Residential Buildings Layouts Generator

Ahmed Soror and Mohamed Hossam Fawzy

German University in Cairo https://www.guc.edu.eg/

Abstract. Spatial Layout is a problem encountered in many real-world domains, e.g., building floor plans in Architecture. An architect typically aims to divide a specific area into different high-level components and try to find suitable placements for some sort of low-level modules inside these high-level components. In the frame of a residential building, these high-level components could correspond to apartments, corridor, stairs, and elevator rooms and may also include units like ducts (next to modules like kitchens or bathrooms), while the low-level modules include typically different room types such as bedrooms, kitchens, bathrooms, living room, sun-room, dressing room, hallways or corridors inside an apartment, etc. A residential building layout plan includes one obvious restriction, no two different units (e.g., rooms) can be placed in the same position on the floor. Moreover, many other constraints could be in the mind of the architect. In this project, we build a solver using swi-prolog and clpfd to find a solution for such a problem.

Keywords: Constraint programming · clpfd.

1 CSP Model

1.1 Conventions

1. Cartesian coordinates system is used.
2. X & Width represent horizontal axes.
3. Y & Height represent vertical axes.
4. (x, y) represents the top left corner of the floor, room, ... etc

1.2 Variables

1. **Floor= [Floor Width, Floor Height, Sides]**
   (a) A list that represents all information about the floor.
   (b) Floor width: the width of the floor.
   (c) Floor height: the height of the floor.
   (d) Sides = [North,West,South,East]:
   represents sides which are open of a building using integer mapping.
      - 0: closed.
      - 1: Open area.
      - 2: Landscape view.
2. **Apartments List:** list of apartment types where:

   (a) Apartment type:
   [Type Count,[Rooms Count,[Room 1,Room 2, ..]]]
   (b) Room:
   [[roomType, Min Area, Width, Height, AssignedRoom], [X, width, Y, Height]]
3. **Corridors Count:** Represents the total number of corridors in the floor. It can be entered as a value or left as a variable.
4. **Ducts Count:** Represents the total number of ducts in the floor. It can be entered as a value or left as a variable.
5. **Soft Constraints options list:** list represents which soft constraint options to apply for each apartment. Each member of the list is a list of size 4 corresponding to the 4 optional soft constraints available and in the order mentioned in subsection 1.4. The 4 entries are either 0 if the constraint is not required or 1 if it is required, except for the second soft constraint where the user wants to specify that the distance between two rooms is either greater or less than some distance limit. A list is entered in this case instead of 1 and it specifies room1, room2, distance should be greater or not and the distance limit itself.
6. **Global Constraints options list:** list represents which global constraint options to apply for each apartment. The list is of size 4 corresponding to the 4 optional global constraints available and in the order mentioned in subsection 1.4. The 4 entries are either 0 if the constraint is not required or 1 if it is required.
7. **Results:** Show the placement of rooms on the floor if there is a feasible solution.

1.3 Domains

1. all X's coordinates: 0.. Floor Width
2. all Y's coordinates: 0.. Floor Height

1.4 Constraints

- **Hard Constraints**
  1. Neither apartments nor their rooms should overlap.
  2. There must be paths (hallways) that makes every room accessible inside in an apartment.
  3. There must be paths (corridors) that makes all apartments as well as stairs and elevators rooms accessible.
  4. Whole space of a floor must be utilized.
  5. Kitchens, and Bathrooms must be adjacent to ducts.
  6. Sun room have got to be exposed to day-lightening.
  7. Dressing room must be adjacent to the bedroom assigned to it.
  8. Minor bathroom if assigned to a room, then must be exactly adjacent to it.

9. If there exists a dining-room, it must be exactly adjacent to the kitchen.

- **Optional Soft Constraints**
  1. All rooms shall be exposed to some form of day-lighting by placing them on open sides of the building.
  2. Distance between two specific rooms shall be greater/less than some value. (Specified by the user input)
  3. Bedrooms in an apartment shall be as close to each other.
  4. Main bathroom (regular ones) shall be as easy to get from every other room, specially the living room.
- **Optional Global View Constraints**
  1. All apartments should have look on landscape view.
  2. All apartments should be of an equal distance to the elevators unit.
  3. Symmetry Constraints over floor or over same type apartments.
  4. Allocate spaces with ratios close to the golden ratio.

2 Implementation[1]

2.1 Solver predicate

- **Input:** Floor Information, Apartments list, Corridors Count, Ducts Count, Optional Soft Constraints, Optional Global Constraints, Results
- **workflow:** First, stairs & elevator, corridors and ducts units are initialized. Corridors and ducts are created based on the user input, whether the user specifies a specific number of corridors and ducts in the floor or the user left it to the system to compute.

  disjoint2/1 built-in predicate is used to make sure that all rooms are not overlapping. Moreover, to ensure that every room is accessible within the apartment, we check that there exist at least one hallway in the apartment that is adjacent to the room, and eventually we make sure that all hallways are adjacent to each other within the same apartment. Same applies in case of accessible apartments and accessible stair & elevator unit where the check is that there exists at least one corridor that is adjacent to a hallway in the apartment (Apartment's door for sure is placed on a hallway), all corridors in the floor are adjacent to each other and stairs & elevator unit is adjacent to a corridor.

