t-Spanner index size and distance evaluations at query time. Every edge needs two machine words of storage.

$n(n-1)/2$	$r = 1$	$r = 2$	$r = 3$
265,017,753	21.83	85.05	495.05

Table 2. AESA structure size and distance evaluations at query time. Every cell entry needs one machine word of storage.

As seen in Tables 1 and 2, our indexes are competitive against AESA and use only a fraction of its space (e.g., only 3.21% for $t = 1.4$). With respect to

t	equivalent # of pivots	$r = 1$	$r = 2$	$r = 3$
1.4	739	539.48	642.65	1251.15
1.5	325	248.13	318.57	1685.52
1.6	230	181.80	268.40	2129.34
1.7	161	132.13	256.17	2845.85
1.8	108	86.75	321.08	3956.21
1.9	78	64.26	465.84	5047.14
2.0	54	49.29	748.81	6082.60

Table 3. Pivot table structure and distance evaluations at query time. Every table cell needs one machine word. We have computed the amount of pivots that corresponds to the t-spanner size for every t.

pivots (Tables 1 and 3), in almost every case the corresponding t-spanners use the space better.

Figures 3, 4 and 5 present the results graphically. We have chosen to draw a line to represent AESA, although, since it permits no space-time tradeoffs, a point would be the correct representation. The position of this point in the x axis would be 132.5, far away from the right end of the plot.

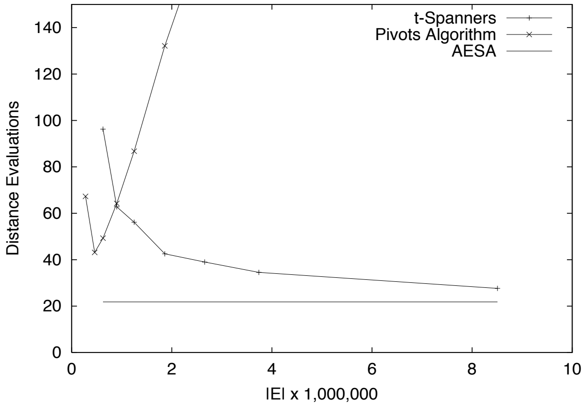

Fig. 3. Distance evaluations, search radius $r = 1$.

4.2 Documents under Cosine Distance

In the space of documents, we select 50 queries randomly from the document database not included in the index, and search with radii chosen to retrieve 1 or 10 documents per query ($r = 0.1325, 0.167$ respectively). Tables 4, 5 and 6

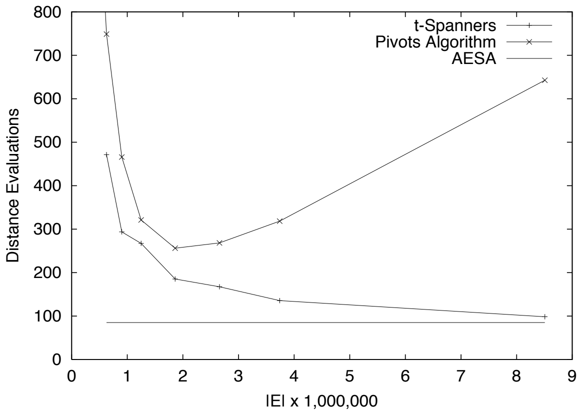

Fig. 4. Distance evaluations, search radius $r = 2$.

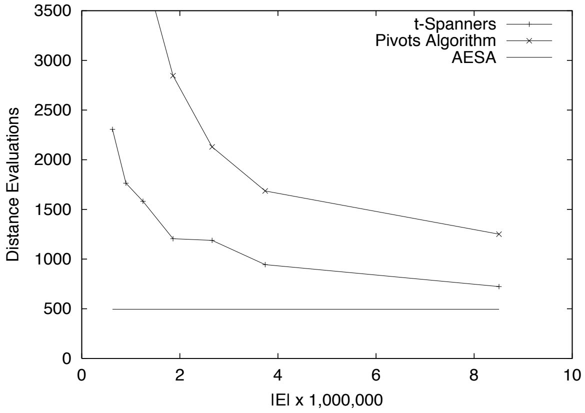

Fig. 5. Distance evaluations, search radius $r = 3$.

show the size of the index structures tested, as well as the distance evaluations required for searching.

As seen in Tables 4 and 5, our indexes are very competitive against AESA and use only a fraction of its space (e.g., only 3.84% for $t = 2.0$). With respect to pivots (Tables 4 and 6), in all cases the corresponding t-spanners use the space better.

Figures 6 and 7 present the results graphically. We have chosen to draw a line to represent AESA.

5 Conclusions

We have presented a new approach to metric space searching, which is based on using a t-spanner data structure as an approximate map of the space. This

t	$\lvert E'\rvert$	retrieving 1 document	retrieving 10 documents
1.4	266,590	191.60	210.84
1.5	190,145	193.04	212.78
1.6	125,358	195.14	212.30
1.7	109,387	194.96	215.30
1.8	87,618	197.20	216.38
1.9	43,336	201.76	218.98
2.0	28,239	205.60	223.02

Table 4. t-Spanner index size and distance evaluations at query time. Every edge needs two machine words of storage.

