Information Processing Letters 40 (1991) 181-188
North-Holland

25 November 1991

Covering a set of points in multidimensional space [*]

Teofilo F. Gonzalez [**]

Department of Computer Science, Utrecht University, Utrecht, Netherlands

Communicated by M.J. Atallah
Received 6 December 1990
Revised 13 September 1991

Abstract

Gonzalez, T.F., Covering a set of points in multidimensional space, Information Processing Letters 40 (1991) 181-188.

Let $P=\{p_1, p_2,\ldots, p_n\}$ be a set of points in d-space. We study the problem of covering with the minimum number of fixed-size orthogonal hypersquares (CS_d for short) all points in P. We present a fast approximation algorithm that generates provably good solutions and an improved polynomial-time approximation scheme for this problem. A variation of the CS_d problem is the CR_d problem, covering by fixed-size orthogonal hyperrectangles, where the covering of the points is by hyperrectangles with dimensions D_1, $D_2,\ldots, D_d$ instead of hypersquares of size D. Another variation is the CD_d problem, where we cover the set of points with hyperdiscs of diameter D. Our algorithms can be easily adapted to these problems.

Keywords: Analysis of algorithms, d-space, covering by hypersquares, hyperdiscs and hyperrectangles, efficient algorithms, polynomial-time approximation scheme

1. Introduction

Let $P=\{p_1, p_2,\ldots, p_n\}$ be a set of points in the plane (E^2). The problem of covering with fixed-size orthogonal squares, CS_2, consists of finding a minimum cardinality set of D by D squares covering all points in P, i.e., each point in P must be inside or on the boundary of at least one of the squares in the cover. A generalization to d dimensions of the CS_2 problem is called the CS_d problem. In this case the points are in d-dimensional space and the covering is by orthogonal hypersquares of size D. A variation of the CS_d problem is the CR_d problem, covering by fixed-size hyperrectangles, where the covering of the points is by orthogonal hyperrectangles with dimension D_1, D_2, $\ldots,D_d$ instead of hypersquares of size D. Another variation is the CD_d problem, where we cover the points with hyperdiscs of diameter D. A related problem, packing of squares, is discussed in Section 4. Hereafter, when we refer to a square (rectangle) or hypersquare (hyperrectangle) we assume it is orthogonal to the coordinate axes.

These problems have many interesting applications [1,4]. The most popular application is the problem of locating the least number of emergency facilities such that all potential users are located within a reasonable small distance from one of the facilities. This corresponds to the CD_2 problem. The CD_d, CR_d and CS_d problems for $d \geqslant 2$ are known to be NP-hard [1,6,7]. Johnson [5] discusses several variations of these problems.

[*] A preliminary version of this paper appeared in the Proceedings of the Twenty-eight Annual Allerton Conference on Communications, Control, and Computing, October 1990.

[**] On Sabbatical leave from the University of California, Santa Barbara.

Heuristics to solve the CS_2 problem have been presented in [8] and [9]. A polynomial time approximation scheme is given in [4], i.e., for every constant $\epsilon > 0$ the algorithm takes $O(n^{O(1/\epsilon)})$ time and generates solutions such that $F_{apx}/F_{opt} \leqslant 1 + \epsilon$, where F_{apx} is the number of hypersquares in the solution generated by the algorithm and F_{opt} is the number of hypersquares in an optimal solution.

For any integer $l \geqslant 1$, the algorithm for the CS_d problem given in [4] has time-complexity bound $O(l^d n^{dl^d+1})$ and the approximation bound is $F_{apx}/F_{opt} \leqslant (1 + 1/l)^d$. The above bound disagrees with the one in [4] because there is a typo in that paper. To achieve an approximation bound of 2^d it takes $O(n^{d+1})$ time and to achieve an approximation bound of 2.25, for $d = 2$, it takes $O(n^9)$ time. In general, the only approximation bound that can be guaranteed within reasonable time constraints is 2^d when d is small, since to guarantee a solution within 2^{d-1} the worst-case time complexity becomes $O(n^9)$ when $d = 2$.

