The complexity of the minimum k-cover problem

October 6, 2003

Richard Cole
Computer Science Department, New York University
New York, NY 10012-1185, USA
e-mail: `cole@cs.nyu.edu`

Costas S. Iliopoulos, Manal Mohamed
Algorithm Design Group, Department of Computer Science
King's College London, London WC2R 2LS, England
e-mail: `{csi,manal}@dcs.kcl.ac.uk`

W. F. Smyth
Algorithms Research Group, Department of Computing & Software
McMaster University, Hamilton ON L8S 4K1, Canada
e-mail: `smyth@mcmaster.ca`

Department of Computing, Curtin University, Perth WA 6845, Australia

Abstract

We consider the problem of determining the minimum cardinality collection of substrings, each of given length $k \geq 2$, that "cover" a given string $\boldsymbol{x}$ of length n. We show first, by reduction from 3-SAT, that this minimum k-cover problem is NP-complete. Then we show further that minimum k-cover may also be thought of as a special case of the vertex cover problem, hence that there exists an efficient greedy algorithm to compute a k-cover of $\boldsymbol{x}$ whose cardinality is at most twice the minimum. In conclusion we describe two open problems that arise naturally out of our research.

1 Introduction

Given a nonempty string $\boldsymbol{x}$ of length n together with a set

$$U_\nu = \{\boldsymbol{u_1}, \boldsymbol{u_2}, \ldots, \boldsymbol{u_\nu}\}$$

of ν substrings of $\boldsymbol{x}$, each of length k, we say that U_ν is a *k-cover* of $\boldsymbol{x}$ if and only if every position of $\boldsymbol{x}$ lies within an occurrence of some $\boldsymbol{u_i}$, $1 \leq i \leq \nu$. If ν is the minimum integer for which such a set U_ν exists, then U_ν is said to be a *minimum k-cover* of $\boldsymbol{x}$. For $k = 1$, U_ν can be a k-cover only if it is the set of distinct letters in $\boldsymbol{x}$, while for $k \geq n/2$, the minimum k-cover size ν can only be 1 or 2, depending on whether or not $\boldsymbol{x}$ has a border of length k. Thus to avoid trivialities, we suppose throughout that $1 < k < n/2$. Observe that in a minimum k-cover, $1 \leq \nu \leq \lceil n/k \rceil$.

The idea of a *cover*, or single covering substring, first arose as a generalization of periodicity, called *quasi-periodicity* [AFI91]. Rather surprisingly, it turned out [LS02] that all the quasiperiods of every prefix of a string could be computed in linear time, just as for regular periods. Thus the computation of k-covers constitutes a further generalization of periodicity, measured by the size ν of the minimum k-cover.

In this paper we consider two closely-related problems:

(P1) for given $\boldsymbol{x}$, k and ν, *decide* whether there exists a k-cover of $\boldsymbol{x}$ of cardinality ν;

(P2) *compute* a minimum k-cover of $\boldsymbol{x}$.

For $\nu = 1$ Problem (P1) can be solved in $\Theta(n)$ time simply by computing all the covers of $\boldsymbol{x}$ [LS02] while checking to determine whether one of them is of length k. Where $\nu \neq 1$, we show in Section 2 by reduction from 3-SAT that (P1) is NP-complete for every $k \geq 2$, refining and generalizing the result given in [CIMSY03]. In Section 3 we go on to show that (P1) is a special case of the vertex cover problem, hence that the solution to (P2) can be approximated to within a factor of 2 by an efficient greedy algorithm. Section 4 discusses the significance of these results and states some open problems.

2 Minimum k-Cover is NP-Complete

In this section we show how to construct a string $\boldsymbol{x}$ whose minimum k-cover can be computed only by solving an instance of 3-SAT, well-known to be NP-complete [C71]. For simplicity we describe the construction in detail for $k = 2$, then indicate how it can be extended to arbitrary $k > 2$.

Recall that for 3-SAT we are given a set of clauses

$$C = \{c_1, c_2, \ldots, c_s\},$$

over a finite set of Boolean variables

$$V = \{v_1, v_2, \ldots, v_r\},$$

each of which assumes one of the values `T` or `F`, where for every $j \in 1..s$,

$$c_j = v_{i_1} \vee v_{i_2} \vee v_{i_3} \tag{1}$$

is a disjunction of variables $v_{i_\ell} \in V$, $\ell = 1, 2, 3$. Our task is to determine whether or not there exists an assignment of `T/F` to the variables of V such that every $c_j =$ `T` (in other words, such that at least one variable in each c_j equals `T`).

