Computational Intelligence

A Logical Approach

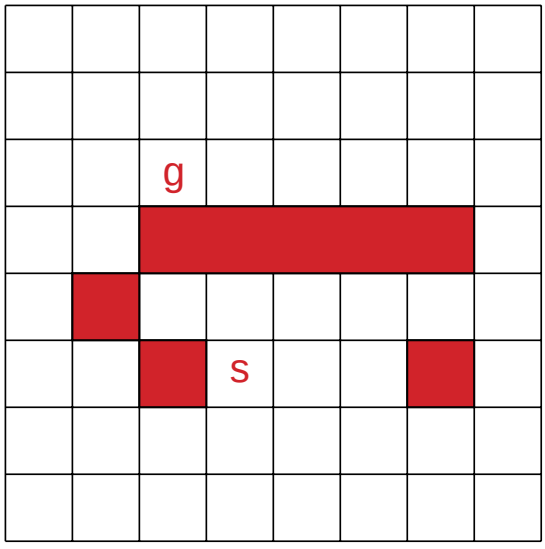
Problems for Chapter 4

Here are some problems to help you understand the material in Computational Intelligence: *A Logical Approach*. They are designed to help students understand the material and practice for exams.

This file is available in html, or in pdf format, either without solutions or with solutions. (The pdf can be read using the free acrobat reader or with recent versions of Ghostscript).

1 Finding Paths in a Grid

Consider the problem of finding a path in the grid shown below from the position *s* to the position *g*. The robot can move on the grid horizontally and vertically, one square at a time (each step has a cost of one). No step may be made into a forbidden shaded area.



(a) On the grid, number the nodes in the order in which they are removed from the frontier in a depth-first search from *s* to *g*, given that the order of the operators you will test is: up, left, right, then down. Assume there is a cycle check.

(b) Number the nodes in order in which they are taken off the frontier for an A^* search for the same graph. Manhattan distance should be used as the heuristic function. That is, $h(n)$ for any node n is the Manhattan distance from n to g. The Manhattan distance between two points

is the distance in the x-direction plus the distance in the y-direction. It corresponds to the distance traveled along city streets arranged in a grid. For example, the Manhattan distance between g and s is 4. What is the path that is found by the A^* search?

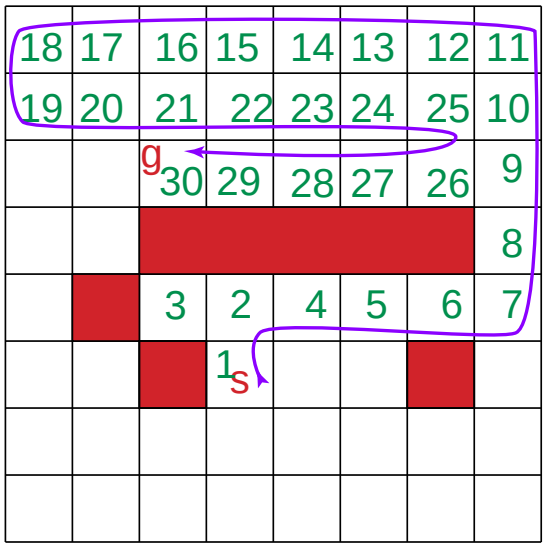
(c) Assume that you were to solve the same problem using dynamic programming. Give the *dist* value for each node, and show which path is found.

(d) Based on this experience, discuss which algorithms are best suited for this problem.

(e) Suppose that the graph extended infinitely in all directions. That is, there is no boundary, but s, g, and the forbidden area are in the same relative positions to each other. Which methods would no longer find a path? Which would be the best method, and why?

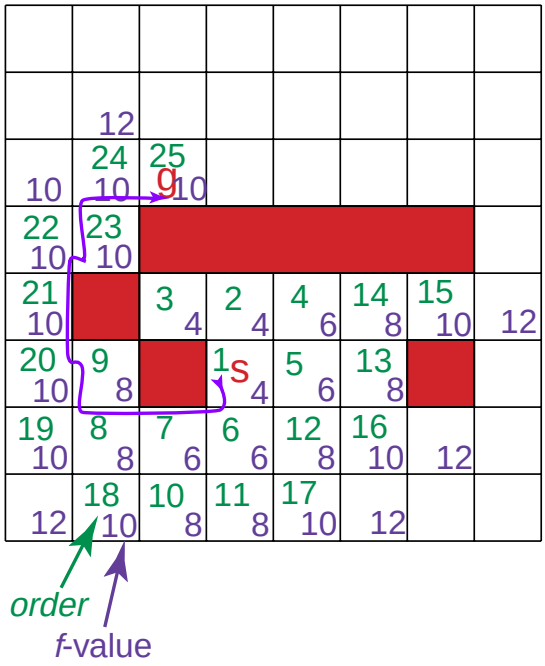
Solution to part (a)

