Overview

In the domain of product configuration, performance of available algorithms and tools for solution finding is important. This document specifies an example problem, which is similar to problems we encountered in different real-world domains of our product configurators.

Problem description

A museum has lots of rooms and doors between some of them. In order to prevent damage to the objects in exhibition, the number of visitors shall be restricted. This is done by a people counting system which consists of following components: door sensors, counting zones, and communication units.

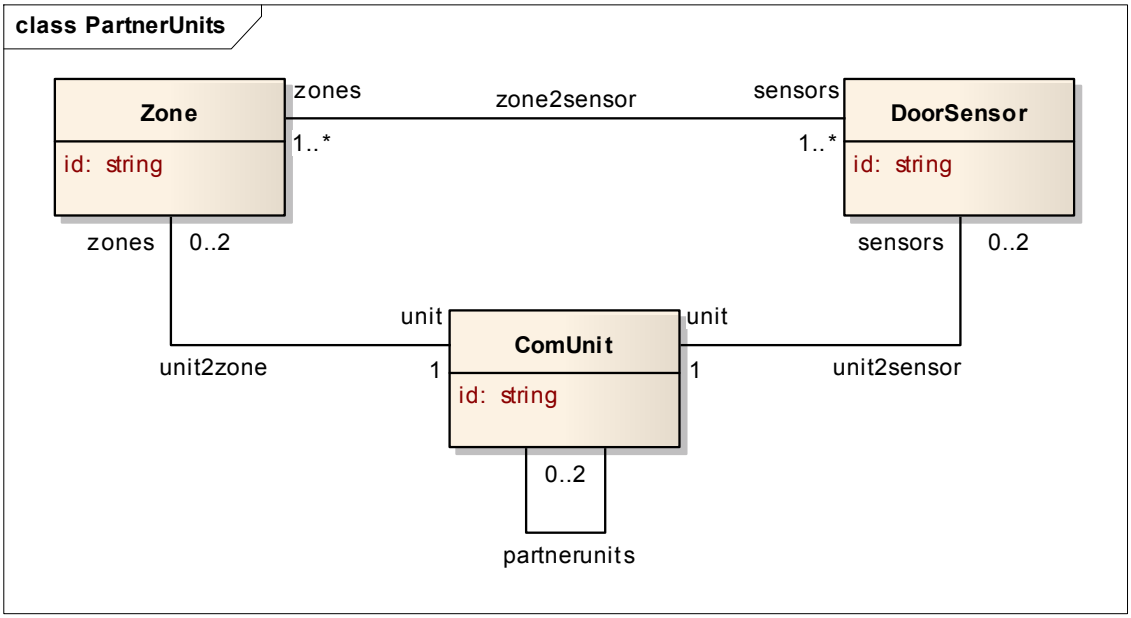
A door sensor detects everybody who moves through its door (directed movement detection). There can be doors without a sensor.

Any number of rooms may be grouped to a counting zone. Each zone knows how many persons are in it (counting the information from the sensors at doors leading outside of the zone - doors between rooms of the zone are ignored). Correct function requires that all doors leading outside a zone have a sensor (the corresponding constraint is not part of this problem). Zones may overlap or include other zones, i.e. a room may be part of several zones.

A communication unit can control at most two door sensors and at most two zones. If a unit controls a sensor which contributes to a zone on another unit, then the two units need a direct connection: one is a partner unit of the other and vice versa. Each unit can have at most N partner units. For the sake of simplicity, we use N=2 throughout this paper, whereas higher values for N are more common in real-life problems of this kind. Of course, the problem diminishes or even vanishes when N is chosen sufficiently high or unbounded, but we assume technical reasons inhibiting high values.

PartnerUnits Problem: Given a consistent configuration of door sensors and zones, find a valid assignment of units (i.e. with max. N partners) striving for a minimal number of units.

UML diagram (for N = 2):



Real-world examples have following properties:

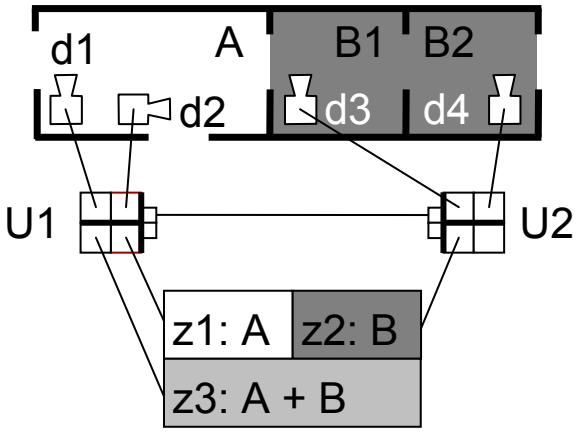
- ca. 100 zones and 100 sensors (could go up to 500 each)
- mostly 2 - 4 sensors per zone
- average 3 zones per sensor (in the simplest case only 2)

