View Graph For Indoor Navigation

Introduction

Recently, several indoor graph models have been proposed to capture topological and geometric information, and generate navigation graphs and route descriptions. Varied modelling approaches in these studies lead to incompatible graph structures that are only suitable for specific use-cases. The inconsistency in the indoor models sources from the differences in modelling approaches that are either focused on capturing the *survey* or the *route* information. For example, the topological and geometric models capture survey information of an indoor environment, while navigation graph and route description models mainly store route information. In this study, we propose a new concept, based on *views*, to capture survey and route information of an indoor environment in a unified manner. We define views as directed line of sights in which the visible spatial objects and their relationships are the same along the view. Using views, moving in the space (route information) can be captured as going from one view to another, and survey information can be attached to each view based on its visibility situation.

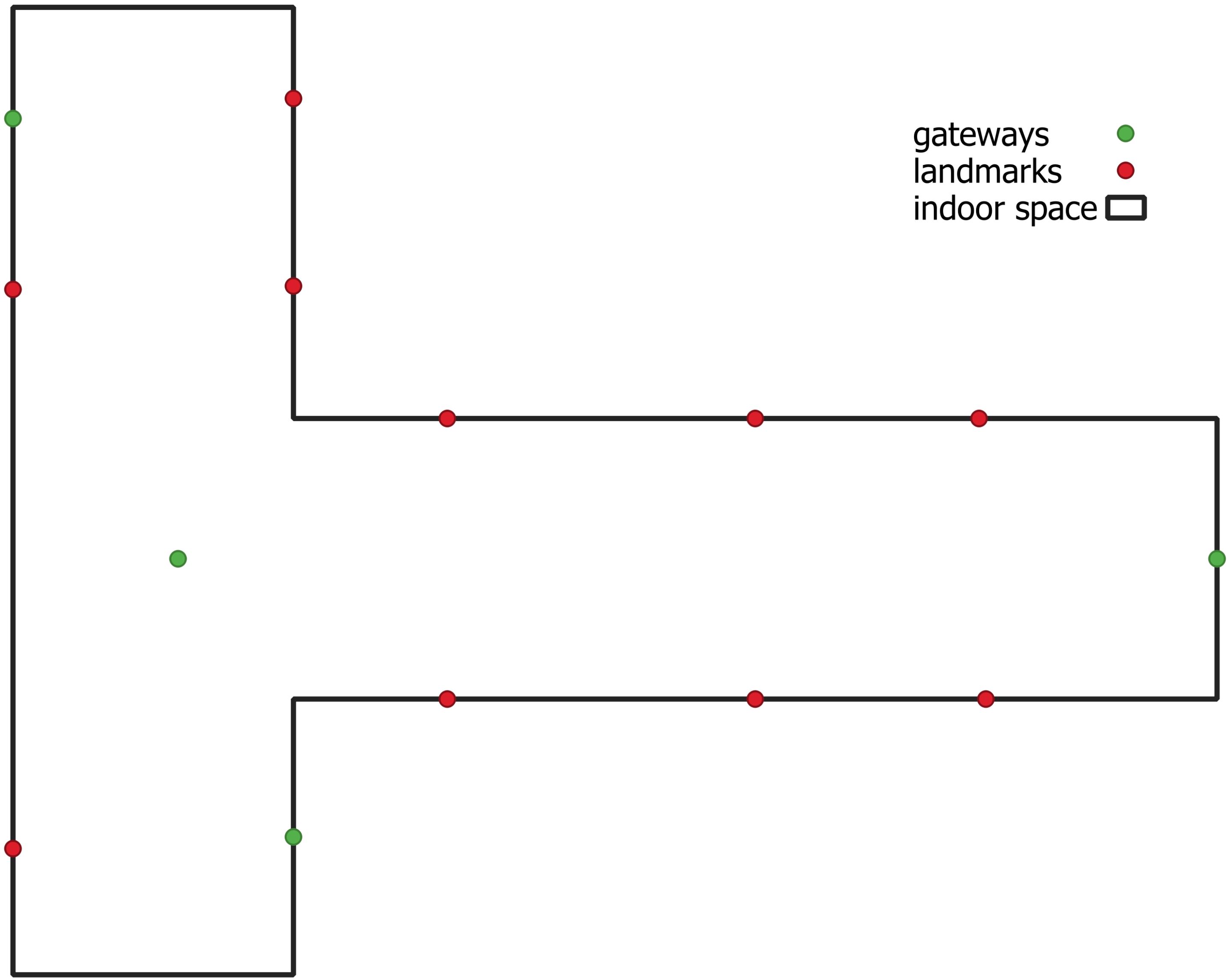
Hypothesis and Research Questions

This study hypothesizes that using the concept of *view* both survey and route knowledge of an indoor environment can be captured. To test the hypothesis, we will address the following research questions:

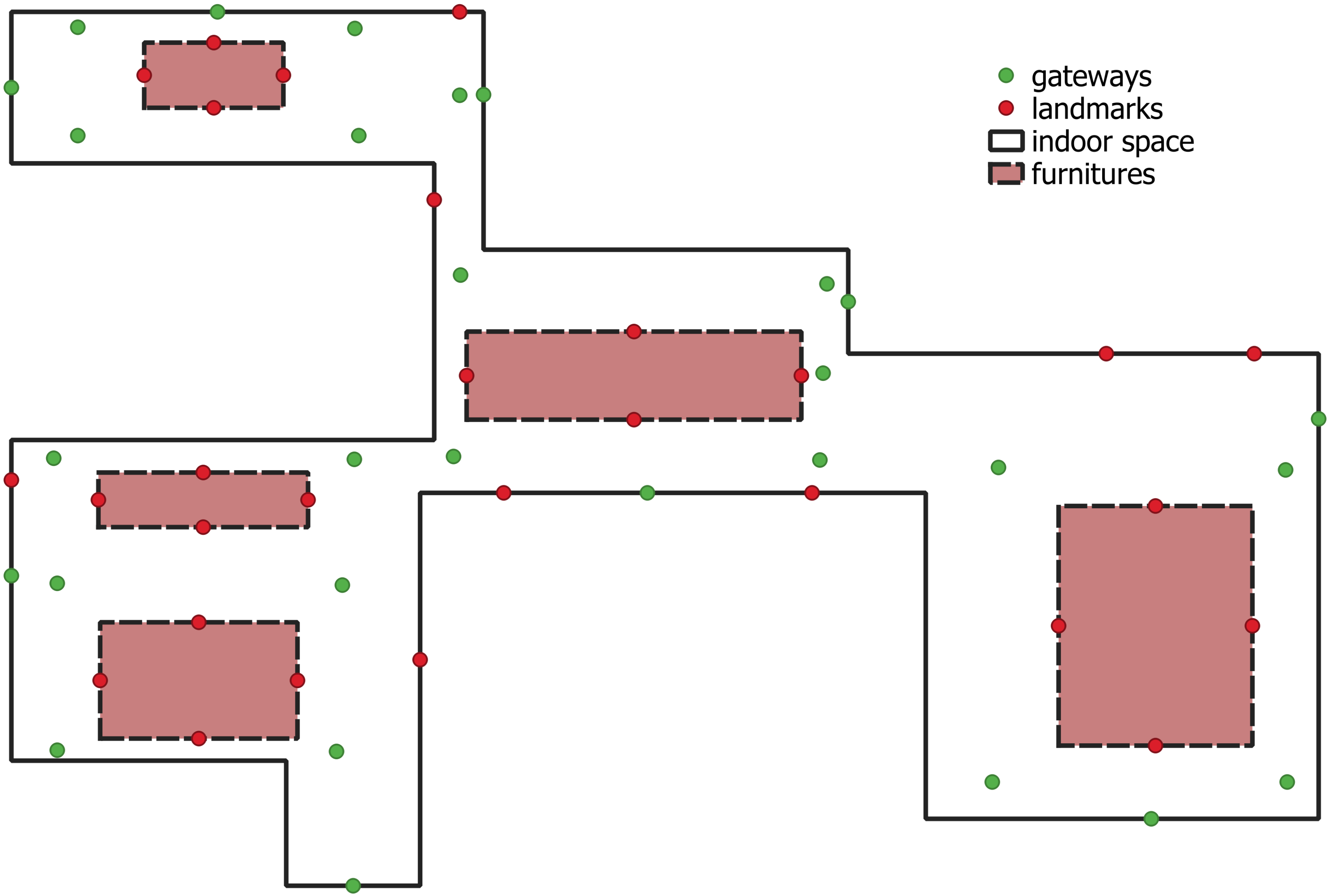
- How can views be extracted from geometric information (e.g., floor plans)?
- How can view graphs be constructed to capture *movement* and *turns* from one view to another?
- Can route descriptions and navigation graphs be generated from the view graph? By this way we can prove that route knowledge of an indoor environment can be captured by views.
- Can spatial relationships of indoor objects be extracted from the view graph? By this way we can demonstrate that survey knowledge of an indoor environment can be captured by views.

