Performing Multicut on Walkable Environments

Obtaining a Minimally Connected Multi-Layered Environment from a Walkable Environment

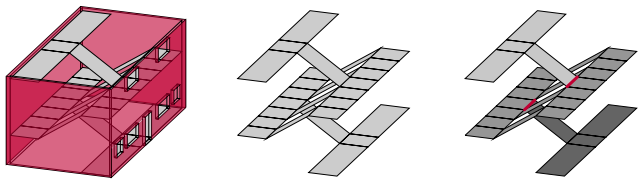
Arne Hillebrand, Marjan van den Akker, Roland Geraerts, and Han Hoogeveen

Institute of Information and Computing Sciences, Utrecht University, 3508 TA Utrecht, the Netherlands
{A.Hillebrand,J.M.vandenAkker,R.J.Geraerts,J.A.Hoogeveen}@uu.nl

Abstract. A multi-layered environment is a representation of the walkable space in a 3D virtual environment that comprises a set of two-dimensional layers together with the locations where the different layers touch, which are called connections. This representation can be used for crowd simulations, e.g. to determine evacuation times in complex buildings. Since the execution times of many algorithms depend on the number of connections, we will study multi-layered environments with a minimal number of connections. We show how finding a minimally connected multi-layered environment can be formulated as an instance of the multicut problem. We will prove that finding a minimally connected multi-layered environment is an NP-Hard problem. Lastly, we will present techniques which shrink the size of the underlying graph by removing redundant information. These techniques decrease the input size for algorithms that use this representation for finding multi-layered environments.

1 Introduction

Evacuation planning and crowd simulations for safety purposes are becoming more and more important in modern society. To perform such simulations in a soccer stadium for example, we need its underlying *polygonal environment* (PE), which is a common format used by architects [17] and 3D modelling tools. A PE is a collection of polygons in $\mathbb{R}^3$ which can be processed through a pipeline to mould it in an appropriate format; an example of such a pipeline is shown in Fig. 1. As is shown in Fig. 1(a), such a PE usually contains unnecessary details for simulations. We only need a filtered version of the PE that contains the polygons that are traversable. Examples of polygons that are not needed in the *walkable environment* (WE) are polygons that are too steep, too close to a ceiling or polygons for which there is not enough clearance for a character to be positioned on the polygon. We assume, without loss of generality, that all polygons $P \in$ WE are convex. Furthermore, we assume that the WE is clean, that is, there is no intersecting or degenerate geometry. Both these properties can be guaranteed when extracting the WE from the PE. An example of a WE is shown in Fig. 1(b). Polygons in a WE can either overlap or be connected.



(a) PE (b) WE (c) MLE

Fig. 1: (a): A polygonal environment. (b): The walkable environment of this model. (c): A multi-layered environment for this walkable environment. Polygons with the same shade of grey are in the same layer. The red edges are connections.

Two polygons *overlap* when they share at least one point when projected on the ground plane that is not on an edge of both polygons. The polygons P and Q are *connected* when they do not overlap and share exactly one edge $e_{P,Q}$. When P and Q are connected, it is possible for a virtual point character to move from P to Q and vice versa.

On a WE we want to perform operations such as constructing visibility graphs [10], which are used for finding shortest paths in the environment, or for creating navigation meshes [15], which enable fast path planning queries that are used for crowd simulations. However, many operations currently are limited to two-dimensional environments. These operations can be extended to layered two-dimensional environments by using a *multi-layered environment* (MLE). An MLE is a decomposition of the WE in layers, such that every layer can be embedded in the plane. When two polygons P and Q share an edge $e_{P,Q}$ and do not overlap but are in different layers, $e_{P,Q}$ connects the two layers. This type of edge is called a *connection*. This set of edges is also stored for an MLE.

Definition 1 (Multi-Layered Environment, Van Toll et al. [16]). *An MLE for a given WE consists of a set* $\mathbf{L} = \{L_1, ..., L_l\}$ *of two-dimensional layers and a set* $\mathcal{C}$ *of connections, such that:*

- *No layer* L_i *(*$i = 1, \ldots, l$*) contains overlapping polygons;*
- *Every polygon in the WE is assigned to exactly one layer* L_i *(*$i = 1, \ldots, l$*);*
- *When two polygons* P *and* Q *are connected, but are in different layers, the connection between* P *and* Q *is part of the set* $\mathcal{C}$*;*
- *Every layer* L_i *(*$i = 1, \ldots, l$*) forms a single connected component, i.e. for any two polygons* P*,* Q *in* L_i *we can walk from* P *to* Q *without leaving* L_i*.*

When we use this definition of an MLE, it is rarely the case that there exists only one possible MLE for a WE. Take for example the MLE given in Fig. 1(c), consisting of three layers and two connections. Another possible MLE for this WE has only two layers, but needs four different connections. We call an MLE with the minimal number of connections a *minimally connected multi-layered environment* (MICLE). In this paper we will focus on MICLEs. This is the most useful type of MLE because subsequent operations are dependent on the

number of connections. For example, van Toll et al. [16] show that constructing a navigation mesh for an MLE can be done in $O(k \times n \log n)$ time, where k is the number of connections in an MLE and n is the number of obstacle points used to describe the boundaries of the individual layers of the MLE.