The construction of $\boldsymbol{x}$ is achieved in three main steps:

(S1) First, corresponding to each variable $v_i \in V$, $i = 1, 2, ..., r$, we form a string $\boldsymbol{w_i}$ that has exactly two minimum k-covers with exactly one k-string in common: one of these corresponds to the choice $v_i = \mathtt{T}$, the other to $v_i = \mathtt{F}$.

(S2) Next, corresponding to each clause $c_j \in C$, $j = 1, 2, \ldots, s$, we construct a string $\boldsymbol{d_j}$ with the property that for $c_j = \mathtt{T}$, the minimum k-cover of $\boldsymbol{d_j}$ has cardinality one less than that corresponding to $c_j = \mathtt{F}$.

(S3) Finally, we form $\boldsymbol{x}$ from the $\boldsymbol{w_i}$ and the $\boldsymbol{d_j}$, and observe that we cannot determine the size ν of the minimum k-cover of $\boldsymbol{x}$ without solving the 3-SAT problem (V, C).

We begin (S1) by defining, for each $i \in 1..r$, the following substrings:

`[a]`	$\#_i a_i \$_i z_i \pounds_i a_i \#_i$	`[b]`	$\&_i b_i \$_i \overline{z_i} \pounds_i b_i \&_i$
`[c]`	$\#_i \&_i$	`[d]`	$\&_i \#_i$
`[e]`	$\#_i$	`[f]`	$\&_i$.

We form $\boldsymbol{w_i}$ by concatenating these six substrings with "padding" at the beginning and end of each one consisting of $2k-1$ occurrences of the letter $*$. This forces $*^k$ to be one of the strings in the k-cover of $\boldsymbol{w_i}$. Although we present the argument here for $k = 2$, we make use of the symbol k in certain contexts below in order to make clearer the extension to the general case. Note that for $k = 2$:

- At least 8 substrings of length k are required to cover both `[a]` and `[b]`, since each is of length $3k + 1$ and together they contain no substring of length k more than once.

- In order to cover `[c-f]` with a minimum of three k-strings, either

  $$C_1 = (*^{k-1}\#_i,\ \&_i *^{k-1},\ \&_i \#_i *^{k-2}) \ \text{ or } \ C_2 = (\#_i *^{k-1},\ *^{k-1}\&_i,\ \#_i \&_i *^{k-2})$$

  must be chosen — any other choice yields more than three k-strings. Note that in each of these sets, the k-string that includes *both* $\&_i$ and $\#_i$ cannot be used to cover `[a-b]`; thus at most two string in each set can be reused.

- It follows that the choice $z_i \pounds_i^{k-1}$ in `[a]` is incompatible with the choice $\overline{z_i} \pounds_i^{k-1}$ in `[b]`; thus 8 substrings will cover `[a]` and `[b]` only if exactly one of the pairs $(*^{k-1}\#_i, \&_i *^{k-1})$ or $(\#_i *^{k-1}, *^{k-1}\&_i)$ is chosen.

We conclude that for $k = 2$ there are exactly two minimum k-covers of $\boldsymbol{w_i}$:

$$\{*^{k-1}\#_i,\ a_i^{k-1}\$_i,\ z_i \pounds_i^{k-1},\ a_i^{k-1}\#_i,\ \&_i b_i^{k-1},\ \$_i \overline{z_i} \pounds_i^{k-2},\ \pounds_i b_i^{k-1},\ \&_i *^{k-1},\ \&_i \#_i *^{k-2},\ *^k\}, \quad (2)$$

$$\{\#_i a_i^{k-1},\ \$_i z_i \pounds_i^{k-2},\ \pounds_i a_i^{k-1},\ \#_i *^{k-1},\ *^{k-1}\&_i,\ b_i^{k-1}\$_i,\ \overline{z_i} \pounds_i^{k-1},\ b_i^{k-1}\&_i,\ \#_i \&_i *^{k-2},\ *^k\}, \quad (3)$$

each of cardinality 10. We select (2) if and only if $v_i = \mathtt{T}$, (3) if and only if $v_i = \mathtt{F}$. Note that of these k-strings, exactly one $(*^k)$ will occur in more than one $\boldsymbol{w_i}$; thus each $\boldsymbol{w_i}$ gives rise to exactly 9 unique distinct k-strings.