Test Environments

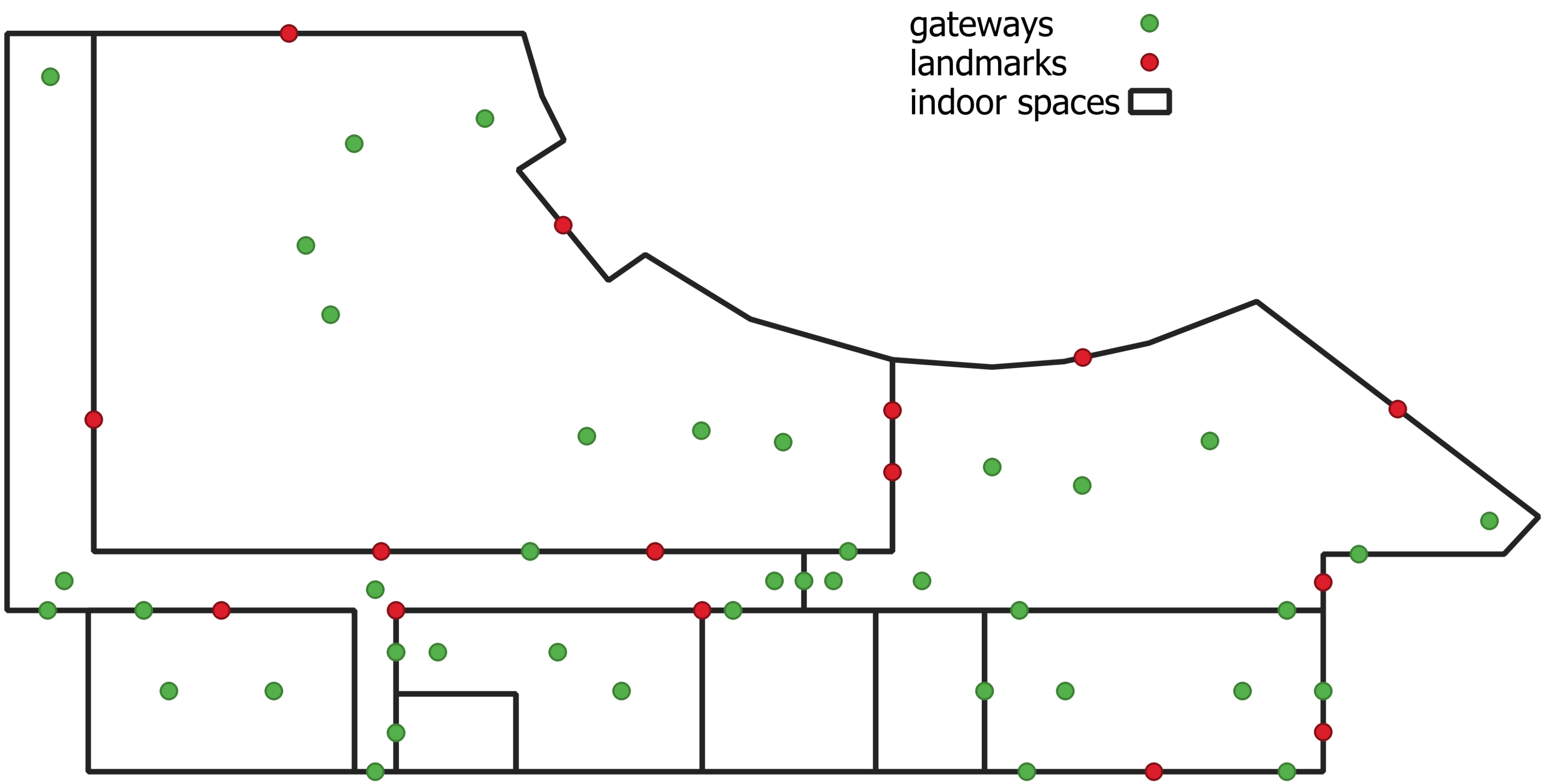
Hypothetical Floor Plan (Basic)



Hypothetical Floor Plan

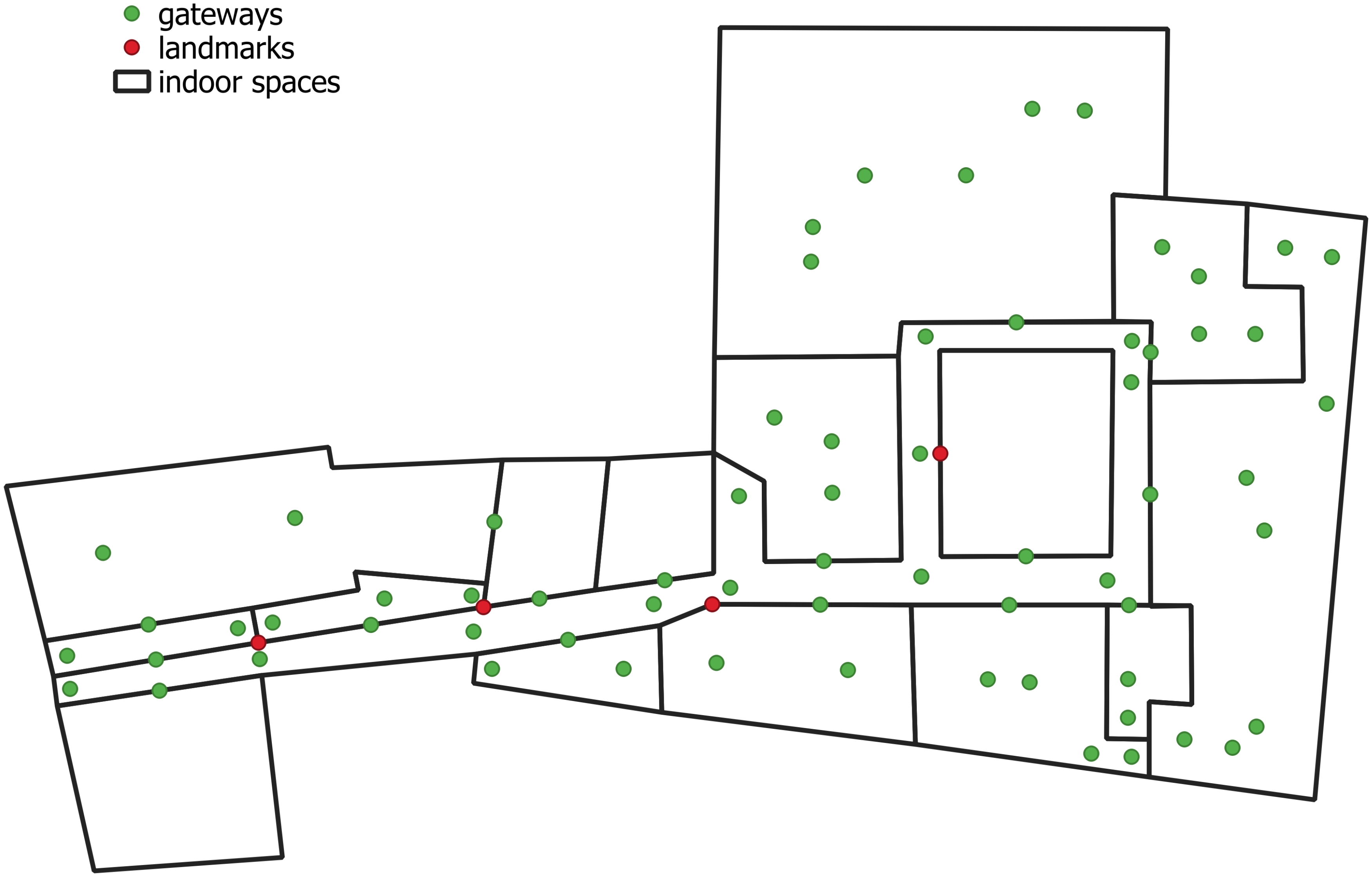


Real World Environment (University Layout)



Real World Environment (Regent Place Shopping Mall)

Constructed from - with minor simplification: https://www.regentplace.com.au/floor-plan



References

- Amoozandeh K, Winter S, Tomko M. Space decomposition based on visible objects in an indoor environment. Environment and Planning B: Urban Analytics and City Science. 2021 Aug 11:23998083211037347.
- Becker T, Nagel C, Kolbe TH. A multilayered space-event model for navigation in indoor spaces. In3D geo-information sciences 2009 (pp. 61-77). Springer, Berlin, Heidelberg.
- Liu L, Zlatanova S. A "door-to-door" path-finding approach for indoor navigation. Proceedings Gi4DM 2011: GeoInformation for Disaster Management, Antalya, Turkey, 3-8 May 2011. 2011.
- Mortari F, Zlatanova S, Liu L, Clementini E. Improved geometric network model (IGNM): A novel approach for deriving connectivity graphs for indoor navigation. ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences. 2014 Apr 23;2(4).
- Pang Y, Zhou L, Lin B, Lv G, Zhang C. Generation of navigation networks for corridor spaces based on indoor visibility map. International Journal of Geographical Information Science. 2020 Jan 2;34(1):177-201.
- Werner S, Krieg-Brückner B, Mallot HA, Schweizer K, Freksa C. Spatial cognition: The role of landmark, route, and survey knowledge in human and robot navigation. In Informatik'97 Informatik als Innovationsmotor 1997 (pp. 41-50). Springer, Berlin, Heidelberg.
- Yang L, Worboys M. Generation of navigation graphs for indoor space. International Journal of Geographical Information Science. 2015 Oct 3;29(10):1737-56.
- Zhou Z, Weibel R, Richter KF, Huang H. HiVG: A hierarchical indoor visibility-based graph for navigation guidance in multi-storey buildings. Computers, Environment and Urban Systems. 2022 Apr 1;93:101751.