For the CD_d problem and any integer $l \geqslant 1$ the algorithm in [4] has worst-case time complexity

$$O\left(l^d (l\sqrt{d})^d (2n)^{2\lceil l\sqrt{d}\rceil^d + 1}\right)$$

and the approximation bound is $F_{apx}/F_{opt} \leqslant (1 + 1/l)^d$. For $d = 2$, to guarantee solutions within 4 (2.25) of optimal, the worst-case time complexity bound is $O(n^9)$ $(O(n^{19}))$. These huge time complexity bounds make the algorithms unusable even when n is small.

In this paper we present a fast approximation algorithm for the CS_d problem with worst-case approximation bound of 2^{d-1}. The algorithm takes

$$O(dn + n \log s)$$

time, where s is the number of hypersquares in an optimal solution. Since our algorithm examines each point $c \log s$ times, where c is a small constant, and each time it performs only simple operations, we know that the constant associated with the time-complexity bound is small. The amount of space required by the algorithm is a small constant times the total input. This is why we say that our algorithm is usable in practical situations. We also present an efficient algorithm for the l_d-slab problem (which we define in Section 3). This algorithm can be easily combined with the polynomial-time approximation scheme given in [4] to generate a solution to the CS_d problem with approximation bound $(1 + 1/l)^{d-1}$ in

$$O\left(l^{d-1} d (2l^{d-1} - 1) n^{d(2l^{d-1}-1)+1}\right)$$

time. Our new algorithms can also be adapted to the CD_d and CR_d problems as well as to the packing problem discussed in [4]. The approximation and time complexity bounds for the CD_d problem are not identical to the ones for the CS_d problem; however, they are improvements over the ones for previous algorithms. We discuss these extensions in Section 4. The approximation algorithm is presented in Section 2 and the improved approximation scheme is given in Section 3.

Before presenting our algorithms, we define some terms. Point p is said to be located at $(x_1(p), x_2(p), \ldots, x_d(p))$, and we assume without loss of generality that $x_j(p) \geqslant 0$, for all p and j. We define the function $i_j(p)$ as $\lfloor x_j(p)/D \rfloor$, for all p and j.

2. Approximation algorithm for the CS_d problem

In this section we present our approximation algorithm for the CS_d problem. The worst-case approximation bound is 2^{d-1} and it takes

$$O(dn + n \log s)$$

time, where s is the number of hypersquares in an optimal solution.

Before we explain our approximation algorithm, let us solve a restricted version of our problem which we call the *slab problem*. In this problem all points are located inside a rectangle (whose sides are orthogonal to the axes) with height D. An optimal cover for the slab problem can be obtained as follows. Consider all points in increasing order with respect to their $x_1(p)$ value (the 1-axis corresponds to the width). The leftmost point (point with smallest $x_1(p)$ value) is covered by the left boundary of the D by D

square whose top boundary coincides with the top boundary of the slab. All the points that fall inside this square are said to be covered. Then the leftmost uncovered point is covered in a similar fashion, and so on. Our algorithm outputs these D by D squares. This procedure, referred to as to *cover the leftmost point first (CLPF)*, has worst-case time complexity of $O(n \log n)$ and generates a minimum cardinality cover.

