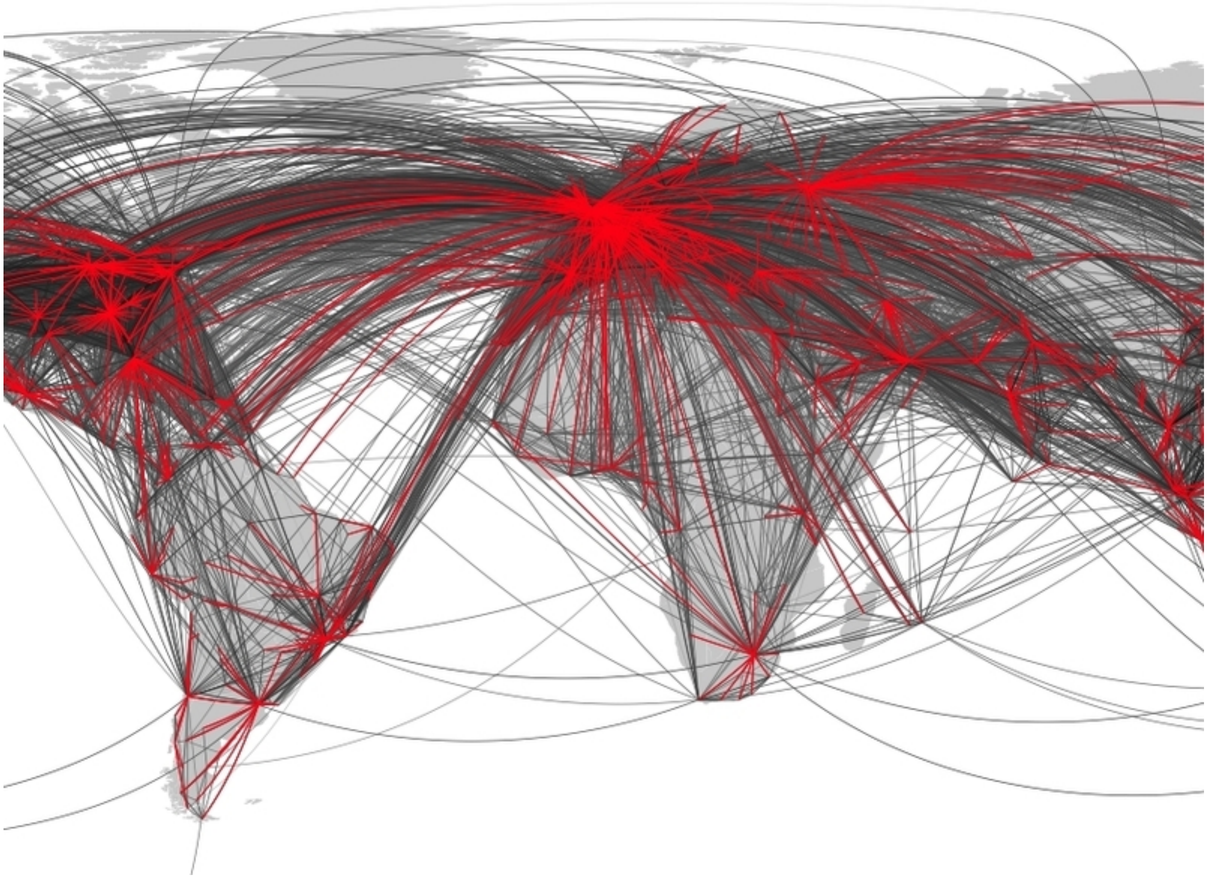
Travel Navigator

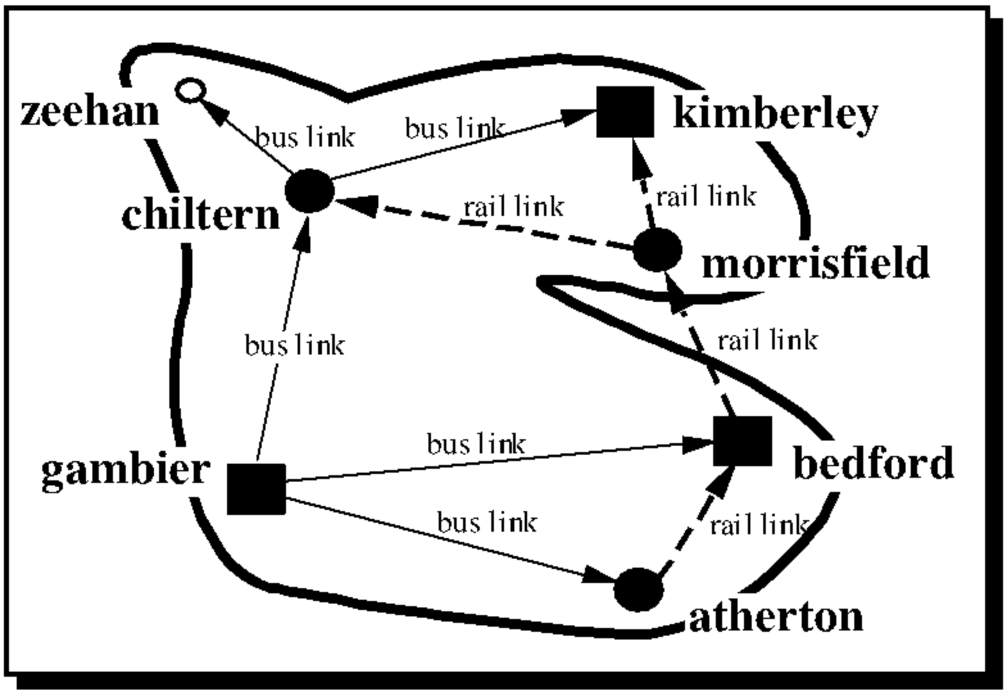


This tutorial has been developed with reference to the following sources:
"Prolog Programming for Artificial Intelligence" (3rd Edition) by I. Bratko, Addison-Wesley, 2001.

A TRAVEL NAVIGATOR SYSTEM

One often used read world application of searching has been in the development of navigation systems, such as e.g. Google Maps, Navman, and Tom Tom. These display a road map of the local neighbourhood and (when used while travelling) give the driver advice on the best route to take and can avoiding high traffic areas and obstacles, or deliberately pass by items of interest etc. Such systems can often also provide walking directions or public transit information and route planning.

To illustrate the application of simple searching, we'll construct a simple travel navigation system to assist a tourist who wishes to visit the scenic island of Jazzmania. The island has 7 centres of population. Some of the centres are linked by rail, others are linked by bus. A schematic map of the island looks like this.



There is an added complication in that due to current travel restrictions, trains and buses are only permitted to travel one way between centres for the duration of your visit. So the bus link between Gambier and Bedford will only permit you to travel in that direction, not from Bedford to Gambier.

REPRESENTATION OF TRAVEL INFORMATION

This schematic map is can be represented as a type of knowledge representation known as a *semantic network* (and in fact, already is an example of a semantic network). We represent the travels location as the nodes and the bus and rail links as directed arcs connecting nodes. It can therefore by represented in Prolog, by turning each link into a Prolog predicate. For example, the predicate:

```
bus(gambier, bedford).
```

represents the link from Gambier to Bedford. Notice that you are only permitted to travel from the first to the second centre, not the other way.

Further links can be represented in the same way, eg.

```
bus(gambier, atherton).
rail(atherton, bedford).
rail(bedford, morrisfield).
etc.
```

1. Proceed to construct the complete semantic net in Prolog and type it into a Prolog file called **travelnavigator.pl** (refer to the previous tutorials if you have forgotten how to create Prolog source files or consult programs in Prolog)

CONSTRUCTING RULES (which use RECURSION)

Now we can construct appropriate rules to embody knowledge that we may have about this network. These rules will be recursive. That is, a rule may be defined in terms of itself. For example, we know that two centres X and Z are connected by rail (directly or indirectly) if X has a (direct) rail link to another centre Y and Y and Z are connected by rail. That is:

```
connectedbyrail(X,Z) :-     rail(X,Y),
                            connectedbyrail(Y,Z).
```

To see how this rule would work, let us ask the question:

```
?- connectedbyrail(atherton, kimberley).
```

The rule would then be tested, with X set to Atherton (ie. X 'instantiated' to Atherton) and Z set to Kimberley. For this conclusion to be true, both premises must be true. ie. There must be some other centre Y, such that there is a direct rail link from Atherton to Y and it must also be true that this centre Y is connected to Kimberley by rail.