Creating View Graph

Loading classes

In [1]:

```
# Parameters
from Parameters import Parameters

# Utitlies for mathematical calculation, isovist and visualization
from Isovist import Isovist
from Plotter import Plotter
from Utility import Utility

# Container -> Environment -> View Graph
from Container import Container
from Environment import IndoorEnvironment
```

```
from ViewGraph import ViewGraph

from pyvis.network import Network
```

Loading environment from GeoJSON files

In [2]:
```
Parameters.set_env("real")  # this can be set to "basic" environment, "hypo" environment as well
Parameters.print_info()
```

```
--------------------------------------
Real-world environment is active

Static Variables:
        epsilon: 0.01
        precision: 2
        alpha: 40
        fov: 160
        min_area: 1000000
        max_area: 1000000
        door_weight: 50
        turn_weight: 0.05
--------------------------------------
```

In [3]:
```
def read_env():
    # Basic environment
    if Parameters.basic:
        address = 'envs/basic/'
        pfiles = ['t_bound.geojson']
        hfiles = [None]
        dfiles = ['t_doors.geojson']
        dpfiles = [None]
        lfiles = ['t_landmarks.geojson']
        # create an indoor environment
        ie = IndoorEnvironment(address, pfiles, hfiles, dfiles, dpfiles, lfiles)

    # Hypo environment
    elif Parameters.hypo:
        address = 'envs/hypo/'
        pfiles = ['hypo_env.geojson']
        hfiles = ['hypo_holes.geojson']
        dfiles = ['hypo_doors.geojson']
        dpfiles = ['hypo_dpoints.geojson']
        lfiles = ['hypo_landmarks.geojson']
```

```
        # create an indoor environment
        ie = IndoorEnvironment(address, pfiles, hfiles, dfiles, dpfiles, lfiles)

    # MC5 real world environment
    else:
        address = 'envs/mc-floor-5/'
        pfiles, hfiles, dfiles, dpfiles, lfiles = IndoorEnvironment.reformat(
            address, 'containers.geojson', 'doors.geojson', 'landmarks.geojson')
        # create an indoor environment
        ie = IndoorEnvironment('', pfiles, hfiles, dfiles, dpfiles, lfiles)
    return ie

ie = read_env()
```

```
environment files -- count is valid
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
```

Decomposing regions into isovists, and create view graph

Here, the following tasks are performed:

- calculate isovists
- decompose containers to regions
- calculate visiblity signature for each region
- create adjacency matrix
- find initial views
- decompose views
- construct view graph
- calculate spatial relationships
- augment the actions in view graphs (to nodes and edges)

In [4]:
```
# create view graph
vgs, isovist_objects = ie.construct_view_graph()
```

```
******************************************
Analyzing: Emergency Stairs
Container environment is valid: True
region initial : 1
regions : 1 -- 1
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 12
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)

******************************************
Analyzing: Women Toilet
Container environment is valid: True
region initial : 3
regions : 4 -- 3
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 49
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)

******************************************
Analyzing: Disabled Toilet
Container environment is valid: True
region initial : 1
regions : 1 -- 1
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 2
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)
```

```
******************************************
Analyzing: Men Toilet
Container environment is valid: True
region initial : 1
regions : 1 -- 1
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 2
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)

******************************************
Analyzing: Corridor
Container environment is valid: True
region initial : 21
regions : 10 -- 21
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 219
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)

******************************************
Analyzing: Active Hub
Container environment is valid: True
region initial : 43
regions : 12 -- 43
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 280
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
```

```
Adding actions to view relations (edges)

*******************************************
Analyzing: Stairs to Lower Floors
Container environment is valid: True
region initial : 1
regions : 1 -- 1
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 2
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)

*******************************************
Analyzing: Ngi-a Djerring Gat-ith
Container environment is valid: True
region initial : 1
regions : 1 -- 1
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 72
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)

*******************************************
Analyzing: UX Lab
Container environment is valid: True
region initial : 16
regions : 12 -- 16
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 191
constructing view graph for regions
calculating all spatial relationships visible in each view
```

```
Adding actions to views (nodes)
Adding actions to view relations (edges)
```

In [5]:

```
ie.containers_names
```

Out[5]:

```
['Emergency Stairs',
 'Women Toilet',
 'Disabled Toilet',
 'Men Toilet',
 'Corridor',
 'Active Hub',
 'Stairs to Lower Floors',
 'Ngi-a Djerring Gat-ith',
 'UX Lab']
```

Shortest Path and Route Instruction

Here, we first generate a shortest path from a region to another. Then we use the augmented actions and relationships in view graph to generate route instructions from its results.

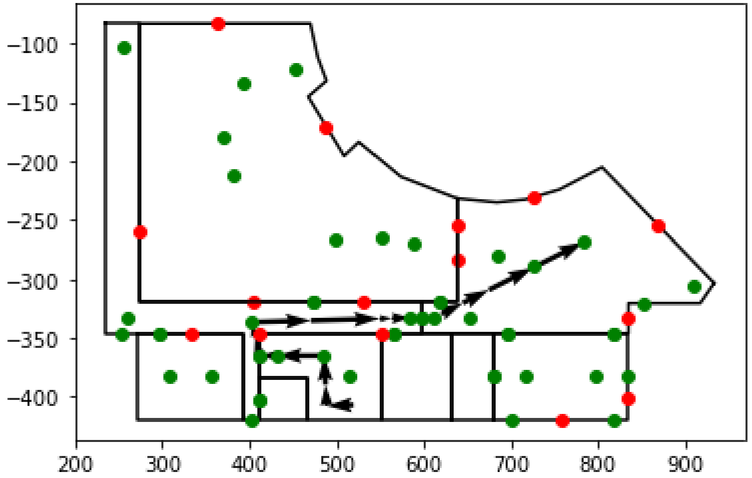
In [8]:

```
# set parameters to Parameters class
start_container = 'Women Toilet'
start_region = 3
end_container = 'Active Hub'
end_region = 3
```

In [9]:

```
# calculate shortest path and generate verbal description
vp, pv = ie.shortest_path(start_container, start_region, end_container, end_region)

# plot shortest path
plotter = Plotter()
for isovist_object in ie.isovist_objects:
    plotter.add_isovist(isovist_object)
plotter.add_views(pv)
plotter.show(False)
plotter.close()
```

```
enter: Corridor
enter: Active Hub
```




```
<Figure size 432x288 with 0 Axes>
```

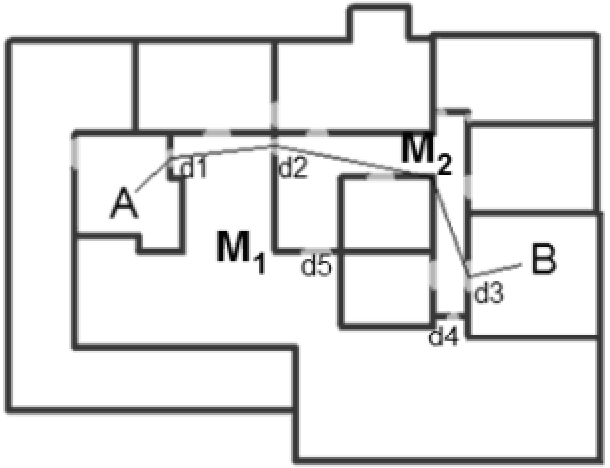
In [10]:

```
# generate route instructions
def generate_route_descriptions(vp):
    container = ''
    container_vids = {}
    finals = {}
    for v in vp[1:-1]:
        info = v.split('-V')
        if container != info[0]:
            container = info[0]
            container_vids[container] = []
        container_vids[container].append(int(info[1]))
    print(container_vids)
    for container, vids in container_vids.items():
        cidx = ie.containers_names.index(container)
        vg = vgs[cidx]
        print(cidx)
        print(vids)
        rds = vg.generate_route_description(vids)
        finals[container] = rds
    return finals

def print_route_descriptions(rd_dictionary):
    containers = list(rd_dictionary.keys())
    for container in containers:
        rd = rd_dictionary[container]
        if containers.index(container) < len(containers) - 1:
```

```
            rd[len(rd)-1] = rd[len(rd)-1].replace('until you reach the destination', 'to enter {}'.format(containers[containers.i
        for r in rd:
            print(r)
# vg.generate_route_description(vp)
print_route_descriptions(generate_route_descriptions(vp))
```

```
{'Women Toilet': [118, 119, 83, 84, 85], 'Corridor': [954, 955, 956, 298, 299, 300, 301, 302, 303, 304, 305], 'Active Hub': [73
3, 734, 48, 49, 50, 51, 52, 53]}
1
[118, 119, 83, 84, 85]
4
[954, 955, 956, 298, 299, 300, 301, 302, 303, 304, 305]
5
[733, 734, 48, 49, 50, 51, 52, 53]
Head towards decision point 1 and turn left
Pass decision point 1 and move forward to enter Corridor
Head towards the door to women toilet
Pass the door to women toilet and veer right
Pass the door to male toilet and move forward to enter Active Hub
Head towards the door to corridor
Pass the landmark 0 and move forward until you reach the destination
```

Derive Door-to-Door Visibility Graph

Example: 

(source: Liu and Zlatanova, 2011)

Node:

- Doors (could be also gateways)

Edge:

- Direct visibility

References:

- Liu L, Zlatanova S. A "door-to-door" path-finding approach for indoor navigation. Proceedings Gi4DM 2011: GeoInformation for Disaster Management, Antalya, Turkey, 3-8 May 2011. 2011.
- Mortari F, Zlatanova S, Liu L, Clementini E. Improved geometric network model (IGNM): A novel approach for deriving connectivity graphs for indoor navigation. ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences. 2014 Apr 23;2(4).

