Rectangular Partitioning

Joe Forsmann and Rock Hymas

Introduction/Abstract

We will look at a problem that I (Rock) had to solve in the course of my work.

> Given a set of non-overlapping rectangles each having top, left, bottom, and right coordinates, divide the x-y plane over which these rectangles exist into the minimum number of rows plus columns, such that each resulting cell intersects at most one rectangle.

We will refer to this problem as Rectangular Partitioning and show that it is NP-complete, but that approximation algorithms exist. We also explore the non-optimal solution used in my work, as well as discussing open problems and future challenges.

Motivation

The motivation for Rectangular Partitioning comes from my work. The application I help create has forms that can be designed by third parties, and it also has a layout engine that makes it easy to change the layout of a form based on the size of the form and its contents. The layout engine takes as it's input a table of cells and their contents, along with attributes specifying how a given cell, row, or column should behave (i.e. expand when the form expands, alignment of controls, etc.). The table of cells may have controls that overlap multiple cells, but each cell may contain at most one control. Third party form designers don't have direct access to this layout engine, and so we created a conversion from the form they designed to the layout engine input by subdividing the form space into rows and columns based on the placement of individual controls, which are all rectangular. Minimizing the number of rows plus columns is important to the running time of the layout engine, so that is our goal.

Additionally, in researching Rectangular Partitioning, we have found that it is related to problems in parallel computation where a computational task is partitioned into subtasks that can be assigned to parallel processors in such a way that minimized communication among the processors in reassembling the solution. Understanding Rectangular Partitioning further may provide insight into the problem of subdividing a computational task for parallel processors.

Formal Problem Definition

Given a set of non-overlapping rectangles $R_1, R_2, \ldots R_n$ each having top coordinates $t(R_i)$, left coordinates $l(R_i)$, bottom coordinates $b(R_i)$, and right coordinates denoted as $r(R_i)$, partition the x-y plane over which these rectangles exist into the minimum number of rows plus columns, such that each resulting cell intersects at most one rectangle. Formally, a partitioning is determined by a set H of horizontal dividers (rows) $h_0 = 0 \leq h_1 \leq \cdots \leq h_p = \max_i r(R_i)$ and a set V of vertical dividers $v_0 = 0 \leq v_1 \leq \cdots \leq v_q = \max_j b(R_j)$. The partitioning creates $p \cdot q$ cells $c_{i,j}$, $0 \leq i < p$ and $0 \leq j < q$, where the cell $c_{i,j}$ is also a rectangle with $t(c_{i,j}) = v_j, b(c_{i,j}) = v_{j+1}, l(c_{i,j}) = h_i, r(c_{i,j}) = h_{i+1}$. We

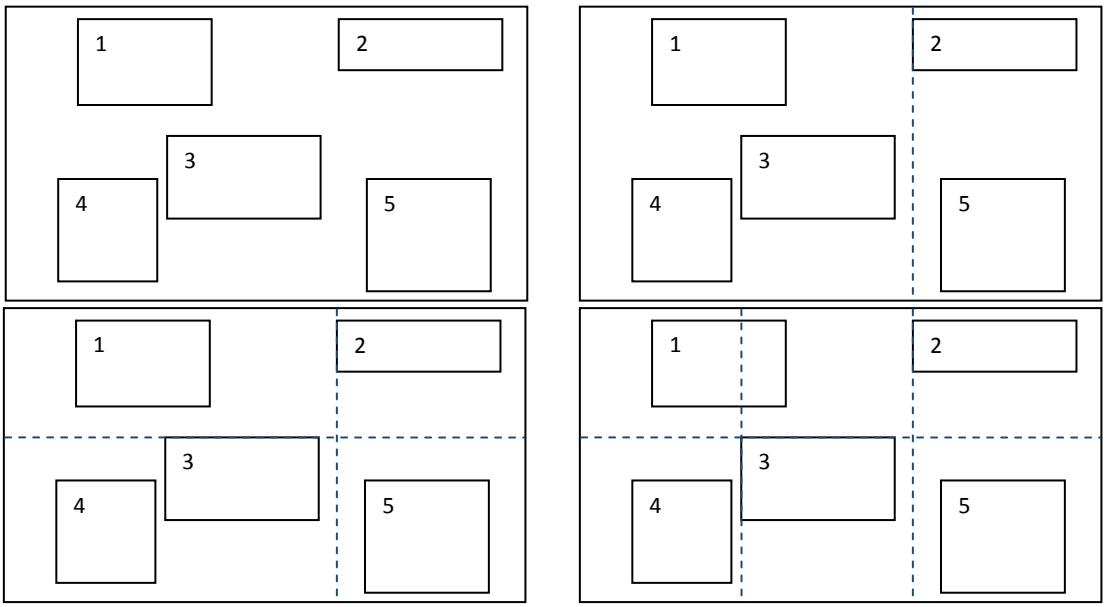
say that a cell $c_{i,j}$ intersects a rectangle R_k if $\max\left(t(c_{i,j}), t(R_k)\right) > \min\left(b(c_{i,j}), b(R_k)\right)$ and $\max\left(l(c_{i,j}), l(R_k)\right) > \min\left(r(c_{i,j}), r(R_k)\right)$. Then we try to find a partitioning $P = (H, V)$ such that $p + q$ is as small as possible, subject to the constraint that $\forall c_{i,j}$, $c_{i,j}$ intersects at most one R_k.

