Refined Algorithms for Hitting Many Intervals

Peter Damaschke*
Department of Computer Science and Engineering
Chalmers University, 41296 Göteborg, Sweden
ptr@chalmers.se

Abstract

Within a study of scheduling problems with gaps, Chrobak et al. (CIAC 2015) have shown how to find γ points on the line that hit a maximum number of intervals, in a given family of n intervals, in $O(\gamma n^2)$ time. The problem is equivalent to finding γ cliques in an interval graph that cover a maximum number of distinct vertices. We give refined algorithms that run faster if the interval graph of the family is sparse.

Keywords: scheduling with gaps, hitting set, interval graph, sparse graph, dynamic programming

1 Introduction

1.1 Problem Setting

This paper is concerned with the following problem.

Partial Interval Hitting Set

We are given n closed intervals $I_j = [r_j, d_j]$ $(j = 1, \ldots, n)$ on the real line, and a positive integer γ. We want to determine a set H of γ points that together hit a maximum number of intervals I_j. Here, a point q is said to *hit* an interval I if and only if $q \in I$.

In [3] this problem is considered as the continuous version of a certain scheduling problem: scheduling short (unit-time) jobs with release times r_j and deadlines d_j, where the goal is to maximize the throughput, that is, the number of jobs actually done, while keeping the number of gaps between busy periods limited to $\gamma - 1$. Limiting the number of gaps can be motivated, for instance, by the costs for switching the machines on and off. Another nice application of Partial Interval Hitting Set is storing various goods (food, chemicals, etc.) at prescribed temperatures. Suppose that the jth good needs a temperature within the interval I_j, and γ cooling rooms are available whose temperatures can be freely chosen. The aim is to store as many goods as possible in these rooms (assuming that the rooms have enough capacity, and no other restrictions apply which rule out the storage of certain goods in the same room).

Definition 1. *The intersection graph of a family of n intervals is its* interval graph. *It has n vertices representing the given intervals, and two vertices are adjacent if and only if the corresponding intervals intersect. Let m denote the number of edges of this interval graph.*

*Tel. 0046 31 772 5405, Fax 0046 31 772 3663

The intervals hit by any fixed point form a complete subgraph of the interval graph (that is, any two vertices of the subgraph are adjacent). The converse is also true, by the Helly property of intervals: For the vertex set C of any complete subgraph of the interval graph, there exists a point that hits all intervals in C. Throughout this paper, a *clique* C is a *maximal* complete subgraph, that is, C is not a subset of another complete subgraph. (Usually all complete subgraphs are named cliques, but for us it will be more convenient to reserve this term for the maximal ones. However, note that $|C|$ is not necessarily maximum.) Our problem is equivalent to covering a maximum number of vertices of an interval graph by γ of its cliques.

For Partial Interval Hitting Set we can assume that no vertex (interval) is isolated, in the sense that it intersects no other interval of the family. Isolated intervals are a trivial matter: Since one point can hit only one isolated interval, for an optimal solution with γ points we need to consider them only when all other intervals are already hit by some of the points. Without isolated intervals, $O(m+n)$ simplifies to $O(m)$.

1.2 Related Work

Hitting set problems for general set families and their dual problems, called covering problems, are among the fundamental NP-hard problems, and they are also well studied from different angles like approximability [4, 7] and parameterized complexity, see [12].

Problem versions where the solution size is prescribed and the aim is to hit or cover as many elements as possible have been considered for special cases, e.g., for the vertex cover problem [11]. General approximation guarantees for greedy partial set cover are given in [6]. In another setting, the fraction of objects to hit or to cover is part of the input, and the goal is to minimize the solution size, e.g., in [5].

For geometrically restricted families like intervals, problems are typically solvable in polynomial time, yet the interesting matter is to get small time bounds. The partial interval set cover problem studied in [5] can be solved in quadratic time by dynamic programming, however, remarkably, this may be too slow for huge data, therefore faster approximation schemes are developed even for this polynomial-time solvable problem, in the same paper. (Similarly, polynomial-time solvable problems allow faster exact algorithms if some input parameter is small; one example is string editing if the edit distance is small [13].)