Step (S2) deals with the construction of 15 substrings that correspond to each clause (1) of C. As in (S1), these substrings are concatenated with padding $*^{2k-1}$ around them in order to form the string $\boldsymbol{d_j}$. Some of the letters in these substrings correspond to the variables v_{i_ℓ}, $\ell = 1, 2, 3$, that occur in clause c_j; specifically, the letters

$$\$_{i_\ell},\ \pounds_{i_\ell},\ z_{i_\ell}.$$

In addition, new letters are introduced that correspond to the clause C itself:

$$d_{j\ell}, h_{j\ell},\ \ell = 1, 2, 3.$$

The following substrings are formed:

`[g]` $\$_{i_1} z_{i_1} \pounds_{i_1} h_{j1}$	`[h]` $\$_{i_2} z_{i_2} \pounds_{i_2} h_{j2}$	`[i]` $\$_{i_3} z_{i_3} \pounds_{i_3} h_{j3}$
`[j]` $\pounds_{i_1} h_{j1} d_{j1} \pounds_{i_2} h_{j2}$	`[k]` $\pounds_{i_2} h_{j2} d_{j2} \pounds_{i_3} h_{j3}$	`[l]` $\pounds_{i_3} h_{j3} d_{j3} \pounds_{i_1} h_{j1}$
`[m]` $\$_{i_1}$	`[n]` $\$_{i_2}$	`[o]` $\$_{i_3}$
`[p]` $\pounds_{i_1}$	`[q]` $\pounds_{i_2}$	`[r]` $\pounds_{i_3}$
`[s]` h_{j1}	`[t]` h_{j2}	`[u]` h_{j3}

For $k = 2$ we make the following observations:

- In order to cover substrings `[m-u]`, we should choose the nine covering k-strings

  $$*^{k-1}\$_{i_\ell},\ *\pounds_{i_\ell}^{k-1},\ h_{j\ell}*^{k-1},\ \ell = 1, 2, 3,$$

  because they are all reusable to partially cover substrings `[g-l]`. Note that neither $*^{k-1}\$_{i_\ell}$ nor $*\pounds_{i_\ell}^{k-1}$ is an element of the minimum k-cover of $\boldsymbol{w_{i_\ell}}$, regardless of whether or not $v_{i_\ell} = \mathtt{T}$.

- If $c_j = \mathtt{F}$, then every $v_{i_\ell} = \mathtt{F}$, and so for every $\ell = 1, 2, 3$,

  $$z_{i_\ell}\pounds_{i_\ell}^{k-1}$$

  cannot have been chosen in the k-cover of $\boldsymbol{w_{i_\ell}}$ (nor can it be chosen now), while

  $$\$_{i_\ell} z_{i_\ell} \pounds_{i_\ell}^{k-2}$$

  must already have been chosen. Thus in order to cover `[g-i]`, we must choose

  $$\pounds_{i_\ell}^{k-1} h_{j\ell},\ \ell = 1, 2, 3, \tag{4}$$

  and then in order to provide a minimum cover of `[j-l]`, we must in addition choose three k-strings to cover the $d_{j\ell}$; for example,

  $$d_{j\ell}\pounds_{i_\ell}^{k-1},\ \ell = 1, 2, 3.$$

  Thus whenever $c_j = \mathtt{F}$, six additional k-strings are required to cover `[g-l]`.

- On the other hand, if $c_j = \texttt{T}$, then we may suppose without loss of generality that $v_{i_1} = \texttt{T}$, hence that $z_{i_1}£_{i_1}^{k-1}$ was already chosen, thus together with $*^{k-1}\$_{i_1}$ and $h_{j1}*^{k-1}$ providing a cover for `[g]`. Note then that in order to cover `[h-l]`, it suffices to choose only five additional k-strings

$$£_{i_2}^{k-1}h_{j2},\ £_{i_3}^{k-1}h_{j3},\ h_{j1}d_{j1},\ h_{j2}d_{j2},\ d_{j3}£_{i_1}^{k-1} \tag{5}$$

  regardless of whether or not $v_{i_2} = \texttt{T}$ or $v_{i_3} = \texttt{T}$.

