Efficient on-line algorithms for maintaining k-cover of sparse bit-strings

Amit Kumar[1], Preeti R. Panda[1], and Smruti Sarangi[1]

1 **Indian Institute of Technology, Delhi**
New Delhi, India

Abstract

We consider the on-line problem of representing a sparse bit string by a set of k intervals, where k is much smaller than the length of the string. The goal is to minimize the total length of these intervals under the condition that each 1-bit must be in one of these intervals. We give an efficient greedy algorithm which takes time $O(\log k)$ per update (an update involves converting a 0-bit to a 1-bit), which is independent of the size of the entire string. We prove that this greedy algorithm is 2-competitive. We use a natural linear programming relaxation for this problem, and analyze the algorithm by finding a dual feasible solution whose value matches the cost of the greedy algorithm.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases On-line algorithms, string algorithms

1 Introduction

In many applications, we need to maintain representations of sparse bit vectors. Such representations should be compact, and one should be able to update them efficiently even in the on-line setting. One way of achieving this is to cluster the 1-bits into small number of intervals. More formally, given a long bit vector and a (relatively) small integer k, we would like to find a set of k (disjoint) intervals such that each 1-bit is in one of these intervals. Such a representation can lead to significant compression of data, and fast retrieval of information. The goal here would be find these intervals such that they contain as few 0-bits as possible, i.e., we would like to minimize the total length of these intervals. We call this the k-`Cover`problem. Such representations have applications in many areas of computer science including compilers, architecture and databases. In this problem, we address this problem in the online or streaming setting : in each time step, one of the 0-bits becomes 1, and we would like to update our representation in time depending on k only. Indeed, in many applications n could be very large, and even running time of $O(\log n)$ per time step could be prohibitive.

Toussi et al. [1] proposed a new algorithm for compressing the information stored in a traditional *points-to*[2] analysis pass in a compiler. In a *points-to* analysis, a compiler first makes a list of all possible objects that are in the scope of a function, and then marks a subset of them that can be possibly accessed by the function under consideration. There are both off-line and online variants of this algorithm. If the analysis is done in one pass, then we can consider it as an off-line version. If the analysis takes multiple passes especially after taking inter-procedural information into account, we can consider it to be an online variant.

[1] proposed to save this information in the form of a bit vector. Each bit corresponds to a unique object/variable in the system. If a bit is 1, then there is a possibility that a given function might access a certain variable. This bit vector can further be compressed to form a *ranged bit vector* that saves only intervals containing the 1-bits. The authors do entertain the possibility of having intermittent 0's similar to our definition of a k-`Cover` problem. In this case, if we decrease the value of k, then we have better compression, yet the compiler is more conservative.

In the area of scientific databases, Sinha et al. [7, 8] reported the use of a scheme similar to the k-`Cover` problem. They consider a variant of binning for bit vector databases that is tailored for storing sparse vectors. They consider a hierarchical structure, where each level stores a bit vector as a sequence of intervals. At the highest level, they store a set of k intervals, where k might possibly be a variable. To filter out false positives, they introduce additional levels to store each interval further as a set of intervals. The last level does not have any false positives. This helps them answer range queries with different degrees of approximation. There is clearly a tradeoff between time and accuracy.

In the areas of computer architecture [9, 10, 11] used similar ideas to compress post-silicon debug information. In each epoch, the authors propose to mark a cache line with 1 if it is written to in that interval. The default state of each line is 0. [11] proposed to save this information in the form of a large bit vector. [10] tried to further improve upon this work by trying to compress the bit vector using LZW compression. We can use the idea of covering by k intervals in this context, because the bit vectors were found to be fairly sparse.