Depth-first search:



Solution to part (b)

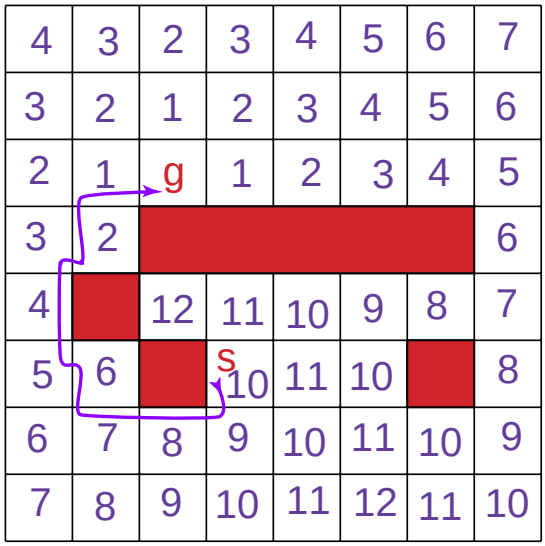
A^* search:



Note that here we assumed that for those nodes whose f-value is the same, that the last added to the queue is used. You can use any other convention.

Solution to part (c)

Dynamic Programming:



Solution to part (d)

It seems as though A^* and dynamic programming are best for this problem. A^* would be best if you were solving it once. If you wanted to solve it multiple times for the same goal dynamic

programming would be good.

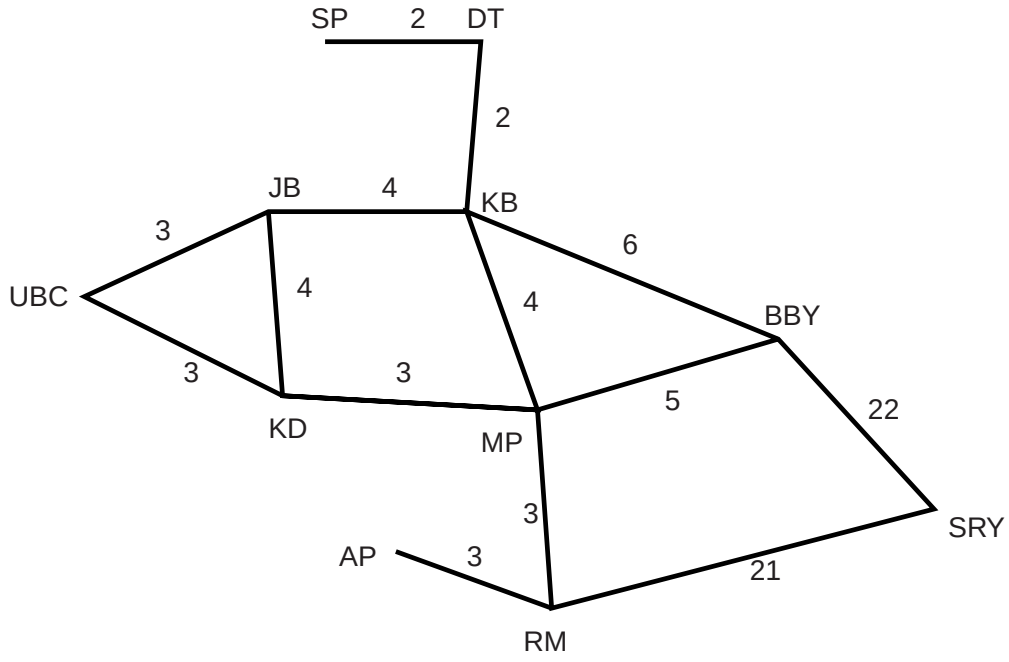
Depth-first search looks pretty stupid given this ordering of neighbors. If you had searched in the order: left, up, right, down, depth-first search would have looked much more sensible.

Solution to part (e)

If the graph had extended infinitely in all directions, A^* would still find a solution. Depth-first search would wander off infinitely. Dynamic programming would find a path from *s* to *g*, but then continue off forever. If you stopped it when it had included *s* in the distance map, it would find the shortest path, but this would be the same as a shortest-first search from *g*, which still isn't as good as A^*. It would, however, let us reuse the distance function built for other starting positions to the same goal.