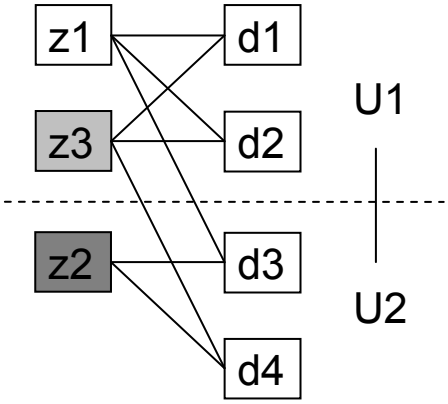
Examples

simple-3 (3 zones, 4 sensors, 2 units):

- with N = 2 (max. 2 partner units)

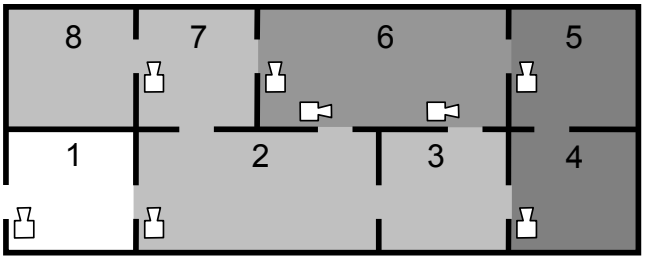
represented as bipartite graph:

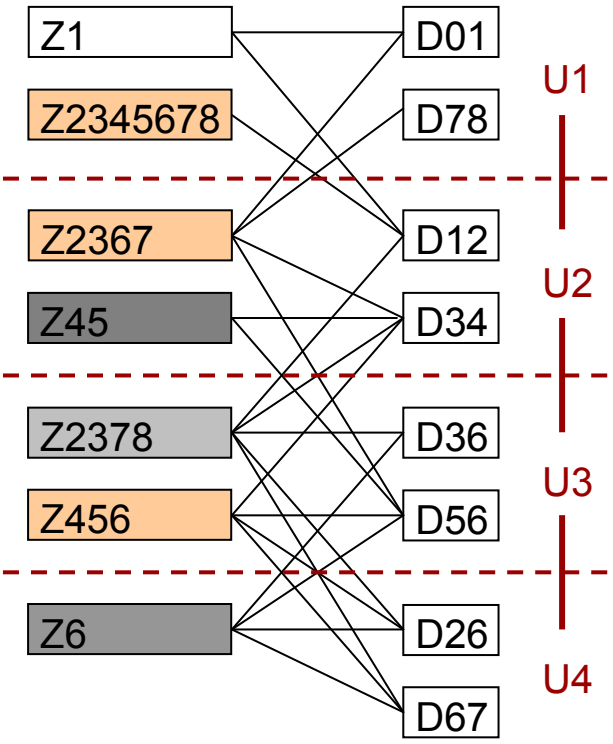




small-7 (7 zones, 8 sensors, 4 units):

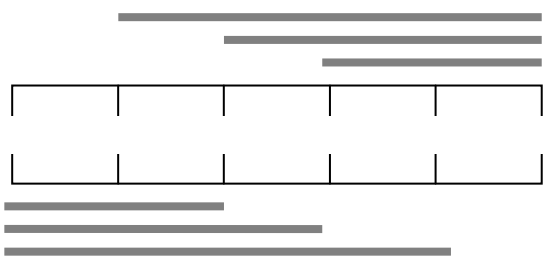
- other small examples are described in FalknerHS_2010_CWS
- with N = 2





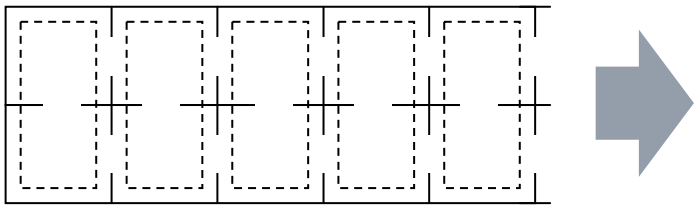
single-11 (11 zones, 6 sensors, 6 units):

- a highly packed configuration with 22 connections between zones and sensors
- with N = 2



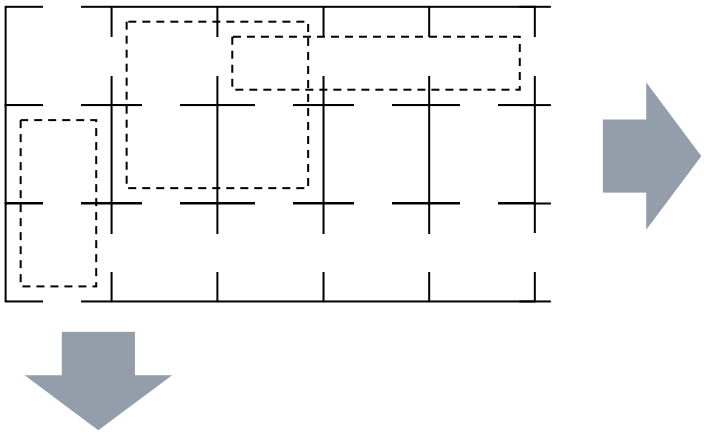
double (number of zones given as parameter, growing to the right):

- a double row of connected rooms, each room being a zone
- a variant has additional zones for each 2 connected rooms vertical to the row
- with N = 3 - or 4 for the variant, respectively



triple (number of zones given as parameter, growing downwards with width 10 to the right):

- a weakly connected group of rooms, each room being a zone
- in some cases with additional 2 or 4 zones consisting of 2-4 rooms
- with N = 4



All examples are available as textfiles in folder dlv: they contain all tuples $zd(z_i,d_j)$ of the relation between zones z_i and door sensors d_j. By that, they list implicitely the names of all instances of zones and sensors. Furthermore, they contain the maximal number of partner units per unit (i.e. N) and a lower bound for the necessary number of units (as comments).

