A Note on Maximum Independent Sets in Rectangle Intersection Graphs

Timothy M. Chan*
School of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1, Canada
tmchan@uwaterloo.ca

September 12, 2003

Abstract

Finding the maximum independent set in the intersection graph of n axis-parallel rectangles is NP-hard. We re-examine two known approximation results for this problem. For the case of rectangles of unit height, Agarwal, van Kreveld, and Suri (1997) gave a $(1+1/k)$-factor algorithm with an $O(n \log n + n^{2k-1})$ time bound for any integer constant $k \geq 1$; we describe a similar algorithm running in only $O(n \log n + n\Delta^{k-1})$ time, where $\Delta \leq n$ denotes the maximum number of rectangles a point can be in. For the general case, Berman, DasGupta, Muthukrishnan, and Ramaswami (2001) gave a $\lceil \log_k n \rceil$-factor algorithm with an $O(n^{k+1})$ time bound for any integer constant $k \geq 2$; we describe similar algorithms running in $O(n \log n + n\Delta^{k-2})$ and $n^{O(k/\log k)}$ time.

Keywords: Approximation algorithm; Computational geometry; Dynamic programming

1 Introduction

The subject of this note is the following (henceforth, "the problem"): given a collection $\mathcal{C}$ of n axis-parallel rectangles in the plane, find a largest subcollection S^* of disjoint rectangles, or equivalently, find a maximum independent set in the intersection graph of the rectangles. The problem has interested several groups of researchers [1, 2, 3, 5, 7, 9], due to applications ranging from map labeling to data mining. As the problem is NP-hard [6, 8], attention is focused on finding efficient approximation algorithms.

See [3] for a more comprehensive discussion of past results. The earliest result was perhaps the "shifting"-based polynomial-time approximation scheme by Hochbaum and Maass [7] for the special case of *unit squares*. Originally, the scheme required $n^{O(k^2)}$ time to guarantee an approximation factor of $1 + 1/k$. Subsequent improvements have reduced the running time somewhat. Notably, Agarwal, van Kreveld, and Suri [1] described a refinement with a time and space bound of $O(n^{2k-1})$ for any constant integer $k \geq 2$. This algorithm worked more generally for *unit-height rectangles*,

*This work was supported in part by an NSERC Research Grant.

which arise in the application to map labeling for a fixed font size. For $k = 1$ (factor 2), the algorithm is almost trivial and runs in $O(n \log n)$ time.

Generalizing in another direction, Erlebach, Jansen, and Seidel [5] and the author [3] have recently obtained polynomial-time approximation schemes for *arbitrary squares*. To get factor $1 + 1/k$, the required running time is $n^{O(k^2)}$ and $n^{O(k)}$, respectively.

For *arbitrary rectangles*, the problem is open. As several researchers have independently observed [1, 9, 10], a logarithmic approximation factor is certainly possible. For example, Agarwal, van Kreveld, and Suri [1] described a straightforward $O(n \log n)$-time algorithm with factor at most $\lceil \log_2 n \rceil$ (in fact, $\lceil \log_2 |S^*| \rceil$—see [10]). Currently, no polynomial-time algorithm is known with $o(\log n)$ approximation factor, although Berman *et al.* [2] have recently observed that the $\log n$ bound can be reduced by an arbitrary constant multiple. Specifically, factor $\lceil \log_k n \rceil$ can be achieved in $O(n^k |S^*|)$ time.

Our new results involve a few modest improvements:

- For *unit-height rectangles*, we simplify and speed up Agarwal *et al.*'s dynamic programming algorithm [1] to run in $O(n^k)$ worst-case time, for the same approximation factor $1 + 1/k$. The time bound is actually $O(n \log n + n\Delta^{k-1})$, where Δ denotes the maximum depth. Here, the *depth* of a point is the number of rectangles containing the point. In map labeling applications, the value of Δ is small (close to a constant), so our result suggests that the dynamic programming approach may not be as impractical as previously thought. (For example, factor 3/2 requires only $O(n \log n + n\Delta)$ time, not cubic.) The space usage is bounded by $O(n + n'\Delta^{k-1})$, where n' is the maximum number of rectangles a horizontal line can intersect; in practice, n' is likely to be much smaller than n.