The CLPF procedure is not a procedure with the best worst-case time-complexity bound for the slab problem. Let us now discuss a better algorithm (**fastCLPF**) whose worst-case time-complexity bound is only $O(n \log s)$, where s is the number of squares in an optimal cover. The idea behind the procedure is to sort the points with respect to their $i_1(p)$ values. This requires $O(n \cdot \log s)$ time, instead of $O(n \log n)$. Then the set of points is partitioned into sets $S_1, S_2, \ldots, S_k$ in such a way that all points in set S_j have identical $i_1(p)$ values and the $i_1(p)$ value of the points in set S_j is smaller than the one for the points in S_{j+1}. Clearly, a point in S_j and a point in S_{j+2} cannot be covered by the same square in a feasible solution. Our new procedure implements the CLPF algorithm by taking advantage of the above properties for the sets $S_1, S_2, \ldots, S_k$.

```
Procedure fastCLPF($P$, $D$)
Sort the points with respect to $i_1(p)$ into sets
    $S_1, S_2, \ldots, S_k$ as described above;
$R \leftarrow S_1 \cup S_2$; $j \leftarrow 2$;
while $R \neq \emptyset$ do
    $q \leftarrow \min\{x_1(p) \mid p \in R\}$;
    Let $Q$ be the set of points in $R$ at a distance at
        most $D$ (with respect to $x_1$ only) from $q$;
    $R \leftarrow R - Q$;
    output the $D$ by $D$ square whose left boundary
        includes point $q$ and whose top boundary
        coincides with the top boundary of the slab;
    while $j < k$ and $R$ contains elements from at
        most one of the sets in $\{S_1, S_2, \ldots, S_k\}$ do
        $j \leftarrow j + 1$; $R \leftarrow R \cup S_j$;
    endwhile
endwhile
end of procedure fastCLPF;
```

We claim that the cover generated by the above procedure is identical to the one generated by procedure CLPF. The reason is that when R has elements from two adjacent sets, S_i and S_{i+1}, q must be in set S_i and no element in S_{i+2} or higher indexed set can be covered by the same square as point q. We also claim that the time complexity is $O(n \log s)$. Sorting takes $O(n \log k)$ time by using balanced binary search trees for the $i_1(p)$ values. Since no point in S_i can be covered by the same square as a point in set S_{i+2}, it must be that $s \geqslant k/2$. Therefore, sorting takes $O(n \log s)$ time. The min operation is performed on each element of S_i at most twice and the test in the conditions of the while statement can be computed in constant time by associating with R the number of different sets S_i where its elements are located. Note that this information can be easily updated when the number of elements of R increases or decreases. Therefore, the overall time complexity is $O(n \log s)$. One can solve the above problem in $O(n)$ time by increasing (substantially in some cases) the space complexity [3]. For more details about this (impractical) method the reader is referred to [3].

Now let us use the above procedure to obtain a fast approximation algorithm for the CS_2 and then for the CS_d problem. Our strategy for $d = 2$ is to divide the problem into two subproblems, R_1 and R_2, and then finding an optimal cover for each of them. Subproblem R_1 contains all the points with $i_2(p)$ odd, and subproblem R_2 contains the remaining points. Let us consider problem R_1. It is simple to see that all the points in R_1 belong to slabs (with height D) which are D units apart. Therefore, an optimal solution to R_1 consists of finding an optimal solution to each of the slabs which we know takes $O(n \log s)$ time. Since the number of squares in an optimal solution to the original problem is at least as large as that of an optimal solution for R_1 or R_2, it then follows that our algorithm has an approximation bound of two. For $d \geqslant 2$, the problem is partitioned into 2^{d-1} subproblems. By following similar arguments, one can easily show that the approximation bound for our procedure is 2^{d-1}. In our implementation, instead of finding a partition and then refining it, we perform both partitions directly. The set of problems generated is P_1, $P_2, \ldots,$ and P_k. Each of these subproblems contains all points with the same $(i_2(p), \ldots, i_d(p))$ value, i.e., each corresponds to a slab in a subproblem R_j. Procedure PARTITION-FIRST based on the above strategy is given below.

```
Algorithm PARTITION-FIRST($P = \{p_1, p_2, \ldots, p_n\}$, $D$)
partition the elements with respect to
    $(i_2(p), \ldots, i_d(p))$ into sets $P_1, P_2, \ldots, P_k$;
apply the fastCLPF procedure to each set;
end of algorithm PARTITION-FIRST
```