Generate door-to-door visibility graph (for doors and decision points)

In [11]:
```
ie.containers_names
```

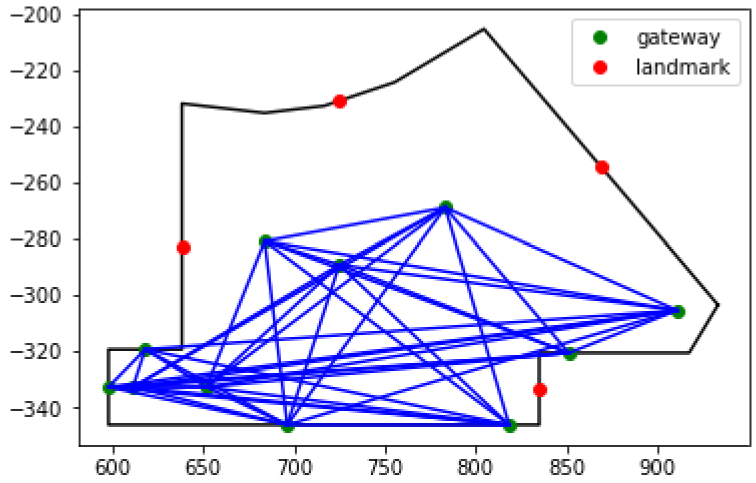
Out[11]:
```
['Emergency Stairs',
 'Women Toilet',
 'Disabled Toilet',
 'Men Toilet',
 'Corridor',
 'Active Hub',
 'Stairs to Lower Floors',
 'Ngi-a Djerring Gat-ith',
 'UX Lab']
```

In [12]:
```
# selecting a space
cidx = ie.containers_names.index('Active Hub')
vg = vgs[cidx]
isovist_object = isovist_objects[cidx]
```

In [13]:
```
# derive door-to-door visibility graph (doors and decision points)
connected, dtd_graph = vg.generate_door_to_door_graph(isovist_object)

print('Press Enter: Door to door visibility (doors+gateways)')
plotter = Plotter()
plotter.add_isovist(isovist_object)
plotter.add_points_lines(connected)
plotter.show()
```

```
plotter.close()
plotter.write_graph('d-t-d-all.html', dtd_graph, is_directed=False)
```

```
generate door-to-door graph, only_doors False from view graph
Press Enter: Door to door visibility (doors+gateways)
```



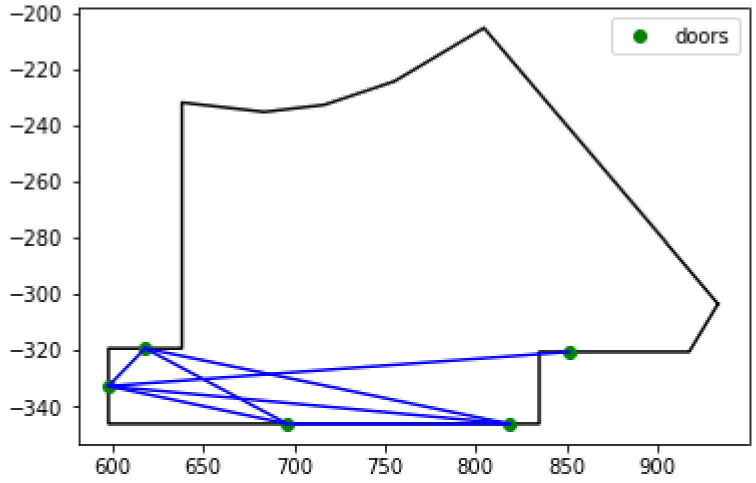

```
<Figure size 432x288 with 0 Axes>
```

Generate door-to-door visibility graph (only for doors)

In [14]:

```
# derive door-to-door visibility graph (only doors)
connected2, dtd_graph2 = vg.generate_door_to_door_graph(isovist_object, only_doors=True)
plotter = Plotter()
plotter.add_poly(isovist_object.space_x, isovist_object.space_y)
plotter.add_holes(isovist_object.holes_x, isovist_object.holes_y)
plotter.add_points(isovist_object.door_points[:isovist_object.door_idx], 'doors')
plotter.add_points_lines(connected2)
plotter.show()
plotter.close()
plotter.write_graph('d-t-d-doors.html', dtd_graph2, is_directed=False)
```

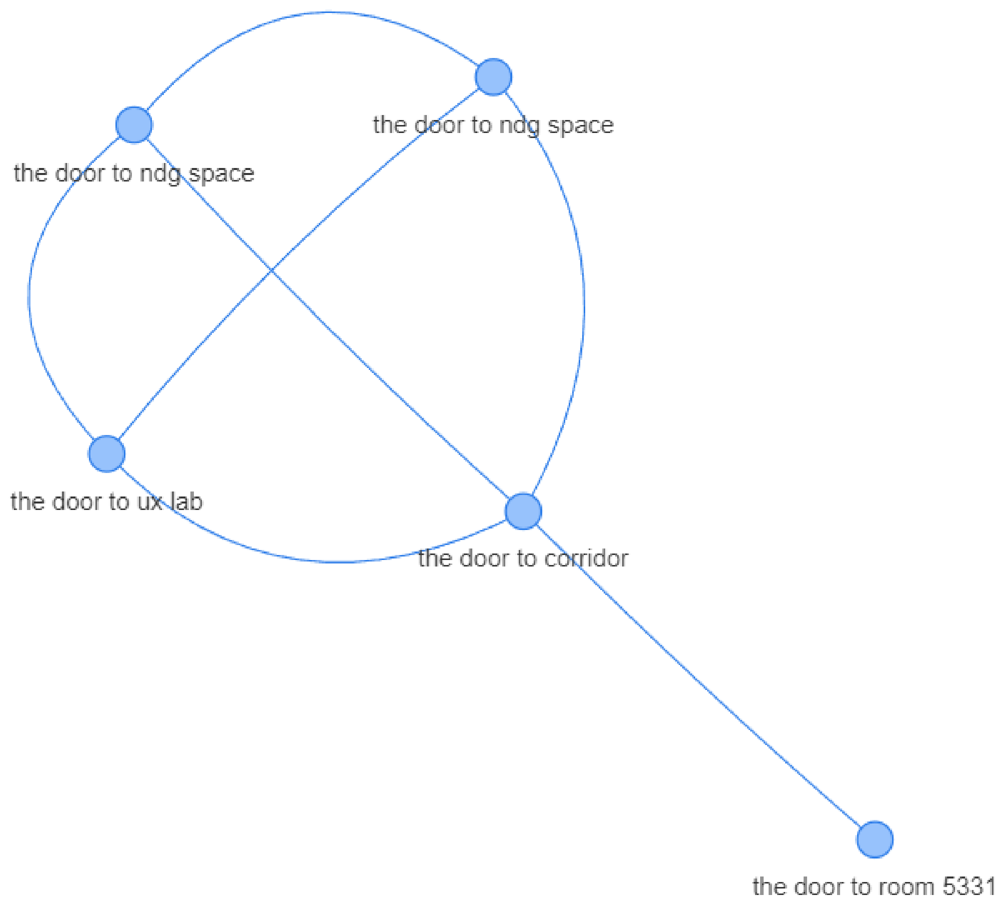
```
generate door-to-door graph, only_doors True from view graph
```



<Figure size 432x288 with 0 Axes>

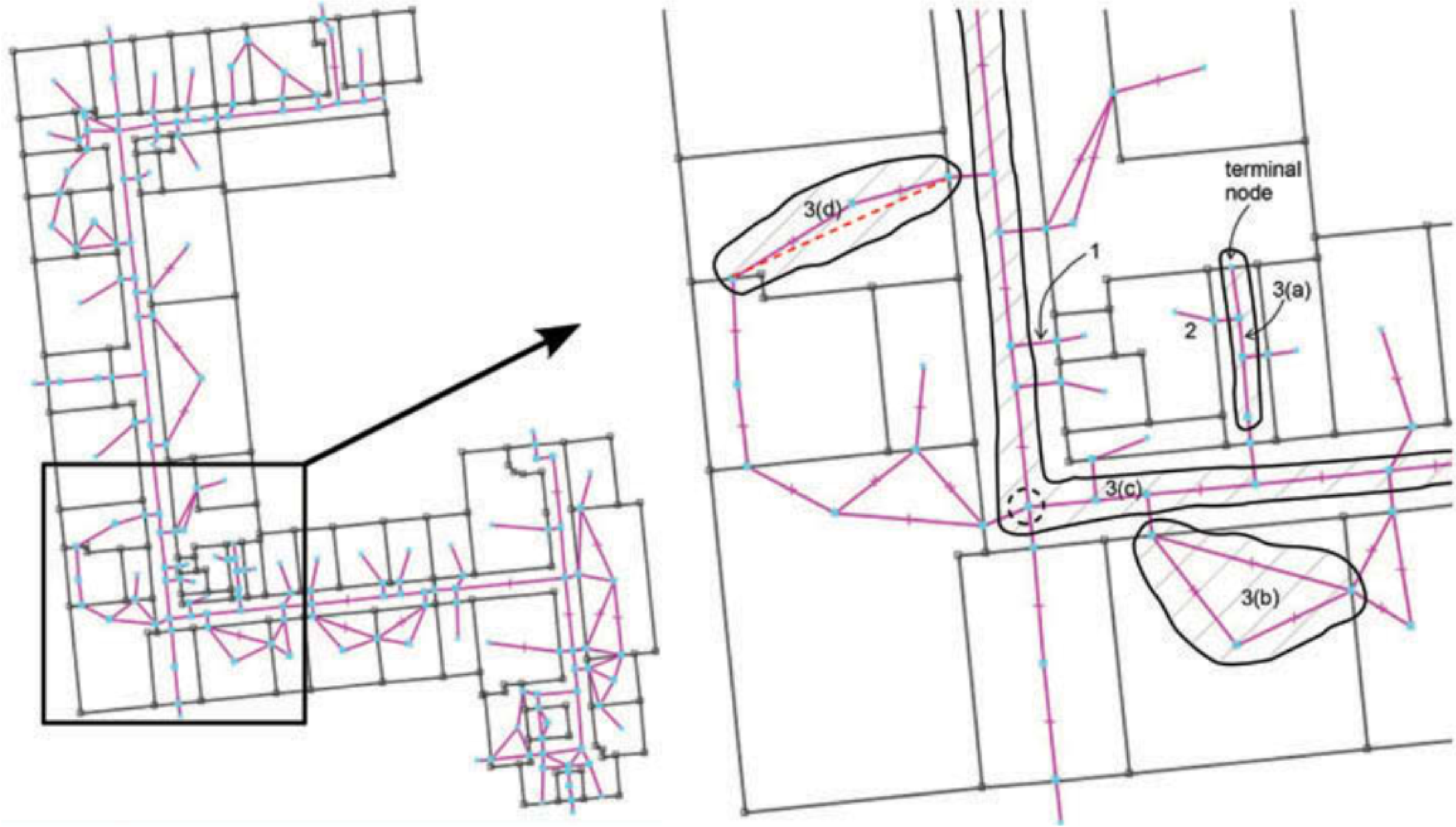
In [15]:
```
nt2 = Network(width='1000px', height='600px', directed=False, notebook=True)
nt2.from_nx(dtd_graph2, show_edge_weights=False)
nt2.options.physics.use_repulsion({'node_distance': 185, 'central_gravity': 0.2, 'spring_length': 200,
                                    'spring_strength': 0.05, 'damping': 0.09})

nt2.show('d-t-d-doors.html')
```