- The (five or six) new k-strings required for the minimum cover of $\boldsymbol{d_j}$ all include either $h_{j\ell}$ or $d_{j\ell}$ for some ℓ, and thus their number cannot be reduced by adding k-strings to the cover of any $\boldsymbol{w_i}$. Furthermore, the minimum number of k-strings required to cover any $\boldsymbol{w_i}$ is unaffected by the number of k-strings used to cover $\boldsymbol{d_j}$.

In view of these comments, we conclude that whenever $c_j = \texttt{T}$, we can cover the corresponding string $\boldsymbol{d_j}$ with $9+5 = 14$ k-strings in addition to those already chosen to cover the strings $\boldsymbol{w_i}$; whereas whenever $c_j = \texttt{F}$, $9+6 = 15$ additional k-strings will be required. However, independent of the value of c_j, of the k-strings required for clauses $\boldsymbol{d_j}$, six are keyed to the $\boldsymbol{w_i}$ and may recur in other clauses; specifically, $*^{k-1}\$_{i_\ell}, *£_{i_\ell}^{k-1}$, $\ell = 1,2,3$. Altogether, over all $j = 1,2,\ldots,s$, r' disjoint sets of these six k-strings will occur for some $r' \leq r$.

Finally Step (S3) constructs $\boldsymbol{x}$ by concatenating the strings $\boldsymbol{w_i}$, $i = 1,2,\ldots,r$, and $\boldsymbol{d_j}$, $j = 1,2,\ldots,s$, and then considers the question

> Does $\boldsymbol{x}$ have a k-cover of cardinality $\nu = 9r + 6r' + 8s + 1$?

As we have seen, such a cover exists if and only if there exists an assignment of the Boolean variables V such that C is satisfied — in other words, if and only if an instance of 3-SAT is solved. Thus the minimum 2-cover problem is NP-complete.

To extend this result to arbitrary k is straightforward: all that is required is to replace the letters

$$a_i, b_i, £_i$$

by unique substrings of length $k-1$; for example, by

$$a_i^{k-1}, b_i^{k-1}, £_i^{k-1},$$

respectively. Then essentially the same argument can be used, with some uninteresting variation possible in the choice of $£_{i_\ell}^{k-1}h_{j\ell}$ in (4) and in the use of $k-2$ occurrences of $£_{i_\ell}$ to pad $h_{j1}d_{j1}$ and $h_{j2}d_{j2}$ in (5).

We conjecture that NP-completeness can similarly be proved for an alphabet of reduced size and sufficiently large k by encoding the letters as substrings on $\{0,1\}$, for example. We state our result formally:

Theorem 2.1 *Over strings $\boldsymbol{x} = \boldsymbol{x}[1..n]$ and integers $k \in 2..\frac{n}{2}-1$, $\nu \geq 2$, Problem (P1) is NP-complete.* $\square$

3 A Special Case of Vertex Cover

In [IS98] an algorithm was described that was claimed to compute a minimum k-cover of a string of length n in $O(n^2(n-k))$ time, a result that in view of Section 2 is not likely to be correct. In fact, algorithms that compute an approximate k-cover in $O(n \log n)$ time were described in [Y00] and shown in certain cases to compute a smaller k-cover than the algorithm of [IS98]; as explained in [Y00], this led to a closer examination of [IS98] and to discovery of an error in the algorithm described there. In this section we put these results in perspective by showing that Problem (P1) is actually a special case of the vertex cover problem.

A *vertex cover* of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$, then either $u \in V'$ or $v \in V'$. The *vertex cover problem* (VCP) asks us to determine, for given graph G and integer ν, whether there exists a vertex cover V' of G such that $|V'| = \nu$. VCP is well-known to be NP-complete [GJ79], but several $O(|V| + |E|)$-time greedy algorithms have been described that compute a vertex cover of G whose cardinality is at most twice that of the minimum-cardinality vertex cover of G. The first such algorithm discovered, due to Gavril ([GJ79], p. 134), is a straightforward application of maximal matching; another [S82] is based on depth-first search.

It is convenient here to define a relaxed version of VCP. We say that a *vertex semi-cover* of $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$, then one of the following conditions holds:

(C1) $u \in V'$;

(C2) $v \in V'$;

(C3) there exist $w_u \in V'$ such that $(w_u, u) \in E$ **and** $w_v \in V'$ such that $(w_v, v) \in E$.