The partition can be obtained by a lexicographic sort of the points with respect to $(i_2(p), \ldots, i_d(p))$. Sorting by the algorithm given in [2] (or an equivalent algorithm) takes

$$O(d\,n + n \log k)$$

time. The second step applies procedure fastCLPF which takes overall time $O(n \log t)$, where t is the number of hypersquares in the solution generated by the algorithm. Since $k \leqslant t \leqslant 2^{d-1}S$, it the follows that the overall time complexity for our procedure is

$$O(dn + n \log s).$$

Theorem 1. *Algorithm* PARTITION-FIRST *generates a solution for the CS_d problem with approximation bound 2^{d-1}. The time complexity for our algorithm is*

$$O(dn + n \log s)$$

time, where s is the number of hypersquares in an optimal solution.

Proof. By the above discussion. □

3. Polynomial-time approximation scheme

Let us consider the l_2-*slab* problem, i.e., all points lie in E^2 inside a rectangle with height lD, for some integer $l \geqslant 1$. First we show that the l_2-slab problem can be solved in $O(4ln^{4l})$ time via dynamic programming. Then we generalize our procedure to solve the l_d-slab problem (i.e., all points lie in E^d inside a hyperrectangle in which all dimensions, except the first, have length LD). Finally, we present an improved polynomial-time approximation scheme by combining the algorithm for the l_d-slab problem with the polynomial-time approximation scheme in [4].

The idea behind the dynamic-programming procedure is to find all covers that satisfy certain properties and cover at least points $p_1, p_2, \ldots, p_i$. The fact that the cover satisfies some special properties (which we specify later) is important as otherwise there is an infinite number of such covers. These properties are defined in such a way that the number of distinct covers is bounded by a polynomial on n.

Let us assume that the points are sorted with respect to their first coordinate value, i.e.,

$$x_1(p_1) \leqslant x_1(p_2) \leqslant \cdots \leqslant x_1(p_n).$$

Square c_i in a cover C is said to be an *a-square* (*anchored square*) if there is a point $r \in P$ located on the left boundary of c_i and a point $s \in P$ (not necessarily different from r) located on the top boundary of c_i that are not contained (are not located inside or on the boundary) in another square in the cover C. The point p_j with least index located on the left (top) boundary of the a-square c_i in a cover C that is not contained in another square in C is called the *left* (*top*) *anchor of c_i in C*. We shall also refer to the two points as simply *anchors*. A cover is said to be an *a-cover* (*anchor cover*) if all the squares in it are a-squares. It is simple to show that any cover can be transformed to an a-cover without increasing the number of squares in it. Therefore, for each problem instance there is at least one a-cover in the set of optimal covers. We say that a vertical line *intersects* a square if the intersection of the set of points that form the line and the square is nonempty. We say than an a-cover is an *s-cover* if every vertical line does not intersect more than $2l - 1$ squares in the cover. To show that the time complexity of our algorithm is bounded by a polynomial on n, we establish in Lemma 2 that every problem instance has an s-cover among the set of optimal covers.

Before we prove lemma 2 we need to define more terms. Let C_{opt} be the set of optimal a-

covers for some problem instance. Assume that the number of squares in each of these covers is t. We define b_i as the x_1 coordinate value of the left anchor of square c_i in the optimal a-cover C. Assume without loss of generality that the squares in each optimal a-cover C have been rearranged so that $b_1 \leqslant b_2 \leqslant \cdots \leqslant b_t$.

Define C_t as

$$\{C \mid C \in C_{opt} \text{ and } b_t \text{ for } C \text{ is maximum amongst all covers in } C_{opt}\}$$

and C_j as

$$\{C \mid C \in C_{j+1} \text{ and } b_j \text{ for } C \text{ is maximum amongst all covers in } C_{j+1}\},$$

for $1 \leqslant j < t$. Each cover C in C_1 is called a *maximum index cover*.