1.1 Related Work

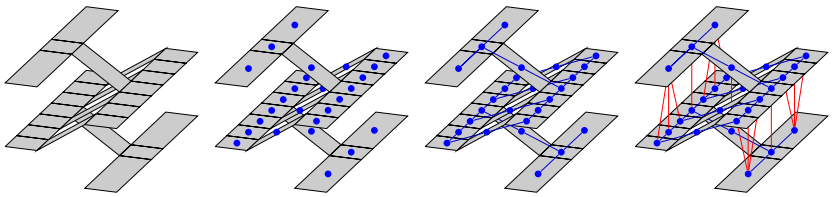
There are several applications and algorithms that are already using some form of an MLE. However, the MLEs that they use are often of poor quality. They do not cover all of the walkable space or contain a high number of connections. Van Toll et al. [16] use an MLE to create a multi-layered navigation mesh, which allows for fast path planning queries. Their MLE is a decomposition of the walkable environment into layers. One strength of this type of MLE is that the representation of the corresponding WE is exact. However, they do not describe any methods to find an MLE with a low number of connections.

Deusdado et al. [3] use a discretized height map to automatically extract walkable surfaces from a PE. The locations of the connections are determined by comparing the height information of different walkable surfaces. Oliva and Pelechano [11] overlay the environment with a three-dimensional grid and mark each grid cell positive when it contains walkable geometry. These grid cells are grouped into layers which are then used to create a multi-layered navigation mesh. Pettré et al. [12] create multiple elevation maps of a PE by using the graphics card. From these elevation maps, slopes that are too steep are filtered and the elevation maps are joined, resulting in a WE. The downside of these techniques is that they are not general enough and that the resulting MLE is only an approximation of the WE and does not cover all of the WE.

Instead of extracting an MLE from a PE or WE, Jiang et al. [9] propose a method which models the environment in simple blocks that can be described in two dimensions. The blocks are linked together in a floor plan-like fashion. When an environment is described this way, an MLE is easy to extract since the layers and connections are explicitly defined.

1.2 Our Contribution

In Sect. 2, we will show that the search for a MICLE can be solved in theory by using *multi-commodity minimal-cut* (`MULTICUT`). For this we will encode the WE as a graph. `MULTICUT` is a problem in the class of NP-Hard problems [1], for which fixed parameter tractable (FPT) algorithms exist [5]. We will prove that finding a MICLE is also in the class of NP-Hard problems. In Sect. 3 we will identify situations in which edges, vertices and overlaps can be removed from the graph, without influencing the size of the cut needed for finding a valid MICLE. We have also implemented algorithms to handle these situations and we experimentally evaluate them in Sect. 4. The graph reduction algorithms have varying results on real-world environments.



(a) WE (b) Vertices added (c) Edges added (d) Overlaps added

Fig. 2: Constructing the graph representation of a walkable environment. (a): The walkable environment. (b): Vertices are added for every polygon in the walkable environment. (c): Undirected edges are added for connected polygons. (d): Overlapping vertices are annotated (represented by red edges).

2 Finding a MICLE

We first convert the WE into a graph $G = (V, E, O)$. In this graph, a vertex is added to V for every polygon in the WE. An undirected edge (v, w) is added to E whenever the polygons corresponding to the vertices v and w are connected in the WE. When two polygons with corresponding vertices v' and w' overlap, the unordered pair (v', w') is added to O. This process is illustrated in Fig. 2. The resulting graph is called the *walkable environment graph* (WEG). Sometimes, we want to assign different weights to edges in the WEG. One such situation will be described in Sect. 3. For a weighted WEG we have $G = (V, E, O, w)$. Here w is a function that maps every edge $e \in E$ to a real number. The weight of the cut set $\mathcal{C}$ is defined as $|\mathcal{C}| = w(\mathcal{C}) = \sum_{e \in \mathcal{C}} w(e)$. Using the WEG, the problem of finding a MICLE can now be formulated as follows:

Problem 1 (Finding a MICLE). Given WEG $G = (V, E, O)$ of WE W. Finding a MICLE is now the same as finding the set $\mathcal{C} \subseteq E$ for which:

- $\forall v, w$ such that $(v, w) \in O$: v and w are in different graph components for the graph $G' = (V, E \setminus \mathcal{C})$;
- $|\mathcal{C}|$ is minimal.

From the different connected components in the graph $G' = (V, E \setminus \mathcal{C})$, we can construct the different layers in the MLE. When $|O| = 1$, this problem is the same as the s-t cut problem, which can be solved using the max-flow min-cut theorem [4]. If $|O| > 1$, we have an instance of the `MULTICUT` problem [14]. In the `MULTICUT` framework, multiple source-sink pairs (s_i, t_i) exist, where each pair has to be separated. By using `MULTICUT` it is possible to find an MLE with a minimal number of connections for a WEG by using the overlapping pairs $(v, w) \in O$ as the terminal pair set. Unfortunately, `MULTICUT` has been proven to be NP-Hard when $|O| \geq 3$ [1]. For $|O| = 2$, polynomial time algorithms exist to solve `MULTICUT` [8]. However, since finding a MICLE is a special case of `MULTICUT`, we still need to prove that finding a MICLE for a WEG is indeed an NP-Hard problem. We will show this below.

2.1 Preliminaries

The proof that finding a MICLE is in the class of NP-Hard problems is heavily based on the work done by Dahlhaus et al. [2] on the MULTITERMINAL-CUT (MTC) problem. The decision version of this problem is defined below:

Definition 2 (MULTITERMINAL-CUT, Dahlhaus et al. [2]). *Given a graph $G = (V, E, w)$, a terminal set $T \subseteq V$ and a positive bound B. The decision version of MTC is now the same as finding a set $E' \subseteq E$ for which:*

- *$\forall v, w \in T$ such that $v \neq w$: v and w are in different components for the graph $G' = (V, E \setminus E')$;*
- *$\sum_{e \in E'} w(e) \leq B$.*

To prove that the MTC problem is NP-Hard, Dahlhaus et al. first prove a restricted version of PLANAR 3-SAT (P3R) to be NP-Complete.