It is easy to solve the off-line version of the k-`Cover` problem. Indeed, the following simple greedy algorithm works – define a gap in a bit-string as a maximal sequence of 0's. Then, the optimal solution should construct the k intervals such that it leaves out the largest $k-1$-gaps (excluding the leftmost and rightmost gaps). In fact, we can easily convert this to an on-line algorithm which maintains a suitable data-structure for storing the gaps – each update can be achieved in $O(\log n)$-time. But as pointed out above, in many applications, k could be much smaller than n, and we would like an algorithm whose running time per update depends on k only.

Our Results We give a simple greedy 2-competitive algorithm for the k-`Cover` problem. Further our algorithm only takes $O(\log k)$-time per update. The analysis of our algorithm requires non-trivial ideas. We do not have any natural charging or potential function based arguments for bounding the competitive ratio of our algorithm. Instead, we use a dual fitting based argument. We first write a natural LP relaxation for the k-`Cover` problem (which has integrality gap of 1). We then show that we can assign values to the variables in the dual LP such that the values are feasible and the objective function is close to the cost of the greedy algorithm. We expect our techniques to be useful for analyzing other on-line string algorithms.

Related Work There is a vast amount of literature on string algorithms and string compression techniques [5, 6]. To the best of our knowledge, this problem has not been considered before in this context. Our problem is a special case of the clustering problem which minimizes the sum of cluster diameters [4]. A constant factor approximation algorithm is known for the latter problem (for any arbitrary metric) [3].

In Section 2, we define the problem formally and describe the greedy algorithm. Subsequently, we analyze the greedy algorithm by first giving an LP relaxation for our problem (Section 3) and then setting the dual variables (Section 4). Finally, we show that the greedy

algorithm cannot be better than 2-competitive (Section 5).

2 Preliminaries

We are given a sequence of n bits. At time t, let S_t denote the sequence of these bits. Let $b_i^{(t)}$ denote the i^{th} bit at time t. Initially, in S_0, all the bits are 0. At each point of time t, one of the bits which was 0 at time $(t-1)$ becomes 1. The input instance also specifies a parameter k. The algorithm needs to maintain a set of k disjoint intervals (which we will call *covers*) at all points of time t – these covers must satisfy the property that any bit which is 1 should lie in one of these covers. Let $C_1^{(t)}, \ldots, C_k^{(t)}$ denote the set of covers at time t. For an interval I, $l(I)$ and $r(I)$ will denote the left and the right end-points of I. Define a *gap* as the sequence of 0's between two consecutive covers. We shall use $G_i^{(t)}$ to denote the gap between $C_i^{(t)}$ and $C_{i+1}^{(t)}$, i.e., the open interval $(r(C_i^{(t)}), l(C_{i+1}^{(t)}))$. Our goal is to minimize the total length of covers at any time, i.e., $\sum_{i=1}^{r} |C_i^{(t)}|$. Let $\texttt{opt}_t$ be the value of the optimal solution at time t.

We propose a simple greedy algorithm $\mathcal{A}$. Suppose at time t, the bit d_t becomes one. If d_t lies in one of the covers at time t, we do not need to do anything. Otherwise, we tentatively create one more cover consisting of the singleton element d_t. Now we have a set of $k+1$ covers (which cover all the 1 bits). Among these $k+1$ covers, we find two consecutive covers with the minimum gap between them – call these C and C', and assume that C lies to the left of C'. We merge them into a single cover, i.e., we replace C and C' by $[l(C), r(C')]$. Thus we again have only k covers at time $t+1$ and they contain all the 1 bits. Note that we always grow the current covers.

For the sake of simplifying the analysis, we shall make the following assumption : at any point of time t, the bit d_t that becomes 1 does not lie in any of the existing covers. This assumption is without loss of generality – if we forbid such bits from becoming 1, we do not affect our algorithm $\mathcal{A}$, and this can only reduce the value of the optimum solution.