Lemma 2. *Every l_2-slab problem has an s-cover among the set of optimal covers.*

Proof. As mentioned above, every problem instance has at least one a-cover among the set of optimal covers. We now show that at least one of these a-covers is an s-cover. The proof is by contradiction. Suppose that all optimal a-covers are not s-covers. Let C' be a maximum-index cover among all optimal a-covers. By assumption there is a vertical line, w, that intersects at least $2l$ squares in C'. Transform C' by the rule given in Fig. 1 at line w and then move the newly introduced squares down and to the right until all of them are anchored or some can be deleted because all points have been covered. Let C be the cover. It is simple to see that the cardinality of C is not larger than the cardinality of C'. If C has fewer squares than C', then it contradicts that C' is an optimal cover. On the other hand, if C and C' have the same cardinality, then we contradict the assumption that C' is the maximum-index cover, since one can easily show that the maximum-index cover in $\{C, C'\}$ is not C'. So it must be that there is at least one s-cover in the set of optimal covers. This completes the proof of the lemma. □

Before presenting our algorithm we define more terms. Deleting a subset of squares from an a-cover C results in an *a-subcover* which we call C'. If points $p_1, p_2, \ldots, p_i$ are covered by an a-subcover C', and all the left anchors of the squares in C' are points from the set $\{p_1, p_2, \ldots, p_i\}$, then C' is called an *i–a-subcover*. Two i–a-subcovers are said to be *t-equivalent* (*tail equivalent*) if every square in each of these i–a-subcovers intersected by a vertical line through p_i is also in the other i–a-subcover. Two i–a-subcovers are said to be *ct-equivalent* (*cardinality and tail equivalent*) if they are t-equivalent and have the same cardinality. An i–a-subcover C *dominates* the i–a-subcover C' iff C and C' are t-equivalent and C has fewer squares than C'. An i–a-subcover is said to be an *optimal i–a-subcover* if it is the i–a-subcover of an s-cover in the set of optimal covers. A set of a-subcovers is said to be *irreducible* if every pair of a-subcovers in the set are incomparable, i.e., one does not dominate the other and both are not ct-equivalent. Procedure DP scans the points in the order $p_1, p_2, \ldots, p_n$ and generates at each iteration a set of irreducible i–a-subcovers. To show correctness we prove in Lemma 3 that every optimal i–a-subcover has a ct-equivalent i–a-

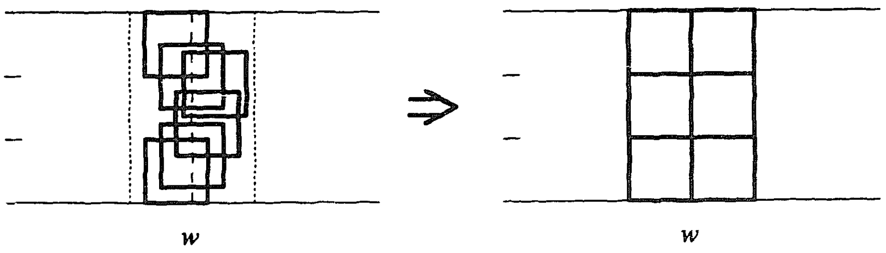

Fig. 1. Transformation rule when $l = 3$.

subcover in the set of irreducible i–a-subcovers constructed by the procedure.

```
Algorithm DP ($p_1, p_2, \ldots, p_n$)
sort the elements so that $x_1(p_1) \leqslant x_1(p_2) \leqslant \cdots \leqslant x_1(p_n)$;
$FS \leftarrow \{\emptyset\}$; /* a set with one element whose
    value is an empty list of squares */
for $i = 1$ to $n$ do
    $newFS \leftarrow \emptyset$;
    for each $a$-subcover $C$ in $FS$ do
        case
        :$p_i$ is covered in $C$: Add $\{C\}$ to $newFS$;
        :less than $2l - 1$ squares are intersected by
         a vertical line through $p_i$:
            Add $\{C' \mid C'$ is $C$ plus an $a$-square with
              $p_i$ as its left anchor$\}$ to $newFS$;
        :else: /* $newFS$ remains unchanged */;
        endcase
    endfor;
    Let $FS$ be a maximal cardinality subset of
      irreducible $a$-covers in $newFS$;
endfor;
Output any cover with the least number of squares
    in $FS$
end of algorithm DP
```

