PDDL Generators

Jendrik Seipp, Álvaro Torralba, Jörg Hoffmann

A collection of PDDL generators, some of which have been used to generate benchmarks for the International Planning Competition (IPC).

Contents

1 Agricola

Instance generator for the Agricola domain
submitted by Tomas de la Rosa <trosa@inf.uc3m.es>

Description: This domain is loosely based on the board game Agricola. The game models a farm with some workers. The game has a number of rounds and stages, in which the player must select actions for every worker. Some of this actions will obtain resources, and others will increase the number of workers. This allows the player to perform more actions per turn but also increases the amount of food consumed at the end of the turn, potentially leading into dead ends.

2 Assembly

* Origin: Drew McDermott. Used in the AIPS-1998 competition.
* Adaptions: None.
* Description: Typed ADL domain using complex pre- and
  effect-conditions, as well as conditional effects. Assemble one goal
  item by repeatedly putting together several items that are part of the
  same meta-item. The part-of relation defines a tree, where the root is
  the goal item and the leafs are the base items that do not need to be
  assembled. Any item can need a resource (like an ofen) during the
  process of its being assembled. It may be that a ‘‘transient’’ part
  has to be assembled somewhere and removed afterwards, that parts of
  the same meta-item need to be assembled in a specified order (obeying
  assemble order constraints) and that some other part needs to be
  assembled before the transient part is removed again (obeying remove
  order constraints).
* Parameters:
  * -d depth of part-of tree
  * -m maximal number of sons to any node in the tree
  * -h probability that a node has any sons
  * -n number of distinct resources
  * -r probability that a non-base node requires a resource
  * -t probability that an item is transient part for any higher-in-tree item
  * -a probability that a pair of items has an assemble order constraint
  * -o probability that an item has a remove ordering constraint with a transient part
  * -s random seed

```
* Generation: Create a tree of depth -d. Nodes have sons with
  probability -h. If they do have sons, then a random number between 1
  and -m, biased to be lower the deeper the node is in the tree. With
  probability -r, the node needs a random resource. An item is assigned
  a transient part relation to any item in a higher tree level with
  probability -t, parts or transient parts of the same item have an
  assemble ordering constraint with probability -a, and for a transient
  part relation (A, B) the parts of B are given a remove ordering
  constraint to A with probability -o. Cycles in the assemble and remove
  order constraints are avoided by arbitrarily ordering the respective
  items, and allowing constraints only between pairs A and B with A
```

3 Barman

```
Usage: barman-generator.py <num_cocktails> <num_ingredients> <num_shots> [<random_seed>]
  num_cocktails (min 1)
  num_ingredients (min 2)
  num_shots (min max[1,num_cocktails+1])
  random_seed (min 1, optional)


IPC'11 OPT
  1-4: 3 3 4
  5-8: 4 3 5
  9-12: 5 3 6
  13-16: 6 3 8
  17-20: 7 3 9


IPC'14 OPT:
   4 3 5
   5 3 6
   6 3 8
   7 3 9
   8 3 9
```

4 Blocksworld

```
The blocksworld generators use the random bw state generator provided
by John Slaney and Sylvie Thiebaux. This is contained in the bwstates.1
subdirectory. The directories 3ops/2pddl and 4ops/2pddl contain
programs that translate the generator's output to PDDL. The random
instances can then be generated by running the script

./blocksworld <ops> <num>

which takes as input the number of operators (either 3 or 4) and blocks
in the problem to be generated. This is currently maximal 200 due to
the random state generator. To increase that, edit the Makefile in
bwstates.1 and increase SIZE.

## Blocksworld-3ops
```

*   Origin: Goes back to the annals of AI, seems to be first mentioned in Terry Winograd's PhD thesis 1972. In a 1974 planning paper by Gerald Sussman. Taken from the [IPP][6] domain collection.
*   Adaptions: None.
*   Description: Classical untyped STRIPS domain, where stackable blocks need to be re-assembled on a table with unlimited space. Representation uses 3 operators, moving a block from the table to another block, a block from another block to the table, or a block from a block to another block. Semantically, the representation does not use a robot arm, in difference to the 4 operator representation below. The initial state specifies a complete state, the goal state specifies only the on relations required between any two blocks.
*   Parameters: Number of blocks.
*   Generation: Uses random blocksworld state generator provided by John Slaney and Sylvie Thiebaux. Simply translates two such states into PDDL, and prints them out as the initial state and the goal state, where the latter state has all facts removed except the on relations between blocks.

Blocksworld-4ops

*   Origin: See above. Used in the AIPS-2000 competition. Taken from the [IPP][6] domain collection.
*   Adaptions: None.
*   Description: Like above, but uses a robot arm that can be used for stacking a block onto a block, unstacking a block from a block, putting down a block, or picking up a block.
*   Parameters: Number of blocks.
*   Generation: Like above.

5 Briefcaseworld

Briefcaseworld

*   Origin: First mentioned by Edwin Pednault. Taken from the [IPP][6] domain collection.
*   Adaptions: None.
*   Description: Typed classical ADL domain, using conditional effects. Transport a number of objects from their start- to their goal-locations, using a briefcase. Each location is accessible from each other location, objects can be put into the briefcase or taken out of the briefcase. When a move is made between locations, then all objects inside the briefcase are also moved, which is encoded by a conditional effect.
*   Parameters:
    * -o number of objects
    * -s random seed
*   Generation: randomly distribute the start locations of all objects and the briefcase over -o %2B 1 locations. Do the same for the goal locations.

6 Cavediving

The cave system is represented by an undirected acyclic graph. Divers have a single point of entry. Certain leaf nodes of the cave branches are objectives that the divers must photograph. Swimming and photographing both consume air tanks. Divers must exit the cave and decompress at the end. They can therefore only make a single trip into the cave.

Certain divers have no confidence in other divers and will refuse to work if someone they have no confidence in has already worked. Divers have hiring costs inversely proportional to how hard they are to work with.

A typical plan involves a diver laying tanks at layer 1 for a diver to use to lay tanks at level 2, etc.

Anyway, the domain generator and a single problem instance are attached. The generator comes with a decent help menu and the code is reasonably commented throughout.

The generator requires python 2.x.

Help is obtained by running: python generator.py --help

Here is how to generate a typical problem:

python generator.py -domain_file ../domain.pddl -problem_file ../problem.pddl -problem_name 01 -neg_link_prob 0.5 -perturb_hiring_costs 0.1 -cave_branches 3:2 -objectives 3:2 -order_tanks True

cave_branches and objectives specify lists of the depths of the branches of the cave and the objectives, respectively.

-cave_branches 3:2:2 means there will be 1 branch with depth 3, and two with depth 2. The starting points for these branches is determined randomly, but their depths correspond to the total depth from the cave entrance. Objectives must be specified at the ends of cave branches, but not all branches need to have an objective.

For example, -cave_branches 3:2:2 -objectives 2:2 is valid, but -cave_branches 3:3:2 -objectives 2:2 is not.

The plan length is exponential in the depths of the caves.

To make increasingly difficult problems, add duplicate caves and objectives at the same depth, and then increase the depth.

If you are after much harder problems, I suggest setting -order_tanks False and also possibly -other_action_cost 0

Unsolvable instances can be created by creating negative cycles in the diver relationships (set this option > 0)

7 Childsnack

./child-snack-generator.py pool {seed} {num_children} {num_trays} {gluten_factor} {const_ratio}

const_ratio = 1.3 in all opt and sat instances

trays: 2 or 3

OPT:
child-snack_pfile01-2.pddl:; child-snack task with 6 children and 0.4 gluten factor
child-snack_pfile01.pddl:; child-snack task with 6 children and 0.4 gluten factor
child-snack_pfile02-2.pddl:; child-snack task with 7 children and 0.4 gluten factor
child-snack_pfile02.pddl:; child-snack task with 7 children and 0.4 gluten factor
child-snack_pfile03-2.pddl:; child-snack task with 8 children and 0.4 gluten factor
child-snack_pfile03.pddl:; child-snack task with 8 children and 0.4 gluten factor
child-snack_pfile04-2.pddl:; child-snack task with 9 children and 0.4 gluten factor
child-snack_pfile04.pddl:; child-snack task with 9 children and 0.4 gluten factor
child-snack_pfile05-2.pddl:; child-snack task with 10 children and 0.4 gluten factor
child-snack_pfile05.pddl:; child-snack task with 10 children and 0.4 gluten factor
child-snack_pfile06-2.pddl:; child-snack task with 11 children and 0.4 gluten factor
child-snack_pfile06.pddl:; child-snack task with 11 children and 0.4 gluten factor
child-snack_pfile07-2.pddl:; child-snack task with 12 children and 0.4 gluten factor
child-snack_pfile07.pddl:; child-snack task with 12 children and 0.4 gluten factor
child-snack_pfile08-2.pddl:; child-snack task with 13 children and 0.4 gluten factor
child-snack_pfile08.pddl:; child-snack task with 13 children and 0.4 gluten factor
child-snack_pfile09-2.pddl:; child-snack task with 14 children and 0.4 gluten factor
child-snack_pfile09.pddl:; child-snack task with 14 children and 0.4 gluten factor
child-snack_pfile10-2.pddl:; child-snack task with 15 children and 0.4 gluten factor
child-snack_pfile10.pddl:; child-snack task with 15 children and 0.4 gluten factor

trays: 3 or 4

SAT:
child-snack_pfile05-2.pddl:; child-snack task with 10 children and 0.4 gluten factor
child-snack_pfile05.pddl:; child-snack task with 10 children and 0.4 gluten factor
child-snack_pfile06-2.pddl:; child-snack task with 11 children and 0.4 gluten factor
child-snack_pfile07-2.pddl:; child-snack task with 12 children and 0.4 gluten factor
child-snack_pfile08-2.pddl:; child-snack task with 13 children and 0.4 gluten factor
child-snack_pfile08.pddl:; child-snack task with 13 children and 0.4 gluten factor
child-snack_pfile09-2.pddl:; child-snack task with 14 children and 0.4 gluten factor
child-snack_pfile09.pddl:; child-snack task with 14 children and 0.4 gluten factor
child-snack_pfile10-2.pddl:; child-snack task with 15 children and 0.4 gluten factor
child-snack_pfile10.pddl:; child-snack task with 15 children and 0.4 gluten factor
child-snack_pfile11-2.pddl:; child-snack task with 16 children and 0.4 gluten factor
child-snack_pfile11.pddl:; child-snack task with 16 children and 0.4 gluten factor
child-snack_pfile12.pddl:; child-snack task with 17 children and 0.4 gluten factor
child-snack_pfile13-2.pddl:; child-snack task with 18 children and 0.4 gluten factor
child-snack_pfile13.pddl:; child-snack task with 18 children and 0.4 gluten factor
child-snack_pfile14.pddl:; child-snack task with 19 children and 0.4 gluten factor
child-snack_pfile15-2.pddl:; child-snack task with 20 children and 0.4 gluten factor
child-snack_pfile16-2.pddl:; child-snack task with 21 children and 0.4 gluten factor

```
child-snack_pfile19-2.pddl:; child-snack task with 24 children and 0.4 gluten factor
child-snack_pfile19.pddl:; child-snack task with 24 children and 0.4 gluten factor
```