2 Searching on a simple graph

Consider the graph (not drawn to scale) with arc lengths shown on the arcs:



Suppose we have the following heuristic values for the distance to *sp*.

h(sp)=0	h(dt)=2	h(kb)=3
h(jb)=3	h(ubc)=5	h(kd)=6
h(mp)=7	h(bby)=8	h(ap)=8
h(rm)=9	h(sry)=29	

(a) Show the nodes expanded (taken off the frontier), in order, and the f-value for each node added to the frontier for an A^* search from *ubc* to *sp*. Assume that multiple-path pruning is used, and that the search stops after the first path is found. Show clearly the path found. [Explain clearly what your notation means.]

(b) Show how dynamic programming can be used to find a path from *ubc* to *sp*. Show all distance values that are computed, and how these are used to find the shortest path. What path is found?

(c) Suppose you were contacted by TecnoTaxi to advise on a method for finding routes between locations in your city. What method would you recommend, and why. Give one shortcoming of the method you propose. You must use full sentences.

Solution to part (a)

The following shows the nodes added to and removed from the queue:

Node	f-value	Order Removed
UBC	5	1
JB	6	2
KD	9	3
KB	10	4
MP	13	
BBY	21	
DT	11	5
SP	11	6

Note that the other three nodes (AP, RM and SRY) are never placed on the queue, and their f-value is not computed.

The final path computed is:

$$UBC \rightarrow JB \rightarrow KB \rightarrow DT \rightarrow SP$$

Solution to part (b)

Dynamic programming builds the following distance map to *sp*

Node	Distance
SP	0
DT	2
KB	4
JB	8
MP	8
BBY	10
UBC	11
KD	11
RM	11
AP	14
SRY	32

To get from UBC to SP, you need to compare going via JB (with distance 3+8) with going via KD (distance 3+11). Thus the first step is to JB. From JB, you need to compare going via KB (distance 8), via KD (distance 15) and UBC (distance 14) and choose KB. From KB, you choose DT and then SP.

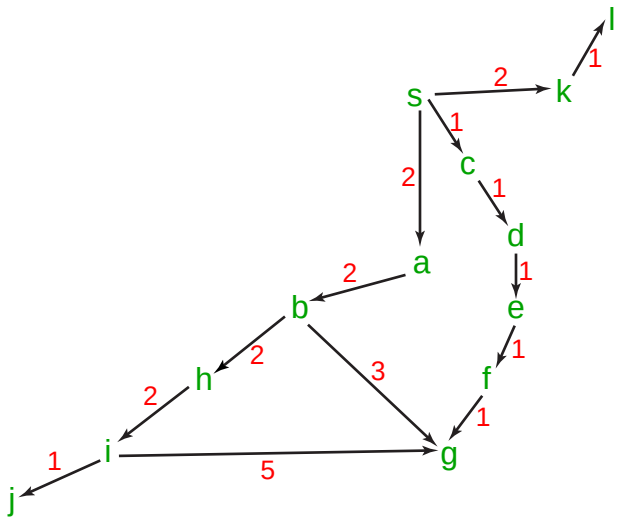


Figure 1: Search Graph

The final route is:

$UBC \rightarrow JB \rightarrow KB \rightarrow DT \rightarrow SP$

Solution to part (c)

You could have suggested dynamic programming, in that once the distance tables have been computed they can be accessed quickly. The problem is that you need a distance function to every possible destination.

You could have suggested A^* that will find shortest routes. The problems is that the time and space is exponential in the path length.

The best solution is probably some hierarchical route finder, finding distances between major intersections and then locally searching from a particular location to these intersections.

3 Comparting Different Search Strategies

Consider the graph shown in Figure 1. Suppose the neighbours are given by the following relations:

```
neighbours(s,[a,c,k]).
neighbours(a,[b]).
neighbours(b,[h,g]).
neighbours(c,[d]).
neighbours(d,[e]).
neighbours(e,[f]).
neighbours(f,[g]).
neighbours(g,[]).
neighbours(h,[i]).
neighbours(i,[j]).
```

```
neighbours(j,[]).
neighbours(k,[l]).
neighbours(l,[]).
```

Suppose the heuristic estimate of the distance to g is:

```
h(a,2).     h(b,3).
h(c,4).     h(d,3).
h(e,2).     h(f,1).
h(g,0).     h(h,4).
h(i,5).     h(j,6).
h(k,5).     h(l,6).
h(s,4).
```

For each of the following search strategies to find a path from s to g:

(a) Depth-first search

(b) Breadth-first search

(c) Least-cost first search

(d) Best-first search

(e) A* search

Specify:

i) What is the final path found?

ii) How many nodes were expanded?

iii) Explain why it selected nodes during the search that were not on the shortest path from s to g.

iv) Explain why it may have been led astray in the final solution. (Either state that it found the shortest part, or explain why the shortest path was not found).

Note there are 20 parts to this question. By brief and concise. You can use the search applet available on the web page and at `~cs322/tools/search/search`.

Solution to part (a): Depth-first search

i) What is the final path found?
$s \rightarrow a \rightarrow b \rightarrow h \rightarrow i \rightarrow g$.

ii) How many nodes were expanded?
7.

iii) Explain why it selected nodes during the search that were not on the shortest path from s to g.
It selects nodes in order irrespective of where the goal is.

iv) Explain why it may have been led astray in the final solution.
It reports whatever path it finds first; this could be any path depending on the order of the neighbours.

Solution to part (b): Breadth-first search

i) What is the final path found?
$s \rightarrow a \rightarrow b \rightarrow g$.

ii) How many nodes were expanded?
9.

iii) Explain why it selected nodes during the search that were not on the shortest path from s to g.
It selects all nodes that are two steps from the start node, irrespective of where the goal is, before it expands nodes that are three steps away and finds the goal.

iv) Explain why it may have been led astray in the final solution.
It finds the path with the fewest arcs, not the shortest path.

Solution to part (c): Least-cost-first search

i) What is the final path found?
$s \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g$.

ii) How many nodes were expanded?
10

iii) Explain why it selected nodes during the search that were not on the shortest path from s to g.
It explores the paths in order of length, irrespective of where the goal is.

iv) Explain why it may have been led astray in the final solution.
It wasn't; least cost first always finds the shortest path to the goal.

Solution to part (d): Best-first search

i) What is the final path found?
$s \rightarrow a \rightarrow b \rightarrow g$.

ii) How many nodes were expanded?
4

iii) Explain why it selected nodes during the search that were not on the shortest path from s to g.
It chooses the node closest to the goal, and doesn't take into account the path length from the start node.

iv) Explain why it may have been led astray in the final solution.
Node a was closer to the goal than node c, once it had nodes on the frontier that were close to goal, it never considered c.

Solution to part (e): A* search

i) What is the final path found? $s \to c \to d \to e \to f \to g$.

ii) How many nodes were expanded?
7.

iii) Explain why it selected nodes during the search that were not on the shortest path from s to g.
Node a looked as thought it was on a direct route to the goal. But, the deviation to go to b was greater than the cost of going via c.

iv) Explain why it may have been led astray in the final solution. .
It wasn't; A* always finds the shortest path to the goal.

4 Generating Graphs Good for Different Search Strategies

For each of the following, give a graph that is a tree (there is at most one arc into any node), contains at most 15 nodes, and has at most two arcs out of any node.

(a) Give a graph where depth-first search is much more efficient (expands fewer nodes) than breadth-first search.

(b) Give a graph where breadth-first search is much better than depth-first search.

(c) Give a graph where A* search is more efficient than either depth-first search or breadth-first search.

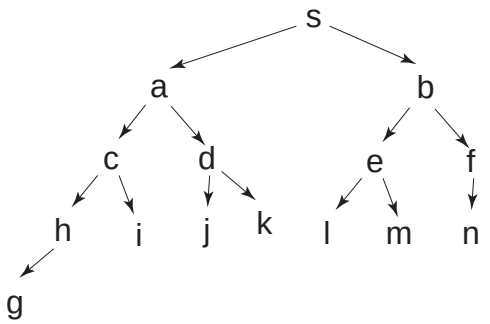
(d) Give a graph where depth-first search and breadth-first search are both more efficient than A* search.

You must draw the graph and show the order of the neighbours (this is needed for the depth-first search). Either give the arc costs and heuristic function or state explicitly that you are drawing the graph to scale and are using Euclidean distance for the arc costs and the heuristic function.

Solution to generating graphs

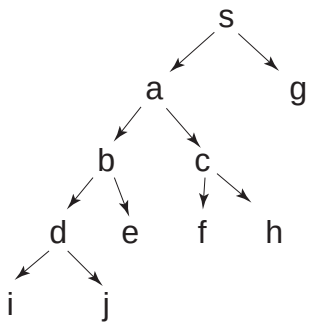
In all of these graphs, we assume that we are on a plane, with Euclidean distance (straight-line distance) as the arc cost and as the heuristic function. We also assume that the neighbours are ordered from left to right. The start node is s and the goal node is g.