3 A Linear Programming Relaxation and its dual

In order to bound the cost of $\mathcal{A}$, we need to have a handle on the value of the optimum (for any time t). Although it is easy to understand what the optimum is, it will be more convenient to deal with a lower bound given by a linear programming relaxation. For a time t, consider the following LP relaxation. For any interval I, it has variables $x_I^{(t)}$, which is supposed to be 1 if I gets chosen in the set of covers at time t, otherwise it should be 0. Let $l(I)$ denote the length of I. The objective function in (1) measures the total length of the chosen intervals. The first constraint (2) says that for any bit which is 1, one of the covers must contain it. The second constraint (3) requires that we can choose only k intervals. Given any solution to our problem instance, we can define a solution to this LP (where we set $x_I^{(t)}$ to 1 iff we take I in our set of covers at time t) which satisfies all the constraints. Therefore, the optimal value of this LP is a lower bound on $\texttt{opt}_t$.

$$\min \sum_I l(I) x_I^{(t)} \tag{1}$$

$$\sum_{I: i \in I} x_I^{(t)} \geq 1 \qquad \text{for all bits } i \text{ such that } b_i^{(t)} = 1 \tag{2}$$

$$\sum_I x_I^{(t)} \leq k \tag{3}$$

$$x_I^{(t)} \geq 0 \qquad \text{for all intervals } I$$

Now, we write the dual of the above LP. Note that the optimal value of the dual is at most the optimal value of the LP above. We have variables $\alpha_i^{(t)}$ for all bits i such that $b_i^{(t)} = 1$ (corresponding to constraints (2)) and $\beta^{(t)}$ for constraint (3).

$$\max \sum_{i: b_i^{(t)} = 1} \alpha_i^{(t)} - k\beta^{(t)} \tag{4}$$

$$\sum_{i: i \in I, b_i^{(t)} = 1} \alpha_i^{(t)} - \beta^{(t)} \leq l(I) \qquad \text{for all intervals } I \tag{5}$$

$$\alpha_i^{(t)}, \beta^{(t)} \geq 0$$

Our goal is to show that for any t, we can construct a feasible solution $\alpha_i^{(t)}, \beta^{(t)}$ to the dual LP above such that the cost of the solution is at least half of the cost of our algorithm $\mathcal{A}$ at time t. Once we can show this, then the cost of our algorithm will be at most twice the cost of the dual LP, and hence at most twice the cost of the primal LP. This will show that $\mathcal{A}$ is 2-competitive. In the next section, we show how to set these dual variables.

4 Setting the Dual Variables

Recall that $C_1^{(t)}, \ldots, C_k^{(t)}$ denote the set of covers constructed by $\mathcal{A}$ at time t. Further $G_i^{(t)}$ denotes the gap between $C_i^{(t)}$ and $C_{i+1}^{(t)}$. Note that there will be $k-1$ gaps. At time t, let $\gamma^{(t)}$ denote the length of the smallest gap. We set $\beta^{(t)} = \max_{t' \leq t} \gamma^{(t)}$, i.e., $\beta^{(t)}$ is the highest among all time t of the smallest gap. The values $\alpha_i^{(t)}$ will be more subtle. Instead of specifying their exact value, we will write some invariants which will be satisfied at all times. Once we prove these invariants, it will be easy to check that the dual constraints are satisfied. Suppose bit $b_i^{(t)} = 1$ and it lies in cover $C_r^{(t)}$. The values $\alpha_i^{(t)}$ will satisfy the following conditions :

(i) If $b_i^{(t)}$ is the right end-point of $C_r^{(t)}$, then $\alpha_i^{(t)} = \min(\beta^{(t)}, |G_i^{(t)}|) + 1$ (if $b_i^{(t)}$ lies in the rightmost cover, we treat $|G_i^{(t)}|$ as infinity).

(ii) If $b_i^{(t)}$ is not the right end-point of $C_r^{(t)}$, let $b_u^{(t)}$ be the next bit to the right of $b_i^{(t)}$ which is 1 (so $b_u^{(t)}$ also lies in $C_r^{(t)}$). Let $d(i,u) = u - i - 1$ be the gap between the two bits (it counts the number of 0's in between). Then $\alpha_i^{(t)}$ will have a value which will be at most $\min(\beta^{(t)} + 1, 2d(i,u) + 2)$.

