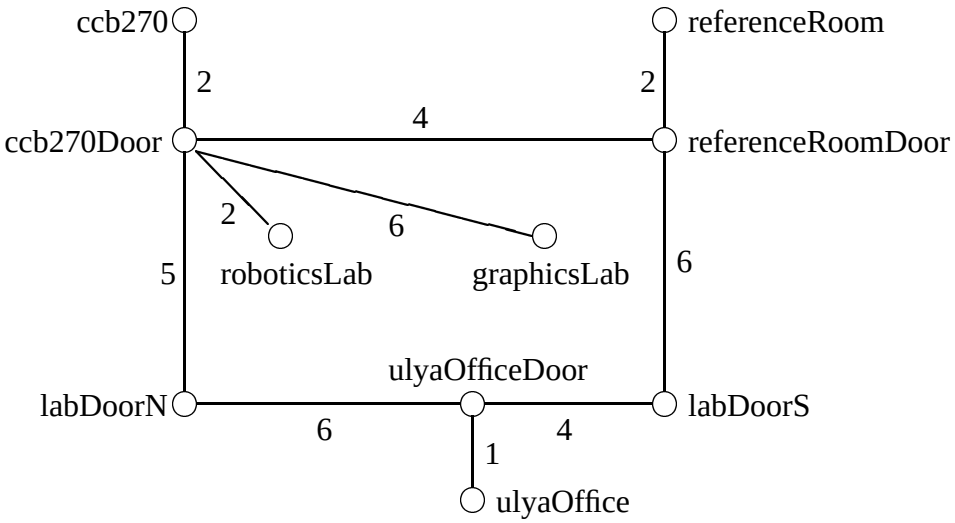
# An IndiGolog Robot Control Exercise

Yves Lespérance
Department of Computer Science
York University

April 30, 2003

Write an IndiGolog program to control a simulated robot that delivers packages on a floor of a building. Your controller will use a simple graph representation of the area in which the robot moves. To test your program, use the following graph (representing the department's old 2nd floor lab area in CCB):




Assume that the robot can perform the following primitive actions:

- `goTo(Place)`, i.e. move the robot to the place specified, which need not be adjacent to where the robot is currenty locate (in one `goTo` action, the robot can move several edges away);
- `pickUp(Package)`, have the robot pick up the specified package; the package must be at the place where the robot is currently located for this action to be possible; the robot may have several packages on board at the same time;
- `dropOff(Package)`, have the robot drop off the specified package at the place where the robot is currently located; the package must be on board the robot for this action to be possible.

Implement an appropriate action theory. Your program should use the following fluents:

- `locatedAt(RobotOrPackage)`, a functional fluent whose value is the place where the robot or package specified is located in the situation;

- `onBoard(Package)`, a predicate fluent that holds iff the specified package is on board the robot in the situation;
- `deliveryOrdered(Package,FromPlace,ToPlace)`, a predicate fluent that holds iff the package must be delivered by the robot from the origin place to the destination place specified;
- `delivered(Package)`, a predicate fluent that holds iff the package has been delivered by the robot;
- `distanceTravelled`, a functional fluent whose value is the total distance travelled so far by the robot.

Note that IndiGolog does not really support predicate fluents, and that these have to be implemented as binary-valued functional fluents, which may take the values `true` or `false`.

Use the following initial situation axioms to test your program:

```
initially(locatedAt(robot),roboticsLab).
initially(locatedAt(package1),graphicsLab).
initially(locatedAt(package2),ccb270).
initially(locatedAt(package3),ulyaOffice).
initially(deliveryOrdered(package1,graphicsLab,ulyaOffice),true).
initially(deliveryOrdered(package2,ccb270,graphicsLab),true).
initially(deliveryOrdered(package3,ulyaOffice,referenceRoom),true).
initially(distanceTravelled,0).
```

Your program should include two procedures for the robot to perform its task:

- A procedure `deliverRandom` that gets the robot to make all the deliveries that have been ordered, without trying to minimize the distance travelled by the robot; the procedure simply chooses packages at ramdom (i.e. non-deterministically) and delivers them.
- A procedure `deliverOptimal(Increment)` that gets the robot to make all the deliveries that have been orderred and minimizes the distance travelled by the robot, so that it is no more than `Increment` – 1 units from the optimal distance. This procedure should be implemented by performing iterative deepening search. In this method, one sets an initial limit for the paths to be considered, in this case, a distance travelled of 0, and searches for a path (sequence of action) that achieves the objective; if a path is found, it is returned; if none is found the limit is increased by the specified increment and the search is performed again. Thus, progressively longer path/sequence of actions are considered. (See the URL below for examples involving search.)

Note that in implementing these procedures, you need to determine whether a place (a node in the graph) is reachable from another place, and what is the shortest distance/path between the two places (we assume that the `goTo(Place)` action uses the shortest path). You should implement a

Prolog predicate `distanceBetween(Place1,Place2,Dist)` to compute the shortest distance `Dist` between `Place1` and `Place2`. Thus, you don't need to compute shortest paths in IndiGolog, you can do it in Prolog. But in `deliverOptimal(Increment)`, your program must use IndiGolog search to find the best sequence of order pick up and drop off to deliver all the orders in the shortest distance.

Test both procedures on the initial situation (and graph) specified above. Hand in a trace that shows the sequence of actions performed and the total distance travelled. You can obtain this by running:

```
indigolog([deliverRandom,?(distanceTravelled = Dist)]).
```

In addition, your program should also include a third procedure `plan(Goal)` that finds a sequence of actions that achieves the given goal and executes it. The goal can be an arbitrary IndiGolog condition. This procedure should also use iterative deepening search, but here you should only minimize the number of actions performed. Perform the following tests:

```
indigolog(plan(onBoard(package1)=true)).
indigolog(plan(and((onBoard(package1)=true),
                   (locatedAt(package1)=roboticsLab)))).
```

Finally, modify the program to handle new orders while the program is running. For this add a new exogenous action

```
makeNewOrder(PackageID,PickUpLocation,DropOffLocation)
```

and modify the program to deal with it. When a new order is made, `deliverOptimal` should compute a new shortest delivery route to deliver the new order as well as the old orders. Test this feature.

Implement your program using the implementation of IndiGolog found in `http://www.cs.yorku.ca/~lesperan/IndiGolog/`; see this URL for the interpreter, some examples, and instructions on how to write and run programs. Hand in a listing of your program's code with appropriate documentation and your test runs.