Our approach

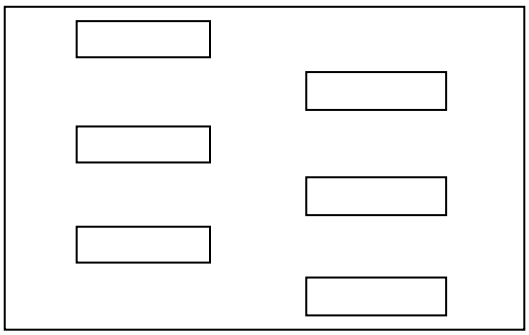
In my application, the existing algorithm for solving this problem is non-optimal. We give the algorithm here for completeness. In this description, increases along the y axis move downward.

```
Sort the rectangles first by increasing t(R_i), then by increasing l(R_i)
Create a partition with one row and one column
    h_0 = 0, h_1 = max_i r(R_i)
    v_0 = 0, v_1 = max_i b(R_i)
For each rectangle R_i
    For each cell c_{i,j} which intersects both R_i and some other R_j
        if R_i and R_j can be separated by creating a new row
            Create a new row, with new h_i = max(t(R_i), t(R_j))
            Insert h_i into the correct place in H
        else
            Create a new column, with new v_i = max(l(R_i), l(R_j))
            Insert v_i into the correct place in V
        endif
    end for
end for
```

The following sequence shows the progression of dividing rows and columns that the above algorithm goes through. The rectangles are numbered according to their order in this sorting.



The following is an example where the algorithm above does not find the optimal solution. The algorithm will divide it into 6 rows, but an optimal solution requires only 3 rows and 2 columns



Rectangular Partitioning is NP-Complete

Rectangular Partitioning has an equivalent decision problem that is defined as follows:

> Given the values p, q, is there a partitioning (H, V) of the x-y plane such that $|H| = p$ and $|V| = q$ and such that each cell $c_{i,j}$ intersects at most one of the rectangles R_i.

We show this by reducing the Balanced Bipartite Cover (BBC) problem to Rectangular Partitioning. BBC is defined as follows:

> Given a bipartite graph $G = (V_1, V_2, E)$ with $|V_1| = |V_2|$, $E \subseteq V_1 \times V_2$ and a positive integer k, are there subsets $U_1 \subseteq V_1$ and $U_2 \subseteq V_2$ such that $|U_1| = |U_2| = k$ each edge $(u, v) \in E$ has either $u \in U_1$ or $v \in U_2$.

BBC is shown to be NP-Complete in [2]. The following proof is analogous to the proof in [same reference] that BBC can be reduced to the Generalized Block Distribution.

First we create an instance of Rectangular Partitioning from a given instance of BBC in the following way. Let $n = |V_1| = |V_2|$ in the instance of BBC. Let $R_{i,j}$ be defined such that $l(R_{i,j}) = 2i + 1, t(R_{i,j}) = 2j + 1, r(R_{i,j}) = 2i + 2, b(R_{i,j}) = 2j + 2$. Then the instance of RP has $q = p = n + k + 2$ and includes the following rectangles:

1. $R_{0,0}$
2. $R_{0,4k+1}$ and $R_{0,4k+2}$ for $0 \leq k < \lceil n/2 \rceil$
3. $R_{1,4k}$ and $R_{1,4k+3}$ for $0 \leq k < \lfloor n/2 \rfloor$
4. $R_{4k+1,0}$ and $R_{4k+2,0}$ for $0 \leq k < \lceil n/2 \rceil$
5. $R_{4k,1}$ and $R_{4k+3,1}$ for $0 \leq k < \lfloor n/2 \rfloor$
6. $R_{2i,2j}$ and $R_{2i+1,2j+1}$ for all $(i, j) \in E$

If we find a solution to this RP then the rectangles from the first five rules force is to create at least $n + 2$ rows and $n + 2$ columns no matter what the rectangles in rule 6 do. This is demonstrated by the hollow rectangles in the first two rows and first two columns in Figure 1. Forcing columns and rows, with the minimum set of rows and columns indicated by dotted lines. This leaves us with k rows and k

columns to add in hopes of satisfying the requirements of the rectangles in rule 6. For each edge in G, rule 6 constructs two rectangles (green) not divided by the the rows/columns forced into existence by rules 1 through 5. Each of these sets of rectangles can and must be divided either by adding a new row or a new column, or both, if we are to meet the requirement that each cell intersect at most one rectangle. Splitting them with a new row corresponds to choosing a vertex from V_1 in BBC and splitting them with a new column corresponds to a choosing a vertex from V_2. It is clear from the construction of this RP instance that there exists a solution to it if and only if the corresponding BBC problem has a solution. Thus, RP is NP-hard. A certificate of a solution to the decision version of RP that is verifiable in polynomial time is simple a partitioning H of the x-y plane with p rows and q columns that satisfies the requirement of RP. Thus, RP is in NP and is NP-Complete.