Definition 3 (P3R, Dahlhaus et al. [2]). *Given a set of variables $X = \{x_1, \ldots x_n\}$ and set of clauses $C = \{c_1, \ldots, c_m\}$, where:*

- *$\forall c_j \in C$: $c_j = (x_k \vee x_l)$ or $c_j = (x_k \vee x_l \vee x_m)$ for some $x_k, x_l, x_m \in X$;*
- *$\forall x_i \in X$: x_i occurs exactly once in three different clauses, and;*
- *$\forall x_i \in X$: Both the literals x_i and $\bar{x}_i$ occur at least once.*

Solving P3R for a formula $F = c_1 \wedge \ldots \wedge c_m$ is now equal to finding a suitable truth assignment T for all the variables $x_i \in X$ such that F equals true.

Dahlhaus et al. show how an instance of P3R can be reduced to the decision version of MTC. Some of the widgets that they use are given in Fig. 3. In this figure, the circles represent the terminals of the MTC problem. Every variable x_i with two x_i literals is replaced with the widget shown in Fig. 3(a). The clauses of size three are replaced with the widget from Fig. 3(c). Similar widgets exist for the clauses of size two and for the variables with two $\bar{x}_i$ literals.

Next, the widgets representing the variables and the clauses are connected using weight two *link edges*. The link edges are attached to the vertices labelled $l_{i,x}$, $\bar{l}_{i,1}$ and $\hat{l}_{j,x}$ $(x = 1, 2, 3)$. For details on how these widgets are connected using the link edges, we refer the reader to reference [2]. The component induced by the vertices of the link edge and the neighbours of these vertices is called the *link structure*. While proving that the decision version of the MTC problem is NP-Complete, Dahlhaus et al. also proved the following two lemmas:

Lemma 1 (Dahlhaus et al. [2]). *Given are a P3R instance (X, C) with X the set of variables and C the set of clauses. Let $G_{X,C}$ be the bipartite graph representing this instance and $G'_{X,C}$ the transformed graph of $G_{X,C}$. There exists a set of edges E' with total weight $B = 10|X| + 4|C|$ separating the terminals in $G'_{X,C}$ if and only if the P3R instance (X, C) has a satisfying truth assignment.*

Lemma 2 (Dahlhaus et al. [2]). *When a cut set E' for MTC is found where the separation cost $\sum_{e \in E'} w(e) \leq B$, E' only contains edges from the connector triangles or link edges. Furthermore, for each link structure it will only contain one of the connector triangles or the link edge.*

These two lemmas are of key importance for the proof that finding a MICLE is an NP-Hard problem.

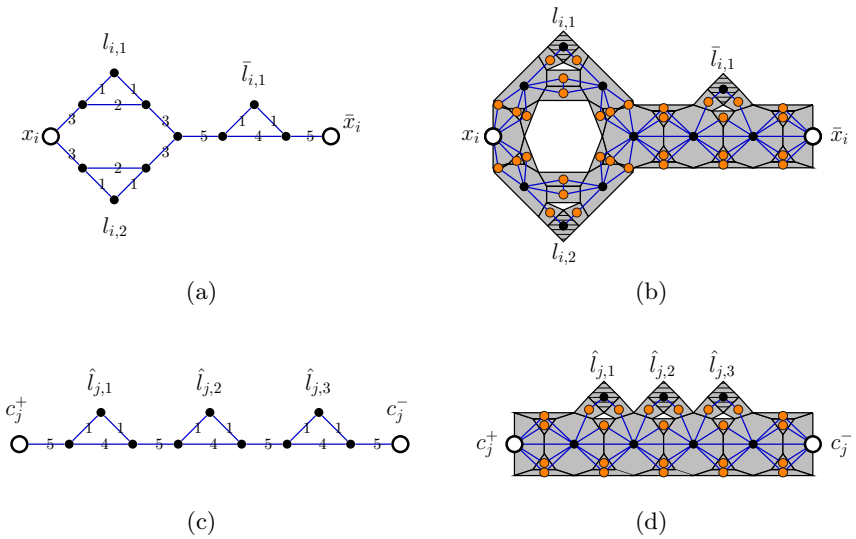


Fig. 3: The widgets used by Dahlhaus et al. [2] and their polygonal counterparts. The terminals are depicted by circles and the vertices by discs. The widget in (a) is used for a variable x_i with two x_i literals. The widget in (c) is used for clauses with three variables. For the polygonal counterparts (Figs. (b) and (d)), the newly created vertices are coloured orange.

2.2 Finding a MICLE is NP-Hard in the Strong Sense

First, we will show that the transformed graph $G'_{X,C}$ can exist as a WEG. Next, we will prove that it is possible to encode the terminals of MTC in a WE. Finally, we will show that this can be done in polynomial time.