8 Citycar

```
## CITYCAR DOMAIN GENERATOR.
## IPC 2014
## author: Mauro Vallati -- University of Huddersfield
## modifed-by: Masataro Asai

usage: generator.py [-h] [--density DENSITY] [--seed SEED] rows columns cars garages

positional arguments:
  rows               the number of grid rows.
  columns            the number of grid columns.
  cars               how many cars have to go through the network
  garages            the number of starting garages

optional arguments:
  -h, --help         show this help message and exit
  --density DENSITY  The ratio of the available roads in the road network
                     (0.0 <= density <= 1.0). Junctions are randomly set
                     obstructed/unavailable for passage
  --seed SEED        random seed


This model aims to simulate the impact of road building / demolition on
traffic flows. A city is represented as an acyclic graph, in which each
node is a junction and edges are "potential" roads. Some cars start from
different positions and have to reach their final destination as soon as
possible. The agent has a finite number of roads available, which can be
built for connecting two junctions and allowing a car to move between
them. Roads can also be removed, and placed somewhere else, if needed.
In order to place roads or to move cars, the destination junction must
be clear, i.e., no cars should be in there.



pddl-generator note:

See README.old for the usage of the original code by Mauro.

In the original version, sparsity was a 0 / 1 flag which,
when enabled, 20% of the roads become unavailable.

In this version, we can specify the sparsity as a probability (default: 1.0).
To achieve the same sparsity as the original, give 0.8 as a --density parameter.
```

9 Crewplanning

```
This generator has not been used for IPC 2011.
All the problems in this domain came from IPC 2008.
```

IPC 2011 Organizers

pddl-generators note:

Crewplanning domain is a numeric / temporal domain from ipc-2008 .
This generator requires an ocaml and ocamlbuild, but
the source code seems to be missing Utils module, therefore we cannot build it at the moment.

The only part of Utils module being used is two functions:

* Utils.pf, which I assume is similar to printf
* Utils.str, which I assume is similar to sprintf

Therefore, it is relatively easy to rewrite it with Printf.fprintf and Printf.sprintf
in the modern ocaml standard modules.

The original generator produces several files from the same command line argument.
This is rewritten so that each run produces a single pddl file.

10 Data-network

Author: Manuel Heusner

```
usage: generator.py [-h] [--seed SEED] data layers scripts module

positional arguments:
  data         the number of data items
  layers       the number of layers, must be smaller than the number of data items
  scripts      the number of scripts, must be larger or equal than [number of data items]-2
  module       A module describing a network, one of ring-network, small-network, tiny-network.
               See example files in the source code directory.

optional arguments:
  -h, --help   show this help message and exit
  --seed SEED  random seed
```

IPC instances are:

* num_layers in range(2, 6)
* num_data_items in range(num_layers + 2, num_layers * 10, num_layers)
* num_scripts in range(num_data_items + 2, num_data_items * 3, num_data_items // 2)

11 Depots

```
Usage:
  depots -e <#depots> -i <#distributors> -t <#trucks> -p <#pallets> -h <#hoists>
```

```
         -c <#crates> [-s <random_seed>]

        All numbers are positive integers (minimal 1).


IPC instances:
depots from 1 to 6
distributors from 2 to 6
trucks from 2 to 6
pallets from 3 to 20
crates from 2 to 20
hoists from 3 to 15
```

12 Driverlog

```
Usage: dlog [-u|-s|-t|-n|-h] <seed> <#road junctions> <#drivers> <#packages> <#trucks> [distance]
If distance is a positive value then the distances between
locations is set randomly using this value as the bound.

-n: Numeric
-u: Untyped
-t: Timed
-s: Simple time
-h: Hard numeric


 IPC instances

DLOG-2-2-2  to DLOG-8-6-25 (drivers-trucks-packages)
DLOG-2-2-2
DLOG-2-2-3
DLOG-2-2-4
DLOG-3-2-4
DLOG-3-2-5
DLOG-3-3-5
DLOG-3-3-6
DLOG-3-3-7
DLOG-2-3-6
DLOG-2-3-6
DLOG-2-3-6
DLOG-2-3-6
DLOG-2-3-6
DLOG-3-3-6
DLOG-4-4-8
DLOG-5-5-10
DLOG-5-5-15
DLOG-5-5-20
DLOG-5-5-25
DLOG-8-6-25

first instance has 5 locations
last instance has 59 locations
```

Some rules that we could enforce are:

drivers, trucks <= packages, locations

drivers = trucks +- 2

So, I suggest to have three linear scalings: one for trucks/drivers, one for packages, and a final one for locations.

13 Elevators

IPC OPT Instances: All instances have 1 slow elevator

IPC'08 (IPC'11 is a subset)
- 09/2 floors 3 passengers 1 fast elevator
- 09/2 floors 3 passengers 2 fast elevators
- 09/2 floors 4 passengers 1 fast elevator
- 09/2 floors 4 passengers 2 fast elevators
- 09/2 floors 5 passengers 1 fast elevator
- 09/2 floors 5 passengers 2 fast elevators
- 09/2 floors 6 passengers 1 fast elevator
- 09/2 floors 6 passengers 2 fast elevators
- 09/2 floors 7 passengers 1 fast elevator
- 09/2 floors 7 passengers 2 fast elevators
- 13/2 floors 3 passengers 1 fast elevator
- 13/2 floors 3 passengers 2 fast elevators
- 13/2 floors 4 passengers 1 fast elevator
- 13/2 floors 4 passengers 2 fast elevators
- 13/2 floors 5 passengers 1 fast elevator
- 13/2 floors 5 passengers 2 fast elevators
- 13/2 floors 6 passengers 1 fast elevator
- 13/2 floors 6 passengers 2 fast elevators
- 13/2 floors 7 passengers 1 fast elevator
- 13/2 floors 7 passengers 2 fast elevators
- 13/3 floors 3 passengers 1 fast elevator
- 13/3 floors 3 passengers 2 fast elevators
- 13/3 floors 4 passengers 1 fast elevator
- 13/3 floors 4 passengers 2 fast elevators
- 13/3 floors 5 passengers 1 fast elevator
- 13/3 floors 5 passengers 2 fast elevators
- 13/3 floors 6 passengers 1 fast elevator
- 13/3 floors 6 passengers 2 fast elevators
- 13/3 floors 7 passengers 1 fast elevator
- 13/3 floors 7 passengers 2 fast elevators

IPC SAT Instances

3 linear scalings:

passengers: 4-13, 10-26 (+2), 12-39 (+3)

```
floors 08/2, 16/2, 24/3
2 fast elevators
1 slow elevator
capacity 3/2, 4/3, 6/4


There was not Readme.txt in the generator version I have, so this is the way I am using
it. I cannot credit the author as I don't know who was she/he.

- Change the following variables at generate_data.c
   - FLOORS: number of floors
    - AREA_SIZE: a divisor of FLOORS (a slow elevator covers this
     + 1 floors) (a fast elevator skips this/2-1 floors)
- FAST_ELEVATORS: number of fast elevators
- SLOW_ELEVATORS: number of slow elevators covering the same area
- FAST_CAPACITY: persons a fast elevators can hold
- SLOW_CAPACITY: persons a slow elevators can hold

- Use generate_data.c to generate a first txt version of the
  problems. Parameters are:
       -  MinPassengers : the number of passengers of the first problem
    -  MaxPassengers : the number of passengers of the last problem
    -  Step : the step to go from MinPassengers to MaxPassengers
    -  MinID : the start number of number problems of the same size
    -  MaxID : the end number of number problems of the same size

- If you want to change the floors, area size, etc. do again the two
  previous steps.

- Decide which version you want to create by putting to 1 the value
  of the following variables of file generate_pddl.c:
     - int temporal=0;
     - int numeric=0;
     - int net_benefit=0;
  If all are 0, the STRIPS with action-costs version is generated.

- Use generate_pddl.c to create the pddl problems. Parameters are:
  - Min number of floors of the problems (first number of problem name)
  - Max number of floors of the problems (first number of problem name)
  - Floor step: difference of floors from one problem to the next one
  - MinPassengers : the number of passengers of the first problem
   (second number of problem name)
  - MaxPassengers : the number of passengers of the last problem
   (second number of problem name)
  - Step : the step to go from MinPassengers to MaxPassengers
  - MinID : the start number of the first problem of the same size
   (third number of the problem name)
  - MaxID : the end number of the last problem of the same size
   (third number of the problem name)

EXAMPLE

If in generate_data.c we have:
#define FLOORS 25
```

```
#define AREA_SIZE 6
#define FAST_ELEVATORS 3
#define SLOW_ELEVATORS 2
#define FAST_CAPACITY 6
#define SLOW_CAPACITY 4

and we execute:

generate_data 40 50 3 1 1

we will obtain the following files:
p25_40_1.txt
p25_43_1.txt
p25_46_1.txt
p25_49_1.txt


Then, we can call

generate_pddl 25 25 1 40 49 3 1 1

to obtain files

p25_40_1.pddl  p25_43_1.pddl  p25_46_1.pddl  p25_49_1.pddl
```

14 Ferry

```
## Ferry

*   Origin: ?. Taken from the [IPP][6] domain collection.
*   Adaptions: Sail operator modified so that moves can only happen
    between different locations.
*   Description: Untyped STRIPS domain. Transport a number of cars from
    their start- to their goal-locations, using a ferry. Each location is
    accessible from each other location, cars can be debarked or boarded,
    the ferry can always carry only one car at a time.
*   Parameters:
    *   -l number of locations
    *   -c number of cars
*   Generation: randomly distribute the start and goal locations of all
    cars over the locations.
```

15 Floortile

```
./floortile-generator.py <name> <num_rows> <num_columns> <num_robots> <mode_flag(seq|time)> [<seed>]

num_robots: 2-4

3x3, 3x4, 4x4, ..., 7x7
```