Out[15]:



Derive navigation graph

Example:



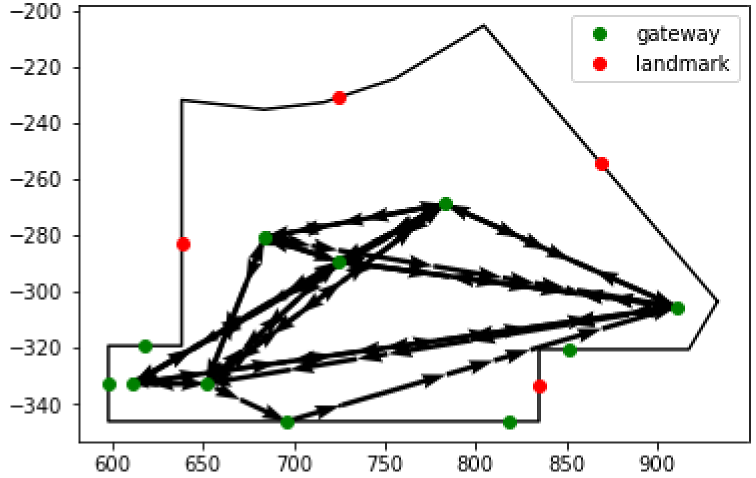
(source: Yang & Worboys, 2015)

Reference

- Yang L, Worboys M. Generation of navigation graphs for indoor space. International Journal of Geographical Information Science. 2015 Oct 3;29(10):1737-56.
- Pang Y, Zhou L, Lin B, Lv G, Zhang C. Generation of navigation networks for corridor spaces based on indoor visibility map. International Journal of Geographical Information Science. 2020 Jan 2;34(1):177-201.

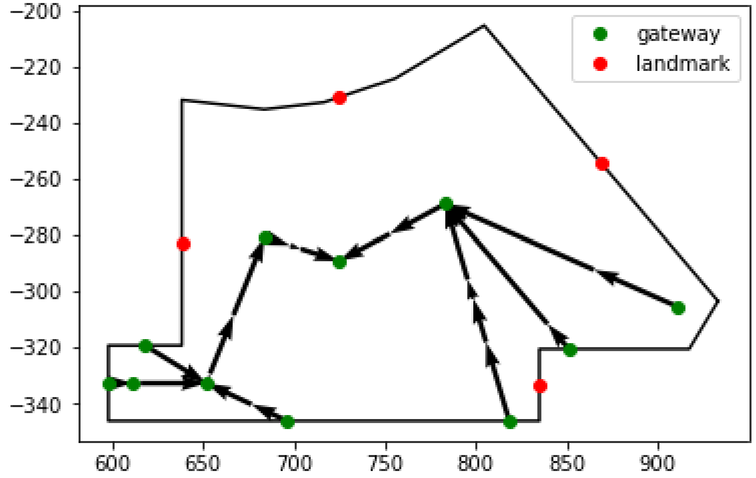
In [16]:
```
# derive all shortest path visibility graph and spanning tree
vps, pvs, st_vps, st_pvs, nvgraph = \
    vg.generate_navigation_graph(isovist_object, indirect_access=False)

plotter = Plotter()
plotter.add_isovist(isovist_object)

for pv in pvs:
    plotter.add_views(pv)
plotter.show()
```

```
derive navigation graph using spanning tree from viewgraph
```



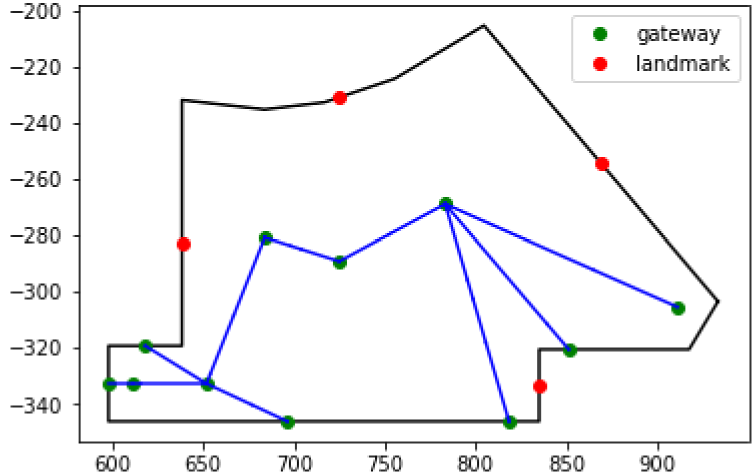
In [17]:
```
plotter.refresh()
for pv in st_pvs:
    plotter.add_views(pv)
plotter.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [18]:
```
plotter.refresh()
for pv in st_pvs:
    plotter.add_points_lines(pv, is_vis=False)
plotter.show()
```

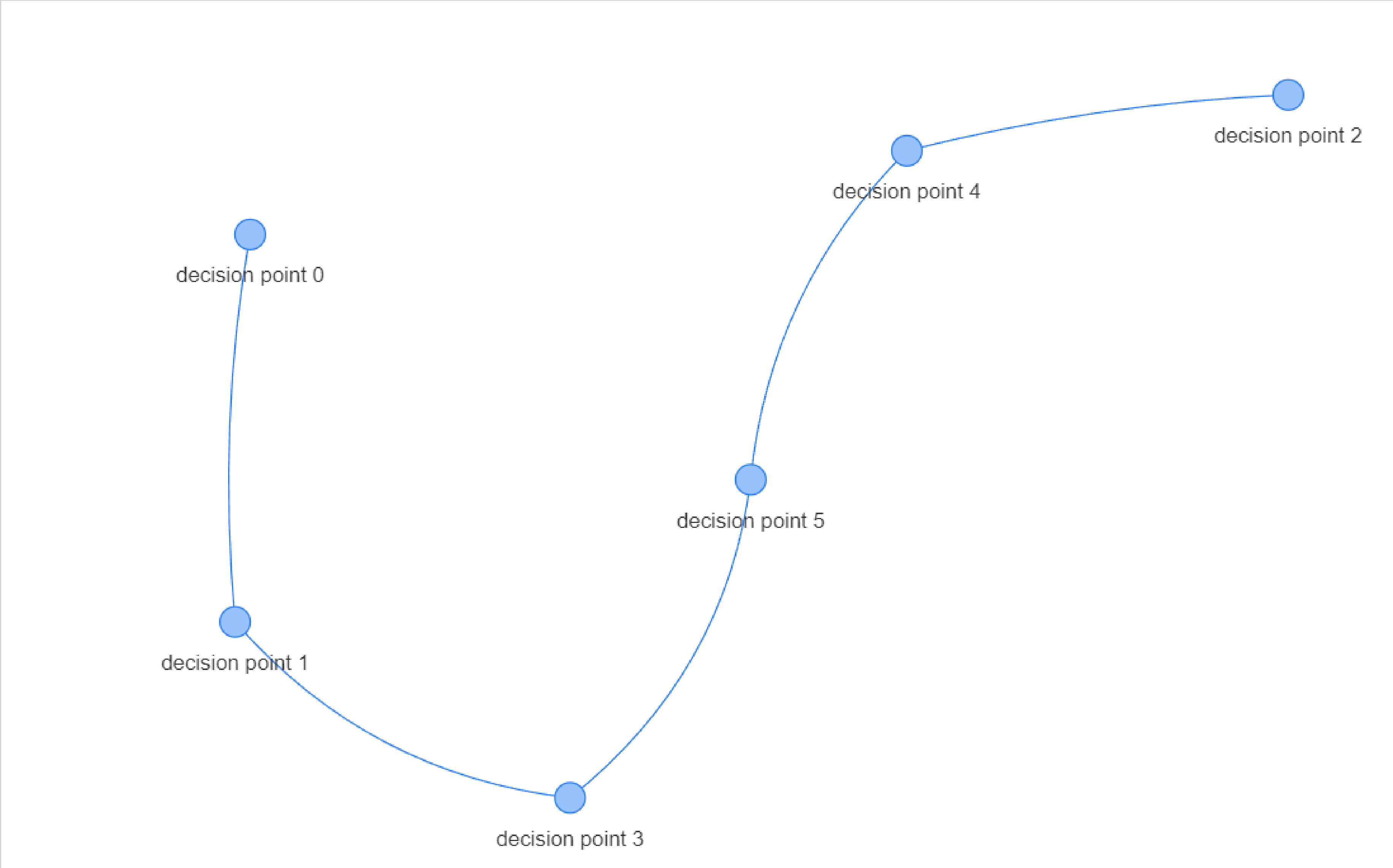
```
<Figure size 432x288 with 0 Axes>
```



In [19]:
```
# generate navigation 'graph' for gateways + doors
nt2 = Network(width='1000px', height='600px', directed=False, notebook=True)
nt2.from_nx(nvgraph, show_edge_weights=False)
nt2.options.physics.use_repulsion({'node_distance': 185, 'central_gravity': 0.2, 'spring_length': 200,
                                   'spring_strength': 0.05, 'damping': 0.09})
```