The *relaxed VCP* (RVCP) asks, for given G and ν, whether there exists a vertex semi-cover of G such that $|V'| = \nu$. Of course a vertex semi-cover will exist whenever a vertex cover exists, and so RVCP is in a sense easier than VCP.

Suppose now that a string $\boldsymbol{x} = \boldsymbol{x}[1..n]$ and an integer k are given. We initialize a graph $G_{\boldsymbol{x}} = (V, E)$, where $V = \{1, 2, \ldots, n\}$ and $E = \emptyset$. At every position $i = 1, 2, \ldots, n$ of $\boldsymbol{x}$, determine the leftmost position $i' \leq i$ in $\boldsymbol{x}$ at which the substring $\boldsymbol{u} = \boldsymbol{x}[i..i+k-1]$ occurs: if $i' = i$, compute

$$E \leftarrow E \cup \big\{(i', i+1), (i', i+2), \ldots, (i', i+k-1)\big\};$$

while if $i' < i$, compute

$$E \leftarrow E \cup \big\{(i', i), (i', i+1), \ldots, (i', i+k-1)\big\}.$$

For example, given $\boldsymbol{x} = abaab$ and $k = 2$, the corresponding graph $G_{\boldsymbol{x}}$ will be given by

$$V = \{1, 2, 3, 4, 5\}, \quad E = \big\{(1, 2), (1, 4), (1, 5), (2, 3), (3, 4)\big\}.$$

Thus the position of the leftmost occurrence of every distinct k-string $\boldsymbol{u}$ of $\boldsymbol{x}$ will be adjacent to every other position covered by $\boldsymbol{u}$. If the alphabet of $\boldsymbol{x}$ is ordered, an algorithm to compute $G_{\boldsymbol{x}}$ from $\boldsymbol{x}$ can be implemented in $O(kn \log n)$ time using a straightforward approach, somewhat faster using a suffix tree to sort the k-strings. Let us designate this algorithm $A(\boldsymbol{x}, k)$.

Consider a set $U_\nu = \{\boldsymbol{u_1}, \boldsymbol{u_2}, \ldots, \boldsymbol{u_\nu}\}$ of distinct substrings of $\boldsymbol{x}$, each of length k, and let j_i denote the leftmost position in $\boldsymbol{x}$ at which $\boldsymbol{u_i}$ occurs, $i = 1, 2, \ldots, \nu$. Let $J_\nu = \{j_1, j_2, \ldots, j_\nu\}$. Now observe that if in fact U_ν is a k-cover of $\boldsymbol{x}$, a vertex semi-cover of $G_{\boldsymbol{x}}$ is given by $V' = J_\nu$: for every $(u, v) \in E$, if neither (C1) nor (C2) is true, then by the construction of E, (C3) must be true. Conversely, if $V' = \{j_1, j_2, \ldots, j_\nu\}$ is a vertex semi-cover of a graph $G_{\boldsymbol{x}}$ formed using $A(\boldsymbol{x}, k)$ for some string $\boldsymbol{x}$ and some $k \geq 2$, then

$$U_1 = \big\{\boldsymbol{x}[j_1..j_1{+}k{-}1], \boldsymbol{x}[j_2..j_2{+}k{-}1], \ldots, \boldsymbol{x}[j_\nu..j_\nu{+}k{-}1]\big\}$$

must be a k-cover of $\boldsymbol{x}$. Thus:

Theorem 3.1 *Problem (P1) is equivalent to RVCP restricted to graphs $G_{\boldsymbol{x}}$ formed according to Algorithm $A(\boldsymbol{x}, k)$.*

In other words, (P1) is a special case of RVCP, hence of VCP. Therefore greedy algorithms such as those described in [GJ79, S82] can be adapted to compute a k-cover of cardinality at most twice the minimum.

