Evacuation of Rectilinear Polygons*

Sándor Fekete, Chris Gray, and Alexander Kröller

Department of Computer Science, TU Braunschweig, Germany
{fekete,gray,kroeller}@ibr.cs.tu-bs.de

Abstract. We investigate the problem of creating fast evacuation plans for buildings that are modeled as grid polygons, possibly containing exponentially many cells. We study this problem in two contexts: the "confluent" context in which the routes to exits remain fixed over time, and the "non-confluent" context in which routes may change. Confluent evacuation plans are simpler to carry out, as they allocate contiguous regions to exits; non-confluent allocation can possibly create faster evacuation plans. We give results on the hardness of creating the evacuation plans and strongly polynomial algorithms for finding confluent evacuation plans when the building has two exits. We also give a pseudo-polynomial time algorithm for non-confluent evacuation plans. Finally, we show that the worst-case bound between confluent and non-confluent plans is $2 - O(\frac{1}{k})$.

1 Introduction

A proper evacuation plan is an important requirement for the health and safety of all people inside a building. When we optimize evacuation plans, our goal is to allow people to exit the building as quickly as possible. In the best case, each of a building's exits would serve an equal number of the building's inhabitants. However, there might be cases in which this can not happen. For instance, when there is a bottleneck between two exits, it might make sense for most of the building's inhabitants to stay on the side of the bottleneck closer to where they begin, even if this means that one exit is used more than the other.

In this research, we study the computation of evacuation plans for buildings that are modeled as grid polygons. We make the assumption that every grid square is occupied by exactly one person and that at most one person can occupy a grid square at any given time. This assumption arises from the inability of building designers to know exactly where people will be in the building in the moments before an evacuation, and this pessimistic view of the situation is the only sensible one to take. Also remember that in some cases, such as in airplanes, the situation in which nearly every bit of floor space is occupied before an evacuation is more common than the alternative.

Evacuation plans can be divided into two distinct types. In the first, signs are posted that direct every person passing them to a specific exit. In the second,

* This research was funded by the German Ministry for Education and Research (BMBF) under grant number 03NAPI4 "ADVEST", which fully funded Chris Gray.

W. Wu and O. Daescu (Eds.): COCOA 2010, Part I, LNCS 6508, pp. 21–30, 2010.

every person is assigned to a distinct exit that does not necessarily depend on the exits to which his or her neighbors are assigned. The first type of evacuation plan generates what is known as a *confluent flow* and the second generates a *non-confluent flow*. More precise definitions of these terms will be given later.

It is clear that in a non-confluent flow, people can be evacuated as quickly as in a confluent flow, and we show that in certain instances people can be evacuated significantly more quickly in a non-confluent flow than in a confluent flow. However, a confluent flow is significantly easier to carry out than a non-confluent one, so we also give results related to it. We first show that the problem of finding an optimal confluent flow belongs to the class of NP-complete problems if the polygon has "holes"—that is, if it represents a building with completely enclosed rooms or other spaces. We then give an algorithm with a running time linear in the description complexity of the region (which can be exponentially smaller than the number of cells) that computes an evacuation plan for buildings without holes that have two exits; a generalization to a constant number of exits is more complicated, but seems plausible. Finally, we show that the worst-case ratio between the evacuation times for confluent flows and non-confluent flows for k exits is $2 - O(\frac{1}{k})$.

1.1 Preliminaries

We are given a rectilinear polygon P on a grid. There exists, on the boundary of P, a number of special grid squares known as *exits*. We call the set of exits $\mathcal{E} = \{e_1, \ldots, e_k\}$. We assume that every grid square in P contains a person. A person can move vertically or horizontally into an empty grid square or an exit. The goal is to get each person to an exit as quickly as possible. When an exit borders more than one grid square, we specify the squares from which people can move into the exit.

The area of P is denoted by A and the number of vertices of P is denoted by n. Note that A can be exponential in n. The set of people that leave P through the exit e is called the *e-exit class*. We also write *exit class* to refer to the set of people who leave through an unspecified exit. The grid squares that are adjacent to the boundary of P are known as *boundary squares*.

There are two versions of the problem that we consider. We call these *confluent flows* and *non-confluent flows*. In the first, we add the restriction that every grid square has a unique successor. Thus, for every grid square s, people passing through s leave s in only one direction. This restriction implies that evacuation plans are determined by space only. It does not exist for non-confluent flows. It can be argued that informing people which exit to use is easier in the case of confluent flows since a sign can be placed in every grid square, informing the people who pass through it which exit to use. However, we show that non-confluent flows can lead to significantly faster evacuations.

The major difficulty in the problem comes from bottlenecks. We define a *k-bottleneck* to be a rectangular subpolygon B of P such that two parallel boundary edges of B are the same as two edges of P, where the distance between the common edges is k.

1.2 Related Work

The problem when restricted to confluent flows is similar in many ways to the unweighted Bounded Connected Partition (or 1-BCP) problem [12]. In this problem, one is given an unweighted, undirected graph $G = (V, E)$ and k distinguished vertices. The goal is to find k connected subsets of V, where each subset contains exactly one of the distinguished vertices and where each has the same cardinality. If we define the dual of the grid contained in our polygon to be the graph formed by connecting adjacent faces and let the distinguished vertices be the exits, then 1-BCP is clearly the same as evacuation restricted to confluent flows with k exits.

It was independently shown by Lovász [8] and Györi [7] that a solution to the 1-BCP problem can be found for every graph that is k-connected. However, their proofs are not algorithmic. Thus, there has been some work on finding partitions of size k for low values of k. For example, Suzuki *et al.* give an algorithm for 2-partitioning 2-connected graphs [14]. One algorithm claiming a general solution for k-partitioning k-connected graphs [10] is incorrect. Unfortunately, when P contains 1-bottlenecks, the graph obtained by finding the dual of the grid inside P is 1-connected. Therefore, the results of Lovász and Györi do not apply. Also, since the dual of the grid contained in our polygon can have size exponential in the complexity of the polygon, we would probably need to merge nodes and then assign weights to the nodes of the newly-constructed graph. The addition of weights, however, makes the BCP problem NP-complete, even in the restricted case in which the graph is a grid [3].

Another connection is to the problem of partitioning polygons into subpolygons that all have equal area. The confluent version of our problem, indeed, can be seen as the "discrete" version of that problem. The continuous version has been studied. One interesting result from this study is that finding such a decomposition while minimizing the lengths of the segments that do the partitioning is NP-hard even when the polygons are orthogonal [1]. However, polynomial algorithms exist for the continuous case when that restriction is removed [9].

Baumann and Skutella [2] consider evacuation problems modeled as earliest-arrival flows with multiple sources. They achieved a strongly-polynomial-time algorithm by showing that the function representing the number of people evacuated by a given time is submodular. Such a function can be optimized using the parametric search technique of Megiddo. Their approach is different from ours in that they are given an explicit representation of the flow network as input. We are not given this, and computing the flow network that is implicit in our input can take exponential time. Also, their algorithm takes polynomial time in the sum of the input and output sizes. However, the complexity of the output can be exponential in the input size.

Another, perhaps more surprising, related problem is machine scheduling. Viewed simply, confluent flows correspond to non-preemptive scheduling problems, while non-confluent flows correspond to preemptive scheduling problems. The NP-hardness result for optimal confluent flows is inspired by the hardness of scheduling jobs on non-preemptive machines [5], while the worst-case ratio

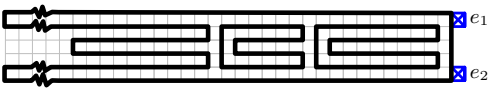

Fig. 1. The polygon P given a PARTITION instance of $\{11, 6, 9\}$. To keep the picture a manageable size, the elements have not been scaled and the left ends of the first and fifth rows are truncated.

between confluent and non-confluent flows is inspired by the list-scheduling approximation ratio [6].

2 Confluent Flows

As mentioned in the introduction, in a confluent flow, every grid square has the property that all people that pass through it use the same exit.

In this section, we present our results related to confluent flows. First, we show the NP-completeness of the problem of finding an optimal evacuation plan with confluent flows in a polygon with holes. This holds even for polygons with two exits. We then give a linear-time algorithm for polygons with two exits.