Since $G'_{X,C}$ is planar, the components from this graph can easily be represented using the polygonal structures given in Figs. 3(b) and 3(d). The link edges connecting the link vertices $l_{i,x}$ or $\bar{l}_{i,x}$ to $\hat{l}_{j,x}$ $(x = 1, 2, 3)$ can be represented using a series of polygons. Since the link edges have weight two, two of such series of polygons are required. These series are attached to the only two free sides of the tetragons $l_{i,x}$ or $\bar{l}_{i,x}$ and $\hat{l}_{j,y}$. The tetragons that represent these vertices are hatched in Figs. 3(b) and 3(d). Every link edge can be represented by using at most five polygons, since there exists a straight line drawing for every planar graph, and it can be found in $O(|V|)$ time [6]. It is easy to see that the resulting WEG has the same properties as the components given by Dahlhaus et al., that is, separating any pair of the vertices from the WEG that directly corresponds to vertices from the original widgets will result in a cut of exactly the same weight.

All the terminals can be encoded by one big set T_{tower} of polygons. The required number of polygons in this set is $2|X| + 2|C|$, one for each terminal in $G'_{X,C}$. These polygons should all be centred around one point c such that, when all the polygons are projected on the ground plane, they all contain this

point. As a result, all these polygons will be placed in different components when searching for any MLE.

The next step is connecting the polygons from T_{tower} to the polygons where the terminals of the MTC problem are located. These are labelled x_i, $\bar{x}_i$, c_j^- and c_j^+ in Figs. 3(b) and 3(d). This should be done in such a way that Lemma 1 remains valid and the final polygonal environment is a WE. The validity of Lemma 1 remains when we connect each terminal in T_{tower} to one of the terminals of MTC using a single weight 7 edge. Cutting such an edge will always increase the weight of the cut set E' to a value higher than B. Edges of weight 7 are sufficient, since the edges of weight 3 and weight 5 connected to the terminals of MTC are guaranteed not to be part of the cut set E' if a satisfying truth assignment exists (Lemma 2).

When we represent these edges of weight 7 by 7 polygonal paths, the polygons can overlap parts of clause and variable structures, generating new overlaps that have to be separated. Fortunately, we can create paths that guarantee that Lemma 1 and Lemma 2 will still hold, even when these paths are added to the WEG.

Lemma 3. *We are given a former terminal t, a path p connecting t to the new terminal $t' \in T_{tower}$ and the polygonal structure S that contains t, the polygonal widget it belongs to and its link structures. If p does not overlap S or itself, Lemma 1 and Lemma 2 still hold.*

Proof. Since Lemma 1 and Lemma 2 do not state anything about the scenario where no satisfying truth assignment for P3R exists, we do not need to consider this scenario. For this reason, we will assume that there exists a satisfying truth assignment for P3R.

Note that the path p must be one of the 7 polygonal paths that connects t to t'. Therefore, cutting only path p will increase the weight of the cut set but not make t' and t disconnected in the WEG. Therefore, cutting p will also mean cutting the other 6 paths connecting t and t'. We already know that these 7 paths will not be cut when separating all terminals in T_{tower}, unless there is no possible truth assignment for P3R. Furthermore, we know from Lemma 2 that the edges from the link structures separate all the terminals. This also means that all paths connecting vertices in a variable or clause structure will be separated from any other variable or clause structure. Since p connects the former terminal t and the new terminal t', we know that cutting the correct link structures will separate t' from all other terminals in T_{tower}. The terminals that are created when p overlaps parts of the WE are also cut, since we know that these overlaps are connected through link structures, and all link structures are cut. Therefore, both Lemma 1 and Lemma 2 must hold. □

It is easy to see that these paths exists. We know that the various clause and variable structures can be connected using straight paths [6], not considering the polygons from T_{tower} or the paths connecting these structures to T_{tower}. This follows from the fact that we started from an instance of P3R, which can be embedded in the plane. Furthermore, the only non-planar structures are T_{tower}

and the paths that lead to it. We also know that the paths connecting the former terminals to the new terminals in T_{tower} may overlap. Therefore, it is sufficient to have these paths skim the border of the polygonal structure S, until a straight path to T_{tower} is possible. This results in the following theorem:

Theorem 1. *The decision version of finding a MICLE is NP-Complete and finding a MICLE is NP-Hard.*

Proof. First we use the graph $G'_{X,C}$ of Dahlhaus et al. [2]. This graph can be represented using 45 polygons for every variable, 20 polygons for every clause with 2 variables and 32 polygons for every clause of three variables. Every link edge can be represented using 5 polygons, and there are $2|X|$ link edges in total. The number of added polygons to create the link structures and components is linear in $|X|+|C|$; the same holds for the needed number of polygons for T_{tower}. The polygons will also encode the $2(|X|+|C|)$ terminals from the `MTC` problem. For creating the paths to T_{tower}, at most another number of polygons linear in $|X|+|C|$ is needed.

To complete the proof, bounds for the terminal-pairs have to be given. The number of added terminal-pairs while creating the paths to T_{tower} is at most $O((|X|+|C|)^2)$. As a result, the decision version of finding a MICLE is NP-Complete, and, therefore the optimization version of finding a MICLE is NP-Hard. □

3 Reducing the Size of the WEG

In this section we describe situations in which vertices, edges and overlaps can be removed from a WEG without changing the optimal solution. In Sect. 3.1, we will present techniques to reduce the number of edges in the WEG, and in Sect. 3.2, we will discuss the removal of overlaps that are already enforced by other overlaps of the WEG. In this section we will use a weighted WEG, as defined in Sect. 2. We start with $\forall e \in E : w(e) = 1$. We will use $N_v = \{w|(v,w) \in E\}$ for the set of neighbours of a vertex v in a weighted WEG $G = (V, E, O, w)$. Similarly, $O_v = \{w|(v,w) \in O\}$ is the set of vertices that v overlaps. A simple path between the vertices v and w, will be denoted as $[v \to w]$.