Looking through the knowledgebase, Prolog will find that there is a rail link from Atherton to Bedford. So the first premise in the rule becomes true. To determine whether the second premise is true, ie. whether Bedford and Kimberley are connected by rail, it is necessary to call the same rule again (hence the recursion).

Testing the rule again, with X set to Bedford this time but with Z still set to Kimberley, Prolog will find that there is a direct rail link between Bedford and Morrisfield, so it will set Morrisfield to Y and then try to establish the truth of the assertion **connectedbyrail (morrisfield, kimberley).**

Commonsense tells us that this is true, since the two towns have a direct rail link and so are obviously connected by rail but nothing in our knowledgebase tells the computer this. To correct this problem and to ensure that the recursion stops at some point, we need to insert the following extra rule in front of the other rule (it's important it goes first):

```
connectedbyrail(X,Y) :- rail(X,Y).
```

Now Prolog will be able to establish the truth of the assertion:

```
connectedbyrail(morrisfield, kimberley).
```

and so will answer **yes** to the original question, meaning that it is possible to get from Atherton to Kimberley by rail.

2. Enter the two new rules for determining whether two centres are connected by rail.

3. Query the knowledgebase to see if it is possible to travel by rail from Atherton to Morrisfield. Does the system behave as would be expected from the previous discussion?

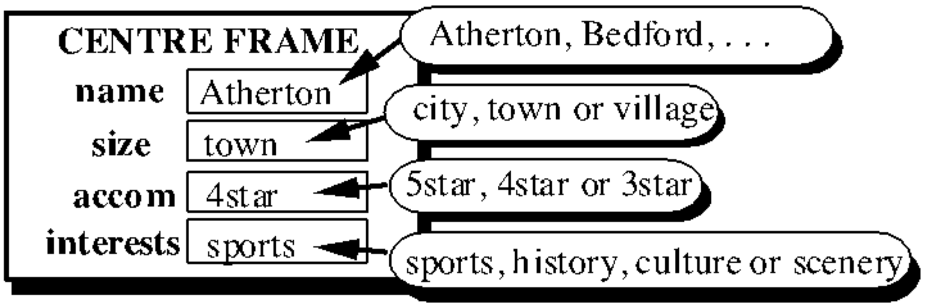
4. Write similar rules to indicate which centres are connected by bus, as has already been done for the rail connections.

5. Query the expanded system to see if it is possible to go from Gambier to Morrisfield by bus. Also find out what towns you can travel to by bus from Chiltern.

REPRESENTING NODES by FRAMES

The travel navigator system would be much more useful if it could include more information than just the name of each population centre. It would be useful to know how large it is, how good the accommodation is, and what interesting features it offers for a tourist. Thus we need to provide a “bundle” of information for each centre; the most appropriate way to do this is to use a knowledge representation called *frames* (frames are similar to classes in Java/C++/Python and structs in C).

Let us construct a **Centre** frame containing the following information:



This particular frame can be represented in Prolog, using the predicate **centre** (technically, this kind of predicate is called a *functor*), as follows:

centre(atherton, town, fourstar, sports).

6. Insert appropriate clauses to store the following information on all of the centres of population. (Be careful with your spelling of centre!)

atherton	town	fourstar	sports
gambier	city	fivestar	history
bedford	city	fourstar	sports
morrisfield	town	threestar	sports
chiltern	town	fourstar	culture
zeehan	village	threestar	scenery
kimberley	city	fivestar	culture

These clauses resemble a simple database. Prolog can act as a very effective database query language. In fact the most popular query language for relational databases, SQL (structured query language), is based on similar predicate logic principles.

Say we wanted a list of all cities with fivestar accommodation available. We could obtain this by typing in the Prolog query :-

new syntax

?- centre(X, city, fivestar, _).