2.1 Hardness

Weak NP-hardness with two exits. We first sketch that the evacuation problem with confluent flows is NP-hard if we allow P to have holes. We reduce from the well-known NP-complete problem PARTITION; the idea is shown in Fig. 1.

Theorem 1. *The problem of finding an optimal confluent flow in a polygon with holes is NP-complete.*

Strong NP-hardness. Theorem 1 shows that the problem is *weakly NP-complete*. This means that the hardness of the problem depends on the areas of subpolygons being exponential in the complexity of the input. This implies that a pseudo-polynomial algorithm might exist.

However, we can show that if we allow $O(n)$ exits, the problem is *strongly NP-complete*. This means that the problem is still NP-complete when all of its numerical parameters are polynomially bounded in the size of the input.

Our reduction is from CUBIC PLANAR MONOTONE 1-IN-3 SATISFIABILITY (or CPM 1-IN-3 SAT for short). This is a variant of the SATISFIABILITY problem in which every clause contains exactly 3 literals, every variable is in exactly 3 clauses, the graph generated by the connections of variables to clauses is planar, every literal in every clause is non-negated, and a clause is satisfied if exactly one of its variables is true.

We use a reduction that is almost equivalent to the one showing that tiling a finite subset of the plane with right trominoes is NP-complete [11]. Details are contained in the full paper [4].

Theorem 2. *The problem of finding an optimal confluent flow in a polygon with holes and $O(n)$ exits is strongly NP-complete.*

2.2 Two Exits

When the polygon P has no holes and only two exits, e_1 and e_2, we can find an optimal confluent flow in $O(n)$ time. We first present an algorithm that takes cubic time that can be modified fairly easily into an algorithm that takes linear time.

Naïve algorithm. Notice that the case in which P has two exits is simpler than the case in which P has more exits because of the fact that both exit classes must each have one contiguous connection to the boundary of P.

We begin with a decomposition of P into rectangles. This decomposition is the overlay of two simpler decompositions: the vertical and horizontal decompositions. The vertical decomposition of a rectilinear polygon P is the partition of P into rectangles by the addition of only vertical line segments. Similarly, the horizontal decomposition of P is the partition of P into rectangles by the addition of only horizontal line segments. We call the overlay ω and its dual graph ω^*. We add the vertical and horizontal line segments from the grid points on opposite sides of e_1 and e_2 to ω as well. We then do the following for every pair of rectangles r_1 and r_2 in ω that have at least one edge of the boundary of P. We first ensure that e_1 is between r_1 and r_2 and that e_2 is between r_2 and r_1. We also ensure that there are no bottlenecks of size 1 between r_1 and r_2. If either of these conditions are not met, we proceed to the next pair of rectangles. We then set the e_1-exit class to be all the grid squares along the boundary of P between r_1 and r_2. We then add all the grid squares surrounded by the e_1-exit class to the e_1-exit class. We call the area of the e_1-exit class A_1. We define the e_2-exit class similarly and call its area A_2. We call the larger of the two areas A_ℓ and its corresponding exit e_ℓ. If A_ℓ is less than $A/2$, we can divide the rest of the grid squares evenly among the exit classes—see Lemma 2 for details—and return the solution. Otherwise, we attempt to make the e_ℓ-exit class as small as possible (while staying above $A/2$) inside r_1 and r_2 and assign the rest of the grid squares to the other exit class. We maintain a variable that tracks the area of the smallest such exit class e_{min}. If we get through all the possible pairs r_1 and r_2 without finding a pair that we can return, we return e_{min}. The proof of the following summarizing lemma can be found in the full paper.

Lemma 1. *The algorithm presented above is correct and takes $O(n^3)$ time.*

Linear algorithm. The algorithm above has two steps that lead to it taking cubic time: the loop over all pairs of rectangles on the boundary of P and the computation of the minimum area for an exit class that has a connection to the boundary that begins in one of the rectangles and ends in the other. In this section, we sketch a more clever solution that avoids these problems; again, full details are described in the full paper.

We begin by observing that if we update the area, each time we change the starting and ending points of the connection of the e_1-exit class to the boundary of the polygon rather than computing it anew, the total time spent computing the area depends on the sum of the complexities of the updated areas.