(iii) $\sum_r \alpha_r^{(t)} - k\beta^{(t)}$ is at least $\sum_{r=1}^{k} |C_r^{(t)}|$.

We prove this by induction on time t.

Base Case : Consider time $t = k$: we have exactly k bits which are 1. The k covers will contain these singleton elements each. We set $\alpha_i^{(k)} = \gamma^{(k)} + 1$ for each $i = 1, \ldots, k$. Also, $\beta^{(k)} = \gamma^{(k)}$. Clearly, condition (i) holds. The requirements of condition (ii) do not hold for any cover, so it is vacuously true. For condition (iii), note that $\sum_r \alpha_r^{(k)} = k\gamma^{(k)} + k$, and so it is satisfied as well.

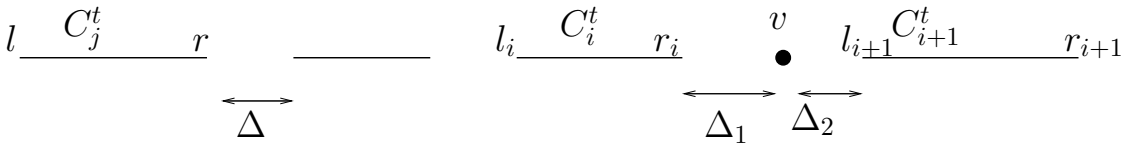

Figure 1 Covers at time t and arrival of new 1-bit at v

Induction Step : Suppose the invariants above hold at some time t. We will prove that they hold at time $t+1$ as well. Suppose at time $t+1$, the bit b_v becomes 1. Assume that b_v lies between the covers $C_i^{(t)}$ and $C_{i+1}^{(t)}$. For the ease of notation, let us denote the end-points of these covers as $C_i^{(t)} = [l_i, r_i], C_i^{(t+1)} = [l_{i+1}, r_{i+1}]$. Let the sequence of 0's between r_i and v be of length Δ_1, and that between v and l_{i+1} be of length Δ_2 (see figure above). There are three possibilities on what our algorithm can do. We show how to set the dual variables in each of these cases.

(a) We merge v with the right end point r_i of $C_i^{(t)}$: first of all, observe that $\gamma^{(t+1)} \leq \gamma^{(t)}$, because we are only reducing the size of one of the gaps. Hence, $\beta^{(t+1)}$ stays at $\beta^{(t)}$. Also, it must be the case that $\Delta_1 \leq \Delta_2$, and $\Delta_1 \leq \gamma^t$. We change the dual variables as follows :

$$\alpha_v^{(t+1)} = 1+\min(\beta^{(t+1)}, \Delta_2), \alpha_{r_i}^{(t+1)} = 1+\Delta_1+\min(\Delta_1+\Delta_2+1, \beta^{(t+1)})-\min(\Delta_2, \beta^{(t+1)}).$$

Rest of the dual variables do not change. Let us now check the invariants. We need to check condition (i) for v because it is the new right end-point of its cover. But the definition of $\alpha_v^{(t+1)}$ is in accordance with condition. Condition (ii) now needs to be checked for r_i : recall that $\Delta_1 \leq \gamma^t$, which is at most $\beta^{(t+1)}$. So,

$$\alpha_{r_i}^{(t+1)} \leq 1 + \Delta_1 + \Delta_1 + \min(\Delta_2, \beta^{(t+1)}) - \min(\Delta_2, \beta^{(t+1)}) = 2\Delta_1 + 2,$$

and

$$\alpha_{r_i}^{(t+1)} \leq 1 + \Delta_1 + \beta^{(t+1)} - \min(\Delta_2, \beta^{(t+1)}) \leq 1 + \beta^{(t+1)}.$$