$n(n-1)/2$	retrieving 1 document	retrieving 10 documents
737,505	187.32	206.26

Table 5. AESA structure size and distance evaluations at query time. Every cell entry needs one machine word of storage.

permits us trading space for query time. We have shown experimentally that the alternative is competitive against existing solutions. In particular we have shown that t-spanners are specially competitive in applications of interest to Information Retrieval: strings under edit distance and documents under cosine distance. For example, in an approximate string matching scenario typical of text databases, we show that t-spanners provide better space-time tradeoffs compared to the classical pivot-based solutions. It also permits approximating AESA, which is an unbeaten index, within 50% of extra time using only about 3% of the space it requires. This becomes a feasible approximation to AESA, which in its

t	equivalent # of pivots	retrieving 1 document	retrieving 10 documents
1.4	438	256.54	288.54
1.5	312	273.80	281.28
1.6	206	281.98	307.46
1.7	180	275.42	319.20
1.8	144	279.48	307.04
1.9	71	251.30	269.70
2.0	46	232.98	252.20

Table 6. Pivot table structure and distance evaluations at query time. Every table cell needs one machine word. We have computed the amount of pivots that corresponds to the t-spanner size for every t.

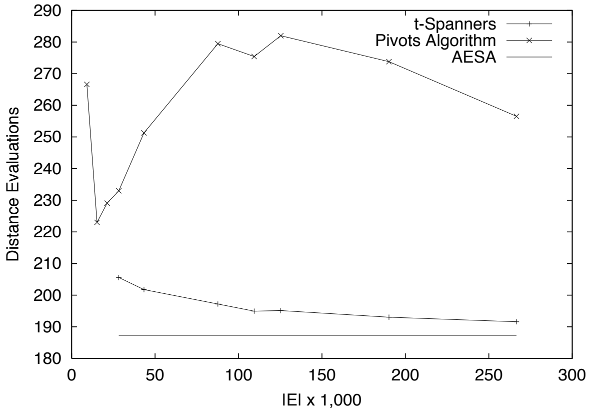

Fig. 6. Distance evaluations, search radius $r = 0.1325$, retrieving 1 document.

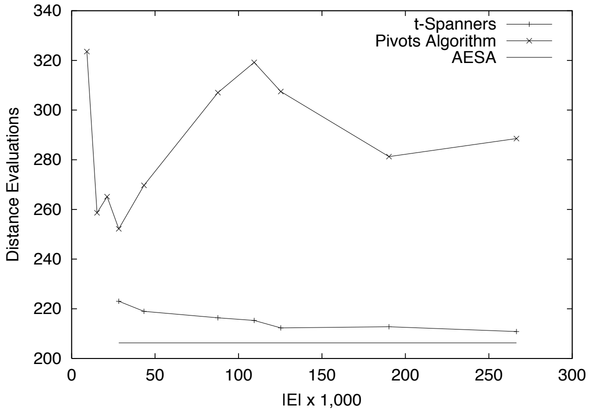

Fig. 7. Distance evaluations, search radius $r = 0.167$, retrieving 10 documents.

original form cannot be implemented in practice because of its quadratic memory requirements. Furthermore, for document retrieval in a textual database, we need a 9% extra time over AESA, using only 4% of its memory requirement.

On the other hand, t-spanners have a large potential for improvements we are pursuing. A first one is that we do not really need the same precision t for all the edges. Shorter edges are more important than longer edges, as Dijkstra tends to use shorter edges to build the shortest paths. Using a t that depends on the distance to estimate may give us better space-time tradeoffs.

Another idea is that we can build a t-spanner and use it as a t'-spanner, for $t' < t$. This may lose some relevant elements but improves the search time. The result is a probabilistic algorithm, which is a new successful trend in metric space searching [3, 2]. In particular, we have observed that in order to build a t-spanner, many distances are estimated better than t times the real one, so this idea seems promising. For example, a preliminary experiment in the string metric space shows that, with a 2.0-spanner and using $t' = 1.9$, we need only 53% of the distance computations to retrieve the 92% of the result.

Finally, another idea is to use the t-spanner as a navigational device. A pivot is much more effective if it is closer to the query, as the ball of candidate elements has much smaller volume. We can use the t-spanner edges to start at a random node and approach the query by neighbors.

References

1. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley, 1999.
2. B. Bustos and G. Navarro. Probabilistic proximity searching algorithms based on compact partitions. In *Proc. 9th International Symposium on String Processing and Information Retrieval (SPIRE 2002)*, LNCS. Springer, 2002. To appear.
3. E. Chávez and G. Navarro. A probabilistic spell for the curse of dimensionality. In *Proc. 3rd Workshop on Algorithm Engineering and Experiments (ALENEX'01)*, LNCS 2153, pages 147–160, 2001.
4. E. Chávez, G. Navarro, R. Baeza-Yates, and J.L. Marroquin. Proximity searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, September 2001.
5. E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM J. on Computing*, 28:210–236, 1998.
6. V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.
7. J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Improved greedy algorithms for constructing sparse geometric spanners. In *Proc. 7th Scandinavian Workshop on Algorithm Theory (SWAT 2000)*, LNCS v. 1851, pages 314–327, 2000.
8. G. Navarro and R. Paredes. Practical construction of metric t-spanners. Technical Report TR/DCC-2002-4, Dept. of Computer Science, Univ. of Chile, July 2002.
9. D. Peleg and A. Schaffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.
10. E. Vidal. An algorithm for finding nearest neighbors in (approximately) constant average time. *Patt. Recog. Lett.*, 4:145–157, 1986.