We also observe that we loop over the rectangles of ω with at least one edge attached to the boundary of P. This means that we do not really need to compute the entire overlay ω—only the intersections of ω with the boundary of P. These intersections can be computed in $O(n)$ time by computing the vertical and horizontal decompositions of P separately. See Fig. 2 for the idea.

Theorem 3. *In the confluent setting, the above algorithm finds the optimal evacuation plan for a polygon P with two exits in $O(n)$ time, where n is the number of vertices in P.*

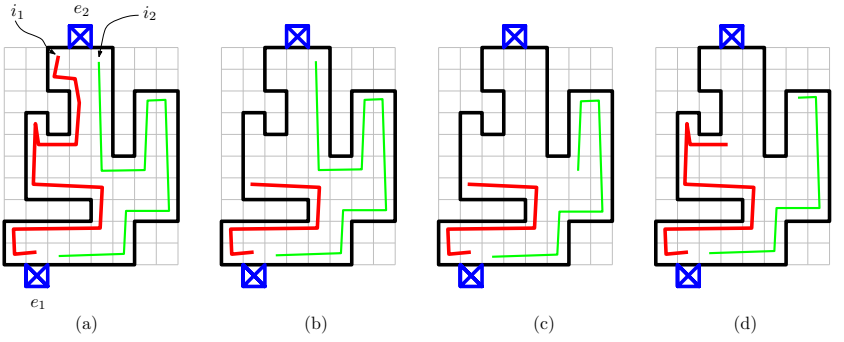

Fig. 2. An example of the linear-time algorithm. The pointers i_1 and i_2 denote the endpoints of the connection of the e_1-exit class to the boundary of the polygon.

If the overall number of grid pixels is exponential in the number of vertices of the polygon, particular care is necessary to ensure a small output complexity. Using a refined output encoding (which is described in the full paper), it is possible to note the following.

Lemma 2. *The above algorithm runs in $O(n)$ time and has output size that is linear in the input size.*

We conjecture that one can use algorithms similar to the naïve algorithm given above to compute the evacuation of any polygon with a constant number of exits, but the details become much more involved. We therefore leave this question to future work.

3 Non-confluent Flows

Compared with confluent flows, non-confluent flows are clearly a stronger model. We note that any confluent flow is a non-confluent flow, but not *vice versa*. We show that non-confluent flows can be as much as $2 - 2/(k + 1)$ times as fast as confluent flows by giving an example in which this is the case. We then argue that the ratio our example achieves is optimal.

3.1 Pseudo-polynomial Algorithm

In contrast to the case with confluent flows, for which we showed that finding an assignment of people to exits is strongly NP-complete when we are dealing with polygons with holes and $O(n)$ exits, we can show that, for non-confluent flows, a pseudo-polynomial algorithm exists.

The algorithm is based on the technique of using time-expanded networks to compute flows over time [13]. Therefore, we compute a flow network from the input polygon as follows. We create a source vertex s and a sink vertex t. For each grid square in P, we create two vertices—an *in* vertex and an *out* vertex. We connect the in vertex to the out vertex with an edge that has capacity 1 for every grid square. We then make, for some integer $T \geq 1$, T copies of the polygon $P_1, \ldots, P_T$, where each copy has these vertices and edges added. For every grid square of P_1, we connect s to the in vertex of the grid square with an edge that has capacity 1. We then connect the out vertex of every grid square in P_i to the in vertex of all its neighbors in P_{i+1} for all $1 \leq i \leq T-1$. Again, the edges we use all have capacity 1. Finally, we connect the out vertex of every exit to t with an edge that has capacity 1. We call this flow network G.

It is fairly easy to see that if we are able to find a maximum flow of value A through G, then we are able to evacuate P in T time steps. However, we note that both T and $|G|$ can be exponential in the complexity of P, making this a pseudo-polynomial algorithm.

Theorem 4. *There exists a pseudo-polynomial algorithm to find an evacuation of a polygon with a non-confluent flow.*

3.2 Differences to Confluent Flows

The example that shows a large gap between confluent and non-confluent flows is a horizontal rectangle of width 1 with length $2k + mk$, for some integer $m \geq 1$. Attached to this rectangle are k vertical rectangles of width 1 and length mk—one at every other square for the first $2k$ squares. Between each vertical rectangle is an exit. Each exit can only be entered from the square to the left. See Figure 3 (a).