```
nt2.show('nvgraph.html')
```

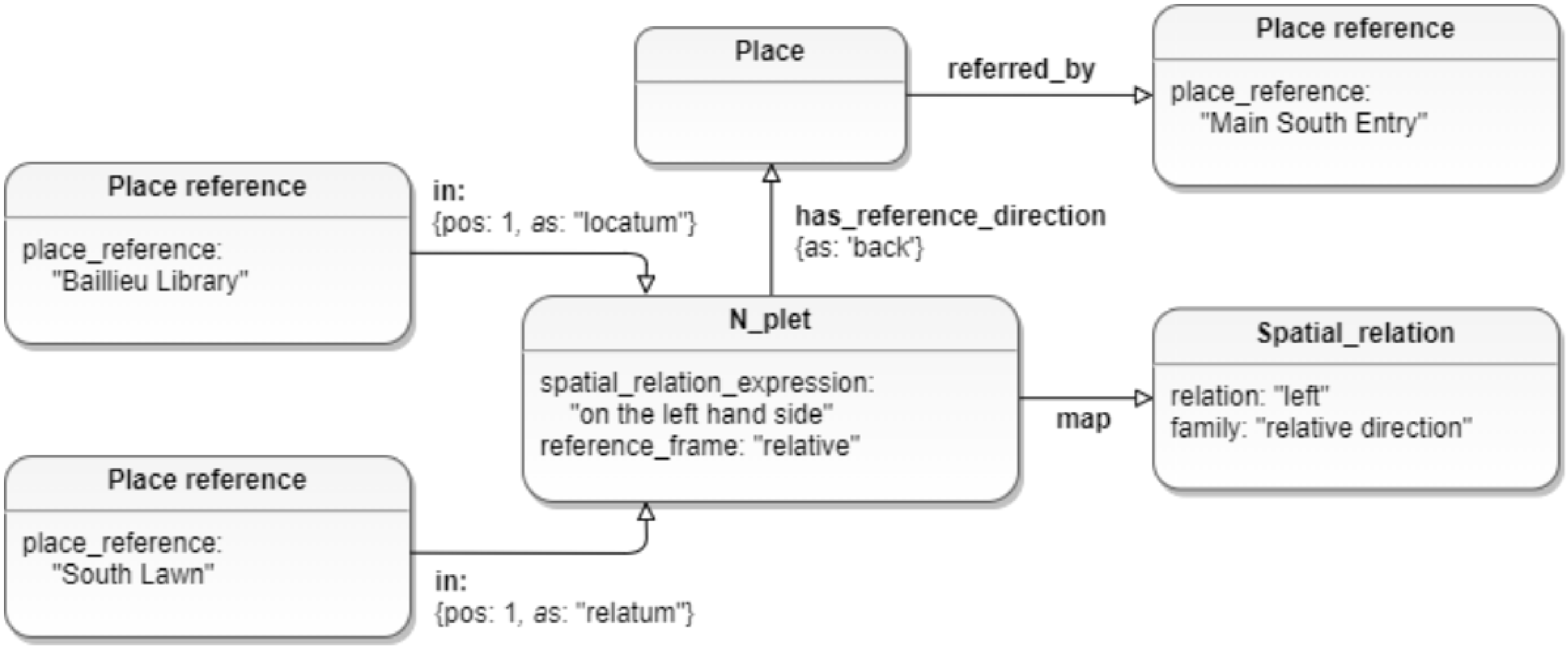
Out[19]:



Derive place graph from view graph

Example:

"… coming from the **Main South Entry**, the **Baillieu Library** will be *on the left hand side* of the **South Lawn** …"



(Source: Chen et. al. 2018)

Nodes:

- place:
- reference:
- n-plet:

- spatial relationship:

Edges:

- locatum:
- relatum:
- map:
- referred by:
- has reference direction:

Reference: Chen H, Vasardani M, Winter S, Tomko M. A graph database model for knowledge extracted from place descriptions. ISPRS International Journal of Geo-Information. 2018 Jun;7(6):221.

In [20]:
```
# derive place graph
place_graph = vg.generate_place_graph(isovist_object)
```

```
derive place graph from view graph
```

In [21]:
```
print('Place graph generation; visualize for all and only for landmark 2')
plotter.write_graph('placegraph.html', place_graph)
```

```
Place graph generation; visualize for all and only for landmark 2
```

In [22]:
```
# selecting a space
cidx = ie.containers_names.index('Corridor')
vg = vgs[cidx]
isovist_object = isovist_objects[cidx]
```

In [23]:
```
place_graph = vg.generate_place_graph(isovist_object)
```

```
derive place graph from view graph
```

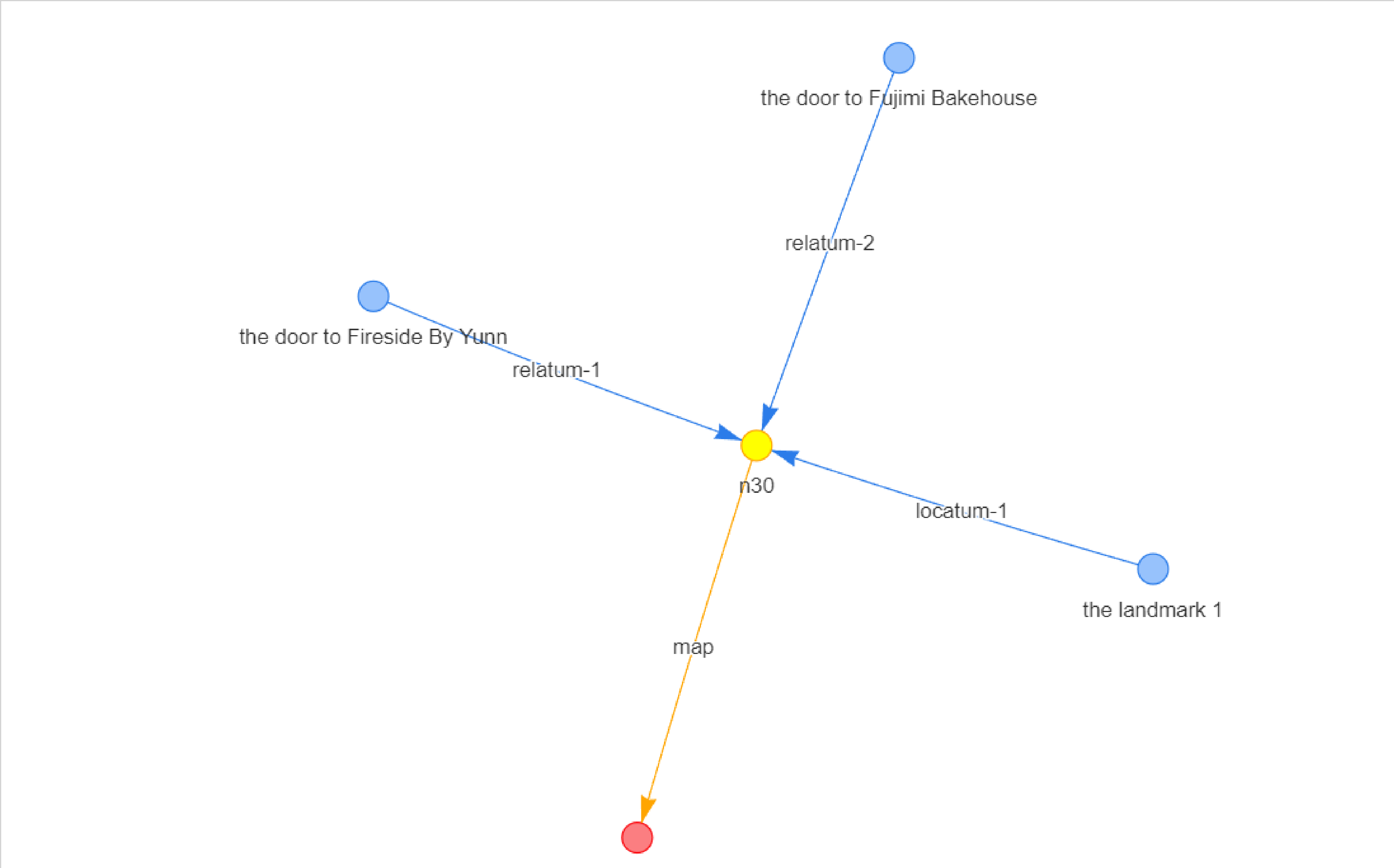
Single Nplet

Select a single n-plet from different spatial relationships and visualize the graph and spatial configuration

1. between
2. near
3. left/right

In [24]:
```
# selecting a space
cidx = ie.containers_names.index('Corridor')
vg = vgs[cidx]
isovist_object = isovist_objects[cidx]

def nplet_extraction(nplet_id):
    ## nplet_id = 'n830'
    place_graph[nplet_id]  # left
    # place_graph['n100'] # between

    # nodes = ['n830', 'left', 'place12', 'gateway 12', 'landmark 20', 'gateway 1']
    nodes = [nplet_id]
    nodes.extend(list(dict(place_graph[nplet_id]).keys()))

    additional = []
    for node in nodes:
        if node.startswith('place'):
            additional.extend(list(dict(place_graph[node]).keys()))
    nodes.extend(additional)

    for v in list(place_graph.edges):
        if v[1] == nplet_id:
            nodes.append(v[0])
            if v[0].startswith('place'):
                nodes.extend(list(dict(place_graph[v[0]]).keys()))

    nplets = place_graph.subgraph(nodes)
    nt2 = Network(width='1000px', height='600px', directed=True, notebook=True)
    nt2.from_nx(nplets, show_edge_weights=False)
    nt2.options.physics.use_repulsion({'node_distance': 185, 'central_gravity': 0.2, 'spring_length': 200,
                                    'spring_strength': 0.05, 'damping': 0.09})
    return nt2, nodes

spatial_expression = 'the door to women toilet between the landmark 2 and the door to disabled toilet'
for n in place_graph.nodes:
    if n.startswith('n') and 'exp' in place_graph.nodes[n].keys() and place_graph.nodes[n]['exp'] == spatial_expression:
        nplet = n
        break
nt2, nplet_nodes = nplet_extraction(nplet)
nt2.show('nplet_1.html')
```

Out[24]:



Regent Place - Shopping Mall

Testing the view graph capabilities in computing shortest path, generating route descriptions, navigation graphs and place graphs in another test environment: Regenet Place Shopping Mall (https://www.regentplace.com.au/floor-plan)

Reading the Floorplan Files

In [27]:
```
# reading the new floorplan dataset
Parameters.set_env(env="real", mc=False)
address = 'envs/RegentPlace/'
pfiles, hfiles, dfiles, dpfiles, lfiles = IndoorEnvironment.reformat(
    address, 'containers.geojson', 'doors.geojson', 'landmarks.geojson')
# create an indoor environment
ie = IndoorEnvironment('', pfiles, hfiles, dfiles, dpfiles, lfiles)
```