3.1 Reducing the Number of Edges

The first reduction we will discuss is called `1-CONTRACT`. It is only applicable for a vertex v with $|O_v| = 0$ and $|N_v| = 1$. Such a vertex can easily be ignored. There is no reason to add this vertex to any other layer than the layer of its only neighbour.

The second graph simplification applies to a vertex with $|N_v| = 2$ and $|O_v| = 0$. Assume the two neighbours of v are u and w. Since this vertex v has no vertices it overlaps with, merging v and u or v and w will not force any new overlaps. The newly created edge (u, w) does not create any new connections

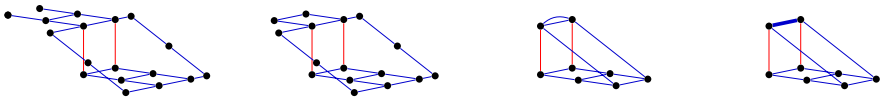


Fig. 4: An example of the application of the contraction operations. In (a) the WEG with which we start is given. In (b), (c) and (d) the WEG is shown after the application of `1-CONTRACT`, `2-CONTRACT` and `E-REDUCE`, respectively. The thick edge in (d) is of weight 2.

between u and w, nor does the minimal cost of separating u and w change. This operation is called `2-CONTRACT`.

The third graph simplification is only useful in combination with the previous simplification. The operation applies to two vertices connected by multiple edges. This situation can not exist in the original WEG because of the convexity constraint for the polygons in a WE. In this situation it would be nice if we could apply the `2-CONTRACT` operation, but this is not possible since the degree of v is three, not two. A simple solution solving this problem is merging all the double edges and combining their respective weights. This method is called `E-REDUCE`. An example of a WEG, and the resulting WEGs after we have applied each of these operations, can be seen in Fig. 4.

3.2 Reducing the Number of Overlaps

A logical next step is removing overlaps from the WEG. This situation will be subdivided into two categories. First, we have the trivial cases, where there is only one possible cut to separate two vertices. Second, there is the case of vertices with degree 2 that overlap. In this scenario several overlaps can be removed under specific circumstances.

Trivial Cases A case is considered trivial when there exists only one vertex-disjoint path between v and w, $(v, w) \in E$, and v and w are overlapping. When this happens, there is only one possible way to separate v and w, which is done by cutting the edge (v, w). Since this action guarantees that v and w are separated, the overlap (v, w) can be removed from O. When there are other vertex-disjoint paths between v and w not containing the edge (v, w), cutting the edge (v, w) will not separate these paths. For this reason, the overlap (v, w) cannot be removed from O in this situation.

Another trivial case is when a vertex v has degree 1 and $(v, w) \in O$. In this scenario the overlap can be removed when all the neighbours of w that are on a vertex-disjoint path $[v \to w]$ are overlapped by the single neighbour of v. In this situation, every path connecting v and w has a subpath connecting a neighbour of v to a neighbour of w. This subpath has to be cut, since the neighbour of v

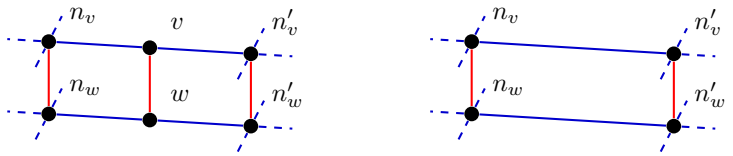


Fig. 5: An example in which the overlap (v, w) can be removed as long as the vertices have degree 2.

is assumed to overlap all the neighbours of w. We call this operation `1-REMOVE`. For a vertex v, we can check if an overlap can be removed in $O(|N_w|)$ time if all members of N_w are overlapped by the single member of N_v, say n_v. If n_v does not overlap all members of N_w, this check is more expensive. We need to check if the members of N_w that are not overlapped by n_v are in a different graph component after the temporary removal of w. This can be done using a Breadth-first search (BFS), resulting in an expensive $O(|E|)$ check to see if this operation is applicable. Performing this step on an entire WEG can take $O(|V||O||E|)$ time. Since $|O|$ can be $O(|V|^2)$, we end up with an $O(|V|^3|E|)$ algorithm, which is a costly operation for larger environments. However, we think that this worst-case scenario rarely happens in practice, because $|O|$ is only $O(|V|^2)$ when almost all polygons overlap all other polygons.

Removing Overlaps from Vertices of Degree 2 When considering a vertex v with degree 2 and its overlaps, some of these overlaps might be removed. Assuming $(v, w) \in O$, this overlap can be removed when, on each path connecting v and w, there is a pair of vertices $(x, y) \in O$. That is, for every possible path connecting v to w, there is a pair of vertices $(x, y) \in O$ and both x and y are on this path. An example of such a situation is given in Fig. 5(a). We will call such a structure a stack, and the operation that removes overlaps from this stack we will call `STACK-REMOVE`. In this situation, all edges have weight one and both v and w have degree two. Furthermore, the neighbours of v and w do also overlap. A formal proof of the validity of `STACK-REMOVE` is given in Appendix A.

Overlaps in similar cases can still be removed from a WEG whenever the edges (n_w, w) and (w, n'_w) have the same weights a, and the edges (n_v, v) and (v, n'_v) also have the same weights b. However, if edges (n_w, w) and (w, n'_w) each have different weights, this is not longer true, since these edges cannot be easily replaced by edges from the original graph.