16 Freecell

```
## Freecell

* Origin: Fahiem Bacchus. Used in the AIPS-2000 competition.
* Adaptions: Domain encoding modified such that cards, freecells, and columns
  all have their own natural numbers. This helps avoiding superfluous action
  instances.
* Description: Typed STRIPS encoding of a card game (similar to Solitaire)
  that comes free with Microsoft Windows. Given a random configuration of cards
  across some columns, move all cards in a specified order onto some goal
  stacks, obeying a number of stacking rules, and using a number of freecells as
  a resource.
* Parameters:
  * -f number of freecells (minimal 0)
  * -c number of columns (minimal 1)
  * -s number of suits (minimal 1, maximal 4)
  * -i number of initial stacks (minimal 1)
  * -0 .. -3 number of cards in each suite
  * -r random seed (optional)
* Generation: As long as there is a card that has not yet been placed somewhere,
  choose one such card at random, and place it randomly on one initial stack.

IPC Instances

In the IPC there were two separate sets. All instances have 4 suits.

SET1:
4 suits, 3 to 13 cards per suite, from 4 to 8 columns

SET2: Scales cards from 2 to 13, 4 instances per size.
```

17 Fridge

```
## Fridge

*   Origin: Tony Barrett. Taken from the [IPP][6] domain collection.
*   Adaptions: Several adaptions made to allow for a flexible number of
    screws on each backplane, and to force a fridge being turned off when
    the screws of the backplane are removed.
*   Description: Typed ADL domain using comlex ADL preconditions (that
    simplify to STRIPS constructs after instantiation). Original was
    STRIPS domain, adaption uses quantification over all screws in
    preconditititions, to allow for a flexible number of those. For a number
    of fridges, unfasten the screws holding the backplane, then remove the
    backplanes and exchange the broken compressor with a new one, then
    re-assemble the fridge and turn it on.
*   Parameters:
    *   -f number of fridges
    *   -s number of screws per backplane
    *   -r random seed
*   Generation: No randomization. Simply specify all static relations as
    well as the initial and goal situations.
```

18 Goldminer

A robot is in a mine and has the goal of reaching a location that
contains gold. The mine is organized as a grid with each cell either
begin hard or soft rock. There is a special location where the robot
can either pickup an endlesssupply of bombs or pickup a laser cannon.
The laser cannon can shoot through both hard and soft rock, whereas the
bomb can only penetrate soft rock. However, the laser cannon also will
destroy the gold if used to uncover the gold location. The bomb will
not destroy the gold. The problem difficulty is scaled by increasing
the size of the grid.

This domain has a simple optimal strategy:
1) get the laser cannon,
2) shoot through the rock until reaching a cell bordering the gold.
3) go and get a bomb.
4) blast away the rock at the gold location.
5) pickup the gold.

19 Grid

Grid

* Origin: Drew McDermott. Used in the AIPS-1998 competition.
* Adaptions: None.
* Description: Untyped STRIPS domain. A robot moves along a rectangular grid
  where positions can be locked. Locks can be opened with matching keys, and
  the goal is to have some of these keys at specified locations.
* Parameters:
  * -x horizontal extension of grid
  * -y vertical extension of grid
  * -t number of different key and lock types
  * -p probability, for any key, to be mentioned in the goal
  * -k number of keys vector (one 0 .. 9 entry for each type)
  * -l number of locks vector (one 0 .. 9 entry for each type)
  * -r random seed (optional)
* Generation: Randomly distribute the robot, lock and key positions over the
  grid. No two locks can be at the same location, and the robot must not start
  on a locked position. If a key is required to be mentioned in the goal, then
  generate a random goal location for it.

One current issue is that locked spots are distributed entirely randomly. It
would be more interesting if we have some kind of "rooms" scenario where is more
likely that one must traverse via some locked places.

IPC instances:

-x 5 -y 5 -t 4 -k 8 -l 8 -p ? -s 0

20 Gripper

Gripper

```
*   Origin: Jana Koehler. Used in the AIPS-1998 competition.
*   Adaptions: None.
*   Description: Untyped STRIPS domain. Given a robot with two gripper
    hands, transport a number of balls from a room A to another room B.
*   Parameters: -n number of balls.
*   Generation: No randomization. Place all balls in room A and require
    them to be in B instead.
```

21 Grippers

```
Gripper variant with multiple robots and more than two rooms.
```

22 Hanoi

```
## Hanoi

*   Origin: ?. Taken from the [IPP][6] domain collection.
*   Adaptions: None.
*   Description: Untyped STRIPS encoding of the well-known Towers of Hanoi
    problem.
*   Parameters: -n number of discs.
*   Generation: No randomization.
```

23 Hiking

```
Hiking
======

Author: Lee McCluskey.

Imagine you want to walk with your partner a long clockwise circular route over several
days (e.g. in the "Lake District" in NW England), and you do one "leg" each day. You want
to start at a certain point and do the walk in one direction, without ever walking
backwards. You have two cars which you must use to carry your tent/luggage and to carry
you and your partner to the start/end of a leg, if necessary. Driving a car between any
two points is allowed, but walking must be done with your partner and must start from the
place where you left off. As you will be tired when you've walked to the end of a leg, you
must have your tent up ready there so you can sleep the night before you set off to do the
next leg the morning.


usage: generator.py <n_couples> <n_cars> <n_places> [<seed>]
         for solvability, cars should be at list n_couples + 1.
```

24 Logistics

```
## Logistics

* Origin: First version by Manuela Veloso, AIPS-1998 version created by Bart Selman
  and Henry Kautz. Used in both the AIPS-1998 and AIPS-2000 competitions.
* Adaptions: None.
```

```
* Description: Classical untyped STRIPS domain. Transport packages within cities via
  trucks, and between cities via airplanes. Locations within a city are directly
  connected (trucks can move between any two such locations), and so are the
  cities. In each city there is exactly one truck, each city has one location
  that serves as an airport.
* Parameters:
  * -c number of cities
  * -s size of each city, i.e. number of locations within cities
  * -p number of packages
  * -a number of airplanes
* Generation: Place trucks randomly within their cities, place airplanes randomly at
  airports. Distribute start and goal locations of packages randomly over all
  locations.

OPTIONS   DESCRIPTIONS

-a <num>    number of airplanes
-c <num>    number of cities (minimal 1)
-s <num>    city size (minimal 1)
-p <num>    number of packages (minimal 1)
-t <num>    number of trucks (optional, default and minimal: same as number of cities;
            there will be at least one truck per city)
-r <num>    random seed (minimal 1, optional)


IPC instances:

IPC00: from 4 to 15 packages, smaller instances have 2 cities, larger ones have 5 cities.
There are instances with 1 or 2 airplanes.

IPC98: cities are scaled linealy, with one more city per instance. There is an outlier
with 47 cities, all others have around 30 something. There are instances with up to 15
airplanes. Packages scale up to 57.  Sometimes there is more than one truck per city.
```

25 Maintenance

```
Appeared in IPC2014.
Author: Jussi Rintanen.

This is a simple planning/scheduling problem. There are mechanics/equipment who
on any day may work at one of several airports (hubs) where the maintenance
facilities are present. There are airplanes each of which has to be checked or
repaired during the given time period. The airplanes are guaranteed to visit
some of the airports on given days. The problem is to schedule the presence of
the mechanics/equipment so that each plane will get maintenance once during the
time period.


Usage: maintenance <days> <planes> <mechanics> <cities> <visits> [<seed>]

parameters for generating interesting instances

days      :  30
```

```
planes    :  3*days
mechanics :   1
hubs      :   3
visits    :   5
instances : 100

if the number of visits is made smaller, it becomes less likely that
a schedule exists. With sufficiently few visits by the airplanes to
the chosen airports, almost certainly no schedules exist. Also, with
far more planes than 3*days no maintenance visits can be scheduled
for some of the planes, turning the problem not solvable.
Increasing the number of planes past 3*days and simultaneously increasing
the number of visits keeps the problem solvable and probably also
challenging, but we have not investigated the impact of different
parameter combinations on the difficulty of the problem.


./maintenance 60 180 1 3 5 100
./maintenance 80 240 1 3 5 100
./maintenance 100 300 1 3 5 100
./maintenance 120 360 1 3 5 100
./maintenance 140 420 1 3 5 100
./maintenance 160 480 1 3 5 100
./maintenance 180 540 1 3 5 100
./maintenance 200 600 1 3 5 100
```

26 Matchcellar

```
# Match Cellar (Temporal, Satisficing)

## Domain Description

Domain is inspired by [a paper by Long and
Fox](https://www.aaai.org/Papers/ICAPS/2003/ICAPS03-006.pdf). The main feature
of this domain is that a lighted match is concurrently required to fix a fuse.

## Authors

Bharat Ranjan Kavuluri

## Original File Names

| file             | original name |
|------------------|---------------|
| domain.pddl      | domain.pddl   |
| instance-1.pddl  | p15.pddl      |
| instance-2.pddl  | p16.pddl      |
| instance-3.pddl  | p17.pddl      |
| instance-4.pddl  | p18.pddl      |
| instance-5.pddl  | p19.pddl      |
| instance-6.pddl  | p20.pddl      |
| instance-7.pddl  | p21.pddl      |
| instance-8.pddl  | p22.pddl      |
```

```
| instance-9.pddl  | p23.pddl      |
| instance-10.pddl | p24.pddl      |
| instance-11.pddl | p25.pddl      |
| instance-12.pddl | p26.pddl      |
| instance-13.pddl | p27.pddl      |
| instance-14.pddl | p28.pddl      |
| instance-15.pddl | p29.pddl      |
| instance-16.pddl | p30.pddl      |
| instance-17.pddl | p31.pddl      |
| instance-18.pddl | p32.pddl      |
| instance-19.pddl | p33.pddl      |
| instance-20.pddl | p34.pddl      |
```

27 Miconic

```
** Miconic-STRIPS **

* Origin: Jana Koehler. Used in the AIPS-2000 competition.
* Adaptions: None.
* Description: Typed STRIPS domain. Like above, but with explicit
  control over the passengers that get in or out of the lift.
* Parameters:
  * -f number of floors
  * -p number of passengers
* Generation: Original generator used in the AIPS-2000 competition.
  Simply distribute origin and destination floors at random.


OPTIONS   DESCRIPTIONS

-f <num>    number of floors (minimal 2)
-p <num>    number of passengers (minimal 1)
-r <num>    random seed (optional)


IPC instances: linear scaling where passengers go from 1 to 30. Number of floors
is passengers * 2.  5 random instances
```