(a) Give a graph where depth-first search is much more efficient (expands fewer nodes) than breadth-first search.
Here depth-first search expands five nodes, whereas breadth-first search expands every node:



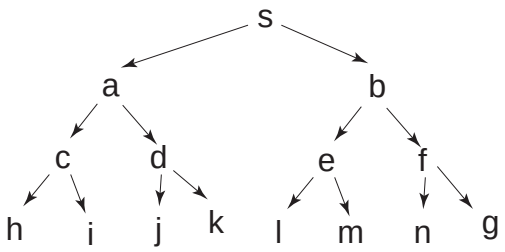
(b) Give a graph where breadth-first search is much better than depth-first search.

Here depth-first search expands every node, whereas breadth-first search expands three nodes:



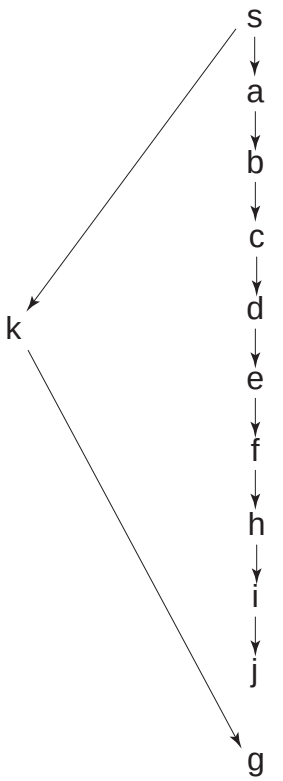
(c) Give a graph where A* search is more efficient than either depth-first search or breadth-first search.

Here depth-first search and breadth-first search expand every node, whereas A* search expands 4 nodes.



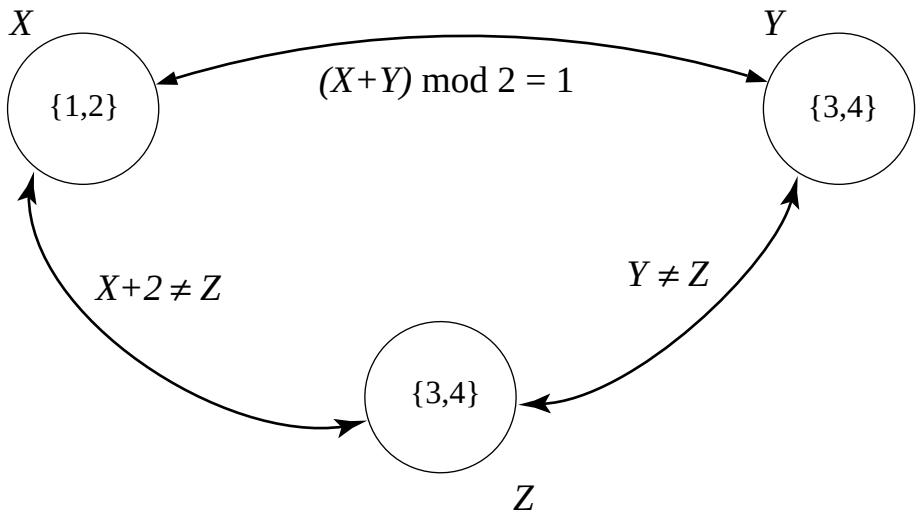
(d) Give a graph where depth-first search and breadth-first search are both more efficient than A* search.

Here depth-first search expands three nodes, breadth-first search expands 4, yet A* search expands every nodes.



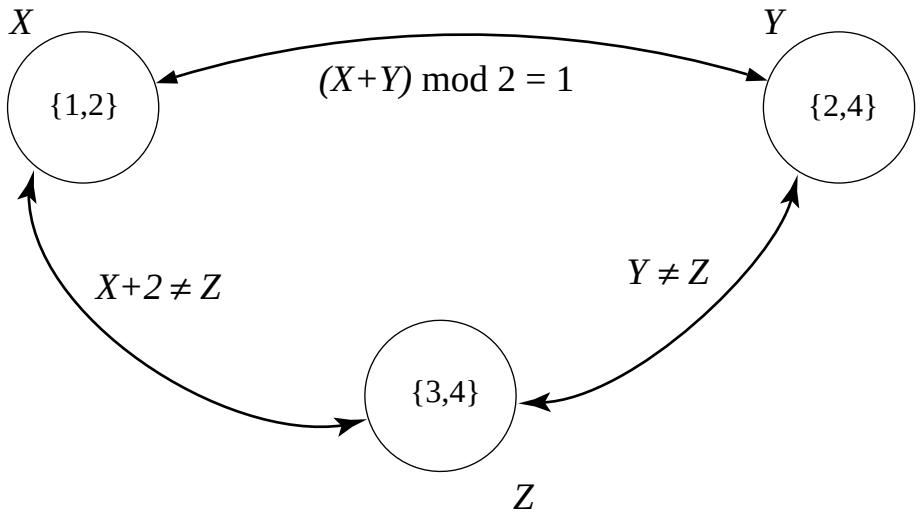
5 Arc Consistency

(a) Consider the following constraint network. Note that $(X + Y) \bmod 2 = 1$ means that $X + Y$ is odd.