Let us now check condition (iii). The induction hypothesis applied to (i) says that the dual value $\alpha_{r_i}^{(t)}$ was $1 + \min(\Delta_1 + \Delta_2 + 1, \beta^{(t+1)})$. So,

$$\begin{aligned}\left(\sum_r \alpha_r^{(t+1)} - k\beta^{(t+1)}\right) - \left(\sum_r \alpha_r^{(t)} - k\beta^{(t)}\right) &= \alpha_v^{(t+1)} + \alpha_{r_i}^{(t+1)} - \alpha_{r_i}^{(t)} \\ &= \Delta_1 + 1,\end{aligned}$$

which is exactly the increase in the cost of our solution.

(b) We merge v with the left end point l_{i+1} of $C_{i+1}^{(t)}$: As argued above, $\Delta_2 \leq \Delta_1, \Delta_2 \leq \gamma^{(t)}$ and $\beta^{(t+1)} = \beta^{(t)}$. We again change the dual variables for v and r_i only :

$$\alpha_v^{(t+1)} = 1+\Delta_2+\min(\Delta_1+\Delta_2+1, \beta^{(t+1)})-\min(\Delta_1, \beta^{(t+1)}), \alpha_{r_i}^{(t+1)} = 1+\min(\Delta_1, \beta^{(t+1)}).$$

Let us check the invariants. Condition (i) holds for r_i again. We need to check (ii) for v. Again, note that

$$\alpha_v^{(t+1)} \leq 1 + \Delta_2 + \min(\Delta_1, \beta^{(t+1)}) + 1 + \Delta_2 - \min(\Delta_1, \beta^{(t+1)}) = 2\Delta_2 + 2,$$

and

$$\alpha_v^{(t+1)} \leq 1 + \Delta_2 + \beta^{(t+1)} - \min(\Delta_1, \beta^{k+1}) \leq 1 + \beta^{(t+1)}.$$

Finally, we see the change in the objective function. As in the previous case, $\alpha_{r_i}^{(t)} = 1 + \min(\Delta_1 + \Delta_2 + 1, \beta^t)$. So the increase in the objecive function is

$$\alpha_v^{(t+1)} + \alpha_{r_i}^{(t+1)} - \alpha_t = 1 + \Delta_2,$$

which is also the increase in our cost.

(c) We merge covers $C_j^{(t)}$ and $C_{j+1}^{(t)}$, where $j \neq i$. So, we add a new cover consisting of the singleton element v. Let Δ denote the length of the gap $G_j^{(t)}$ (between $C_j^{(t)}$ and $C_{j+1}^{(t)}$). So, $\Delta \leq \min(\Delta_1, \Delta_2)$. Further $\Delta = \gamma^t$. Also, it is possible that $\gamma^{(t+1)}$ is larger than $\gamma^{(t)}$. So $\beta^{(t+1)} = \max(\beta^{(t)}, \gamma^{(t+1)})$. Let the end-points of $C_j^{(t)}$ be $[l, r]$. For this, we consider two sub-cases.

- $\beta^{(t+1)} = \beta^{(t)}$: We set

$$\alpha_{r_i}^{(t+1)} = \min(\Delta_1, \beta^{(t+1)}) + 1, \alpha_v^{(t+1)} = \min(\Delta_2, \beta^{(t+1)}) + 1,$$

and

$$\alpha_r^{(t+1)} = 2\Delta + 1 + \min(\Delta_1 + \Delta_2 + 1, \beta^{(t+1)}) - \min(\Delta_1, \beta^{(t+1)}) - \min(\Delta_2, \beta^{(t+1)}).$$