Note:

- that when the underline character is used in the place of an argument (or as the start of an argument, e.g. **_anything**), it means that the query will match this clause irrespective of the current value of that argument. ie. underline means “don't care”.

Our knowledgebase is more complex than before but much more as it is now possible to ask questions such as “is there a centre, with 5star accommodation, which can be reached by rail from Atherton?”. For example, the following query asks “Is there a centre X, such that atherton is connected by rail to X and X has fivestar accommodation. The other properties of X are unimportant.”

```
?- connectedbyrail(atherton,X),  centre(X, _, fivestar, _).
```

What is the following query asking?

```
?- bus(gambier, X),  centre(X, _, fivestar, culture).
```

7. Type in the these three queries and see if the Travel Navigator works the way you would expect it to.

```
?- centre(X, city, fivestar, _ ).
?- connectedbyrail(atherton, X),  centre(X, _, fivestar, _).
?- bus(gambier, X),  centre(X, _, fivestar, culture).
```

8. Ask the Travel Navigator to list out any cultural centre which can be reached by bus from Gambier (either directly or indirectly).

TRAVEL NAVIGATOR QUESTION SHEET

NAME ....

STUDENT ID .... **LOGIN** ....

Question 1
Query the **travelnavigator** knowledgebase to find out if you can travel (directly or indirectly) by bus from Atherton to Zeehan. Write a Prolog query, to determine this:

```
connectedbybus(atherton, zeehan).
```

What is answer to this query?

```
no
```

Can you explain the reason for this?

```
the only route out of Atherton is by rail
```

Question 2
Assume you are a tourist who is not restricted to bus or rail travel and is quite happy to change from rail to bus whenever needed. It is possible to construct a rule which says that two centres X and Y are linked directly if X and Y are linked by bus, or if X and Y are linked by rail. To construct rules such as this, with an OR between premises, you should construct two Prolog rules thus:

linked(X, Y) :- rail(X, Y).
linked(X, Y) :- bus(X, Y).

Add these rules to the knowledgebase; together with another a rule, called the **connected** rule, which says that X is **connected** to Z if X is **linked** directly to some other centre Y and Y is **connected** to Z.

```
connected(X, Z) :- linked(X, Y), connected(Y, Z).
```

Remember that this rule is recursive so ensure that you insert an extra rule to terminate the recursion.

```
connected(X, Y) :- linked(X, Y).
```

Now query the knowledgebase to see if you can travel from Atherton to Zeehan, provided you are willing to take bus or train links where available.

What is the answer to this query?

```
true
```

Question 3
Modify the **connectedbybus** rule so that Prolog will print out the name of each centre as it is being tested. The new rule makes use of some rudimentary Prolog output facilities; the **write** and newline (**nl**) statements:

new syntax

```
connectedbybus(X,Z) :-    bus(X,Y),
                          write(X), nl, write(Y), nl,
                          connectedbybus(Y,Z).
```

You will need to insert write and newline statements in the terminator rule as well. Now query the knowledgebase to see if you can travel by bus from Gambier to Zeehan. Note down the centres being tried, in the order they are being printed out.

`gambier` .... `bedford` .... `atherton` .... `chiltern` .... `zeehan` ....

Now see if you can travel by bus from Gambier to Morrisfield. Again note down the centres as they are tried.

`gambier` .... `bedford` .... `atherton` .... `chiltern` .... `zeehan` ....

If you trace these out on the original map of Jazzmania, you will see that Prolog tries out one specific path, until it is stopped, then it will **backtrack** and try out another path, until it finally reaches its goal. With reference to the searching lecture, what kind of search is this?

`depth-first search`

Question 4
Construct an appropriate Prolog query to give the names and types of accommodation for all cities (note: cities) in Jazzmania:

```
centre(X, city, Y, _).
```

Type in the query and write out the information that is given:

`fivestar in gambier, fourstar in bedford, fivestar in kimberly`

Question 5
Construct an appropriate Prolog query to determine whether there are two centres, connected to each other (by rail or bus), both of which have sports features of interest.

```
centre(X, _, _, sports), centre(Y, _, _, sports), connected(X, Y).
```

Type in the query and write out the information that is given:

`atherton and bedford, atherton and morrisfield, bedford and morrosfield`