28 Miconic-fulladl

```
## Miconic-ADL

*   Origin: Jana Koehler. Used in the AIPS-2000 competition.
*   Adaptions: None.
*   Description: Typed ADL domain using complex preconditions and
    conditional effects. Transport a number of passengers with an elevator
    from their origin- to their destination floors. Obeye several
    restrictions: some passengers must be transported directly, the vips
    shall be served first, some must be transported non-stop, some must be
    attended by others, some groups of people must not meet each other,
    some people do not have access to certain floors. When the lift stops
    at some floor, all passengers waiting there get in, and all passengers
    wanting to go there get out, by a conditional effect.
```

```
*   Parameters:
    *   -f number of floors
    *   -p number of passengers
    *   -u percentage of passengers with direct transportation
    *   -v percentage of vips
    *   -g percentage of non-stop passengers
    *   -n percentage of passengers that must be attended
    *   -a percentage of passengers that can attend the above type
    *   -A percentage of passengers in conflict group A
    *   -B percentage of passengers in group B, which must not meet the above group
    *   -N percentage of people with no access to some floors
    *   -F percentage of of floors not to be accessed by those
    *   -r random seed
*   Generation: Original generator used in the AIPS-2000 competition. Make
    sure that the specified percentage values are all met by randomly
    assigning types to the passengers (a passenger can have several
    types). Randomly assign origin and destination floors to all
    passengers, considering several heuristics to help problem becoming
    solvable (like not placing conflicting people at the same origin floor
    etc.).
```

29 Miconic-simpleadl

```
## Miconic-SIMPLE

*   Origin: Jana Koehler. Used in the AIPS-2000 competition.
*   Adaptions: None.
*   Description: Typed ADL domain using conditional effects. Like above,
    but without any additional constraints. The conditional effects make
    sure that all waiting passengers get in or out.
*   Parameters:
    *   -f number of floors
    *   -p number of passengers
    *   -r random seed
*   Generation: Original generator used in the AIPS-2000 competition.
    Simply distribute origin and destination floors at random.
```

30 Movie

```
## Movie

* Origin: Corin Anderson. Used in the AIPS-1998 competition.
* Adaptions: None.
* Description: Untyped STRIPS domains. Buy one each out of five
  types of snacks, then rewind the movie and reset the counter.
* Parameters: -n number of snacks of each type
* Generation: No randomization.
```

31 Mprime

```
## Mprime
```

* Origin: Drew McDermott. Used in the AIPS-1998 competition.
* Adaptions: Translated all predicate names to get a more intuitive notation. Also use typing. Operator for passing on fuel from one location to another could, in the original version, be instantiated with the same location as origin and destination city, which caused the amount of fuel in that city to increase one unit. Changed that such that origin and destination cities must be different. NOTE: in the aibasel benchmarks repo, the reformulated domain no-mprime does a similar translation of predicate names but doesn't use typing. It also doesn't use the described fix (which the mprime domain of that repo does, however).
* Description: Typed STRIPS domain; the name results from Mystery' (see below). Logistics variant where trucks move on a map of locations. Additionally, trucks have only limited transportation capacity, and there are constraints on the amount of fuel. Each location has initially a certain amount of fuel available. Moving a truck away from a location decreases the amount of fuel at that location by one. If a location has more than one fuel item, then it can pass a fuel item over to a different location.
* Parameters:
  * -l number of locations
  * -f maximal amount of fuel at a location
  * -s maximal amount of transportation capacity (space)
  * -v number of trucks (vehicles)
  * -c number of cargos
* Generation: Create a simple map of -l locations such that location i is linked to location i%2B1, and location -l is linked to location 1. Randomly assign transportation capacity between 1 and -s to all vehicles, and fuel between 0 and -f to all locations. Distribute cargo origin and destination locations randomly over all locations, likewise for the starting locations of the vehicles.

32 Mystery

Mystery

* Origin: Drew McDermott. Used in the AIPS-1998 competition.
* Adaptions: Translated all predicate names to get a more intuitive notation. Also use typing. NOTE: in the aibasel benchmarks repo, the reformulated domain no-mystery does a similar translation of predicate names but doesn't use typing. This is not to be confused with the IPC'11 domain nomystery for which there is also a generator here.
* Description: Typed STRIPS domain; the name is because the original specified the semantics in a disguised manner by using unintuitive names for the predicates and constants. The domain is the same like the Mprime domain above, except that there is no way of passing on fuel between locations.
* Parameters:
  * -l number of locations
  * -f maximal amount of fuel at a location
  * -s maximal amount of transportation capacity (space)
  * -v number of trucks (vehicles)
  * -c number of cargos
* Generation: Exactly like in Mprime.

33 Nomystery

Nomystery is a transportation domain designed to study resource constrained planning [1,2]. In this domain, a truck moves in a weighted graph; a set of packages must be transported between nodes; actions move along edges, and load/unload packages; each move consumes the edge weight in fuel. In brief, Nomystery is a straightforward problem similar to the ones contained in many IPC benchmarks. Its key feature is that it comes with a domain-specific optimal solver allowing to control the constrainedness of the resources. The generator first creates a random connected undirected graph with n nodes, and it adds k packages with random origins and destinations. The edge weights are uniformly drawn between 1 and an integer W. The optimal solver computes the minimum required amount of fuel M, and the initial fuel supply is set to [C×M], where C >= 1 is a (float) input parameter of the generator. The parameter C denotes the ratio between the available fuel vs. the minimum amount required. The problem becomes more constrained when C approaches 1.

A ubiquitous feature of planning problems is the need to economize limited resources such as fuel or money. While heuristic search, mostly based on relaxation heuristics, is currently the superior method for most varieties of planning, its ability to solve critically resource-constrained problems is limited: relaxation heuristics basically ignore the resource consumption by actions. Nomystery generator is a nice test domain to study the behavior of planning algorithms in planning problems with resources.

There are two ways to scale problem difficulty in Nomystery: increasing the number of packages and locations, and decreasing C. The generator is very fast up to 18 locations and 18 packages. Two types of encoding can be used for the problems: Hard and Hard-cost. The only difference is in the action costs. While all actions have a unit cost in Hard encoding, in Hard-cost version action costs are equal to the amount of fuel consumed by the action. Therefore, the total cost of a plan for Hard-cost encoding equals the amount of fuel consumed in the plan. In Hard encoding, problems with 12 locations and 12 packages become quite challenging for state of the art planners when C is close 1 [1]. Hard-cost encoding makes the problems easier for the current planners. The reason is that the heuristic functions that consider costs are not any more completely ignorant to the resource consumption of actions. However, this type of encoding is not always feasible for resource planning; there might be several resources and the cost of the plan might be different from the amount of the resource consumption. However, our initial experiments show problems in this encoding with 12 locations and 15 packages become challenging for current planners when C is close to 1.

From the two versions of the domain we have chosen the non-cost one (cost of the plan and resource consumption are not related). As the domain creator says, this is the most difficult one of the two formulations. In addition, it is the less similar to the Transport domain. According to the domain creator planners find it difficult to solve problems with more than 12 locations and 12 packages when the constraint on the resource tends to 1. For the satisficing track, problems 1-10 start with 6 locations and 6 packages with c=1.5 and each problem increases 1 package and 1 location to the previous one (so problem 10 has 15 packages and 15 locations). Problems 11-20 also start in 6-6 and increase parameters by 1, but in this case c=1.1.

For the optimal track, we developed a first version where problems 1-10 start with 2 locations and 1 package with c=1.5 and each problem increases 1 location and 1 package to the previous one (so problem 10 has 10 packages and 11 locations). Problems 11-20 also start in 2-1 and increase parameters by 1, but in this case c=1.1. This version resulted to be quite simple, so we generated a tougher one. We started with 4 locations and 3

packages (last problem had 13 locations and 12 packages) with c=1.5 and c=1.1. That version was also quite simple, so a third one has been generated. From experiments it seems the lower the constraint the easier the problems are for the optimal planners: the search space is pruned quickly as the cost limits the number of applicable actions. Taking that into account we have generated problems with c=1.1, 1.5 and 2.0. Problems 1-6 have c=1.1 and start in 9 locations and 8 packages till 14-13. From 7-13 c=1.5, starting in 8 locations and 7 packages. Problems 14-20 are like 7-13 but with c=2.0.

34 Npuzzle

N-Puzzle Domain:

This is the classic NxN sliding puzzle domain. It has received a large
amount of attention from search researchers and in particularly has
been studied in the context of macro learning.

35 Nurikabe

Instance Generator for Nurikabe

Alvaro Torralba <torralba@cs.uni-saarland.de>
Florian Pommerening <florian.pommerening@unibas.ch>

This is a version of the Floortile domain where a robot must paint a certain pattern in a grid and cannot move into locations that have already been painted. The pattern is not decided in advance, but instead it must fulfill the constraints of a simplified Nurikabe puzzle. The grid has some positions marked with a number. When the robot is on top of a cell with a number N on it, it can start the paint action so that the next N cells (including the current position) will be painted. There is an additional constraint that two adjacent cells cannot belong to cells of different groups.

36 Openstacks

generate_problems.cpp:

Ioannis's program for generating the text files that define the
Openstacks instances. The ICAPS benchmark set was created with
something like the following calls:

```
./generate_problems  5  34  1 1 1 XXX
./generate_problems  5  20  5 1 3 XXX
./generate_problems 30  60 10 1 3 XXX
./generate_problems 80 100 20 1 3 XXX
```

where XXX is the density parameter which has unfortunately been lost
(20? 25? 30? 40?). For generating additional benchmark instances, the
Python script (below) should be preferred.

Notes on attribute selection:
  - A very high-density seems to simplify the problem with blind search.

- For OPT in IPC'08 they scaled both products and orders at the same time,
  with a linear scaling of +1
- For SAT, in IPC'14 they started at 170 for both. The scaling factor was +20
  in both cases.

generate_problems.py:

A Python version of Ioannis's program which fixes a few issues with
the C++ program:

 * When an order is empty, some product is chosen uniformly randomly
   to make it nonempty. Due to an off-by-one error, the last product
   could never be chosen for this.

 * Similarly, unused products would randomly be added to one order.
   Due to an off-by-one error, the last order could never be selected
   for this.

 * The original code shuffled the orders after generating the problem
   matrix. However, this shuffling was not completely uniform. To see
   this, consider e.g. a 5x5 problem, where the shuffling algorithm
   would uniformly compute ten random numbers in the range {0, ...,
   4}, which gives 5^10 atomic events with uniform probability. From
   this random information, it's not possible to select between 5! =
   120 permutations uniformly since 120 does not divide 5^10.

 * More seriously, the original code did not shuffle the products.
   Adjacent products are correlated in that they tend to be useful for
   the same orders, so this may introduce some bias for planners that
   for whatever (tie-breaking) reason process the products in some
   kind of sequential order.

generate_pddl.cpp:

Ioannis's program for generating PDDL code from the text files
generated by generate_problems. Need to change various global
variables (opt, sat, all, ADL, negated, semiground, temporal, numeric,
net_benefit) to affect which problem set is generated.

generator.py:

Combination of generate_problems.py and generate_pddl.cpp in Python. This
generator version can only generate the sequential STRIPS version with
negative preconditions. In contrast to the original generator files, this
script only generates a single task per call.