Lemma 3. *For every l_2-slab problem, algorithm* DP *generates an optimal cover.*

Proof. It is simple to show that the proof of the lemma reduces to showing that if at the ith iteration every optimal $(i-1)$–a-subcover has a ct-equivalent $(i-1)$–a-subcover in FS_b (set FS at the beginning of the iteration), then every optimal i–a-subcover has a ct-equivalent i–a-subcover in FS_e (set FS at the end of the iteration). Let T be any s-cover in the set of optimal covers. Let T_{i-1} (T_i) be the $(i-1)$–a-subcover (i–a-subcover) of T. By assumption there is an $(i-1)$–a-subcover in FS_b (say C_{i-1}) ct-equivalent to T_{i-1}. If p_i is covered by the squares in T_{i-1}, then $C = C_{i-1}$ is added to $newFS$ and T_i is the same as T_{i-1}. On the other hand, if p_i is not covered by the squares in T_{i-1}, then when C_{i-1} is considered in the **for**-loop an i–a-subcover C t-equivalent to T_i is added to $newFS$. In either case the i–a-subcover $C' = C$ in $newFS$ ends in FS_e, or an i–a-subcover C' t-equivalent to C and with at most the same number of elements as in C ends in FS_e. Since T is an optimal cover, it must be that C' has the same cardinality as C. Therefore, C' which is ct-equivalent to T_i ends in FS_e, and the result follows. □

To establish our time-complexity bound we need to specify some implementation details. We find a maximal subset of irreducible a-covers in $newFS$ as follows. Each a-cover in $newFS$ is characterized by the anchors of the squares intersected by a vertical line through p_i and the number of squares in the cover. We sort via radix sort the covers in $newFS$ using as keys the indices of the points which are anchors (in sorted order) of the squares that characterize the covers. Associated with each key there is the number of squares in the cover. Deletion of dominated i–a-subcovers and ct-equivalent i–a-subcovers can be easily done by traversing the list once. Thus, if $newFS$ has m elements, the overall time taken to execute the step is $c(m+n)$, where c is the number of anchor points that characterize an i–a-subcover. The above bound can be achieved by sorting the elements via radix sort since the keys are c dimensional points whose value along each axis is an integer in the range $[1, n]$.

Lemma 4. *For every l_2-slab problem, algorithm* DP *takes* $O(4l\, n^{4l})$ *time.*

Proof. By Lemma 2, the definition of s-covers, and the fact that there are at most two anchors per square, we know that $|FS| \leqslant n^{4l-2}$. Since for each element in FS at most n covers are added to $newFS$, $|newFS| \leqslant n^{4l-1}$. Using the procedure discussed above the maximal set of irreducible covers in $newFS$ can be identified in

$$O((4l-2)n^{4l-1})$$

time. The above operation is repeated n times, therefore the overall time complexity is $O(4l\, n^{4l})$. □

For the l_d-slab problem, the number of squares partitioned by a hyperplane passing through p_i is at most $2l^{d-1} - 1$ instead of $2l - 1$, and the number of anchors per square is d instead of two. Therefore, the time complexity for the l_d-slab problem is

$$O\left(d(2l^{d-1}-1)n^{d(2l^{d-1}-1)+1}\right).$$

This algorithm can be easily incorporated with the polynomial-time approximation scheme given in [4]. The idea is to apply the algorithm to l^d problem instances of the l_d-slab problem. The l^d problem instances can be partitioned into l^{d-1} groups such that the total number of points in all problems in each group is n. This results in an overall time complexity bound

$$O\left(l^{d-1}d(2l^{d-1}-1)n^{d(2l^{d-1}-1)+1}\right).$$

The approximation bound is $(1+1/l)^{d-1}$.