```
environment files -- count is valid
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
reading GeoJSON files (boundary, holes, doors and decision points)
```

Creating View Graph

In [28]:
```
# creating view graph
vgs, isovist_objects = ie.construct_view_graph()
```

```
******************************************
Analyzing: Ice Kirin Bar
Container environment is valid: True
region initial : 1
regions : 1 -- 1
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 20
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)

******************************************
Analyzing: Shop
Container environment is valid: True
region initial : 3
regions : 2 -- 3
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 14
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)

******************************************
Analyzing: Fireside By Yunn
Container environment is valid: True
region initial : 1
regions : 1 -- 1
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 2
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)
```

```
******************************************
Analyzing: Yakitori Yokocho
Container environment is valid: True
region initial : 2
regions : 3 -- 2
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 51
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)

******************************************
Analyzing: Loading Dock
Container environment is valid: True
region initial : 9
regions : 5 -- 9
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 69
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)

******************************************
Analyzing: Fujimi Bakehouse
Container environment is valid: True
region initial : 3
regions : 3 -- 3
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 43
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
```

```
Adding actions to view relations (edges)

*******************************************
Analyzing: Daiso
Container environment is valid: True
region initial : 14
regions : 10 -- 14
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 165
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)

*******************************************
Analyzing: The Parks Sydney
Container environment is valid: True
region initial : 12
regions : 7 -- 12
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 90
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)

*******************************************
Analyzing: Mido Mart
Container environment is valid: True
region initial : 48
regions : 22 -- 48
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 238
constructing view graph for regions
calculating all spatial relationships visible in each view
```

```
Adding actions to views (nodes)
Adding actions to view relations (edges)

*******************************************
Analyzing: Toilet
Container environment is valid: True
region initial : 2
regions : 3 -- 2
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 34
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)

*******************************************
Analyzing: Yakinku Yokocho
Container environment is valid: True
region initial : 6
regions : 5 -- 6
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 71
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)

*******************************************
Analyzing: Dioa by Devon
Container environment is valid: True
region initial : 1
regions : 1 -- 1
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 12
constructing view graph for regions
```

```
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)

*******************************************
Analyzing: Arctic White
Container environment is valid: True
region initial : 1
regions : 1 -- 1
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 12
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)

*******************************************
Analyzing: Edomae Sushi
Container environment is valid: True
region initial : 1
regions : 1 -- 1
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 2
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)

*******************************************
Analyzing: FraserSuites
Container environment is valid: True
region initial : 1
regions : 1 -- 1
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 2
```

```
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)

*******************************************
Analyzing: Corridor
Container environment is valid: True
region initial : 577
regions : 51 -- 577
calculating the visibility signatures...
calculating adjacency matrix for regions
finding regions that contains doors/gateways and decision points
decompose views
len: 1068
constructing view graph for regions
calculating all spatial relationships visible in each view
Adding actions to views (nodes)
Adding actions to view relations (edges)
```

Shortest Path Computation

In [29]:
```
ie.containers_names
```

Out[29]:
```
['Ice Kirin Bar',
 'Shop',
 'Fireside By Yunn',
 'Yakitori Yokocho',
 'Loading Dock',
 'Fujimi Bakehouse',
 'Daiso',
 'The Parks Sydney',
 'Mido Mart',
 'Toilet',
 'Yakinku Yokocho',
 'Dioa by Devon',
 'Arctic White',
 'Edomae Sushi',
 'FraserSuites',
 'Corridor']
```

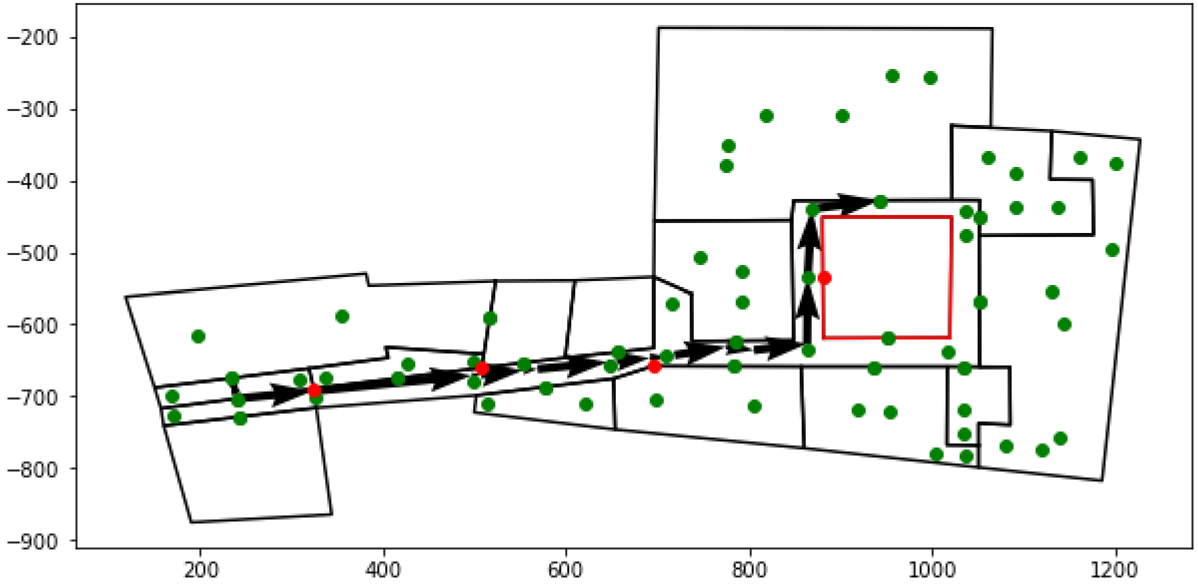
In [30]:
```
# set parameters to Parameters class
start_container = 'Ice Kirin Bar'
```

```
start_region = 0
end_container = 'Daiso'
end_region = 1
```

In [31]:

```
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [10, 5]

# calculate shortest path and generate verbal description
vp, pv = ie.shortest_path(start_container, start_region, end_container, end_region)

# plot shortest path
plotter = Plotter()
for isovist_object in ie.isovist_objects:
    plotter.add_isovist(isovist_object)
plotter.add_views(pv)
plotter.show(False)
plotter.close()
```

```
enter: Corridor
enter: Daiso
```




```
<Figure size 720x360 with 0 Axes>
```

Generating Route Description

In [32]:

```
# generate route instructions
def generate_route_descriptions(vp):
    container = ''
    container_vids = {}
    finals = {}
    for v in vp[1:-1]:
        info = v.split('-V')
        if container != info[0]:
            container = info[0]
            container_vids[container] = []
        container_vids[container].append(int(info[1]))
    print(container_vids)
    for container, vids in container_vids.items():
        cidx = ie.containers_names.index(container)
        vg = vgs[cidx]
        print(cidx)
        print(vids)
        rds = vg.generate_route_description(vids)
        finals[container] = rds
    return finals

def print_route_descriptions(rd_dictionary):
    containers = list(rd_dictionary.keys())
    for container in containers:
        rd = rd_dictionary[container]
        if containers.index(container) < len(containers) - 1:
            rd[len(rd)-1] = rd[len(rd)-1].replace('until you reach the destination', 'to enter {}'.format(containers[containers.i
        for r in rd:
            print(r)
# vg.generate_route_description(vp)
print_route_descriptions(generate_route_descriptions(vp))
```

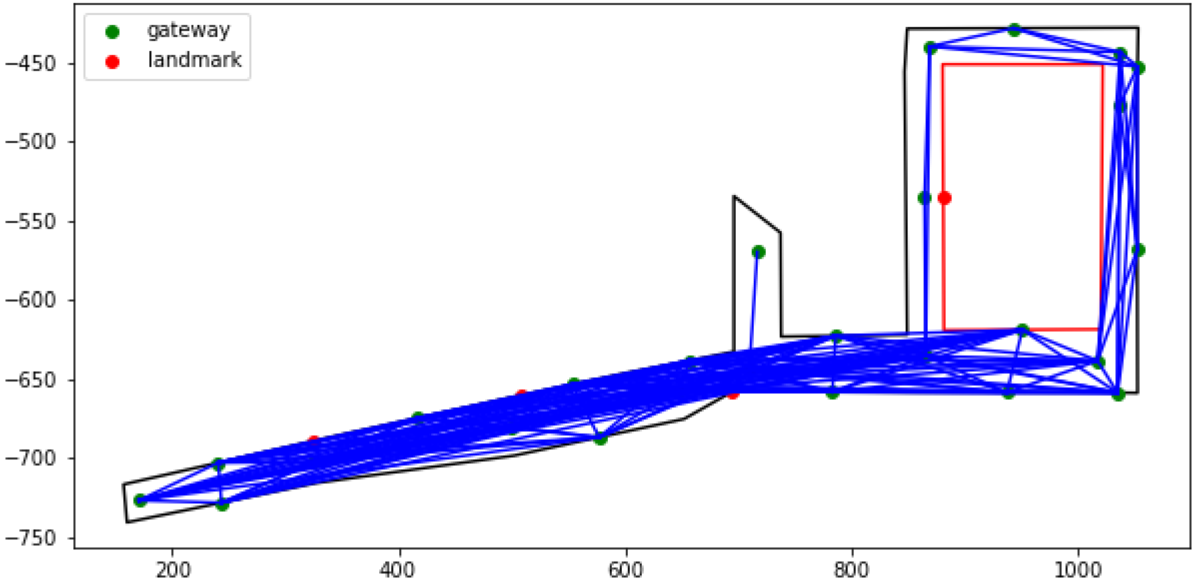
```
{'Ice Kirin Bar': [9, 10], 'Corridor': [1566, 1567, 1568, 1569, 1570, 1571, 1572, 1573, 1574, 1575, 1576, 2159, 2160, 2161, 247
1, 287, 633, 2680, 2681, 4001, 4002]}
0
[9, 10]
15
[1566, 1567, 1568, 1569, 1570, 1571, 1572, 1573, 1574, 1575, 1576, 2159, 2160, 2161, 2471, 287, 633, 2680, 2681, 4001, 4002]
Head towards the door to Corridor and move forward to enter Corridor
Head towards the door to Ice Kirin Bar
Pass decision point 11 and turn left
Follow decision point 7 on the front and turn right and move forward until you reach the destination
```

Creating Navigation Graphs - Door-to-Door

In [33]:
```
# selecting a space
cidx = ie.containers_names.index('Corridor')
vg = vgs[cidx]
isovist_object = isovist_objects[cidx]
```

In [34]:
```
# derive door-to-door visibility graph (doors and decision points)
connected, dtd_graph = vg.generate_door_to_door_graph(isovist_object)

plotter = Plotter()
plotter.add_isovist(isovist_object)
plotter.add_points_lines(connected)
plotter.show()
plotter.close()
```