General Overlap Removal In the general case, overlaps can be removed whenever the separation of overlapping vertices is already forced by the surrounding environment, just as with `STACK-REMOVE`. Unfortunately, checking this requires

an exponential amount of time in worst case scenarios. A simple BFS or DFS will not suffice since not all paths connecting v and w are traversed.

Since such a worst-case runtime is not usable in practice (especially when you consider running that algorithm for every vertex that has an overlap), it is prudent to limit the search depth at the cost of the number of overlaps that will be removed. Such an algorithm is d-`REMOVE`. The parameter d is a bound on the path length considered when searching for overlaps that can be removed.

For each simple path of length d originating in v, we temporarily remove the vertices that are overlapped by vertices on the simple path from the WEG. After doing this, a BFS is started from the last vertex on the simple path. If we encounter a vertex o that overlaps v during this BFS, we remember it. This process is repeated for all the simple paths. For overlaps that were not encountered during this process, there must be vertices on the simple paths of length d that guarantee the separation. Therefore, we can safely remove the overlaps that were not encountered during this process from the WEG.

The running time for this algorithm is $O(x^d \times (|E| + |V| - 2d) + x^d \times y)$, where x is the maximal branching factor and y is the maximal number of overlaps associated with a vertex. In this algorithm, x^d calls are made to a BFS algorithm. Since we already traversed d vertices and edges, the BFS does not have to visit them any more. Registering the encountered overlaps on the simple paths accounts for the remaining $x^d \times y$ time.

4 Experiments and Results

We have implemented the WEG reductions described in Sect. 3 in C++ and tested them on a number of environments. The details of the used environments are given in Table 1. The environments As_oilrig, Library and Parking lot were taken from Saaltink [13], Station 1 and Station 2 were provided by Movares, an engineering and consultancy company. The environments Halo, Cliffsides and Hexagon were taken from the Google Sketchup warehouse[1]. The Tower environment was created by the authors, based on a flat in Utrecht, the Netherlands.

Besides the size of these environments (which we can see in column $|V|$), there are two other important aspects of the environments. The first one is the ratio $\frac{|O|}{|V|}$, which is an indication of how layered the environment is. The second aspect is what types of geometric primitives were used to model these environments. If an environment consists solely of triangles, it might be easier to reduce the underlying graph. We tested our algorithms on models of real buildings (type R) and on game levels (type V).

All our experiments were performed on a machine with an Intel i5-4670 clocked at 3.4 GHz with 16GB of DDR3 RAM. They only used a single thread and were repeated 20 times. The OS and compiler that we used are Linux Mint 13.2 (64 bit) and g++ version 5.3.0, respectively.

[1] `https://3dwarehouse.sketchup.com/model.html?id=`{13c3078fa52d14554b9e17-7bc9ee06a9, 2ac949d235d65acb46697ff0ff0b9b2c, 33b2c337108275568c09573a9753f4-fd}

Table 1: The different environments we have tested. Column **T.** gives the type of environment. V stands for "real" virtual environment and R stands for real world environment. A ✓ in column **Tri.** means that the environment is triangulated.

Environ.	**T.**	**Tri.**	$\|V\|$	$\|E\|$	$\|O\|$
As_oilrig	V	✓	2077	2399	10717
Halo	V	✓	179	184	346
Cliffsides	V	✓	748	764	162
Hexagon	V	✓	2368	2419	20207

Environ.	**T.**	**Tri.**	$\|V\|$	$\|E\|$	$\|O\|$
Library	R	✗	298	420	775
Tower	R	✗	5932	8033	116983
Station 1	R	✓	206	209	1026
Station 2	R	✓	82	86	115

For each environment, we first ran experiments that only used the edge reduction algorithms described in Sect. 3.1. Next we added the overlap reduction algorithms from Sect. 3.2. We did not include the results of experiments that only used the algorithms in Sect. 3.2, because of space limitations. We tested for d from d-`REMOVE` in the range 1 through 8. The results for all environments can be seen in Fig. 6. This figure shows the relative changes for $|V|$, $|E|$ and $|O|$. Here, a value of 1 means that no graph reduction was performed. A value of 0 means that all the vertices (or edges or overlaps) were removed.

The first thing that we notice is that the Cliffsides environment benefits greatly from the edge-reduction algorithms. We believe this to be because of the relatively low ratio of $\frac{|O|}{|V|}$ for this environment. This ratio is a measure of how layered the environment is. Another important factor is the fact that the environment is triangulated, which is also illustrated in Fig. 7. In this figure, we see the Library environment and the corresponding WEG for different values of d. When $d = 5$, there are many vertices that do not have any overlap, but cannot be removed because the edge degree is too high. The edge degree would have been lower, if the environment had been triangulated beforehand.

We can also see that all environments benefit from the use of the overlap-reductions algorithms. There is a steep decline in the number of vertices and edges when it is first used.

5 Conclusion

We have given a definition of MLEs that can be used as input for already existing 2D algorithms. A special type of MLE, the MICLE, can be useful for solving problems in the multi-layered problem domain. Since a MICLE has the lowest number of connections possible, cross-layer operations will occur less frequently. Finding such a MICLE for a given WE is an NP-Hard problem. It is a version of the well studied `MULTICUT`, which is also in the class of NP-Hard problems.

Furthermore, we have described some algorithms that can reduce the size of the WEG. This is accomplished by merging vertices and removing overlaps in such a way, when searching for a MICLE, that we find a solution that is also optimal for the unmodified WEG, or one that can be adjusted to an optimal solution for the unmodified WEG. These algorithms have been implemented and tested for different environments.