All other dual variables remain unchanged. Note that the bit at position r is no longer the end-point of a cover, so we need to check condition (i) for r_i and v only. Again, by definition, the conditions hold here. We need to check condition (ii) for bit r now. Observe that

$$\alpha_r^{(t+1)} \leq 2\Delta + 1 + 1 = 2\Delta + 2,$$

and because $\Delta \leq \Delta_1, \Delta_2, \beta^{(t+1)}$,

$$\begin{aligned}\alpha_r^{(t+1)} \leq 1 + \min(\Delta_1, \beta^{(t+1)}) + \min(\Delta_2, \beta^{(t+1)}) + \min(\Delta_1 + \Delta_2 + 1, \beta^{(t+1)}) \\ - \ \min(\Delta_1, \beta^{(t+1)}) - \min(\Delta_2, \beta^{(t+1)}) \leq 1 + \beta^{(t+1)}.\end{aligned}$$

Finally, we calculate the change in the objective function. By induction hypothesis and condition (i), we know that

$$\alpha_r^{(t)} = \Delta + 1, \alpha_{r_i}^{(t)} = \min(\Delta_1 + \Delta_2 + 1, \beta^{(t+1)}) + 1.$$

So, the change in objective function is

$$\begin{aligned}\alpha_r^{(t+1)} + \alpha_{r_i}^{(t+1)} + \alpha_v^{(t+1)} - \alpha_r^{(t)} - \alpha_{r_i}^{(t)} \ &= \ 2\Delta + 3 + \min(\Delta_1 + \Delta_2 + 1, \beta^{(t+1)}) - \Delta - 1 \\ &\quad - \min(\Delta_1 + \Delta_2 + 1, \beta^{(t+1)}) - 1 = \Delta + 1,\end{aligned}$$

which is exactly the increase in the cost of the algorithm.

- $\beta^{(t+1)} = \gamma^{(t+1)} > \beta^{(t)}$: It must be the case that $\Delta_1, \Delta_2 > \beta^{(t)}$. For all the right end points r_j of the covers in the solution at time $(t+1)$ (which includes the singleton element v), we set $\alpha_{r_j}^{(t+1)} = \beta^{(t+1)} + 1$. Rest of the dual variables remain unchanged. Note that the bit at position r is no longer the end-point of a cover, and so, $\alpha_r^{(t+1)} = \alpha_r^{(t)} = \Delta + 1$. Condition (i) is satisfied because gap between any two covers is at least $\beta^{(t+1)}$. Condition (ii) is also trivially satisfied – for bit r, note that the gap to the next 1 bit on the right is Δ. So, we just need to check the change in the objective function. Note that we have change the dual value of the right end point of all covers

except r at time t. Also, we added a new cover $\{v\}$. Since $\alpha_i^{(t)} \leq \beta^{(t)} + 1$ for all bits $b_i^{(t)}$ (invariant (i)), the change is

$$\left(\sum_r \alpha_r^{(t+1)} - k\beta^{(t+1)}\right) - \left(\sum_r \alpha_r^t - k\beta^t\right) \geq k(\beta^{(t+1)} + 1) - \Big((k-1)(\beta^{(t)}+1)) - k(\beta^{(t+1)} - \beta^{(t)})\Big)$$
$$= \beta^{(t)} + 1 \geq \Delta + 1$$

which is again the increase in our cost.

Thus, we have shown that the invariants hold at all times. We now need to show that the dual variables are feasible to the dual LP.

▶ **Lemma 1.** *For any interval I and time t,*

$$\sum_{i: i \in I, b_i^{(t)}=1} \alpha_i^{(t)} - \beta^{(t)} \leq 2l(I).$$

Proof. Consider an interval $I = [a, b]$. Suppose $a \in C_j^{(t)}$ and $b \in C_l^{(t)}$ $(j \leq l)$. If $i \in I$ and $b_i = 1$, let $n(i)$ be the next bit (on the right) which is 1. Let A be the set of all bit positions which are 1 in I except for the last such bit. So if $i \in A$, then $n(i) \in I$. Also, $l(I) - 1 \geq \sum_{i \in A}(n(i) - i)$ (we are not counting the last 1-bit, so we subtract 1 from LHS). For any $i \in A$, $\alpha_i^{(t)} \leq 2(n(i) - i)$. Indeed, for any bit $i \in A$ which is not the last bit in the cover containing it, this follows from condition (ii). For bits i which are the right end-point of a cover, this follows from condition (i) (in fact, we do not need the factor 2 here). Let u be the last bit in I which is 1. By condition (ii) or (i), $\alpha_u^{(t)} \leq \beta^{(t)} + 1$. Putting, everything together, we have