Is this constraint network arc consistent? If it is, explain why. If it isn't, explain which arc is not arc consistent and why it isn't arc consistent.

(b) Consider the following constraint network:



Is this constraint network arc consistent? If it is, explain why. If it isn't, explain which arc is not arc consistent and why it isn't arc consistent.

Solution to part (a)

Yes, this is arc consistent.

The relation, $(X + Y) \bmod 2 = 1$ is true of $X = 1, Y = 4$ and $X = 2, Y = 3$. For each X-value there is a Y-value and for each Y-value there is an X-value.

The relation $X + 2 \neq Z$ is true of $X = 1, Z = 4$ and $X = 2, Z = 3$. For each X-value there is a Z-value and for each Z-value there is an X-value.

The relation $Y \neq Z$ is true of $Y = 3, Z = 4$ and $Y = 4, Z = 3$. For each Y-value there is a Z-value and for each Z-value there is an Y-value.

Thus it is arc consistent, but there are no solutions!

Solution to part (b)

No, this constraint network is not arc consistent.

The arc $\langle X, Y\rangle$ is not arc consistent. For $X = 2$ there is no Y such that $(X + Y) \bmod 2 = 1$ is true.

6 Arc Consistency

Suppose you have a relation $r(X, Y)$ that is true of there is a word in the word list below with first letter X and second letter Y.

The word list is:

add	arc	bad	bud
cup	dip	fad	odd

Suppose the domain of X is $\{a, b, c, d\}$ and that of Y is $\{a, d, i, r\}$.

(a) Is the arc $\langle X, Y\rangle$ arc consistent? If so, explain why. If not, show what element(s) can be removed from a domain to make it arc consistent.

(b) Is the arc $\langle Y, X\rangle$ arc consistent? If so, explain why. If not, show what element(s) can be removed from a domain to make it arc consistent.

Solution to part (a)

Is the arc $\langle X, Y\rangle$ arc consistent?

no.

Arc $\langle X, Y\rangle$ is arc consistent means that for every X there exists a Y such that $r(X, Y)$ is true.

The following table shows for each value of X a value of Y such that $R(X, Y)$ is true:

X	Y
a	d
b	a
c	***
d	i

There is no element Y in $\{a, d, i, r\}$ such that $r(c, Y)$ is true. That is there is no word starting with *ca, cd, ci,* or *cr* in the word list. *c* can be removed from the domain of X to make $\langle X, Y\rangle$ arc consistent.

Solution to part (b)

Is the arc $\langle Y, X\rangle$ arc consistent?

yes.

Arc $\langle Y, X\rangle$ is arc consistent means that for every Y there exists an X such that $r(X, Y)$ is true.

The following table shows for each value of Y a value of X such that $R(X, Y)$ is true:

Y	X
a	b
d	a
i	d
r	a

Thus the arc $\langle Y, X\rangle$ is arc consistent.

The most common mistake: The relation $r(X, Y)$ corresponds to two arcs, $\langle X, Y\rangle$ and $\langle Y, X\rangle$. You do not swap the arguments to $r(X, Y)$.

7 Solving a CSP via backtracking, arc consistency, hillclimbing

In this question you will look at backtracking, arc consistency, and hill climbing for solving the same CSP problem.

Consider a scheduling problem, where there are five variables *A, B, C, D,* and *E,* each with domain $\{1, 2, 3, 4\}$. Suppose the constraints are: $E - A$ is even, $C \neq D$, $C > E$, $C \neq A$, $B > D$, $D > E$,$B > C$.

(a) Show how backtracking can be used to solve this problem, using the variable ordering A, B, C, D, E. To do this you should draw the search tree generated to find all answers. Indicate clearly the satisfying assignments.

To indicate the search tree, write it in text form with each branch on one line. For example, suppose we had variables X, Y and Z with domains t, f, and constraints $X \neq Y$, $Y \neq Z$. The corresponding search tree can be written as:

```
X=t Y=t failure
    Y=f Z=t solution
        Z=f failure
X=f Y=t Z=t failure
        Z=f solution
    Y=f failure
```

Hint: the easiest way to solve this problem is to write a program to generate the tree (using whatever programming language you like).

(b) Is there a different variable ordering that results in a smaller tree? Give the variable ordering that results in the smallest tree (or a small tree). Explain how you determined this was the optimal ordering. (E.g., what was the search strategy through the space of orderings that you used to solve this. A good explanation as to why your ordering is good is more important than the right answer.)