TODO: In real-world, there is not so much regularity as in the examples above, except for the stochastic values mentioned at the end of section "Problem description" above: the vast majority of sensors per zone is 2 and 3, sometimes 1 or 4, higher values occuring very rarily.

There is a big subclass of the problem where sensors have max. 2 zones (i.e. there are no overlapping zones). Mostly, there are only few overlapping zones (< 20%) in the remaining scenarios.

For near-realistic examples, one could take a rectangle of rooms (e.g. with x=8, y=10) and let every room be a zone. Assuming that every side of a room may have at most one door, there are 2x+2y possible doors to the outside and 2xy-x-y possible inside doors. Randomly choose 6 of the 36 outside and 94 of the 142 inside ones, so that you have 100 doors (each having a sensor). Repeating the selection leads to a set of distinct examples without overlapping zones (for testing the mentioned subclass). Now one can randomly create additional zones by combining neighbouring rooms (2 rooms, 3 rooms, 4 rooms, even more). By that, each examples grows with additional zones, e.g. four times adding 5 zones leading to a series of examples with 80, 85, 90, 95, and 100 zones. There is a scenario in the real world where those additional zones are not distributed over the whole rectangle, but start from the outside doors (similar to the "single" example above).

Results

The examples were processed with various programs (see sections below).

The following table summarizes the results as the time in seconds on a 2 GHz PC (WindowsXP) for finding a valid solution with various algorithms (or in the case of small-no, for finding a proof that the problem has no solution):

Example	Back-tracking	Greedy	Domain-specific	Simulated Annealing	CSP (Choco)	ASP (DLV)	SAT (Kodkod)
small-7	1	1	-	1	1	1	1
small-8	1	-	-	1490	251	8	1
small-no	12	-	-	-	-	-	79
single-11	6	-	-	-	155	2	1
double-20	1	1	3	35	1	-	25
doublev-30	-	55	15	-	-	-	130
double-40	1	2	37	149	9	-	-
doublev-60	-	-	16	-	-	-	-
double-60	2	6	135	138	-	-	-
double-80	4	153	-	84	-	-	-
double-100	6	134	-	350	-	-	-
triple-30	3	5	1	5	3	-	602
triple-32	3	6	5	5	-	-	-
triple-34	-	76	1	150	-	-	-
triple-60	11	35	1	113	150	-	-
triple-64	-	-	9	-	-	-	-
triple-90	29	93	1	1209	-	-	-
triple-120	65	1473	-	-	-	-	-

A "-" means that no solution was found within the given timeframe of 1500 seconds or the algorithms gave up before time-out, e.g. due to running out of memory (was limited to 600 MB) or when a domain-specific algorithm got stuck in a local optimum.

Programs

TODO: Describe included Programs: DLV, Alloy, Prolog

DLV:

- Download latest version of DLV-Complex (static-linked exe from http://www.mat.unical.it/dlv-complex)
- Put in folder dlv, rename to dlvc.exe for shorter call, execute directly in a command shell

Problem properties resemble the real world:

- Computational complexity: unknown (seems to be NP)
- Variety of easy and difficult problem instances (more difficult if the bipartite graph is denser)
- Easy but large problem instances are the vast majority (good performance for them is important)
- There might be problem instances without a solution (over-constrained problem)
- Optimization problem: minimize number of units? minimize number of partner unit connections
- Solutions are fragile w.r.t. changes of the problem statement (small adjustments may require completely different solving)

Problem is customizable:

- upper bound of partnerunits cardinality (e.g. 4 instead of 2)
- sparse and dense zone2sensor relation (sparser is easier)
- over-constrained and under-constrained problem instances
- number of zones and door sensors

Variants:

- optimization: minimize number of partner units
- higher decision degree: different types of zones / sensors (see FalknerH_2010_ECAI_IKBET)

Literature

FalknerHS_2010_CWS:

- A. Falkner, A. Haselböck, G. Schenner: "Modeling technical product configuration problems", in Proceedings of the ECAI 2010 Workshop on Configuration, p.40-45, 2010.

FalknerHSS_2011_AIEDAM:

- to appear in AIEDAM Special Issue on Configuration.

FalknerH_2010_ECAI_IKBET:

- A. Falkner, A. Haselböck: "Challenges of knowledge evolution in practice", in Proceedings of the ECAI 2010 Workshop on Intelligent Engineering Techniques for Knowledge Bases (IKBET), p.67-72, 2010.
- Contains a variant of the PartnerUnits problem which cannot be simply represented by a bipartite graph.