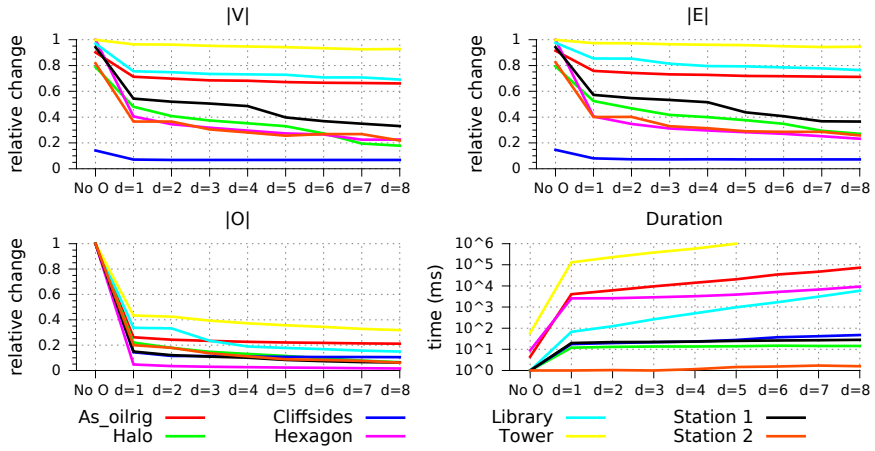


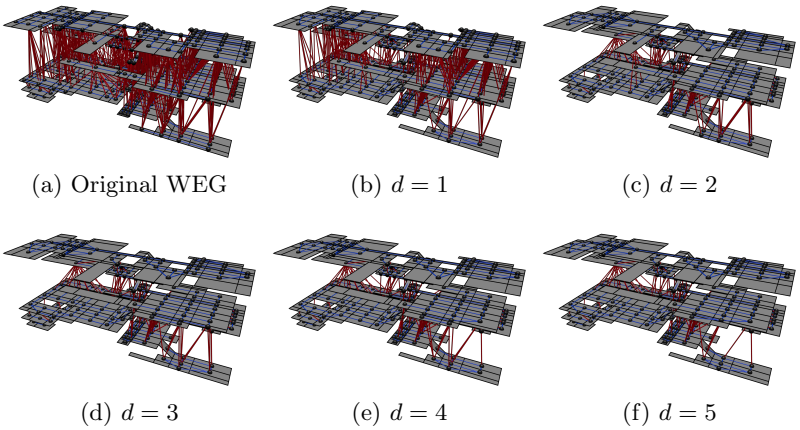
Fig. 6: Plots showing the relative changes of $|V|$, $|E|$, $|O|$ and the running time for the different values of d. For the results in column 'No O' only `1-CONTRACT`, `2-CONTRACT` and `E-REDUCE` were used.

Working with MICLEs can increase the efficiency of operations performed on this MLE. Examples of such operations include, but are not limited to, performing simulations of large crowds or constructing a visibility graphs for finding shortest routes. However, as we have proven, an MLE with the smallest number of connections is hard to find.

Currently, we are investigating techniques that can generate a WE from a PE and strategies that can construct an MLE with a small number of connections from a WE. Our first results on extracting MLEs from WEs can be found in reference [7].

References

1. Călinescu, G., Fernandes, C.G., Reed, B.: Multicuts in unweighted graphs with bounded degree and bounded tree-width. In: Integer Programming and Combinatorial Optimization, pp. 137–152. Springer (1998)
2. Dahlhaus, E., Johnson, D., Papadimitriou, C., Seymour, P., Yannakakis, M.: The complexity of multiterminal cuts. SIAM Journal on Computing 23(4), 864–894 (1994)
3. Deusdado, L., Fernandes, A.R., Belo, O.: Path planning for complex 3D multilevel environments. Proc. 24th Spring Conference on Computer Graphics pp. 187–194 (2008)
4. Ford, L., Fulkerson, D.: Solving the transportation problem. Management Science 3(1), 24–32 (1956)
5. Guo, J., Hüffner, F., Kenar, E., Niedermeier, R., Uhlmann, J.: Complexity and exact algorithms for multicut. In: Software Seminar. pp. 303–312 (2006)
6. Harel, D., Sardas, M.: An Algorithm for Straight-Line Drawing of Planar Graphs. Algorithmica 20, 119–135 (1998)



(a) Original WEG (b) $d = 1$ (c) $d = 2$

(d) $d = 3$ (e) $d = 4$ (f) $d = 5$

Fig. 7: Different WEGs for the Library environment. The black circles are vertices, the blue segments are edges and the red segments are overlaps. In (a) the unmodified WEG is shown. (b) through (e) show the reduced WEGs for different values of d.