We can see that the example has an optimal confluent flow that requires $2mk + 3$ time steps: three to remove the people directly to the left of each exit and all the people below the exits, mk to remove the people in the vertical rectangles, and another mk for the people in the horizontal "tail" to go through the rightmost exit. On the other hand, in the optimal non-confluent flow, all exits can remain continuously busy. One way that this can happen is for m people from the horizontal rectangle to leave through successive exits, while people from the vertical rectangles are leaving through the other exits. Since the exits are continuously busy, the amount of time for all people to leave is $(k^2m + (2+m)k)/k = mk + 2 + m$. The ratio between the confluent and non-confluent flows in this case is

$$\frac{2mk+3}{mk+m+2} = 2 - \frac{2m+1}{mk+m+2} \underset{m\to\infty}{\to} 2 - \frac{2}{k+1}\,.$$

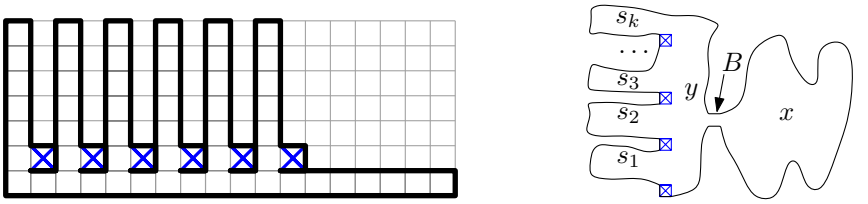

Fig. 3. (a) A polygon whose optimal non-confluent flow is nearly twice as fast as its optimal confluent flow. (b) The general shape of any polygon that realizes the maximum ratio between the confluent and non-confluent flows.

We now show that the ratio achieved in this example is tight. Our ratio is similar to (and inspired by) the upper bound for the list-scheduling approximation ratio [6] in machine scheduling.

Theorem 5. *The maximum ratio between the confluent flow and non-confluent flow in any grid polygon P is $2 - (2/(k+1))$.*

Proof. Let the ratio between the confluent flow and the non-confluent flow for a given polygon P be known as R_P. When calculating R_P, we assume that the confluent and non-confluent flows are calculated optimally for P.

We begin by observing that by reducing the size of the smallest bottleneck in a polygon P can only increase R_P. Suppose we have polygons P_a and P_b, where P_a has a minimum bottleneck size of at least 2, and P_b is the same as P_a, except that one grid square has been removed from the minimum bottleneck. The number of exit classes on one side of the smallest bottleneck in the confluent case can only decrease in P_b relative to P_a, while in the non-confluent case, they may stay the same. Thus, there may be an exit in the non-confluent solution that is used for longer than in the confluent solution. This implies that the number of steps that is required to evacuate the building in the confluent setting increases faster than the number of steps required under the non-confluent setting. Therefore, R_{P_a} is at least as large as R_{P_b}, and may be larger.

This implies that R_P is maximized when the size of the minimum bottleneck is minimized, so we can assume that the size of the minimum bottleneck is 1. We call the subpolygons on either side of the bottleneck P_1 and P_2.

Furthermore, we can easily see that increasing the difference between the number of exits in P_1 and P_2 can only increase R_P. This is because the number of people that must go through the bottleneck that separates P_1 and P_2 can only be increased by increasing this difference. Therefore, we can assume that all the exits are on one side of the bottleneck between P_1 and P_2. Without loss of generality, assume that all exits are in P_1.

Given this setup, we attempt to construct P so that as many exits as possible are used during as many time steps as possible in the non-confluent case. This implies that there must be some source of people in P_1. This is because only

one person can go through the bottleneck between P_1 and P_2 per time step. Therefore, at most one person from P_2 can reach an exit per time step. However, by creating a supply of people in P_1, we allow the people from P_2 to queue in front of the exits.

So that the people from P_2 can queue in front of the exits, the route taken by the people in P_1 from the supply to the exit must not interfere with the paths of the people from P_2 to the exits. This means that the number of people in P_1 must be split and distributed to each exit.

Therefore, P has the form sketched in Figure 3(b). There is a bottleneck B. The number of people behind B is x, the amount of space for these people to queue in is y, and the supply of people for each exit e_i is s_i.

