German University in Cairo

Snap Plan: A Constraint Programming Approach to Architectural Floor Planning

Abdulrahman Muhammad

Mohamed Farid

January 4, 2021

Contents

1 Introduction

1.1 Floor Planning

Humans need shelter. It is a basic need, so buildings are built to accommodate various sheltering needs. Different modules coexist in a building on the same floor and on multiple floors. Relationships between such modules vary from proximity to similar modules to distality and from reachability to inaccessiblity. In architectural design, planning a space such as a floor is performed to allow these different modules to be contained in such space with their relationships maintained.

In this report, we discuss an implemented approach to plan a single floor that contains multiple modules constrained by various relationships to other modules as well as to the floor itself.

1.2 Definitions and Assumptions

We assume a defined rectangular floor of known length and width. We shall refer to the floor sides using the cardinal directions (N, S, E, W). For such floor, we assume a known view or lack thereof for each side, e.g. the N side has a view on a landscape.

We define a floor-level module as a module that is contained in the floor directly and not inside another module. We assume that for a floor, its floor-level modules are a set of apartments, a set of corridors, a stairwell and an elevator. Of these aforementioned modules, only an apartment can contain other modules, such modules are called apartment-level modules.

For floor-level modules other than the apartments and corridors, we impose a constant known rectangular size. We restrict the type of apartment-level modules to 9 different types of rooms. Additionally, each apartment has a defined number of apartment-level hallways and ducts. Ducts are rectangular of constant size. The apartment type is defined by the apartment-level modules contained in such apartment.

The apartment-level modules (excluding ducts) and the corridors are of variables size. All modules are of variable location. We define the location of a module by the X and Y coordinates of its SW and NE corners.

2 Constraint Satisfaction Problem

We define the CSP formally as a 3-tuple (**V**, **D**, **C**) where:

- **V** is a set of locations of all apartment-level modules and non-apartment floor-level modules,
- **D** is the domains of variables in V as 0 to floor width for the X component and 0 to floor length for the Y component,
- **C** is the list of constraints.

We will further discuss **C** in the upcoming subsections.

2.1 Hard Constraints

The following constraints are hard constraints and will prove the problem infeasible if they are inapplicable.

- Pairwise no-overlap over all modules defined in **V**.
- The area of the floor is the total area enclosed by all modules defined in **V**.
- Adjacency of specific apartment-level modules.
- Adjacency of specific apartment-level modules to a defined floor side.

- An apartment-level module (other than a duct) is adjacent to a hallway in the same apartment.
- An apartment is adjacent to a corridor.
- A hallway is adjacent to another hallway and similarly for corridors.

2.2 Soft Constraints

The following soft constraints are maximized over but they are not necessarily enforced.

- Adjacency of all room apartment-level modules to specific floor sides.
- Center-distance between specific apartment-level modules is bound by a defined value.

2.3 Optional Hard Constraints

The following hard optional constraints are applied when requested.

- Adjacency of apartments to specific floor sides.
- Variations in center-distance between apartments and a specific floor-level module is less than a defined value.
- For every apartment type, for every apartment in that apartment type class there exists another apartment in the same class that is the reflected version of the former apartment over a central horizontal or vertical axis of the floor.
- The ratio between the dimensions of an apartment-level (other than a duct or a hallway is within a defined margin of error of the golden ratio.

3 Implementation

We implemented the CSP using Google's OR-Tools in Python. It is to be noted that using such implementation library has required the creation of multiple reified boolean variables to aid in applying the constraints in **C**.

Our implementation consisted of the following:

- Models: containing an OOP structure of the floor, apartment, floor-level module, and other relevant modules.
- IO: handling of parsing the JSON input and visualization.
- Constrainers: reification of base constraints, enforcement of adjacency, distance, symmetry, area and other derived or deriving constraints.

4 Discussion

This approach of floor planning appears capable of handling floor sizes in the double digits. As the size, the number of components or relationships increase the solver takes considerably more time to produce the solution. This increase in time is due to the exponential increase in the number of possible floor plannings that are constrained.

It is to be noted that this approach despite requiring reifications due to the lack of support in the used library, appears significantly easier to implement than an imperative or functional-based approach. This ease of programming is, however, hampered by arcane errors of the used library, terse documentation as well as lack of community support.