(c) Show how arc consistency can be used to solve this problem. To do this you need to

i) Draw the constraint graph,

ii) Show which elements of a domain are deleted at each step, and which arc is responsible for removing the element.

iii) Show explicitly the constraint graph after arc consistency has stopped.

iv) Show how splitting domains can be used to solve this problem. Include all arc consistency steps.

(d) Show how hill climbing can be used for the problem. Suppose a neighbor is obtained by increasing or decreasing the value of one of the variables by 1, the heuristic function to be maximized is the number of satisfied constraints, and you always choose a neighbor with the maximal heuristic value.

i) Show what happens when we start with the assignment $A = 1, B = 1, C = 1, D = 1, E = 1$.

ii) Show what happens when we start with $A = 3, B = 3, C = 2, D = 1, E = 4$.

iii) Can you think of a better heuristic function? Explain why or why not.

Solution to part (a)

See the web page for the solution and for a java program and a Prolog program that can generate the solution.

Solution to part (b)

The optimal orderings are

```
[ B, C, E, D, A]
[ B, D, E, C, A]
[ C, B, E, D, A]
[ C, E, B, D, A]
[ D, B, E, C, A]
[ D, E, B, C, A]
[ E, C, B, D, A]
[ E, D, B, C, A]
```

There each have 49 failing branches.

Some close orderings are:

```
[ B, C, D, E, A]
[ B, D, C, E, A]
[ C, B, D, E, A]
[ C, E, D, B, A]
[ D, B, C, E, A]
[ D, E, C, B, A]
[ E, C, D, B, A]
[ E, D, C, B, A]
```

There each have 61 failing branches.

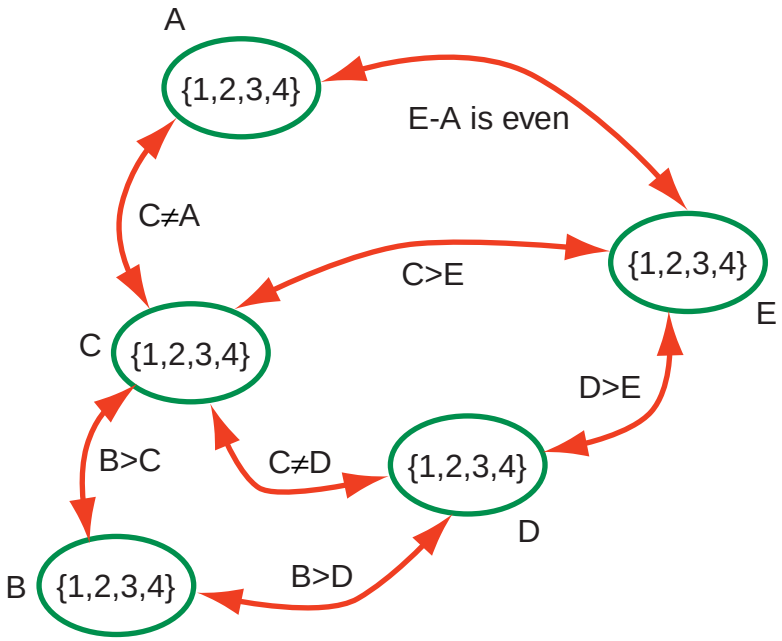
```
[ B, C, E, A, D]
[ C, B, E, A, D]
[ C, E, B, A, D]
[ E, C, B, A, D]
```

These each have 64 failing branches.

So how would one go about arguing for a good ordering? We can look at a strategy that tries to cut out as big a proportion of the branches as possible at each stage. It doesn't matter what we choose first. The second choice should make the biggest pruning which can be done if we choose a node that uses the $>$ constraint. This has cut the possibilities to a few of the pairs. The third constraint then has to prune as much as possible again. The ones that prune the most are those that have constraints with both of the first two variables. The first three variables should either be $\{B, C, D\}$ or $\{C, D, E\}$. Note that if a branch gets pruned on the third step, it doesn't matter what the ordering of the variables is.