In both the confluent and non-confluent solutions, it takes time $2y/k$ to remove the y people in the queueing area. In the non-confluent solution, it is necessary that this is the first step performed. After this is done, the people from behind the bottleneck begin entering the queueing area. The people must therefore take turns exiting from the queueing area and exiting from the supplies that are attached to each exit. This implies that y is as small as possible (while satisfying $y \geq 2k$) and that $s_i \geq x$ for all $1 \leq i \leq k$. Using different values of s_i yields no advantage, so we assume that the value of s_i is some value s_x for all i.

The ratio between the confluent flow and non-confluent flow is thus

$$\frac{2y/k + 2s_x + 2x}{2y/k + 2s_x + 2x/k}$$

which is maximized according to our constraints when $s_x = x$ and when $y = 2k$. This gives a ratio of

$$\frac{2x + 2k/k}{x + x/k + 2k/k} = \frac{2x + 2}{x + x/k + 2} = 2 - \frac{2x + 2k}{xk + x + 2k} \underset{x \to \infty}{\to} 2 - \frac{2}{k+1}$$

which is the claimed result. □

4 Conclusions

We have discussed evacuations in grid polygons. We first showed that finding evacuations with confluent flows in polygons with holes is hard, even for polygons with only two exits. We then looked at algorithms to find evacuations with confluent flows. Finally, we showed that, while the difference between confluent and non-confluent flows is potentially significant, it is bounded.

Our work raises some questions that require further study. For simple polygons, there is evidence that a constant number of exits allows strongly polynomial solutions, even though some of the technical details are complicated. What is the complexity of finding an evacuation plan with a confluent flow when the number of exits is not constant? Next, can we find an fixed-parameter-tractable algorithm to find the confluent evacuation of polygons? Finally, can we find a polynomial algorithm that gives the optimal evacuation using non-confluent flows? Note that it is not even clear that the output size of such an algorithm is always polynomial.

Acknowledgments

We thank Estie Arkin, Michael Bender, Joe Mitchell, and Martin Skutella for helpful discussions; Martin Skutella is also part of ADVEST.

References

1. Bast, H., Hert, S.: The area partitioning problem. In: Proc. 12th Can. Conf. Comput. Geom. (CCCG 2000), Fredericton, NB, Canada, pp. 163–171 (2000)
2. Baumann, N., Skutella, M.: Earliest arrival flows with multiple sources. Math. Oper. Res. 34(2), 499–512 (2009); Journal version of 2006 FOCS article Solving evacuation problems efficiently
3. Becker, R., Lari, I., Lucertini, M., Simeone, B.: Max-min partitioning of grid graphs into connected components. Networks 32(2), 115–125 (1998)
4. Fekete, S.P., Gray, C., Kröller, A.: Evacuation of rectilinear polygons. Technical report (2010), `http://arxiv.org/abs/1008.4420`
5. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, New York (1979)
6. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics 17, 416–429 (1969)
7. Győri, E.: On division of graphs to connected subgraphs. Combinatorics, Keszthely, 485–494 (1978)
8. Lovász, L.: A homology theory for spanning trees of a graph. Acta Mathematica Hungarica 30(3), 241–251 (1977)
9. Lumelsky, V.: Polygon area decomposition for multiple-robot workspace division. Int. Journal of Computational Geometry and Applications 8(4), 437–466 (1998)
10. Ma, J., Ma, S.: An $O(k^2n^2)$ algorithm to find a k-partition in a k-connected graph. Journal of Computer Science and Technology 9(1), 86–91 (1994)
11. Moore, C., Robson, J.: Hard tiling problems with simple tiles. Discrete and Computational Geometry 26(4), 573–590 (2001)
12. Salgado, L.R., Wakabayashi, Y.: Approximation results on balanced connected partitions of graphs. Electr. Notes in Discrete Mathematics 18, 207–212 (2004)
13. Skutella, M.: An introduction to network flows over time. In: Research Trends in Combinatorial Optimization, pp. 451–482. Springer, Heidelberg (2009)
14. Suzuki, H., Takahashi, N., Nishizeki, T.: A linear algorithm for bipartition of biconnected graphs. Information Processing Letters 33(5), 227–231 (1990)