```
generate door-to-door graph, only_doors False from view graph
```

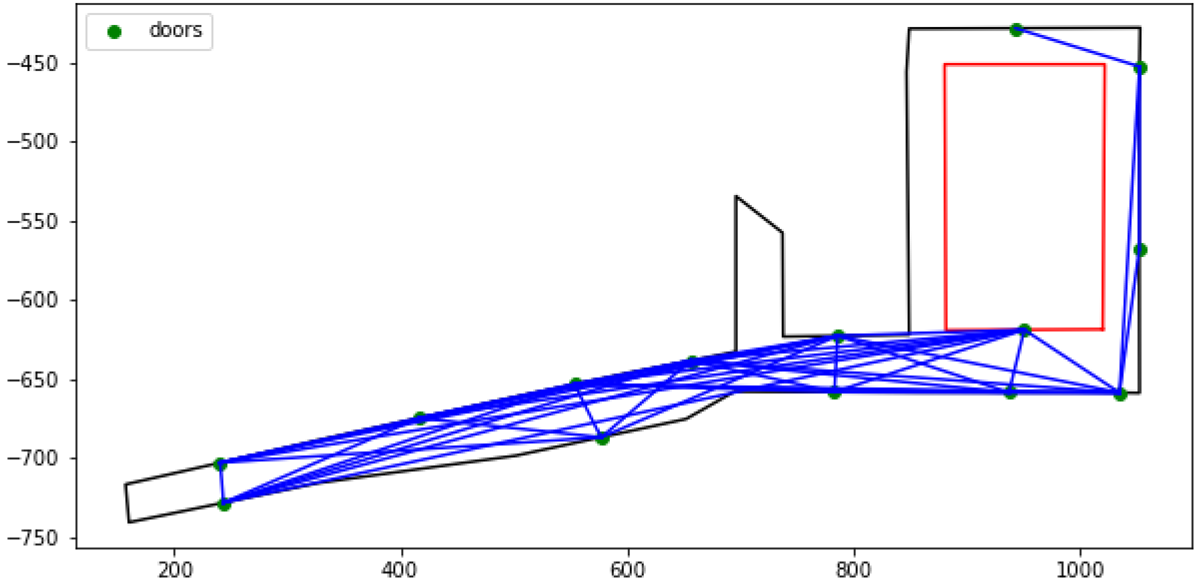



```
<Figure size 720x360 with 0 Axes>
```

In [35]:
```
# derive door-to-door visibility graph (only doors)
connected2, dtd_graph2 = vg.generate_door_to_door_graph(isovist_object, only_doors=True)
plotter = Plotter()
plotter.add_poly(isovist_object.space_x, isovist_object.space_y)
```

```
plotter.add_holes(isovist_object.holes_x, isovist_object.holes_y)
plotter.add_points(isovist_object.door_points[:isovist_object.door_idx], 'doors')
plotter.add_points_lines(connected2)
plotter.show()
plotter.close()
```

```
generate door-to-door graph, only_doors True from view graph
```




```
<Figure size 720x360 with 0 Axes>
```

Creating Place Graph

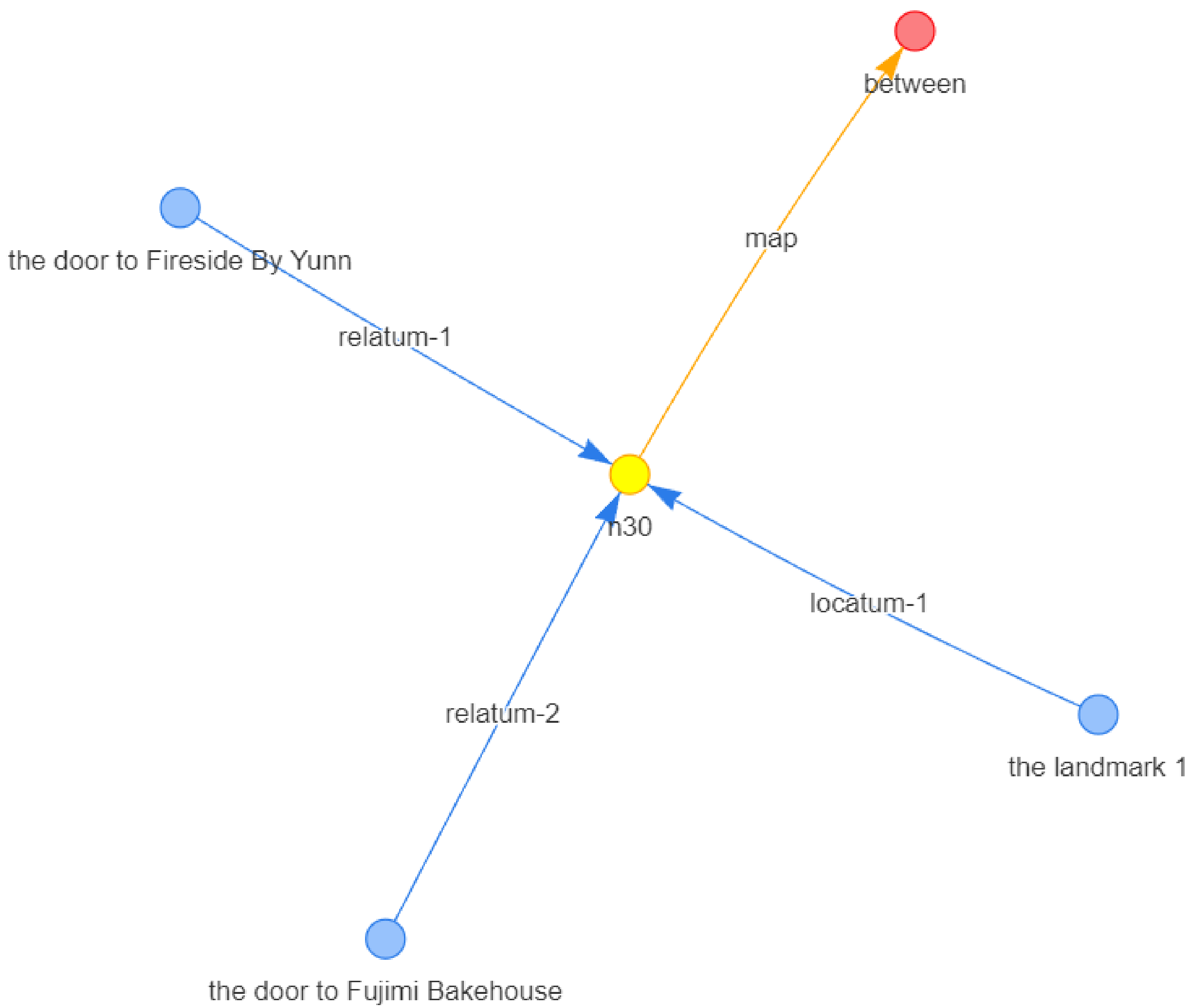
In [36]:
```
# derive place graph
place_graph = vg.generate_place_graph(isovist_object)
```

```
derive place graph from view graph
```

In [37]:
```
# selecting a space
cidx = ie.containers_names.index('Corridor')
vg = vgs[cidx]
isovist_object = isovist_objects[cidx]

def nplet_extraction(nplet_id):
    ## nplet_id = 'n830'
    place_graph[nplet_id]  # left
```

```
    # place_graph['n100'] # between

    # nodes = ['n830', 'left', 'place12', 'gateway 12', 'landmark 20', 'gateway 1']
    nodes = [nplet_id]
    nodes.extend(list(dict(place_graph[nplet_id]).keys()))

    additional = []
    for node in nodes:
        if node.startswith('place'):
            additional.extend(list(dict(place_graph[node]).keys()))
    nodes.extend(additional)

    for v in list(place_graph.edges):
        if v[1] == nplet_id:
            nodes.append(v[0])
            if v[0].startswith('place'):
                nodes.extend(list(dict(place_graph[v[0]]).keys()))

    nplets = place_graph.subgraph(nodes)
    nt2 = Network(width='1000px', height='600px', directed=True, notebook=True)
    nt2.from_nx(nplets, show_edge_weights=False)
    nt2.options.physics.use_repulsion({'node_distance': 185, 'central_gravity': 0.2, 'spring_length': 200,
                                        'spring_strength': 0.05, 'damping': 0.09})
    return nt2, nodes

spatial_expression = 'the landmark 1 between the door to Fireside By Yunn and the door to Fujimi Bakehouse'
for n in place_graph.nodes:
    if n.startswith('n') and 'exp' in place_graph.nodes[n].keys() and place_graph.nodes[n]['exp'] == spatial_expression:
        nplet = n
        break
nt2, nplet_nodes = nplet_extraction(nplet)
nt2.show('nplet_1.html')
```

Out[37]:


between
map
the door to Fireside By Yunn
relatum-1
n30
locatum-1
relatum-2
the landmark 1
the door to Fujimi Bakehouse

In [ ]: