

1 Building-like random game generator

Building-like random game generator creates game graphs that try to mimic layout of a multi-story office building. The game graph consists of elevator shafts, main corridors on each story, an entrance on the ground level and office rooms. The procedure has a number of parameters. The parameters are summarized in Table 1;

The building structure is based on rectangular grids of $n \times m$ vertices. The vertices are connected in a specific way to mimic layout of the office building. In the rest of this description vertices will be referred as (x, y) coordinates will be used where x means number of column and y number of row the vertex is in. Each grid represents one story of the building. The generation process of the structure is as follows:

1. **Randomly choose (x, y) positions for e elevator shafts.** The positions are drawn from uniform RNG with the restriction that no two shafts can be placed in neighboring rows or columns. Connect vertices from neighboring floors which belong to the same elevator shaft.
2. **Place main corridors** in all rows and columns that contain elevator shaft. The corridors are straight and go from one edge of the building to another. Connect all neighboring vertices in such rows and columns.
3. **Place building entrance.** If there is no elevator on north end of the first N-S corridor, place a building entrance here. Otherwise place in on the south end of this corridor.
4. **Place doors to offices.** The remaining vertices of the graph are considered to be office space of the building. For each pair of neighboring vertices where one is a part of the corridor and another is a part of office space put an edge with p_d probability. Those are considered doors to offices.
5. **Make sure all office chunks are reachable.** For each chunk of office space (chunks are cut by corridor layout) if there is no door to this chunk add a door to random vertex next to corridor being part of this space.
6. **Generate office space layout.**
 - (a) Consider all office vertices being part of the door as *connected*.
 - (b) Take random vertex that has a neighbor in not connected office space from *connected* set.
 - (c) Connect the vertex with randomly chosen not connected neighbor. And add the neighbor to connected set.
 - (d) Repeat two above steps as long as there are not connected vertices.
 - (e) Now offices have structure of trees.
 - (f) For each of not connected pair of neighboring vertices in offices add a connection with p_o probability.

Once a building structure is finished a positions of targets and a defender base are placed in random vertices in office space of the building. Attacker entry vertex is in the building entrance.

The game payoffs are set as follows:

- Penalty for the attacker for being caught in a target is set to $P_a t$, fixed value for each target.
- Penalty for the attacker for being caught elsewhere is set to $P_a v$, fixed value for each vertex
- Reward for the attacker for attacking a vertex is chosen randomly from uniform range $[0, A_{max}]$, independently for each target.
- Reward for the defender for catching the attacker in a target is set to $R_a t$, fixed value for each target.
- Reward for the defender for catching the attacker elsewhere is set to $R_a v$, fixed value for each vertex.
- Penalty for the defender is chosen randomly from uniform range $[0, D_{max}]$, independently for each target.

Example games generated with the presented method are presented in figs. 1 and 2. Please note that the goal of the presented figures is to visualize graph generator and elements like elevators, corridors and offices are not relevant to game rules which exploit connections between nodes and locations of targets.

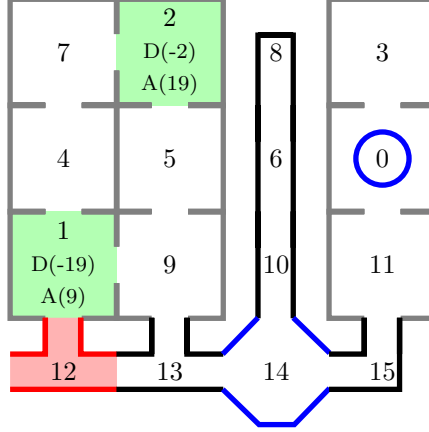


Figure 1: Example of building graph generated with parameter set from Table 2. Graph nodes are in grid layout, each numbers defines a node with given id. Black narrow elements are corridors. Corridor shape defines node connections. Gray squares are offices. If there is a gap in line between offices there is an edge connecting this nodes. Octagonal blue node is an elevator. Red shaded corridor element is attacker's starting point (entrance to the building), green shaded office elements are targets labeled with payoffs. Defender's start is denoted with a circle around a number.