From IPC'11:

Openstacks

From the original domain description: This domain was first used at IPC-2006. The openstacks domain is based on the "minimum maximum simultaneous open stacks" combinatorial optimization problem, which can be stated as follows: A manufacturer has a number of orders, each for a combination of different products, and can only make one product at a time.

The total required quantity of each product is made at the same time (because changing from making one product to making another requires a production stop). From the time that the first product included in an order is made to the time that all products included in the order have been made, the order is said to be "open" and during this time it requires a "stack" (a temporary storage space). The problem is to order the making of the different products so that the maximum number of stacks that are in use simultaneously, or equivalently the number of orders that are in simultaneous production, is minimized (because each stack takes up space in the production area).

The problem is NP-hard and known to be equivalent to several other problems. It was recently posed as a challenge problem for the constraint programming community (see http://www.dcs.st-and.ac.uk/~ipg/challenge/).

Note that this is an optimization problem: for any instance of the problem every ordering of the making of products is a solution, which at worst uses as many simultaneously open stacks as there are orders. Thus, finding a plan is quite trivial (in the sense that there exists a domain-specific linear-time algorithm that solves the problem), but finding a plan of high quality is hard (even for a domain-specific algorithm).

There are two versions of this domain, STRIPS and ADL. As there are few ADL compliant planners and in the last IPC there were not too much differences in results between both versions, we have not generated ADL problems for this IPC. In the STRIPS version the goal is to minimize the total number of stacks. The number of stacks is represented using symbols and universally quantified preconditions have been replaced by multiple semi-ground actions. Note that this requires a different domain file for each problem.

For the satisficing track, in the 2008 competition, objects ranged from 5 to 100. For the first 12 problems, three versions with the same difficulty were created and objects were increased by 5 from one triplet to the next (i.e. problems 1, 2 and 3 were similar, with 5 objects; problems 4, 5 and 6 had 10 objects...). From 13 to 24 objects were increased by 10, while for the last 6, objects were increased by 20. The density parameter has been lost. For IPC 2011, as in the other reused domains, we reused the 10 first problems. From 11 to 20, objects are increased starting from 130 by 30, and 2 problems of the same size have been created, one with density = 20 and the other with density = 80.

The selected problems are (numbers in parenthesis are number of objects of each new problem):

problem old best problem old best
p01 p21 6 p11 (130)
p02 p22 16 p12 (130)
p03 p24 19 p13 (160)
p04 p25 19 p14 (160)

p05 p26 15 p15 (190)
p06 p27 18 p16 (190)
p07 p20 13 p17 (220)
p08 p28 33 p18 (220)
p09 p29 31 p19 (250)
p10 p30 35 p20 (250)

For the optimal track, in the IPC 2008 a total of 9 problems were not solved by any planner (problem 20 and from 23 to 30). As in the satisficing version, there is no record of the density used to generate the problems. The first problem in IPC 2008 started with 5 objects and each problem had 1 object more than the previous one. So problem 30 had 34 objects.

Five first problems were quite simple so they have been removed from the selection. This results in 16 problems previously solved and 4 non solved for this edition. The selected problems are:

problem old optimal problem old optimal
p01 p6 2 p11 p15 4
p02 p7 5 p12 p16 4
p03 p8 5 p13 p17 4
p04 p9 3 p14 p18 3
p05 p10 3 p15 p19 4
p06 p11 4 p16 p21 3
p07 p12 3 p17 p20 -
p08 p13 4 p18 p23 -
p09 p14 4 p19 p24 -
p10 p22 4 p20 p25 -

37 Parking

This domain involves parking cars on a street with N curb locations where cars can be double parked but not triple parked. The goal is to move from one configuration of parked cars to another configuration by driving cars from one curb location to another. The problems in the competition contain 2*(N-1) cars, which allows one free curb space and guarantees solvability. The problem difficulty is scaled by increasing the number of curb locations. This domain is isomorphic to a bounded-table blocks world domain where there are N table locations and 2*(N-1) blocks and towers are only allowed to contain 2 blocks.

OPT 11: from 7 to 12 curbs, several instances of the same size
OPT 14: from 7 to 11 curbs, several instances of the same size

38 Pathways

Pathways-Propositional Problem file generator
Authors: Yannis Dimopoulos, Alfonso Gerevini and Alessandro Saetti
==========

Usage: pathways [options]

Command line options:
 -out <file>   Set the problem file name
 -n <string>   Set the problem name (default PathwaysProb)

```
 -R <num>       Set the min number of reactions used in the reach. analysis
 -G <num>       Set the number of goals
 -L <num>       Set the desired number of initial substances (in metric field)
--seed <num>    Set the seed for the random number generator

The reactions in the generated problems are based on files
"Pathways-SimpleSubs" and "Pathways-Reactions". The generator selects
reactions and defines goals according to an algorithm that performs a
sort of reachability analysis. The command lines used to generate the
IPC5 problems of Pathways-Propositional are:

pathways --seed 2004 -out pfile01 -R 12 -G 01 -L 3 -n Pathways-01 > domain-dummyAct.pddl
pathways --seed 1146 -out pfile02 -R 24 -G 02 -L 3 -n Pathways-02 > domain-dummyAct.pddl
pathways --seed 126053358 -out pfile03 -R 36 -G 03 -L 3 -n Pathways-03 > domain-dummyAct.pddl
pathways --seed 6800456 -out pfile04 -R 48 -G 04 -L 3 -n Pathways-04 > domain-dummyAct.pddl
pathways --seed 387462 -out pfile05 -R 72 -G 06 -L 7 -n Pathways-05 > domain-dummyAct.pddl
pathways --seed 231459 -out pfile06 -R 96 -G 08 -L 10 -n Pathways-06 > domain-dummyAct.pddl
pathways --seed 2211674 -out pfile07 -R 132 -G 11 -L 10 -n Pathways-07 > domain-dummyAct.pddl
pathways --seed 32908700 -out pfile08 -R 144 -G 12 -L 17 -n Pathways-08 > domain-dummyAct.pddl
pathways --seed 21545433 -out pfile09 -R 156 -G 13 -L 12 -n Pathways-09 > domain-dummyAct.pddl
pathways --seed 3286431 -out pfile10 -R 168 -G 14 -L 14 -n Pathways-10 > domain-dummyAct.pddl
pathways --seed 1276765 -out pfile11 -R 180 -G 15 -L 10 -n Pathways-11 > domain-dummyAct.pddl
pathways --seed 1062006 -out pfile12 -R 192 -G 16 -L 17 -n Pathways-12 > domain-dummyAct.pddl
pathways --seed 428365421 -out pfile13 -R 204 -G 17 -L 19 -n Pathways-13 > domain-dummyAct.pddl
pathways --seed 871235621 -out pfile14 -R 216 -G 18 -L 18 -n Pathways-14 > domain-dummyAct.pddl
pathways --seed 281837981 -out pfile15 -R 228 -G 19 -L 18 -n Pathways-15 > domain-dummyAct.pddl
pathways --seed 78854 -out pfile16 -R 240 -G 20 -L 22 -n Pathways-16 > domain-dummyAct.pddl
pathways --seed 1216 -out pfile17 -R 252 -G 21 -L 20 -n Pathways-17 > domain-dummyAct.pddl
pathways --seed 763451 -out pfile18 -R 288 -G 24 -L 20 -n Pathways-18 > domain-dummyAct.pddl
pathways --seed 432102 -out pfile19 -R 312 -G 26 -L 24 -n Pathways-19 > domain-dummyAct.pddl
pathways --seed 40275209 -out pfile20 -R 324 -G 27 -L 21 -n Pathways-20 > domain-dummyAct.pddl
pathways --seed 8465321 -out pfile21 -R 336 -G 28 -L 22 -n Pathways-21 > domain-dummyAct.pddl
pathways --seed 9347325 -out pfile22 -R 348 -G 29 -L 21 -n Pathways-22 > domain-dummyAct.pddl
pathways --seed 723452 -out pfile23 -R 360 -G 30 -L 25 -n Pathways-23 > domain-dummyAct.pddl
pathways --seed 98766 -out pfile24 -R 372 -G 31 -L 28 -n Pathways-24 > domain-dummyAct.pddl
pathways --seed 276314 -out pfile25 -R 396 -G 33 -L 34 -n Pathways-25 > domain-dummyAct.pddl
pathways --seed 82254 -out pfile26 -R 408 -G 34 -L 29 -n Pathways-26 > domain-dummyAct.pddl
pathways --seed 447198776 -out pfile27 -R 420 -G 35 -L 30 -n Pathways-27 > domain-dummyAct.pddl
pathways --seed 8632 -out pfile28 -R 444 -G 37 -L 35 -n Pathways-28 > domain-dummyAct.pddl
pathways --seed 822562 -out pfile29 -R 468 -G 39 -L 27 -n Pathways-29 > domain-dummyAct.pddl
pathways --seed 37463 -out pfile30 -R 480 -G 40 -L 27 -n Pathways-30 > domain-dummyAct.pddl


The atoms in ":goal" field are dummy. Each of these command line also
outputs a set of dummy operators, whose (disjunctive) preconditions
state the real goals of the planning problem.


**NOTE**: The generator may produce unsolvable tasks.
```

39 Pegsol

```
Peg Solitaire
```

40 Rovers

```
2021 June fix:
   visible_from output was not using the correct data, so all objectives were
   visible from waypoint0 to waypoingN. Apart from heavily impacting the
   diversity of the generated instances, this could cause some instances to be
   unsolvable (e.g. when waypoint0 and 1 are not reachable by the rover).


------------------------------


Inspired by planetary rovers problems, this domain requires that a collection of rovers
navigate a planet surface, finding samples and communicating them back to a lander.

Strips: The Strips version is an interesting challenge in its own right. In addition, a
minor subtlety in the encoding is used to prevent parallel communication between rovers
and the lander: communication actions precondition on the channel to the lander being
free, and then both add and delete this fact. Because deletes occur before adds this has
the over all effect of leaving the channel free, but it makes the fact a "moving target"
which prevents a concurrent action from using the fact. Some planners find this mechanism
difficult to handle.

Useage: rovergen [-s|-t|-n|-u|-f <filename>]
                 <seed> <#rovers> <#waypoints> <#objectives> <#cameras> <#n-goals>

IPC instances p01: 1 4 2 1 3
              p10: 4 7 4 6 11
              p20: 8 25 8 7 20
              p30: 10 50 8 14 25
              p40  14 100 11 15 69
```