The problem of covering *all* vertices of a graph by a minimum number of cliques is solvable in $O(n)$ time even on circular-arc graphs [10], a class which includes interval graphs. However, the problem of covering a maximum number of vertices by a fixed number of cliques behaves differently, and time bounds do not simply carry over.

As mentioned above, our problem can also be rephrased as finding a union of γ cliques with a maximum number of vertices. In the complement graph, cliques become independent sets, hence the problem translates into the maximum γ-colorable subgraph problem. Algorithms for this problem are known for comparability graphs and their complements (a class that includes interval graphs), however their time bounds are at least quadratic in n; see [8] and the references to earlier work. For another generalization of interval graphs, the class of chordal graphs, the maximum γ-colorable subgraph problem is solvable in polynomial time for fixed γ, and in interval graphs it is polynomial also when γ is part of the intput [14]. A linear-time algorithm for interval graphs was found later [2]. However, this is the "complement" to our problem. Both problems are NP-complete for split graphs, hence for

chordal graphs and their complements [14]. To the best of our knowledge, our refined time bounds for sparse interval graphs have not been discovered by other related work either.

Finally, Partial Interval Hitting Set can be solved in $O(\gamma n^2)$ time; this is one of many results in [3].

1.3 Our Contributions

We first present an algorithm that solves Partial Interval Hitting Set in $O(\gamma m)$ time. We assume that the interval endpoints are sorted, otherwise an $O(n \log n)$ term must be added for sorting. Since m can be $\Theta(n^2)$, of course, this does not improve upon the known $O(\gamma n^2)$ worst-case bound in [3], however it is a major improvement for sparse interval graphs. Our method is just standard dynamic programming techniques, but we organize the computations differently from [3] where a simple dynamic programming in the order of right endpoints is done. That is, we give a different algorithm. Its time bound is obtained by counting arguments.

We argue that sparse interval graphs are typical in applications like scheduling: If the stream of arriving jobs has a steady-state behaviour, with respect to central parameters like average job length and average workload, then obviously m grows only linear in n in the long run. Very sparse interval graphs as instances, with short intervals spread out over a long line segment, can have the additional feature that γ largest cliques in total do not overlap much, especially if some intervals form clusters while all others have few intersections. (A large number of jobs may simultaneously appear at specific peak times.)

Let $s(\gamma)$ denote the sum of the sizes of the γ largest cliques. (Note that shared intervals are here counted multiple times.) In the special case that the γ largest cliques are disjoint, they form an optimal solution that hits $s(\gamma)$ intervals, that is, a trivial greedy rule succeeds. Starting from this observation we study the less restrictive case where these cliques may share a small number of common vertices. More precisely, we ask for a solution where γ points hit $s(\gamma) - \lambda$ intervals, where λ is some small parameter. By a modification of our dynamic programming scheme we compute an optimal solution to Partial Interval Hitting Set in $O(m+\gamma\lambda n)$ time. By a more sophisticated dynamic programming scheme we can solve the same problem in $O(m+\gamma^2\lambda^2 \log n)$ time, by restricting the candidate points for solutions to a set whose size does not depend on n. Either algorithm can be faster, depending on the parameter values. Such results may be seen as analogues of parameterization and kernelization results in the polynomial-time "world" rather than for NP-hard problems.

2 Preliminaries

For two points $p < q$ on the real line we say that p is to the left of q.