Solution to part (c)

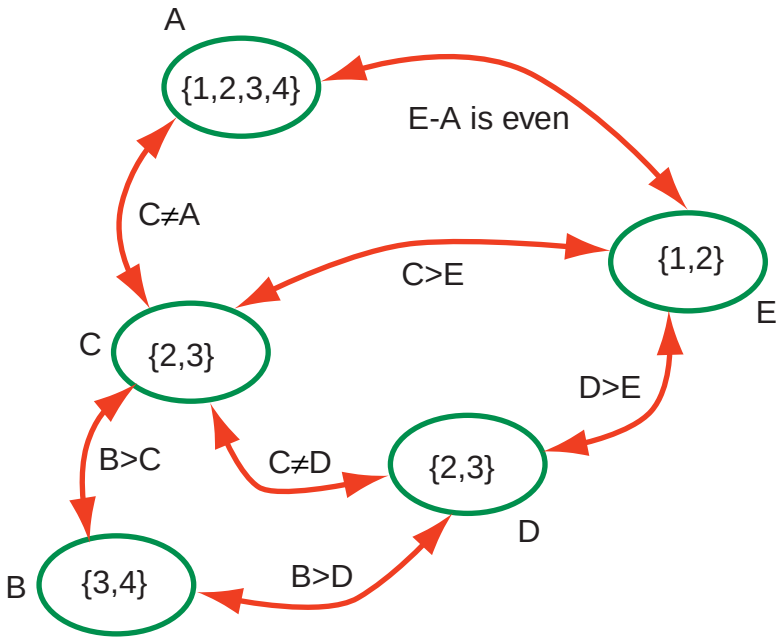
The constraint network is



The following shows a trace of the arc consistency:

```
Arc: <B,C> removes 1 from the domain of B
Arc: <C,B> removes 4 from the domain of C
Arc: <C,E> removes 1 from the domain of C
Arc: <E,C> removes 3 & 4 from the domain of E
Arc: <D,E> removes 1 from the domain of D
Arc: <B,D> removes 2 from the domain of B
Arc: <D,B> removes 4 from the domain of D
```

The constraint network is now arc consistent:



We now have to split a domain. Let's split say the domain of C. We first look at the case with $C = 2$ then with the $C = 3$. That is, we find all of the answers with $C = 2$; then we'll find all of the answers with $C = 3$.

Case I: $C = 2$. We can carry out the following pruning:

```
Arc: <A,C> removes 2 from the domain of A
Arc: <D,C> removes 2 from the domain of D
Arc: <B,D> removes 3 from the domain of B
Arc: <E,C> removes 2 from the domain of E
Arc: <A,C> removes 4 from the domain of A
```

This results in another arc consistent network with

```
domain(A)={1,3}
domain(B)={4}
domain(C)={2}
domain(D)={3}
domain(E)={1}
```

We can split A resulting in two solutions:

```
A=1, B=4, C=2, D=3, E=1
A=3, B=4, C=2, D=3, E=1
```

Case I: $C = 3$. We can carry out the following pruning:

```
Arc: <A,C> removes 3 from the domain of A
Arc: <D,C> removes 3 from the domain of D
Arc: <B,C> removes 3 from the domain of B
Arc: <E,D> removes 2 from the domain of E
Arc: <A,C> removes 2&4 from the domain of A
```

This results in the solution:

```
A=1, B=4, C=3, D=2, E=1
```

Solution to part (c)

Show how hill climbing can be used for the problem. Suppose a neighbor is obtained by increasing or decreasing the value of one of the variables by 1, the heuristic function to be maximized is the number of satisfied constraints, and you always choose a neighbor with the maximal heuristic value.

i) There are many variants. Here is one:

A	B	C	D	E	h
1	1	1	1	1	1
1	1	2	1	1	4
1	2	2	1	1	5
1	3	2	1	1	6
1	3	2	2	1	6
1	3	2	3	1	6
1	4	2	3	1	7

Here is another:

A	B	C	D	E	h
1	1	1	1	1	1
1	1	2	1	1	4
1	2	2	1	1	5
1	2	3	1	1	5
1	3	3	1	1	5
1	3	3	2	1	6
1	4	3	2	1	7

ii) Again there are many variants:

A	B	C	D	E	h
3	3	2	1	4	4
4	3	2	1	4	5
4	4	2	1	4	5
4	3	2	1	4	5

This can meander for a long time, but it never gets out of the local minima.
Here is another run:

A	B	C	D	E	h
3	3	2	1	4	4
3	3	2	1	3	5
3	4	2	1	3	5

Again we can't go anywhere except alternate between the last two assignments. This is a small plateau.

iii) A better heuristic would be to notice that if we have a constraint $X > Y$ that increasing X, even if it doesn't make it greater than Y is a useful move. Hence one heuristic is to count a the achievement of $X > Y$ by 1, but a violation of $X > Y$ by the value $X - Y$ (which could be negative). This would allow us to avoid the second local minima we found.

Another may be to count the inequality and the even/odd constraints as less important on the grounds that they it is often useful to violate them temporarily, and they are easy to achieve (by moving a value by one).