parameter	meaning
f	number of floors
n	width of the building
m	height of the building
e	number of elevator shafts
p_d	probability of placing a door from corridor to office
p_o	probability of additional connections inside the offices
P_{av}	Attacker's penalty in a vertex
P_{at}	Attacker's penalty in a target
A_{max}	Upper limit for attacker's reward
R_{av}	Defender's reward in a vertex
R_{at}	Defender's reward in a target
D_{max}	Limit for defender's penalty

Table 1: Summary of building-like game generator parameters

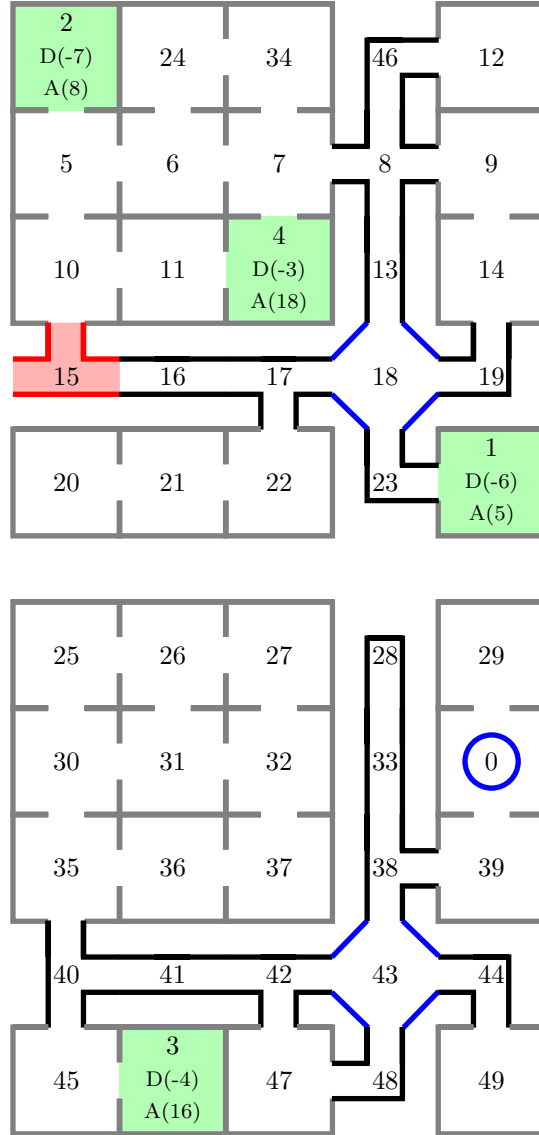


Figure 2: Example of two-story building graph. An elevator is not presented as edge in the figure. It connects vertices 21 and 46. See caption under Figure 1 for symbol description.

parameter	value	meaning
f	1	number of floors
n	4	width of the building
m	4	height of the building
e	1	number of elevator shafts
p_d	0.4	probability of placing a door from corridor to office
p_o	0.5	probability of additional connections inside the offices
P_{av}	1	Attacker's penalty in a vertex
P_{at}	3	Attacker's penalty in a target
A_{max}	20	Upper limit for attacker's reward
R_{av}	1	Defender's reward in a vertex
R_{at}	2	Defender's reward in a target
D_{max}	20	Limit for defender's penalty

Table 2: Summary of building-like game generator parameters

1.1 smallbuilding graph set

Smallbuilding graph set was generated with the presented generator with the parameters presented in Table 2. The parameters, especially dimensions of the building were chosen as a largest value for which solutions can be calculated using 8GB RAM computer. The benchmark set used in experiments was generated as follows:

1. 100 random instances were generated,
2. MILP solution was calculated for each instance if possible within 6GB memory limit,
3. all instances which exceeded memory limit were dropped,
4. all instances where game results for uniform strategy and optimal strategy were equal were dropped.

Finally 25 instances were kept and used as a benchmark for Mixed-UCT method.