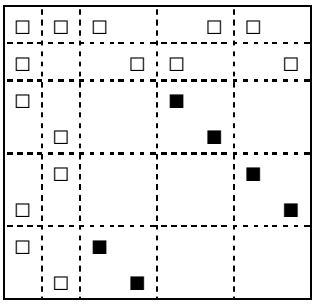


Figure 1. Forcing columns and rows

Paper Discussion

Problem Overview

In their paper [1] Muthukrishnan and Suel discuss partitioning an array of cells containing integers into rectangular 'tiles'. Consider, for example, that integers in each cell represent amount of work, and the job is to partition the work among processors. The goal of the algorithm is to balance the work between the processors. The paper also describes applications of this algorithm in the database and image processing paradigm.

Metrics for Optimal Partitioning

Two metrics are used to quantify the optimality of the algorithm – MAX-SUM and SUM-VAR. The MAX-SUM metric is simply the maximum of all tile weights, where the tile weight is the sum of all cells within that tile. The SUM-VAR metric is the sum of all variance, where variance is defined as the square of the difference between the tile weight and the average of all tile weights. The goal of the algorithm is to find a partitioning that minimizes these metrics.

NP-Completeness

Via references, the paper stated that variation of this problem was proven to be NP-Complete. The MAX-SUM in three dimensions was found to be NP Complete using a reduction to Monotonic 3-SAT. Further, the two dimensional problem was shown to be NP Complete using a reduction to the Balanced Complete bipartite subgraph. For optimizing the slightly different problem of minimizing the number of tiles, a reduction to Set Cover is referenced that provides a $O(\log(n))$ approximation of the number of dividers.

Other Heuristic Algorithms

Having shown that this problem and many variants are NP-Complete, the paper turns toward finding algorithms that approximate solutions. Many other authors have shown that there exist heuristic algorithms that provide good approximations to the MAX-SUM problem. The best of which provides a solution with an approximation factor of about 120. Most of these 'other' algorithms are based on the following principle:
Do alternate scans of each axis, and for each scan find the best possible cuts – taking into account all the cuts that have been made before. The algorithm terminates when there are no more cuts made over a full scan.

The Algorithm
The formal algorithm can be found in section 5.2 of the paper. The core algorithm described in this paper is inspired by a greedy algorithm used for solving Set Cover. Essentially, the author makes optimizations on another referenced solution to provide a better running time and closer approximation to an optimal solution than any other known solution.

Mapping Rectangular Partitioning to Array Partitioning
The concept of returning a list of 'x' and 'y' values that define the partitioning boundaries are the same between the problem discussed in the paper, and the project problem of partitioning controls into cells. Given this, we considered ways in which to map the rectangular partitioning into the array partitioning. In the Rectangular Partitioning problem, if we place a vertical line on the right side of each control, and we place a horizontal line on the top of each control, then this creates an $n \times m$ array with at most one control per cell. If a cell in this array contains a control, then set the value of that cell to be a large value. Finally, set the value of all other cells that do not contain a control to 0. Then, we run the version of the algorithm described in this paper with the metric to minimize the number of partitions to obtain the solution to the Rectangular Partitioning problem.

To ensure that each cell in the end will contain at most one control, we set the value of a control-containing cell to be a large value. Therefore, the extreme cost of having two controls in one cell, and one cell in another would force all cells to have a single control.

Open Problems and Challenges

There are many potential variations on this problem, even within the framework of our motivation section. For example, the layout engine might have $O(pq)$ time complexity, and so the optimization would change to optimizing for minimum pq, rather than minimum $p + q$. One could easily imagine removing the initial constraint that all rectangles are non-overlapping. We could then rephrase the question in terms of minimizing the number of cells that intersect multiple rectangles. If the layout engine could handle multiple controls in a given cell, but doing so was expensive, we could modify what we're optimizing for by making it $p + q + cI$, where c is a constant penalty factor and I is the number of cells with multiple intersecting controls.

Bibliography

[1] S. Muthukrishnan and Torsten Suel. *Approximation Algorithms for Array Partitioning Problems.* citeseer.ist.psu.edu/582593.html

[2] M. Grigni, and F. Manne, "*On the Complexity of the Generalized Block Distribution*," Proc. 3rd Int. Workshop on Parallel Algorithms for Irregularly Structured Problems (IRREGULAR'96),1996, pp. 319--326.

[3] S. Muthukrishnan, Viswanath Poosala, and Torsten Suel. *On rectangular partitionings in two dimensions: Algorithms*, complexity, and applications. 7th International Conference on Database Theory, January, 1999.

[4] S. Khanna, S. Muthukrishnan and S. Skiena. Efficient array partitioning. Proc. 24th ICALP, 616626, 1997. http://citeseer.ist.psu.edu/article/khanna97efficient.html