Some assumptions on the given intervals can be made without loss of generality: All $2n$ endpoints r_j and d_j are distinct, and the endpoints are sorted (otherwise we can sort them using $O(n \log n)$ extra time). The r_j are indexed such that $r_1 < \ldots < r_n$. Moreover, it suffices to restrict solutions H to subsets of $\{r_1, \ldots, r_n\}$. (The argument is simple: In any solution, any point which is not an r_j can be moved to the left until the left endpoint of some interval is met.) Moreover, only points r_b that hit a clique, i.e., a maximal complete subgraph, need to be considered for an optimal solution. (Otherwise, some other point $r_{b'}$ hits further intervals in addition to those hit by r_b, and we can replace r_b with $r_{b'}$ in H.)

Definition 2. *Let $I(b)$ denote the set of all intervals hit by r_b.*

In particular, every clique is some of the $I(b)$. (But the converse does not hold. Note that $I(b)$ is a clique if and only if the next interval endpoint to the right of r_b is some right endpoint d_j. We will not need this property, we only mention it for the sake for completeness.) Sometimes we loosely identify a point r_b and its set $I(b)$ if this causes no confusion. For formal reasons we also insert a dummy point $r_0 < r_1$. As usual, let ω denote the number of vertices of a largest clique.

Lemma 3. *We have $m = O(\omega n)$ and $\sum_{b=1}^{n} |I(b)| = O(m)$. Moreover, in $O(m)$ time we can prepare all sets $I(b)$, in such a way that the intervals in every $I(b)$ are sorted by their left endpoints.*

Similar observations were known before, even for chordal graphs [9]. We omit the straightforward proof. The first part can be shown by counting arguments, and the $O(m)$-time algorithm only loops through the sorted sequence of the r_b.

Definition 4. *Consider any solution H to* Partial Interval Hitting Set*, where the points have been inserted one by one in a specific order (for instance, from left to right, but it could be any order). Upon insertion of $r_b \in H$ we say that an interval in $I(b)$ is a* new interval*, if it was not already hit by some earlier point of H in the specified order.*

3 Time Complexity Proportional to the Edge Number

Theorem 5. *Given a family of n intervals whose interval graph has m edges, we can determine γ points that together hit a maximum number of intervals, in $O(\gamma m)$ time.*

Proof. First we preprocess the given interval family according to Lemma 3 in $O(m)$ time. Remember that the intervals in every $I(b)$ are sorted by their left endpoints.

We define $h(g, a)$ to be the largest number of intervals that can be hit by $g \leq \gamma$ points, each of which is less than or equal to r_a. In particular, using the dummy point r_0 we define $h(g, 0) := 0$ for all g. Trivially, $h(0, a) = 0$ holds for all a. Note that $O(\gamma n)$ pairs (g, a) exist, which is in $O(\gamma m)$.

For any fixed b^*, suppose that all $h(g^*, a)$, $a < b^*$, are already computed. Below we discuss how we compute all values $h(g^*, b^*)$.

Consider any optimal solution H and any point $r_b \in H$. Let g be the number of points in H to the left of r_b, let k be the number of new intervals in $I(b)$, let r_c be the left endpoint of the first new interval in $I(b)$, and let $a := c - 1$. (In other words, the first new interval is the kth last in the sorted set $I(b)$.) Then the $g + 1$ points in H up to r_b together hit $h(g, a) + k$ intervals. In fact, the contribution from the first g points is $h(g, a)$, since otherwise we could replace them with g points to the left of r_a that hit more intervals, moreover we could still add the k new intervals from $I(b)$, and the rest of H to the right, contradicting the optimality of H.

This shows that $h(g + 1, b)$ equals the maximum of $h(g + 1, b - 1)$ and of all $|I(b)|$ sums $h(g, a) + k$ that we can build as above. The term $h(g + 1, b - 1)$ accounts for the case that by using r_b we cannot hit more intervals than by using only points to the left of r_b. The summations and comparisons are done for all $g < \gamma$. Now the time bound $O(\gamma m)$ follows from the bound $\sum_{b=1}^{n} |I(b)| = O(m)$ in Lemma 3.

An optimal solution is recovered from the $h(g, b)$ values by backtracing, as outlined below. Clearly, $h(g, b)$ is a monotone function in both arguments. We start at $h(\gamma, n)$ which is the number of intervals hit by an optimal solution. In a generic step we are at some entry $h(g, b)$. We decrease b (do iteratively $b := b - 1$) until we find an argument b where $h(g, b)$ decreases for the first time, that is, $h(g, b - 1) < h(g, b)$. Then this b is the index of the leftmost point r_b where a solution with g points, still hitting the same number of intervals, can end. Thus we also know that r_b was the gth point in our optimal solution. We check which of the left endpoints of intervals in $I(b)$ gave the maximum sum $h(g, a) + k$. Say, it was r_c, ties are broken arbitrarily. We set $b := c - 1$ and $g := g - 1$ and continue. Since we only reconsider some old calculations and only decrease the argument b in between, still the time does not exceed $O(\gamma m + n) = O(\gamma m)$. □

4 If the Largest Cliques are Almost Disjoint

The motivation of the previous result was that in typical cases the interval graphs may be sparse, that is, the interval family is "long" but every point hits only a limited number of intervals. Recall the correspondence between points and cliques. If the γ largest cliques happen to be disjoint sets of intervals, then the γ corresponding points form an optimal solution, that is, a greedy algorithm taking the largest cliques succeeds. (Care is needed if certain cliques have equal sizes, but we withhold the details, as we will derive a more general result below.)

This greedy policy fails in general, if the largest cliques overlap. Perhaps the simplest counterexample is a graph with a clique of 4 vertices a, b, c, d and 2 further vertices e, f, with e being adjacent to a, b, and f being adjacent to c, d. This is easily seen to be an interval graph. Let $\gamma = 2$. Taking the 4-clique first, we can cover only 5 vertices, whereas the 2 disjoint 3-cliques cover all 6 vertices.

In the following, remember that "cliques" only refers to maximal complete subgraphs.

Definition 6. *Let ω_i denote the size of the ith largest clique, and let $s(\gamma) := \sum_{i=1}^{\gamma} \omega_i$. A solution that hits $s(\gamma) - \lambda$ intervals is said to have* loss λ.

If γ largest cliques are disjoint, then the corresponding γ points can hit $s(\gamma)$ intervals. An obvious question is how difficult it is to compute a solution that hits $s(\gamma) - \lambda$ intervals, with a small loss λ. Note that $\lambda = 0$ is the special case discussed above. So our question is how simple the problem remains if the largest cliques overlap only a little. Sparse interval families are likely to have this property which is less restrictive than the disjoint case.

Lemma 7. *Any solution H with loss λ has the following property: Every point r_b in H hits at least $|I(b)| - \lambda$ new intervals.*

Proof. Assume that some point r_b in H hits fewer than $|I(b)| - \lambda$ new intervals. Then this causes a loss exceeding λ (even in the best case when the γ points in H are those of the γ largest cliques), a contradiction. □

Theorem 8. *Given a family of n intervals and a maximum loss λ, we can determine γ points that together hit at least $s(\gamma) - \lambda$ intervals (or report that no solution with loss at most λ exists), in $O(m + \gamma\lambda n)$ time.*

Proof. First we preprocess the given interval family according to Lemma 3 in $O(m)$ time. We use the dynamic programming scheme from Theorem 5 and the same notations. The only difference is that, due to Lemma 7, it suffices to consider the last λ intervals in every sorted set $I(b)$. Thus, if our final solution with $h(\gamma, n)$ intervals has a loss exceeding λ, we know that λ was chosen too small. Clearly, the time is $O(\gamma\lambda n)$. □

For λ not being given in advance, we can find the optimal number $s(\gamma) - \lambda$ of intervals by doing unbounded binary search for λ in $O(\log \lambda)$ steps, and solving the existence problem each time as in Theorem 8, which costs an extra $O(\log \lambda)$ factor. (For unbounded search we refer to the classical paper [1], however the $O(\log \lambda)$ bound is obvious.)

By defining the density of the interval graph as $\delta := m/n$ we can compare the factors of n. The algorithm in Theorem 8 is faster than the general algorithm in Theorem 5 if the loss is $\lambda < \delta$. In the following we will devise another algorithm which has a time bound of $O(m+\gamma^2\lambda^2 \log n)$. Note that the second term is only logarithmic in n and otherwise depends only on γ and λ. In other words, the factor of n is asymptotically just δ, compared to $\delta + \gamma\lambda$ in Theorem 8. Hence the new algorithm becomes faster if the number of points is $\gamma > \delta/\lambda$. It is also faster than the algorithm in Theorem 5. Even if λ is not part of the input and the time bound gets an extra $O(\log \lambda)$ factor, the new algorithm is faster than the one in Theorem 5 for $\gamma > \log \lambda$.

5 Dynamic Programming with Restricted Candidate Set

Lemma 9. *Any solution H with loss λ has the following properties:*
(i) H contains every point r_b with $|I(b)| > \omega_\gamma + \lambda$.
(ii) H contains only points r_b with $|I(b)| \geq \omega_\gamma - \lambda$.

Proof. Remember that the size of H is fixed as γ.

For (i), assume that some point r_b as specified is not in H. Then some point that is in H instead of r_b hits at most ω_γ intervals. Thus H hits at most $s(\gamma) - |I(b)| + \omega_\gamma < s(\gamma) - (\omega_\gamma + \lambda) + \omega_\gamma = s(\gamma) - \lambda$ intervals, hence the loss exceeds λ, a contradiction.

For (ii), assume that some point r_b with $|I(b)| < \omega_\gamma - \lambda$ is in H. Then some of the points r_c that hit the γ largest cliques $I(c)$, with sizes at least ω_γ, is not in H (and r_b is there instead). Thus H hits at most $s(\gamma) - \omega_\gamma + |I(b)| < s(\gamma) - \omega_\gamma + (\omega_\gamma - \lambda) = s(\gamma) - \lambda$ intervals, hence the loss exceeds λ, again a contradiction. □

Definition 10. *We call the cliques specified in Lemma 9 (i) and (ii) the major and minor cliques, respectively.*

Due to (ii), cliques smaller than $\omega_\gamma - \lambda$ can be ignored henceforth.

Theorem 11. *Given a family of n intervals and a maximum loss λ, we can determine γ points that together hit at least $s(\gamma) - \lambda$ intervals (or report that no solution with loss at most λ exists), in $O(m + \gamma^2\lambda^2 \log n)$ time.*

Proof. According to (i) we put all major cliques (more precisely, their corresponding points r_b) in the solution. Due to (i) and (ii), the minor cliques have sizes $|I(b)|$ in the range from $\omega_\gamma - \lambda$ to $\omega_\gamma + \lambda$. Due to their minimum size and Lemma 7, each of the further points added to the solution must hit at least $\tau := \omega_\gamma - 2\lambda$ new intervals.

The limited range of interval numbers suggests to use other parameters for the dynamic programming function. From now on let g denote the number of *minor* cliques selected so far, in addition to the major cliques that we are forced to use. As seen above, every selected minor clique hits some number $\tau + k$ of new intervals, with $k \geq 0$. Let h denote the sum of all these excess numbers k above the threshold τ. Note that $k \leq 3\lambda = O(\lambda)$ for each of these k, by Lemma 7. Finally, we define $t(g, h)$ as the smallest index b such that there exists a partial solution with values g, h, where r_b is the rightmost point corresponding to a selected minor clique. If no such solution exists, then we formally set $t(g, h) := n + 1$, or take any dummy value larger than n. Clearly, we have $h \leq \gamma 3\lambda = O(\gamma\lambda)$, hence only $O(\gamma^2\lambda)$ pairs (g, h) exist.

In order to determine the $O(\gamma^2\lambda)$ indices $t(g, h)$ efficiently among the n possible ones, we need a special-purpose data structure. We take the sorted sets $I(b)$ from Lemma 3 and create certain triples (k, c, b) as described in the following. We keep only the minor cliques $I(b)$ and mark the intervals that are hit by the major cliques. In every minor clique $I(b)$ we reverse the order, such that intervals are sorted in right-to-left order of their left endpoints. We skip the first $\tau - 1$ unmarked intervals and then assign indices $k = 0, 1, 2, 3, \ldots$ to the following unmarked intervals. Note that $\tau + k$ equals the number of new intervals that r_b would hit if the last minor clique selected before were at some r_a just before the left endpoint of the unmarked interval with index k. Finally we create the triple (k, c, b) for the interval in $I(b)$ that received index k and has the left endpoint r_c. Note that the total number of triples is $O(m)$ by Lemma 3, in general it can be much smaller. Next, for every fixed k, we sort all triples (k, c, b) by ascending b. Since the indices b are sorted, this only requires a re-shuffling in $O(m)$ time.

Finally we discard dominated triples: We say that (k, c, b) dominates (k, c', b') if $c \geq c'$ but $b \leq b'$. Instead of adding $r_{b'}$ to a solution and hitting exactly $\tau + k$ new intervals, we can use r_b: This point still hits at least $\tau + k$ new intervals (as the last minor clique selected before was at some point r_a with $a \leq c' \leq c$), and since $b \leq b'$, all new intervals hit by further points to the right remain new intervals in the modified solution, too. In order to erase all dominated triples in $O(m)$ time, we take each k, loop through the list of triples (k, c, b) by ascending b, maintain the maximum c observed so far, and retain a triple only if c gets strictly larger.

For any fixed g, suppose that all $t(g, h)$ are already computed, and that all $t(g + 1, h')$ are initialized by $n + 1$. Consider any such known value $t(g, h) = a$. For each k we need to determine the smallest b such that we can hit $\tau + k$ new intervals by appending point r_b to a solution whose last minor clique is at r_a. To this end, we need the triple (k, c, b) with $a < c$ and minimum b. Since no triple is dominated by others, this is equivalent to determining the triple (k, c, b) with minimum $c > a$. Since the retained values of c are monotone decreasing (for any fixed k), we can find c by binary search in $O(\log n)$ time. Then we update $t(g+1, h+k) := b$ if the returned b is smaller than the current value of $t(g + 1, h + k)$. It is clear that we eventually get the correct indices $t(g + 1, h')$ for all h'.

Let J be the number of major cliques, and L the number of intervals hit by them. A solution has loss at most λ if and only if $L + (\gamma - G)\tau + h \geq s(\gamma) - \lambda$. Hence, a solution with loss at most λ exists if and only if $t(\gamma, h) \leq n$ holds for some h satisfying this inequality. Since there exist only $O(\gamma^2\lambda)$ pairs (g, h), and each one is combined with $O(\lambda)$ values of k, the dynamic programming phase costs $O(\gamma^2\lambda^2 \log n)$ time. As was detailed above, preprocessing takes $O(m)$ time. □

For λ not being given in advance, the same remark as in Theorem 8 applies.

6 Bounded Clique Size

As usual, let $\omega(= \omega_1)$ denote the size of a maximum clique in our interval graph. From Lemma 3 we see that $m = O(\omega n)$ holds in interval graphs. Together with Theorem 5 this implies:

Corollary 12. *Given a family of n intervals whose interval graph has the maximum clique size ω, we can determine γ points that together hit a maximum number of intervals, in $O(\gamma\omega n)$ time.*

More interesting is another observation:

Theorem 13. *Given a family of n intervals, we can determine γ points that together hit a maximum number of intervals, in $O(m + \gamma^2\omega^2 \log n)$ time.*

Proof. The dynamic programming algorithm inside Theorem 11 is only based on the fact that one point can add $k = O(\lambda)$ new intervals. Trivially we also have $k = O(\omega)$. Thus we can proceed similarly, with h as the absolute number of intervals hit so far, and replace λ with ω in the time bound for dynamic programming. □

7 Conclusions

We proposed algorithms for hitting a maximum number of intervals by γ points, that run faster than the worst-case time bound $O(\gamma n^2)$ from [3] suggests, when the interval family has few overlaps. Different dynamic programming versions are tailored to different "shapes" of the given interval family. The refined bounds might also be used for the discrete scheduling problem, of maximizing throughput with a budget for gaps, which is considered in [3].

Some questions for further research are: Can we get rid of the $O(\log n)$ factor in Theorem 11, and of the $O(\log \lambda)$ factors when the maximum loss is not part the input (Theorem 8 and 11)? Can the algorithms be generalized to larger standard graph classes such as circular-arc graphs and (strongly) chordal graphs?

Acknowledgment

The author would like to thank the anonymous referees for spotting an error in an earlier version, and for other valuable comments that led to clarifications.

References

[1] J.L. Bentley, A.C.C. Yao, An almost optimal algorithm for unbounded searching. Info. Proc. Letters 5 (1976) 82–87

[2] M.C. Carlisle, E.L. Lloyd, On the k-coloring of intervals, Discrete Appl. Math. 59 (1995) 225–235

[3] M. Chrobak, M.J. Golin, T.W. Lam, D. Nogneng, Scheduling with gaps: New models and algorithms, in: V.Th. Paschos, P. Widmayer (Eds.), 9th Int. Conf, on Algorithms and Complexity CIAC 2015. LNCS, vol. 9079, Springer 2015, pp. 114–126

[4] V. Chvatal, A greedy heuristic for the set-covering problem, Math. of Oper. Res. 4 (1979) 233–235

[5] K. Edwards, S. Griffiths, W.S. Kennedy, Partial interval set cover – Trade-offs between scalability and optimality, in: P. Raghavendra, S. Raskhodnikova, K. Jansen, J.D.P. Rolim (Eds.), 16th Int. Workshop on Approximation, Randomization, and Combin. Optim. Algor. and Techniques APPROX-RANDOM 2013, LNCS, vol. 8096, Springer 2013, pp. 110–125

[6] T. Elomaa, J. Kujala, Covering analysis of the greedy algorithm for partial cover, in: T. Elomaa, H. Mannila, P. Orponen (Eds.), Algorithms and Applications. Essays Dedicated to Esko Ukkonen on the Occasion of His 60th Birthday, LNCS, vol. 6060, Springer 2010, pp. 102–113

[7] U. Feige, A threshold of ln n for approximating set cover, J. ACM 45 (1998) 634–652

[8] F. Gavril, Algorithms for maximum k-colorings and k-coverings of transitive graphs, Networks 17 (1987), 465–470

[9] M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs (2nd ed.), Annals of Discrete Math., vol. 57, North Holland, Elsevier 2004

[10] W.L. Hsu, and K.H. Tsai, Linear time algorithms on circular-arc graphs. Info. Proc. Letters 40 (1991) 123–129

[11] J. Kneis, A. Langer, P. Rossmanith, Improved upper bounds for partial vertex cover, in: H. Broersma, T. Erlebach, T. Friedetzky, D. Paulusma, (Eds.), 34th Int. Workshop on Graph-Theoretic Concepts in Computer Science WG 2008, LNCS, vol. 5344, Springer 2008, pp. 240–251

[12] R. Niedermeier, Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Math. and its Appl., Oxford Univ. Press 2006

[13] E. Ukkonen, Algorithms for approximate string matching, Info. and Control 64 (1985) 100–118

[14] M. Yannakakis, F. Gavril, The maximum k-colorable subgraph problem for chordal graphs, Info. Proc. Letters 24 (1987), 133–137