41 Satellite

```
Fix June 2021:

Added code to make sure that:
  - all modes are supported by at least one instrument
  - all observations are interesting

This ensures that some observations can be made, so that all instances have at least a goal.


--------------------------------


Usage: satgen [-T <d>|-u|-s|-t|-n|-c] <seed> <#s> <#i> <#m> <#t> <#o>

        where: #s = number of satellites, #i = max instruments/satellite,
               #m = number of modes, #t = number of targets
               #o = number of observations
```

IPC instances:

1 1 3 7 3
10 3 5 205 178

Clearly, what we scale is the number of modes, targets, satellites, and observations.
Number of instruments per satellite we may fix to 3.
Targets should be at least 5

	sat	mod	tar	obs
p01-pfile1.pddl	1	3	7	3
p02-pfile2.pddl	1	3	8	5
p03-pfile3.pddl	2	3	8	4
p04-pfile4.pddl	2	3	10	7
p05-pfile5.pddl	3	3	10	6
p06-pfile6.pddl	3	4	11	7
p07-pfile7.pddl	4	4	12	7
p08-pfile8.pddl	4	4	15	10
p09-pfile9.pddl	5	5	15	10
p10-pfile10.pddl	5	5	17	11
p11-pfile11.pddl	5	5	20	11
p12-pfile12.pddl	5	5	25	19
p13-pfile13.pddl	5	5	30	24
p14-pfile14.pddl	6	5	25	16
p15-pfile15.pddl	8	5	25	18
p16-pfile16.pddl	10	5	25	19
p17-pfile17.pddl	12	5	25	17
p18-pfile18.pddl	5	5	25	13
p19-pfile19.pddl	5	8	25	27
p20-pfile20.pddl	5	10	25	40
p21-HC-pfile1.pddl	5	3	43	34
p22-HC-pfile2.pddl	5	3	53	43
p23-HC-pfile3.pddl	7	3	53	47
p24-HC-pfile4.pddl	7	3	73	66
p25-HC-pfile5.pddl	8	3	73	60
p26-HC-pfile6.pddl	8	4	74	64
p27-HC-pfile7.pddl	10	4	75	64
p28-HC-pfile8.pddl	10	4	105	93
p29-HC-pfile9.pddl	15	5	105	87
p30-HC-pfile10.pddl	15	5	125	108
p31-HC-pfile11.pddl	15	5	155	130
p32-HC-pfile12.pddl	15	5	205	178
p33-HC-pfile13.pddl	15	5	255	226
p34-HC-pfile14.pddl	5	5	205	140
p35-HC-pfile15.pddl	8	5	205	184
p36-HC-pfile16.pddl	10	5	205	175

42 Scanalyzer

IPC instances:

prob_no = 1

```
for half_segment_ids in [ [""], ["a", "b"] ]:
    for size in xrange(1,11):
        for problem_type in [ (1,1), (size,1), (1,size), (size,size), "simple" ]:
            filename = "p%03d.pddl" % prob_no
            create_pddl(half_segment_ids, problem_type, size, prob_no, filename)
            prob_no += 1

Selection in IPC:

(half_segment_ids, problem_type, size)
11: [''], (1, 1), 3
12: [''], (3, 1), 3
14: [''], (3, 3), 3
16: [''], (1, 1), 4
17: [''], (4, 1), 4
19: [''], (4, 4), 4
21: [''], (1, 1), 5
22: [''], (5, 1), 5
24: [''], (5, 5), 5
26: [''], (1, 1), 6
27: [''], (6, 1), 6
29: [''], (6, 6), 6
31: [''], (1, 1), 7
32: [''], (7, 1), 7
34: [''], (7, 7), 7
36: [''], (1, 1), 8
37: [''], (8, 1), 8
39: [''], (8, 8), 8
41: [''], (1, 1), 9
42: [''], (9, 1), 9
44: [''], (9, 9), 9
51: ['a', 'b'], (1, 1), 1
52: ['a', 'b'], (1, 1), 1
54: ['a', 'b'], (1, 1), 1
56: ['a', 'b'], (1, 1), 2
57: ['a', 'b'], (2, 1), 2
59: ['a', 'b'], (2, 2), 2
61: ['a', 'b'], (1, 1), 3
62: ['a', 'b'], (3, 1), 3
64: ['a', 'b'], (3, 3), 3
```

43 Schedule

```
## Schedule

*   Origin: One variation appears in the Prodigy collection by Manuela
    Veloso. Prepared for the AIPS-2000 competition by Fahiem Bacchus.
*   Adaptions: None.
*   Description: Typed ADL domain using conditional effects. Encodes a
    simple Scheduling kind of problem where a number of objects need to be
    processed using a collection of machines. Possible actions are
    polishing, punching holes, painting etc. All actions need uniform
    time, which is modelled by a do-time-step operator. If that operator
```

```
    is applied, then all busy machines are no longer busy, and all
    scheduled objects are no longer scheduled - this is also an example of
    the kind of conditional effects that are used in the representation.
*   Parameters: (without significant changes to the domain, parameters -s
    to -o can not be arbitrarily increased)
    *   -p number of objects (parts)
    *   -s number of shapes (maximal 3: cylindrical, circular, oblong)
    *   -c number of colors (maximal 4)
    *   -w number of different widths of holes (maximal 3)
    *   -o number of orientations of holes (maximal 2)
    *   -Q probability that a part needs to be made cylindrical
    *   -W probability that a part is initially coloured
    *   -E probability that a part needs to be coloured
    *   -R probability that a part has a hole initially
    *   -T probability that a part needs to have a hole
    *   -Y probability that a part needs to have a specific surface condition
*   Generation: All parts are given a random initial shape and surface
    condition. With the respective probability, they are given random
    colours or holes. In the goal state, they need, with the respective
    probabilities, to be cylindrical (which is the only shape that can be
    produced by the machines), to be randomly coloured, to have a random
    hole, and to have a random goal surface condition.
```

44 Snake

```
2021 June bugfix:

- The generator assumed that at least 1 apple will be spawn. Now it is fixed and instances
can be generated where no apples are spawn (you only need to pick the ones that were
already there).

- Reduced the maximum number of apples by 1 when the grid has an odd number of cells, as
  they may be hamiltonian paths. This ensures that all instances using an empty grid are
  solvable (if other types of grids are used solvability needs to be ensured by other
  means).


------------------------

Instance Generator for Snake

Alvaro Torralba <torralba@cs.uni-saarland.de>
Florian Pommerening <florian.pommerening@unibas.ch>


In the game Snake, a snake moves around a grid. There is a number of apples in some grid
cells and when the snake moves into one of these locations, it eats the apple, extends its
length by one, and a new apple may spawn. In this version uncertainty has been removed by
deciding the location where apples will spawn in advance. Except for the end of the game,
there will be more than one apple on the board at the same time.

Interesting properties: dead-end states
```

45 Sokoban

This generator has not been used for IPC 2011.
All the problems have been taken from IPC 2008.

The IPC 2011 organizers

* pddl-generators note

‘build-program.py‘ converts microban / multiban / hexoban data into PDDL.

‘choose-suite.py‘ analyzes instances in ‘rolling_stone_data.py‘ and
select problem instances for satisficing and optimal track.

From the message above, it seems these problem generators (which are based on
existing microban benchmarks etc.) are not used for IPC2011.

Scripts in ‘random/‘ are randomized generators that are used for generating
instances for IPC2008, which was also used for IPC2011.

usage:

OPTIONS DESCRIPTIONS

-n <num> grid size (minimal 5)
-b <num> number of boxes (minimal 1)
-w <num> number of walls (minimal 0)
-s <num> random seed

46 Spanner

A worker is in a shed, containing a number of spanners, and at a gate some
distance away there are a number of nuts that must be tightened. Crucially, once
the worker has left the shed they cannot return. Also, the spanners are fragile,
so once used to tighten a nut they break, and then cannot be used to tighten
another. The goal is planning to pick-up the spanners and tighten the nuts. This
is an interesting domain for learning, as it has a directed search space, and is
challenging for planners employing an ’ignore delete lists’ relaxation.

Origin: Created for IPC 2011 by Amanda Coles, Andrew Coles, Maria Fox and Derek Long.

47 Spider

In the solitaire game "Spider solitaire" stacks of cards of the same suit can be
moved from a pile to another and cards can be placed on a card with a value that
is higher by one (of any suit). In contrast to the real game, all cards are
face-up from the start, to remove uncertainty. Movement of stacks of cards could
be easily modeled with derived predicates but here we use auxiliary 0-cost

```
actions that update the state.

Origin: IPC 2018
```

48 Storage

```
Problem generator for Storage-Propositional
Authors: Alfonso Gerevini and Alessandro Saetti
================================================

storage [options] file

Command line options:
 -h      Display this help
 -p      Set the number of problem (header of the problem file)
 -n      Set the number of hoists (default 3)
 -d      Set the number of depots (default 1)
 -o      Set the number of containers (default 1)
 -s      Set the number of store-areas (default 9)
 -c      Set the number of crates (default 5)
 -e      Set seed for random number generator (must be different from 0)


Constraints:
   Number of number of crates <= store-areas
   Number of number of depots <= store-areas
   Number of number of hoists <= store-areas

   3 < Number of crates / Number of containers <= 4


The command lines used to generate the IPC5 problems of Storage-Propositional are:

storage -p 01 -o 1 -e 2005 -c 1 -n 1 -s 1 -d 1 pfile01
storage -p 02 -o 1 -e 2005 -c 1 -n 2 -s 2 -d 1 pfile02
storage -p 03 -o 1 -e 2005 -c 1 -n 3 -s 3 -d 1 pfile03
storage -p 04 -o 1 -e 2005 -c 2 -n 1 -s 4 -d 1 pfile04
storage -p 05 -o 1 -e 2005 -c 2 -n 2 -s 4 -d 1 pfile05
storage -p 06 -o 1 -e 2005 -c 2 -n 3 -s 4 -d 1 pfile06
storage -p 07 -o 1 -e 2005 -c 3 -n 1 -s 6 -d 1 pfile07
storage -p 08 -o 1 -e 2005 -c 3 -n 2 -s 6 -d 1 pfile08
storage -p 09 -o 1 -e 2005 -c 3 -n 3 -s 6 -d 1 pfile09
storage -p 10 -o 1 -e 2005 -c 4 -n 1 -s 8 -d 1 pfile10
storage -p 11 -o 1 -e 2005 -c 4 -n 2 -s 8 -d 1 pfile11
storage -p 12 -o 1 -e 2005 -c 4 -n 3 -s 8 -d 1 pfile12
storage -p 13 -o 2 -e 2005 -c 5 -n 1 -s 10 -d 2 pfile13
storage -p 14 -o 2 -e 2005 -c 5 -n 2 -s 10 -d 2 pfile14
storage -p 15 -o 2 -e 2005 -c 5 -n 3 -s 10 -d 2 pfile15
storage -p 16 -o 2 -e 2005 -c 06 -n 3 -s 12 -d 2 pfile16
storage -p 17 -o 2 -e 2005 -c 07 -n 3 -s 14 -d 2 pfile17
storage -p 18 -o 2 -e 2005 -c 08 -n 3 -s 16 -d 2 pfile18
storage -p 19 -o 3 -e 2005 -c 09 -n 3 -s 18 -d
```

```
containers 1 crates 1 hoists 1 store_areas 1 depots 1
containers 1 crates 1 hoists 2 store_areas 2 depots 1
containers 1 crates 1 hoists 3 store_areas 3 depots 1
containers 1 crates 2 hoists 1 store_areas 4 depots 1
containers 1 crates 2 hoists 2 store_areas 4 depots 1
containers 1 crates 2 hoists 3 store_areas 4 depots 1
containers 1 crates 3 hoists 1 store_areas 6 depots 1
containers 1 crates 3 hoists 2 store_areas 6 depots 1
containers 1 crates 3 hoists 3 store_areas 6 depots 1
containers 1 crates 4 hoists 1 store_areas 8 depots 1
containers 1 crates 4 hoists 2 store_areas 8 depots 1
containers 1 crates 4 hoists 3 store_areas 8 depots 1
containers 2 crates 5 hoists 1 store_areas 10 depots 2
containers 2 crates 5 hoists 2 store_areas 10 depots 2
containers 2 crates 5 hoists 3 store_areas 10 depots 2
containers 2 crates 6 hoists 3 store_areas 12 depots 2
containers 2 crates 7 hoists 3 store_areas 14 depots 2
containers 2 crates 8 hoists 3 store_areas 16 depots 2
containers 3 crates 9 hoists 3 store_areas 18 depots
```