$$\sum_{i: i \in I, b_i^{(t)}=1} \alpha_i^{(t)} = \sum_{i \in A} \alpha_i^{(t)} + \alpha_u^{(t)} \leq \sum_{i \in A}(n(i) - i) + \beta^{(t)} + 1 \leq 2l(I) + \beta^{(t)}.$$

This proves the lemma. ◀

The lemma shows that $\alpha_i^{(t)}/2, \beta^{(t)}/2$ are dual feasible. Condition (iii) implies that the cost of this solution is at least half the cost of our algorithm. So our algorithm is 2-competitive.

5 Lower Bound for the Greedy Algorithm

In this section, we show that the competitive ratio of our algorithm can be close to 2.

▶ **Lemma 2.** *Given any constant $c < 2$, there is an instance for which our algorithm has competitive ratio at most c.*

Proof. We consider a bit string of length n, where n is large enough. We also fix a parameter k (which is much smaller than n). Our instance is required to maintain k covers. Initially all bits are 0. Now suppose the first $k + 1$ rounds, the bits which become 1 are at positions $0, D, 2D, \ldots, kD$ for some parameter D (assume $n > kD$). Our greedy algorithm must have some interval which contains two of these 1's. Assume that one of the greedy covers includes $[iD, (i + 1)D]$ for some i. Now, we pick a j different from i, and in the next $D - 1$ rounds, we set all the bits in the positions $[jD, (j + 1)D]$ to 1. So the greedy algorithm must have

cost at least $2D$. However, the optimal off-line algorithm will pay at most $D + K$: it will have one of the covers as $[jD, (j + 1)D]$ and the remaining 1-bits can be covered by $k - 1$ intervals of unit length. Now if D is much larger than K, then we get the desired result. ◀

References

1 Hamid A. Toussi, Ahmed Khademzadeh: Improving bit-vector representation of points-to sets using class hierarchy CoRR abs/1108.2683: (2011)
2 Compilers: Principles, Techniques, and Tools Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, Addison Wesley 1986.
3 Moses Charikar, Rina Panigrahy: Clustering to Minimize the Sum of Cluster Diameters.
4 S. R. Doddi, M. V. Marathe, S. S. Ravi, D. S. Taylor, P. Widmayer: Approximation algorithms for clustering to minimize the sum of diameters. Proc. 7th Scandinavian Workshop on Algorithm Theory, 2000: 237-250.
5 Dan Gusfield: Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press (1997).
6 Mark Nelson, Jean-Loup Gailly: The Data Compression Book. John Wiley & Sons.
7 Rishi Rakesh Sinha, Marianne Winslett: Multi-resolution bitmap indexes for scientific data. ACM Trans. Database Syst. 32(3): 16 (2007)
8 Rishi Rakesh Sinha, Soumyadeb Mitra, Marianne Winslett: Bitmap indexes for large scientific data sets: a case study. IPDPS 2006
9 Preeti Ranjan Panda, M. Balakrishnan, Anant Vishnoi: Compressing Cache State for Postsilicon Processor Debug. IEEE Trans. Computers 60(4): 484-497 (2011)
10 Preeti Ranjan Panda, Anant Vishnoi, M. Balakrishnan: Enhancing post-silicon processor debug with Incremental Cache state Dumping. VLSI-SoC 2010: 55-60
11 Anant Vishnoi, Preeti Ranjan Panda, M. Balakrishnan: Online cache state dumping for processor debug. DAC 2009: 358-363