We make the following observations:

- Since by Theorem 2.1 (P1) is NP-complete for every fixed choice of $k \geq 2$, therefore by Theorem 3.1 so is RVCP restricted to graphs computed using Algorithm $A(\boldsymbol{x}, k)$.
- Suppose that the substrings $\boldsymbol{u_i}$ of U_ν are no longer constrained to have length k but may instead have any positive length $\leq k$. This change leads to the idea of a **$\leq k$-cover**, say U'_ν, and a corresponding problem (P1′). The graph $G_{\boldsymbol{x}}$ can be formed in the same way as before by executing Algorithm $A(\boldsymbol{x}, k')$ for every $k' \in 2..k$, while eliminating any duplicate edges that result during the construction process. If we call the new algorithm $A'(\boldsymbol{x}, k)$, then an analogue to Theorem 3.1 holds: (P1′) is equivalent to RVCP restricted to graphs formed according to $A'(\boldsymbol{x}, k)$.
- Observe that, by virtue of the way in which edges are formed in $G_{\boldsymbol{x}}$, all the edges must be expressed in the form (u, v), $u < v$. Thus if we interpret the edges as arcs in a directed graph $\overline{G_{\boldsymbol{x}}}$, the set $\mathcal{G}$ of all such digraphs $\overline{G_{\boldsymbol{x}}}$ is therefore a subset of the set of all acyclic digraphs. $\mathcal{G}$ is moreover a proper subset since none of its members can be a tournament.

4 Conclusions & Open Problems

We have shown that for $k \geq 2$, the k-cover problem (P1) is NP-complete and moreover equivalent to a restriction of RVCP, so that a greedy algorithm can be used to approximate a minimum k-cover to within a factor of two. Interesting questions remain:

(Q1) The set $\mathcal{G}$ of graphs $G_{\boldsymbol{x}}$ formed by Algorithm $A(\boldsymbol{x}, k)$ in some sense describes the structure of all strings. To our knowledge these graphs have not previously been reported in the literature. Can the graphs of $\mathcal{G}$ be characterized in another way? What are their defining properties?

(Q2) The NP-completeness proof given in Section 2 is based upon strings whose length n is a function of three parameters: k (the length of the covering substrings), r (the number of variables in the corresponding 3-SAT problem), and s (the corresponding number of clauses). A short calculation shows that in fact

$$n = (18k+7)r + (42k-3)s + (2k-1),$$

while as we have seen the minimum cover size

$$\nu = 9r + 6r' + 8s + 1,\ r' \leq r.$$

Let us call the ratio $\gamma_k = n/\nu k$ the *k-coverability* of the string $\boldsymbol{x}[1..n]$; observe that γ_k has as an upper bound the average number of occurrences in $\boldsymbol{x}$ of the strings in the minimum k-cover. Since $\nu \leq 15r+8s+1$, we see then that for the class of strings constructed in Section 2, $\gamma_k > 6/5$; in other words, the strings in the k-cover occur on average somewhat frequently in $\boldsymbol{x}$. What happens when $\gamma_k \leq 6/5$? Can we find a polynomial-time algorithm to compute a minimum k-cover given that γ_k falls below a certain threshold? For "most" strings and some sufficiently large k, we expect that $\nu = \lceil n/k \rceil$, so that $\gamma_k \approx 1$; thus such an algorithm would in fact handle most of the cases that arise.

References

[AFI91] Alberto Apostolico, Martin Farach & Costas S. Iliopoulos, **Optimal superprimitivity testing for strings**, *Infor. Proc. Letters 39* (1991) 17–20.

[CIMSY03] Richard Cole, Costas S. Iliopoulos, Manal Mohamed, W. F. Smyth & Lu Yang, **Computing the minimum k-cover of a string**, *Proc. Prague Stringology Conf. '03* (2003) to appear.

[C71] Stephen A. Cook, **The complexity of theorem-proving procedures**, *Proc. Third Annual ACM Symp. on Theory of Computing* (1971) 151–158.

[GJ79] Michael R. Garey & David S. Johnson, *Computing and intractability: a guide to the theory of NP-completeness*, Freeman (1979).

[IS98] Costas S. Iliopoulos & W. F. Smyth, **On-line algorithms for k-covering**, *Proc. Ninth Australasian Workshop on Combinatorial Algorithms* (1998) 107–116.

[LS02] Yin Li & W. F. Smyth, **Computing the cover array in linear time**, *Algorithmica 32–1* (2002) 95–106.

[S82] Carla D. Savage, **Depth first search and the vertex cover problem**, *Inf. Proc. Lett. 14* (1982) 233–237.

[Y00] Lu Yang, *Computing the Minimum k-Cover of a String*, M. Sc. thesis, McMaster University (2000) 86 pp.