49 Termes

```
Instance Generator and solver for TERMES domain:
  Alvaro Torralba <torralba@cs.uni-saarland.de>
  Florian Pommerening <florian.pommerening@unibas.ch>



This domain models the Harvard TERMES robots, based on termites. They cannot only carry
blocks but also climb on them so that they can build complex structures. This domain, in
its full generality, is a multi-agent planning benchmark, but here we model only a single
robot.

Domain submitted by Sven Koenig and Satish Kumar:

S. Koenig and S. Kumar. A Case for Collaborative Construction as Testbed for Cooperative
Multi-Agent Planning. In Proceedings of the ICAPS-17 Scheduling and Planning Applications
Workshop (SPARK), 2017.
```

50 Tetris

```
## TETRIS DOMAIN GENERATOR.
## IPC 2014
## author: Mauro Vallati -- University of Huddersfield

Usage:  generator.py <grid_size> <conf_blocks>
 line numbers is used for defining number of lines of the screen.
 The number of column is fixed at 4. E.g., 8-> 8x4 grid. Only odd numbers accepted.
 conf_blocks:
 1 -> only 1x1 square blocks
 2 -> only 2x1 blocks
```

3 -> only L-shaped blocks
4 -> mix of blocks

** Take care, generator can return unsolvable instances!! **

51 Tidybot

Tidybot
Author: Bhaskara Marthi

From the original description: The Tidybot domain models a household cleaning task, in which one or more robots must pick up a set of objects and put them into goal locations. The world is structured as a 2d grid, divided into navigable locations and surfaces on which objects may lie. Robots have a gripper, which moves relative to the robot, up to some maximum radius. Existing objects block the gripper, so that it may be necessary to move one object out of the way to put another one down. Robots can carry one object at a time in the gripper, but may also make use of a cart, that can hold multiple objects. The instance generator creates worlds that contain rectangular surfaces ("tables"), as well as U-shaped enclosures ("cupboards"), which are the goal locations of objects.

In many real-world problems, the difficulty is due to the large state space and number of objects, rather than due to complex, "puzzle-like" combinatorial constraints. Humans are able to quickly find feasible solutions in such domains, because they seem to be able to decompose the problem into separate parts and make use of the geometrical structure. This domain is thus intended to exercise the ability of planners to find and exploit structure in large but mostly unconstrained problems.

Optimal reasoning in such problems is challenging for humans as well, and a secondary motivation for the domain is to test the ability to do optimal reasoning in geometrically structured worlds. The presence of the carts adds another combinatorial decision: it might be worthwhile to spend some time fetching the cart to avoid later having to go back and forth with each object.

The instance generator is provided as a .jar file, together with the source code (in the Clojure language). To generate an instance: java -cp tidybot.jar tidybot world-size num-tables num-cupboards table-min-size table-max-size cupboard-size

The number of objects will be num-cupboards * objects-per-cupboard, where objects-per-cupboard = (cupboard-size - 2)^2. Thus, num-cupboards should be >= 1, otherwise there will be no objects. It is recommended to set cupboard-size = 4. Scaling world-size will then increase the number of literals/states, while scaling num-cupboards will increase the number of objects and goals. Scaling num-tables will tend to make the domain more constrained (since there are more obstacles), and also to increase the branching factor, since there are more places to set objects down.

Problems with the same size can be quite different in difficulty, so we

will generate various problems of the same size. The automatic generator returns error quite often (9 out of 10 times at least) when the size of the world is lower than 9. But problems with size 9 are quite challenging even for the satisficing planners. In addition some of the generated problems are unsolvable or are equal to other generated problems. This makes it quite hard to generate problems for this domain.

For the satisficing track, problems with world size=8 are solved quite easily, but problems with size=9 are challenging. The following table shows the parameters chosen for the problems at IPC 2011.

problem	size	tables	cupboards	goal positions
p01	9x9	5	1	6
p02	9x9	6	1	6
p03	9x9	3	1	7
p04	9x9	3	1	6
p05	9x9	3	1	5
p06	9x9	6	1	6
p07	9x9	3	1	8
p08	9x9	3	1	6
p09	10x10	9	1	7
p10	10x10	9	1	7
p11	10x10	3	1	7
p12	10x10	2	1	7
p13	10x10	6	1	5
p14	10x10	8	1	7
p15	10x10	6	1	7
p16	10x10	3	1	5
p17	11x11	9	1	8
p18	11x11	7	1	6
p19	12x12	5	3	15
p20	12x12	7	2	11

problem	size	tables	cupboards	goal positions
p01	5	0	1	4
p02	6	0	1	4
p03	6	0	1	4
p04	6	0	1	4
p05	7	0	1	4
p06	7	0	1	4
p07	7	0	1	4
p08	7	0	1	4
p09	8	0	1	4
p10	8	0	1	4
p11	8	0	1	4
p12	8	0	1	4
p13	8	0	1	4
p14	9	3	1	6
p15	9	2	1	7
p16	9	5	1	6
p17	9	4	1	6
p18	9	3	1	7

```
p19    & 9    & 4    & 1    & 5
p20    & 9    & 5    & 1    & 5
```

52 Tpp

```
TPP-Propositional Problem file generator
Authors: Alfonso Gerevini and Alessandro Saetti
================================================

Usage: gen-TPP [options] file

Command line options:
-h              Display this help
-s <num>        Set the seed for the random number generator
-p <num>        Set the number of product (default 2)
-m <num>        Set the number of market (default 4)
-t <num>        Set the number of truck (default 2)
-d <num>        Set the number of depot (default 2)
-l <num>        Set the maximum level of goods (default 10)

The command lines used to generate the IPC5 problems of
TPP-Propositional are:

gen-TPP -s 2006 -m 1 -p 1 -t 1 -d 1 -l 1 pfile01
gen-TPP -s 2006 -m 1 -p 2 -t 1 -d 1 -l 1 pfile02
gen-TPP -s 2006 -m 1 -p 3 -t 1 -d 1 -l 1 pfile03
gen-TPP -s 2006 -m 1 -p 4 -t 1 -d 1 -l 1 pfile04
gen-TPP -s 2006 -m 2 -p 5 -t 2 -d 1 -l 1 pfile05
gen-TPP -s 2006 -m 2 -p 6 -t 2 -d 1 -l 2 pfile06
gen-TPP -s 2006 -m 2 -p 7 -t 2 -d 1 -l 2 pfile07
gen-TPP -s 2006 -m 2 -p 8 -t 2 -d 1 -l 2 pfile08
gen-TPP -s 2006 -m 3 -p 9 -t 3 -d 1 -l 2 pfile09
gen-TPP -s 2006 -m 3 -p 10 -t 3 -d 1 -l 2 pfile10
gen-TPP -s 2006 -m 3 -p 6 -t 3 -d 2 -l 3 pfile11
gen-TPP -s 2006 -m 3 -p 7 -t 3 -d 2 -l 3 pfile12
gen-TPP -s 2006 -m 4 -p 8 -t 4 -d 2 -l 3 pfile13
gen-TPP -s 2006 -m 4 -p 9 -t 4 -d 2 -l 3 pfile14
gen-TPP -s 2006 -m 4 -p 10 -t 4 -d 2 -l 3 pfile15
gen-TPP -s 2006 -m 4 -p 11 -t 4 -d 2 -l 4 pfile16
gen-TPP -s 2006 -m 5 -p 12 -t 5 -d 2 -l 4 pfile17
gen-TPP -s 2006 -m 5 -p 13 -t 5 -d 2 -l 4 pfile18
gen-TPP -s 2006 -m 5 -p 14 -t 5 -d 2 -l 4 pfile19
gen-TPP -s 2006 -m 5 -p 15 -t 5 -d 2 -l 4 pfile20
gen-TPP -s 2006 -m 6 -p 11 -t 6 -d 3 -l 5 pfile21
gen-TPP -s 2006 -m 6 -p 12 -t 6 -d 3 -l 5 pfile22
gen-TPP -s 2006 -m 6 -p 13 -t 6 -d 3 -l 5 pfile23
gen-TPP -s 2006 -m 6 -p 14 -t 6 -d 3 -l 5 pfile24
gen-TPP -s 2006 -m 7 -p 15 -t 7 -d 3 -l 5 pfile25
gen-TPP -s 2006 -m 7 -p 16 -t 7 -d 3 -l 6 pfile26
gen-TPP -s 2006 -m 7 -p 17 -t 7 -d 3 -l 6 pfile27
gen-TPP -s 2006 -m 7 -p 18 -t 7 -d 3 -l 6 pfile28
gen-TPP -s 2006 -m 8 -p 19 -t 8 -d 3 -l 6 pfile29
gen-TPP -s 2006 -m 8 -p 20 -t 8 -d 3 -l 6 pfile30
```

```
gen-TPP -s 2006 -m 8 -p 16 -t 8 -d 4 -l 10 pfile31
gen-TPP -s 2006 -m 8 -p 17 -t 8 -d 4 -l 10 pfile32
gen-TPP -s 2006 -m 9 -p 18 -t 9 -d 4 -l 10 pfile33
gen-TPP -s 2006 -m 9 -p 19 -t 9 -d 4 -l 10 pfile34
gen-TPP -s 2006 -m 9 -p 20 -t 9 -d 4 -l 10 pfile35
gen-TPP -s 2006 -m 9 -p 21 -t 9 -d 4 -l 15 pfile36
gen-TPP -s 2006 -m 10 -p 22 -t 10 -d 4 -l 15 pfile37
gen-TPP -s 2006 -m 10 -p 23 -t 10 -d 4 -l 15 pfile38
gen-TPP -s 2006 -m 10 -p 24 -t 10 -d 4 -l 15 pfile39
gen-TPP -s 2006 -m 10 -p 25 -t 10 -d 4 -l 15 pfile40
```