Theorem 5. *Combining algorithm* DP *with the polynomial-time approximation scheme in* [4] *results in a procedure that generates a solution to the CS_d problem with approximation bound* $(1+1/l)^{d-1}$ *and time complexity*

$$O\left(l^{d-1}d(2l^{d-1}-1)n^{d(2l^{d-1}-1)+1}\right).$$

Proof. By the above discussion. □

4. Discussion

All our techniques can also be adapted to the CR_d and the CD_d problem. For the CR_d problem the approximation and time-complexity bounds are identical to the ones for the CS_d problem. For the CD_d problem the algorithm in Section 2 has identical time-complexity bound; however, the approximation bound is $2^{d-1}\lceil\sqrt{d}\rceil^d$. For the polynomial-time approximation scheme, the approximation bound is $2(1+1/l)^{d-1}$ and the time-complexity bound is

$$O\left(l^{d-1}d\lceil 2\sqrt{d}\rceil\lceil l\sqrt{d}\rceil^{d-1}n^{d\lceil 2\sqrt{d}\rceil^{d-1}+1}\right).$$

The same techniques can be adapted to the packing problem studied in [4]. The approximation and time-complexity bounds are smaller than for the CS_d problem. For brevity we do not discuss this further. As pointed out in [4] it is important to develop fast approximation algorithms for the l_d-slab problem These algorithms would imply a much better time-complexity bound for the CS_d problem at the expense of a slightly larger approximation bound.

From our algorithms one can define a heuristic which we believe has a reasonable behavior with respect to the time complexity and the approximation. The idea is to partition the set of points into two groups. Those which are "far" from other points and those with "close" neighbors. The heuristic then applies our polynomial-time approximation scheme to the problem defined by all the points in the first group. An optimal solution will be generated quickly for this subproblem. The remaining problem is solved by the approximation algorithm in Section 2.

Gonzalez [3] has developed other approximation algorithms for the problems discussed in this paper. The algorithm in this paper outperforms these algorithms because it is either faster, requires less space, or has a smaller worst-case approximation bound. To generate good solutions in practice one should execute all of these algorithms concurrently and then select the best of the solution generated. Proving that this approach has a smaller worst-case approximation bound is not simple, and remains an open problem.

References

[1] R.J. Flower, M.S. Paterson and S.L. Tanimoto, Optimal packing and covering in the plane are NP-complete, *Inform. Process. Lett.* **12** (1981) 133–137.

[2] T. Gonzalez, The on-line d-dimensional dictionary problem, Tech. Rept. RUU-CS-90-31, University of Utrecht, July 1990.

[3] T. Gonzalez, Covering a set of points with fixed size hypersquares and related problems, in: *Proc. 28th Ann. Allerton Conf. on Communications, Control, and Computing* (1990) 838–847.

[4] D.S. Hochbaum and W. Maass, Approximation schemes for covering and packing problems in image processing and VLSI, *J. ACM* 32 (1) (1985) 130–136.

[5] D. Johnson, The NP-completeness column: An ongoing guide, *J. Algorithms* 3 (1982) 182–195.

[6] S. Masuyama, T. Ibaraki, and T. Hasegawa, The computational complexity of the m-center problems on the plane, *Trans. IECE Japan* **E64** (1981) 57–64.

[7] K.J. Supowit, Topics in computational geometry, Rept. No. UIUCDCS-R-81-1062, Dept. Computer Science, University of Illinois, Urbana, IL, 1981.

[8] S.L. Tanimoto, Covering and indexing an image subset, in: *Proc. 1979 IEEE Computer Society Conf. on Pattern Recognition and Image Processing* (IEEE, Chicago, 1979) 239–245.

[9] S.L. Tanimoto and R.J. Fowler, Covering image subsets with patches, in: *Proc. 5th Internat. Conf. on Pattern Recognition* (1980) 835–839.