7. Hillebrand, A., van den Akker, M., Geraerts, R., Hoogeveen, H.: Separating a walkable environment into layers. In: 9th Int. ACM SIGGRAPH Conf. on Motion in Games (2016), to appear
8. Itai, A.: Two-commodity flow. Journal of the ACM 25, 596–611 (1978)
9. Jiang, H., Xu, W., Mao, T., Li, C., Xia, S., Wang, Z.: A semantic environment model for crowd simulation in multilayered complex environment. ACM Symposium on Virtual Reality Software and Technology (2015), 191–198 (2009)
10. Lozano-Pérez, T., Wesley, M.A.: An algorithm for planning collision-free paths among polyhedral obstacles. Communications of the ACM 22(10), 560–570 (1979)
11. Oliva, R., Pelechano, N.: NEOGEN: Near optimal generator of navigation meshes for 3D multi-layered environments. Computers & Graphics 37(5), 403–412 (2013)
12. Pettré, J., Laumond, J.P., Thalmann, D.: A navigation graph for real-time crowd animation on multilayered and uneven terrain. First International Workshop on Crowd Simulation 47(2), 81–90 (2005)
13. Saaltink, W.: Partitioning polygonal environments into multi-layered environments. Master's thesis, Utrecht University (2011)
14. Schrijver, A.: Combinatorial Optimization - Polyhedra And Efficiency, Algorithms and Combinatorics, vol. 24. Springer (2003)
15. Snook, G.: Simplified 3D movement and pathfinding using navigation meshes. In: DeLoura, M. (ed.) Game Programming Gems, pp. 288–304. Charles River Media (2000)
16. van Toll, W., Cook IV, A., Geraerts, R.: Navigation meshes for realistic multi-layered environments. In: Intelligent Robots and Systems, 2011 IEEE/RSJ International Conference on. pp. 3526–3532 (2011)
17. Whyte, J., Bouchlaghem, N., Thorpe, A., McCaffer, R.: From cad to virtual reality: modelling approaches, data exchange and interactive 3d building design tools. Automation in Construction 10(1), 43 – 55 (2000)

A Proof of STACK-REMOVE

Theorem 2. *Given a weighted WEG $G = (V, E, O, w)$, vertex v with $N_v = \{n_v, n'_v\}$, vertex w with $N_w = \{n_w, n'_w\}$, $\{(v, w), (n_v, n_w), (n'_v, n'_w)\} \subseteq O$ and $w(e) = 1$ for all $e \in E$. The optimal cut set $\mathcal{C}'$ for $G' = (V, E, O', w)$ with $O' = O \setminus (v, w)$, either is also optimal for G, or can be adjusted to an optimal cut set $\mathcal{C}$ for G.*

Proof. Assume we have the WEG $G' = (V, E, O', w)$, with $O' = O \setminus (v, w)$. Furthermore, we will also assume that we have found a MICLE for G' with the cut set $\mathcal{C}'$. Are all paths $[v \to w]$ cut by $\mathcal{C}'$? If not, how can we change $\mathcal{C}'$ without increasing the weight so that these paths are cut? This situation is also illustrated in Fig. 5.

First, we observe that all vertex-disjoint paths $[v \to w]$ can be split into four categories, namely $[v, n_v \to n_w, w]$, $[v, n'_v \to n'_w, w]$, $[v, n_v \to n'_w, w]$ and $[v, n'_v \to n_w, w]$. Categories $[v, n_v \to n_w, w]$ and $[v, n'_v \to n'_w, w]$ will be cut by $\mathcal{C}'$, because the vertex-disjoint paths $[n_v \to n_w]$ and $[n'_v \to n'_w]$ need to be cut for any valid MICLE of G'. But what about the paths $[v, n_v \to n'_w, w]$ and $[v, n'_v \to n_w, w]$? We know the following groups of paths will be cut using edges of $\mathcal{C}'$: (a) $[n_v \to n'_w, w, n_w]$; (b) $[n'_v \to n_w, w, n'_w]$; (c) $[n_v, v, n'_v \to n_w]$; (d) $[n'_v, v, n_v \to n'_w]$.

This fact does not guarantee that the vertex-disjoint paths $[v, n_v \to n'_w, w]$ and $[v, n'_v \to n_w, w]$ are cut by $\mathcal{C}'$. The paths $[v, n_v \to n'_w, w]$ and $[v, n'_v \to n_w, w]$ are definitely cut when $\mathcal{C}'$ contains at least one edge of all vertex-disjoint paths $[n_v \to n'_w]$ and $[n'_v \to n_w]$. If this is not the case, we have one of the following three situations:

1. The paths $[n_v \to n'_w]$ are cut by $\mathcal{C}'$, but the paths $[n'_v \to n_w]$ are not;
2. The paths $[n'_v \to n_w]$ are not cut by $\mathcal{C}'$, but the paths $[n_w \to n_v]$ are;
3. Both the paths from $[n_v \to n'_w]$ and $[n'_v \to n_w]$ are not cut by $\mathcal{C}'$.

For these three remaining situations we can replace edges from $\mathcal{C}'$ to also cut all paths $[v \to w]$ without increasing the weight of $\mathcal{C}'$ and cutting all paths from O and thus obtaining the cut set $\mathcal{C}$ for G.

For the first situation, we know that the edges (n_v, v) and (n'_w, w) need to be in $\mathcal{C}'$ to cut the paths of type (b) and (c). Instead of adding the edges (n_v, v) and (n'_w, w) to $\mathcal{C}'$, we can add the edges (n'_v, v) and (n_w, w) to $\mathcal{C}'$ without increasing the weight of $\mathcal{C}'$. When we do this the overlapping vertices of G' will still be separated, but we also separated v from w without increasing the weight of $\mathcal{C}'$. The second situation can be handled analogously.

When we have the third situation, we know that one of the edges $\{(n_w, w), (w, n'_w)\}$ needs to be in $\mathcal{C}'$ to cut the paths of types (a) and (b), and one of the edges $\{(n_v, v), (v, n'_v)\}$ to cut the paths of type (c) and (d). When we just pick the edges (v, n_v) and (w, n_w), we will once again not change the size of $\mathcal{C}'$ and still separate all overlaps of O', but also all overlaps of O. □

The same trick can be applied to prove that overlaps can also be removed in a stack of structures. This can be proven using exactly the same steps as before and can only be applied under the same restrictions.