53 Transport

```
./city-generator.py          <nodes> <size^(1/2)> <degree> <mindistance> <trucks> <packages> <seed>
./two-cities-generator.py    <nodes> <size^(1/2)> <degree> <mindistance> <trucks> <packages> <seed>
./three-cities-generator.py <nodes> <size^(1/2)> <degree> <mindistance> <trucks> <packages> <seed>
```

54 Trucks

```
Problem generator for Trucks-Propositional
Authors: Yannis Dimopoulos, Alfonso Gerevini and Alessandro Saetti
=====================================================================

trucks [options] file

Command line options:
 -t <num>       Set the number of trucks
 -l <num>       Set the number of locations
 -p <num>       Set the number of packages
 -a <num>       Set the number of truck areas
 -n <num>       Set the number of problem
 -seed <num>    Set the seed for random number generator


The command lines used to generate the IPC5 problems of
Trucks-Propositional are:

gen-Trucks -seed 200416 -t 1 -l 3 -p 3 -a 2 -n 1 > pfile01
gen-Trucks -seed 9764618515 -t 1 -l 3 -p 4 -a 2 -n 2 > pfile02
gen-Trucks -seed 12605335814 -t 1 -l 3 -p 5 -a 2 -n 3 > pfile03
gen-Trucks -seed 680045613 -t 1 -l 3 -p 6 -a 2 -n 4 > pfile04
gen-Trucks -seed 1922597212 -t 1 -l 3 -p 7 -a 2 -n 5 > pfile05
gen-Trucks -seed 3846211 -t 1 -l 3 -p 8 -a 2 -n 6 > pfile06
gen-Trucks -seed 764310 -t 1 -l 4 -p 6 -a 3 -n 7 > pfile07
gen-Trucks -seed 234599 -t 1 -l 4 -p 7 -a 3 -n 8 > pfile08
gen-Trucks -seed 908548 -t 1 -l 4 -p 8 -a 3 -n 9 > pfile09
gen-Trucks -seed 78547 -t 1 -l 4 -p 9 -a 3 -n 10 > pfile10
gen-Trucks -seed 2216746 -t 1 -l 4 -p 10 -a 3 -n 11 > pfile11
gen-Trucks -seed 32087005 -t 1 -l 4 -p 11 -a 3 -n 12 > pfile12
gen-Trucks -seed 21454334 -t 1 -l 5 -p 9 -a 4 -n 13 > pfile13
gen-Trucks -seed 314313 -t 1 -l 5 -p 10 -a 4 -n 14 > pfile14
gen-Trucks -seed 1267652 -t 1 -l 5 -p 11 -a 4 -n 15 > pfile15
gen-Trucks -seed 1020061 -t 1 -l 5 -p 12 -a 4 -n 16 > pfile16
```

```
gen-Trucks -seed 4287421 -t 1 -l 5 -p 13 -a 4 -n 17 > pfile17
gen-Trucks -seed 871235621 -t 1 -l 5 -p 14 -a 4 -n 18 > pfile18
gen-Trucks -seed 28183798 -t 1 -l 6 -p 12 -a 5 -n 19 > pfile19
gen-Trucks -seed 78854 -t 1 -l 6 -p 13 -a 5 -n 20 > pfile20
gen-Trucks -seed 1216 -t 1 -l 6 -p 14 -a 5 -n 21 > pfile21
gen-Trucks -seed 92349 -t 1 -l 6 -p 15 -a 5 -n 22 > pfile22
gen-Trucks -seed 93543 -t 1 -l 6 -p 16 -a 5 -n 23 > pfile23
gen-Trucks -seed 76345 -t 1 -l 6 -p 17 -a 5 -n 24 > pfile24
gen-Trucks -seed 532323 -t 1 -l 7 -p 15 -a 6 -n 25 > pfile25
gen-Trucks -seed 432102 -t 1 -l 7 -p 16 -a 6 -n 26 > pfile26
gen-Trucks -seed 40275209 -t 1 -l 7 -p 17 -a 6 -n 27 > pfile27
gen-Trucks -seed 8465321 -t 1 -l 7 -p 18 -a 6 -n 28 > pfile28
gen-Trucks -seed 9347325 -t 1 -l 7 -p 19 -a 6 -n 29 > pfile29
gen-Trucks -seed 832250 -t 1 -l 7 -p 20 -a 6 -n 30 > pfile30
```

55 Tsp

```
## TSP

*   Origin: Obtained from Maria Fox and Derek Long.
*   Adaptions: None.
*   Description: Untyped STRIPS domain. Extremely simple version of TSP.
    The locations are connected by a complete graph, i.e. each location is
    accessible from each other location. The edges all have equal cost -
    one moving operation - and the goal is simply to have all locations
    visited. An optimal solution simply visits all locations once in an
    arbitrary ordering.
*   Parameters: -n number of locations
*   Generation: No randomization.
```

56 Turnandopen

```
# Turn and Open (Temporal, Satisficing)

This is a temporal domain which appeared in ipc2011.

## Domain Description

In this domain, there are a number of robots with two gripper hands and a set of
rooms containing balls. The goal is to find a plan to transport balls from a
given room to another. There are doors that must be open to move from one room
to another. In order to open a given door, the robot must turn the doorknob and
open the door at the same time.

## Authors

Sergio Jiménez Celorrio

## Original File Names

| file             | original name |
|------------------|---------------|
| domain.pddl      | domain.pddl   |
```

```
| instance-1.pddl  | pfile0.pddl   |
| instance-2.pddl  | pfile1.pddl   |
| instance-3.pddl  | pfile2.pddl   |
| instance-4.pddl  | pfile3.pddl   |
| instance-5.pddl  | pfile4.pddl   |
| instance-6.pddl  | pfile5.pddl   |
| instance-7.pddl  | pfile6.pddl   |
| instance-8.pddl  | pfile7.pddl   |
| instance-9.pddl  | pfile8.pddl   |
| instance-10.pddl | pfile9.pddl   |
| instance-11.pddl | pfile10.pddl  |
| instance-12.pddl | pfile11.pddl  |
| instance-13.pddl | pfile12.pddl  |
| instance-14.pddl | pfile13.pddl  |
| instance-15.pddl | pfile14.pddl  |
| instance-16.pddl | pfile15.pddl  |
| instance-17.pddl | pfile16.pddl  |
| instance-18.pddl | pfile17.pddl  |
| instance-19.pddl | pfile18.pddl  |
| instance-20.pddl | pfile19.pddl  |


usage:

OPTIONS   DESCRIPTIONS
-n <num>    number of robots (minimal 1)
-r <num>    number of rooms (minimal 1)
-o <num>    number of balls (minimal 1)
-s <num>    random seed
```

57 Tyreworld

```
## Tyreworld

*   Origin: Stuart Russel. Adaption for multiple tyres by Jana Koehler.
    Taken from the [IPP][6] domain collection.
*   Adaptions: None.
*   Description: Typed STRIPS domain. Replace a flat tyre with a spare
    one. This involves fetching the tools (wrench, jack, pump) from the
    boot, undoing the nuts on the flat tyre, jacking up the (appropriate)
    hub(s), removing the tyre, doing up the spare one, etc. Adapted for
    several tyres by simply increasing the number of flat tyres to be
    replaced.
*   Parameters: -n number of tyres.
*   Generation: No randomization.
```

58 Visitall

```
usage:

 OPTIONS   DESCRIPTIONS

 -n <num>    size of square grid
```

```
 -r <num>    ratio of cells in the goal state
 -u <num>    number of unavailable locations ---which are randomly arranged

 -s <num>    random seed (optional)


IPC: -u 0 -n x Full (-r 1)  and Half (-r 0.5) instances

IPC 14 x from 5 to 18

As in the IPC, we follow two linear scalings: full and half
```

59 Woodworking

```
How to use the problem generator:

The problem generator creates X instances for the seq-sat and seq-opt
tracks. It allows to change the available percentages of wood and the
parts.
To change them go to the end of the file and look for the tag tasks:

tasks = [generate_instance(size, wood_factor)
         for wood_factor in [1.4, 1.2, 1.0]
         for size in range (3, 33, 3)]

The former generates 30 instances, 10 for each wood factor, starting
in 3, ending in 33 and increasing by 3 from one to each other.

By default problems are created in sub-folders named "seq-sat" and
"seq-opt" which have to exist before the script is run.

The default behavior is to generate problems with only one tool of
each type, if more tools want to be created, the _default_machines
constant has to be changed.

In its current form, the script generates the last 4 problems of IPC 2011:
p17-->p17 IPC
p18-->p19 IPC
p19-->p18 IPC
p20-->p20 IPC



    wood_factor = float(sys.argv[1])
    size = int(sys.argv[2])
    num_machines = int(sys.argv[3])
    seed = int(sys.argv[4])

tasks = [generate_instance(size, wood_factor)
         for wood_factor in [1.4, 1.2, 1.0]
         for size in range (3, 33, 3)]
```

60 Zenotravel

```
Usage: ztravel [-n|-s|-t|-c|-u] <seed> <#cities> <#planes> <#people> [distance]


IPC instances:

           planes person cities
p01.pddl:   1    2       3
p02.pddl:   1    3       3
p03.pddl:   2    4       3
p04.pddl:   2    5       3
p05.pddl:   2    4       4
p06.pddl:   2    5       4
p07.pddl:   2    6       4
p08.pddl:   3    6       5
p09.pddl:   3    7       5
p10.pddl:   3    8       5
p11.pddl:   3    7       6
p12.pddl:   3    8       6
p13.pddl:   3    10      6
p14.pddl:   5    10      10
p15.pddl:   5    15      12
p16.pddl:   5    15      14
p17.pddl:   5    20      16
p18.pddl:   5    20      18
p19.pddl:   5    25      20
p20.pddl:   5    25      22
```