  Next, apply hard constraints, optional soft constraints and optional global constraints on the apartments then search for a feasible solution. The search is performed to maximize the total used area of the floor and minimize all costs obtained from applying the soft constraints. A combination of search techniques is used to accelerate the search process.

[1] Please refer to the code file that contains all implemented predicates. All predicates are well documented.

2.2 Constraints predicates

Constraints are gathered in predicts: consistentApartments, optionalGlobalConstraints.

- **consistentApartments:**
  1. ensures that all rooms are accessible in each apartment consistentRooms(Apartment).
  2. apply hard constraints on rooms within each apartment roomConstraint(Floor,Apartment,Apartment,DuctsList).
  3. ensures that every apartment is accessible within the floor floorConstraint(Apartment, Corridors).
  4. apply optional soft constraints on each apartment and store the cost in the cost variables softContraints(Floor, Apartment, ApartmentSoftConstraintOptions, DayLightCost, RoomsDistanceCost, BedroomsCloseCost, BathroomCloseCost).

- **consistentRooms:** Makes sure that every room in the appartment is accessible.for every room in the apartment, there should exist a hallway in the apartment that is adjacent to the room. In addition, all hallways are adjacent to each other.
- **roomConstraint:** Responsible for applying all hard constraint mentioned in subsection 1.4.
- **floorConstraint:** For every apartment, there must be a hallway in the apartment that is adjacent to a corridor in the floor so that all apartments are accessible.
- **softContraints:**
  - **parameters:** Floor, Apartment, ApartmentSoftConstraintOptions, DayLightCost, RoomsDistanceCost, BedroomsCloseCost, BathroomCloseCost where:
    1. Floor: carries information about the floor.
    2. Apartment: list of rooms of the apartment.
    3. ApartmentSoftConstraintOptions: list of flags represents which soft constraints to be considered in the apartment.
    4. DayLightCost: Represents the cost for applying daylight soft constraint.
    5. RoomsDistanceCost: Represents the cost for applying distance-between-two-rooms soft constraint.
    6. BedroomsCloseCost: Represents the cost for applying close bedrooms-to-each-other soft constraint.
    7. BathroomCloseCost: Represents the cost for applying easy-access-main-bathroom soft constraint.

  - **daylightConstraint:**
    * adds a cost for every room in the apartment that is neither placed on an open-area side nor on a landscape view side.

- **roomsDistanceConstraintApartment:**
  * given two rooms in the apartment, add cost if the distance between them does not satisfy the given constraint.
- **bedroomsAsClose:**
  * get all bedrooms in the apartment using exclude and sub_string predefined predicates
  * calculate distance between every bedroom and every other bedroom
  * add calculated distance to the cost
- **easyAccessBathroom:** for every apartment:
  * get main bathrooms
  * calculate distance between bathroom and every other room in the apartment
  * calculated distance is summed to represent the cost
  * distance between bedroom and living room (if it exists) is given more weight to be prioritized when minimizing the total cost.

- **optionalGlobalConstraints:**
  - **parameters:** Floor,Apartment, Corridors, Stairs, Min, Max and list of constraints to be applied. where:
    1. Floor: carries information about the floor.
    2. Apartment: list of rooms of the apartment.
    3. Corridors: list of corridors in the floor.
    4. Stairs: represents the stairs & elevator unit.
    5. Min, Max: used to add some tolerance when placing apartments at equal distance from the stairs
    6. constraints to be applied: list of flags (0 or 1) that indicates whether each constraint is applied or not.
       [Landscape view, EqualDistanceToElev, Symmetry, Golden ratio]

  - **landScapeView:** for every apartment in the floor:
    * there exists a room belongs to the apartment and is placed on a landscape view side
  - **equalDistancesToElev:** for every apartment:
    * calculate the distance between the apartment and the stair-elevator unit
    * Apartment is represented by the hallway adjacent to the floor's corridor
    * a min and max values are used to add some tolerance to the constraint, where the constraint is satisfied if the distance between the apartment and the stairs-elevator unit us greater than some value and less than some value.
  - **goldenRatio:** for every apartment,adds constraint that the room follows the golden ratio, where assuming that the room sides are A, b where A¿B, then the golden ratio is applied as following:

$$(A+B)/A >= 1.6, and (A+B)/A =< 1.7 \quad (1)$$

2.3 Helpers predicates

- **getAppartmentsNTimes(N,ApartmentType,R):**
  repeats apartment type N times with different variables to be assigned later on.
- **getConstraintsNTimes((N,SoftConstraintsOptions, Result)):**
  repeats soft constraints options N times to match the N apartments of the same type .
- **getRects(Apartments List, Rects List, VarsX, VarsY, Total Used-FloorArea)**
  converts rooms in the apartment list to a list that can be used as a parameter to disjoint2 predicate. the result is a list of rect(X, W, Y, H). In addition all X's coordinates, Y's coordinates as well as the total area are returned in the header of the predicate, to be used in determining the domain and maximizing the total used area of the floor.
- **getters predicates**
  used to get specific room types. e.g., getHallways, getKitchens, getAssigned, .. etc.
- **adjacent(Room A, Room B)**
  checks if the two rooms are adjacent or not. Two rooms are considered to be adjacent if and only they intersect in more than one point and they are not the same rooms.