- For *arbitrary rectangles*, we show that a modification of Agarwal *et al.*'s $\lceil \log_2 n \rceil$ divide-and-conquer algorithm that incorporates the dynamic programming subroutine can find a factor-$\lceil \log_k n \rceil$ solution in $O(n \log n + n\Delta^{k-2})$ time. The resulting algorithm is simpler and faster (and thus more practical) than Berman *et al.*'s [2]. The space usage is $O(n + n'\Delta^{k-2})$.

  On the theoretical side, we also derive a better worst-case time bound in terms of n. We show that factor near $\lceil \log_k n \rceil$ requires $O(n^{f(k)})$ time for some function $f(k) = O(k/\log k)$. For small k, the time bound is actually quite reasonable (for example, $f(3) < 1.369$, $f(4) = 1.5$, $f(5) < 1.635$, and $f(8) = 2$).

The techniques we use are hardly original; the basic dynamic programming strategy, as well as the divide-and-conquer idea, can be found (independently and in different forms) in the papers of Agarwal *et al.* [1] and Berman *et al.* [2]. The improvements come mostly from a more careful usage of these techniques and, at the same time, an attempt to keep things simple.

2 A Dynamic Programming Subroutine

Lemma 2.1 *Fix an integer constant $k \geq 1$. If all the rectangles can be stabbed by k horizontal lines, then we can solve the problem exactly in $O(n \log n + n\Delta^{k-1})$ time.*

Proof: Let $R_1, \ldots, R_n$ be the given rectangles. Let a_i and b_i be the left and right x-coordinate of R_i. By sorting, assume that $a_1 \leq \cdots \leq a_n \leq a_{n+1} = \infty$. Let *next*$[j]$ denote the smallest index i

with $a_i > b_j$. For a set S of rectangles, let $S_{|i}$ denote the subset of rectangles in S intersecting the vertical line $x = a_i$.

A subproblem is created for each index i and each subset S of disjoint rectangles intersecting $x = a_i$ with $|S| \leq k - 1$: define $A[i, S]$ to be the maximum number of disjoint rectangles among $R_i, \ldots, R_n$ that do not intersect the rectangles in S. These numbers can be computed from $i = n+1$ to 1 by the following rules.

1. For the base case, $A[n + 1, \emptyset] = 0$.

2. If R_i intersects some rectangle in S, then

$$A[i, S] = A[i + 1, S_{|i+1}],$$

because R_i cannot be used in the solution to the subproblem.

3. Otherwise, if $|S| < k - 1$, then

$$A[i, S] = \max\left\{A[i + 1, S_{|i+1}],\ 1 + A[i + 1, (S \cup \{R_i\})_{|i+1}]\right\},$$

because the first term corresponds to the case where R_i is not used in the solution, and the second term corresponds to the case where R_i is used.

4. Otherwise, $|S| = k - 1$. Let t be the smallest $next[j]$ value over all rectangles $R_j \in S \cup \{R_i\}$. Then

$$A[i, S] = \max\left\{A[i + 1, S_{|i+1}],\ 1 + A[t, (S \cup \{R_i\})_{|t}]\right\},$$

because if R_i is used in the solution, then all other rectangles in the solution must be to the right of some rectangle in $S \cup \{R_i\}$, and thus t (otherwise, we would have $k + 1$ disjoint rectangles intersecting a vertical line, contradicting the assumption of the lemma). Note that $|(S \cup \{R_i\})_{|t}| \leq k - 1$ by the choice of t.

The maximum number of disjoint rectangles is $A[1, \emptyset]$. The actual collection of disjoint rectangles can be retrieved in the usual way. By the assumption of the lemma, there are $O(n\Delta^{k-1})$ subproblems. So, excluding sorting of x-coordinates (needed to initialize the *next* entries), this dynamic programming algorithm runs in $O(n\Delta^{k-1})$ time.

One implementation issue remains to be addressed: to use $O(n\Delta^{k-1})$ space instead of $O(n^k)$ for the array A, we need to first map tuples (i, S) to indices. This can be done by sorting (lexicographically) all tuples occurring in both sides of the above equations, and assigning a common index to identical tuples. By radix sort, the additional running time is $O(n\Delta^{k-1})$. $\square$

3 The Unit-Height Case

Theorem 3.1 *Fix an integer constant $k \geq 1$. If all rectangles have unit height, then we can solve the problem approximately to within a factor of $1 + 1/k$ in $O(n \log n + n\Delta^{k-1})$ time.*

Proof: We use a shifting idea.

1. For each $i = 0, \ldots, k$, let $\mathcal{C}^{(i)}$ be the subcollection of all rectangles that do not intersect any grid horizontal line $y = \ell$ with $\ell \equiv i \bmod (k+1)$. Now, $\mathcal{C}^{(i)}$ is a union of groups of rectangles, where each group can be stabbed by k horizontal lines (because the rectangles have unit height) and no two rectangles from different groups intersect. We can thus solve the problem for each group by Lemma 2.1, and take the union to obtain the solution $S^{(i)}$ to $\mathcal{C}^{(i)}$.

2. Return the largest set S among $S^{(0)}, \ldots, S^{(k)}$.

The running time sums to $O(n \log n + n\Delta^{k-1})$.

Since each unit-height rectangle belongs to exactly k of the $k+1$ subcollections $\mathcal{C}^{(0)}, \ldots, \mathcal{C}^{(k)}$,

$$k|S^*| = \sum_{i=0}^{k} |S^* \cap \mathcal{C}^{(i)}| \le \sum_{i=0}^{k} |S^{(i)}| \le (k+1)|S|,$$

implying that $|S^*| \le (1 + 1/k)|S|$. □

4 The General Case

Theorem 4.1 *Fix an integer constant $k \ge 2$. If we are given $H > 1$ horizontal lines that stab all rectangles, then we can solve the problem approximately to within a factor of $\lceil \log_k H \rceil$ in $O(n \log n + n\Delta^{k-2})$ time.*

Proof: We use divide-and-conquer.

1. The base case, $H \le k$, can be handled directly by Lemma 2.1.

2. Let $\ell_1, \ldots, \ell_{k-1}$ be the $\lceil H/k \rceil$-th, $2\lceil H/k \rceil$-th, ..., $(k-1)\lceil H/k \rceil$-th lowest horizontal lines (with ℓ_0 at $y = -\infty$ and ℓ_k at $y = \infty$).

3. Let $\mathcal{C}^{(0)}$ be the subcollection of all rectangles stabbed by these $k-1$ lines. We can compute the exact solution $S^{(0)}$ to $\mathcal{C}^{(0)}$ by Lemma 2.1.

4. For each $i = 1, \ldots, k$, let $\mathcal{C}^{(i)}$ be the subcollection of all rectangles that lie entirely between ℓ_{i-1} and ℓ_i. Now, $\mathcal{C}^{(i)}$ is stabbed by $\lceil H/k \rceil$ horizontal lines. We can compute an approximate solution $S^{(i)}$ to $\mathcal{C}^{(i)}$ recursively.

5. Return the larger of the two sets $S^{(0)}$ and $S^{(1)} \cup \cdots \cup S^{(k)}$.

The running time is dominated by step 3 and sums to $O(n \log n + n\Delta^{k-2})$.

To analyze the approximation factor, let S be the final returned solution. Consider the recursion tree generated and let $\mathcal{C}_v^{(0)}$ and $S_v^{(0)}$ be sets corresponding to node v of the tree. Then

$$\sum_v |S^* \cap \mathcal{C}_v^{(0)}| \le \sum_v |S_v^{(0)}| \le |S|,$$

where the sums are over all nodes at any fixed level of the tree. Summing over all levels yield $|S^*| \le |S| \lceil \log_k H \rceil$. □

5 More on the General Case

If Δ is polylogarithmic, we can set $k = \Theta(\log^{\varepsilon} n)$ for a small constant $\varepsilon > 0$ in Theorem 4.1 to get a polynomial-time algorithm with factor $O(\log n / \log\log n)$ (since the hidden constant in the time bound depends polynomially on k). For an arbitrary Δ, however, we are unable to obtain an approximation factor of $o(\log n)$ in polynomial time; instead, we concentrate on improving the dependence on n in the running time.

In the worst case, Theorem 4.1 yields factor $\lceil \log_k n \rceil$ in $O(n^{k-1})$ time ($k \geq 3$). This worst-case bound can be improved, for example, by using larger branching factors (instead of k) at deeper levels of the tree (as n decreases). A still better bound can be obtained by a different approach which we describe now. This approach exploits the "unweightedness" of the problem (in contrast, all the algorithms we have given so far can be extended to solve the maximum-weight independent set problem for input rectangles with weights).

Corollary 5.1 *Fix an integer constant $k \geq 2$. We can solve the problem approximately to within a factor of $\lceil \log_k |S^*| \rceil$ in $O(n \log n + n\Delta^{k-2})$ time.*

Proof: Project the rectangles onto the y-axis. We can find the smallest set of points that stab the resulting (one-dimensional) y-intervals by the standard greedy algorithm, in linear time after sorting. As is well-known, in one dimension, the smallest number of stabbing points, H, is equal to the largest number of disjoint intervals. Thus, $H \leq |S^*|$. The result follows from Theorem 4.1. □

Corollary 5.2 *Fix an integer constant $k \geq 2$. Given d, we can compute a solution S with $|S^*| \leq |S| \lceil \log_k |S^*| \rceil + n/d$ in $O(n \log n + nd^{k-2})$ time.*

Proof: We use a greedy strategy. Repeatedly find a point p of depth $\geq d$ and remove all rectangles stabbed by p, until the remaining collection of rectangles, $\mathcal{C}'$, have maximum depth $< d$. Then return the solution to $\mathcal{C}'$ by Corollary 5.1.

The number of iterations is at most n/d, since at least d rectangles are removed per iteration. Since $\mathcal{C} - \mathcal{C}'$ can be stabbed by at most n/d points, there can be at most n/d disjoint rectangles in $\mathcal{C} - \mathcal{C}'$. Thus, $|S^* - \mathcal{C}'| \leq n/d$, and $|S^* \cap \mathcal{C}'| \leq |S| \lceil \log_k |S^* \cap \mathcal{C}'| \rceil$. The approximation bound follows.

To analyze the running time, observe that a standard sweep-line algorithm can compute the maximum depth in a collection of n axis-parallel rectangles in $O(n \log n)$ time; for example, see [4]. This algorithm stores the rectangles that intersect the sweep line in a tree structure supporting logarithmic-time insertions and deletions, and maintains the maximum depth at the sweep line, as we move from left to right. We can modify the algorithm so that as soon as a point p of depth $\geq d$ is discovered, we delete the rectangles stabbed by p from the data structure, and resume the sweep. As the total number of deletions is still bounded by n, the total time required to implement the greedy phase remains $O(n \log n)$. □

Corollary 5.3 *Fix real constants $b > 2$ and $\varepsilon > 0$. We can solve the problem approximately to within a factor of $(1+\varepsilon)\lceil \log_b n \rceil$ in $O(n^{f(b)})$ time, where*

$$f(b) := 1 + \max_{i=1,\ldots,\lceil b \rceil - 1} (i-1)(1 - \log_b i) = O(b/\log b).$$

Proof: Run the algorithm in Corollary 5.2 for each $k = 2, \ldots, \lceil b \rceil$ with $d = Cn^{1-\log_b(k-1)}$ for a sufficiently large constant C, and return the largest solution S found. The time bound holds.

To analyze the approximation factor, let k be such that $n^{\log_b(k-1)} \leq |S^*| \leq n^{\log_b k}$. Then

$$|S^*| \leq |S| \lceil \log_k |S^*| \rceil + n^{\log_b(k-1)}/C \leq |S| \lceil \log_b n \rceil + |S^*|/C,$$

implying that $|S^*| \leq |S| \lceil \log_b n \rceil / (1 - 1/C)$.

Note that since $(x-1)(1-\log_b x) < x \frac{\ln(b/x)}{\ln b} < b/\ln b$ for all $x \leq b$, we indeed have $f(b) = O(b/\log b)$. □

References

[1] P. K. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Comput. Geom. Theory Appl.*, 11:209–218, 1998.

[2] P. Berman, B. DasGupta, S. Muthukrishnan, and S. Ramaswami. Efficient approximation algorithms for tiling and packing problems with rectangles. *J. Algorithms*, 41:443–470, 2001.

[3] T. M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46:178–189, 2003.

[4] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete Comput. Geom.*, 11:321–350, 1994.

[5] T. Erlebach, K. Jansen, and E. Seidel. Polynomial-time approximation schemes for geometric graphs. In *Proc. 12th ACM-SIAM Sympos. Discrete Algorithms*, pages 671–679, 2001.

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

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

[8] H. Imai and T. Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *J. Algorithms*, 4:310–323, 1983.

[9] S. Khanna, S. Muthukrishnan, and M. Paterson. On approximating rectangle tiling and packing. In *Proc. 9th ACM-SIAM Sympos. Discrete Algorithms*, pages 384–393, 1998.

[10] F. Nielsen. Fast stabbing of boxes in high dimensions. *Theoret. Comput. Sci.*, 246:53–72, 2000.