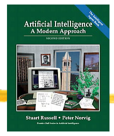
Introduction to Intelligent Systems (co528)

Google "Andy King"

Part	Topic
1	Iterative deepening
2	Puzzles
3	Blind and informed search
4	Minimax and 2-player games
5	Constraint programming

Books and resources



- Search Techniques:
  - Stuart Russell and Peter Norvig, "Artificial Intelligence: A Modern Approach", Prentice Hall, 2002 (£40 used, library)
  - Judea Pearl, "Heuristics: Intelligent Search Strategies for Computer Problem Solving", Addison Wesley, 1984 (library)
  - Nils Nilsson, "Problem Solving Methods in Artificial Intelligence", McGraw-Hill, 1971 (library)
- Constraint Programming:
  - Krzysztof Apt, "Principles of Constraint Programming", Cambridge, 2003 (£32 used, library)
  - Kim Marriott and Peter Stuckey, "Programming with Constraints", MIT Press, 1998 (library)

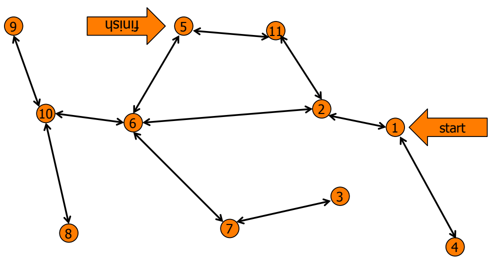
Part I

Route finding and iterative deepening

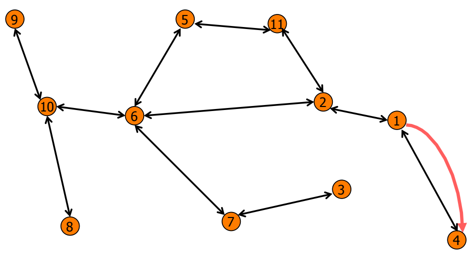
Historical perspective on "Look Ma, no hands" era

- [Newell and Ernest, *IFIP Congress*, 1965] introduced the phrase "heuristic search"
- [Doran and Michie, *Proceedings of the Royal Society of London*, 294, 1966] developed heuristics for the 8-puzzle and the 15-puzzle
- [Hart, Nilsson and Raphael, *Systems Science and Cybernetics*, SSC-4, 1968] developed A* search; [Erratum in *SIGART*, 1972]
- [Haralick and Elliot, *Artificial Intelligence*, 14, 1980] developed heuristics for constraint programming (see survey by Apt)

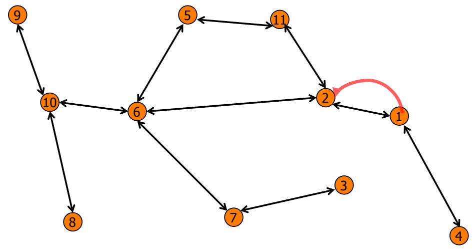
Iterative deepening



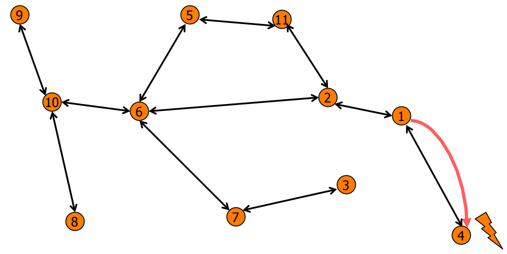
Phase 1 start



Phase 1 finish

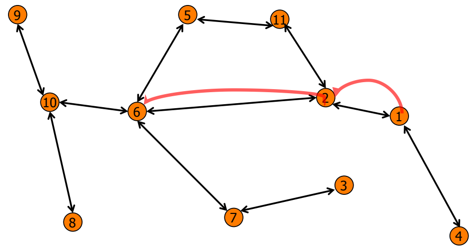


Phase 2 start

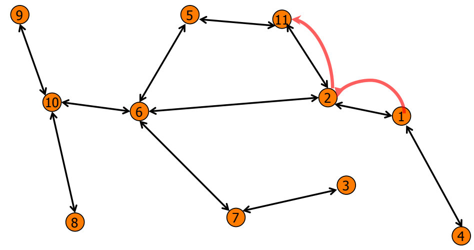


But cannot extend path through 4

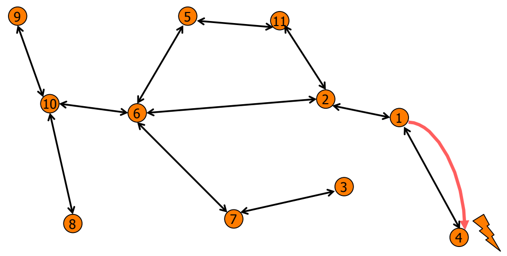
Phase 2 continued



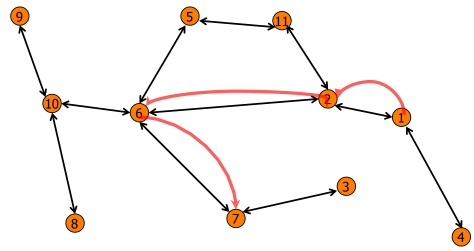
Phase 2 finish



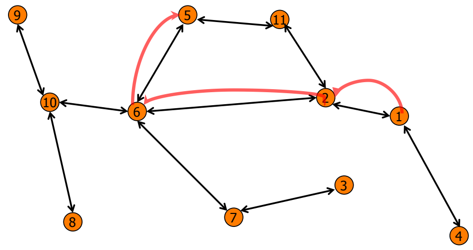
Phase 3 start



Phase 3 continued



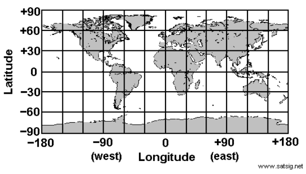
Phase 3 finish



Routefinder for Kent

Latitude and longitude details (towns.xls)

ashford	ash	51.1534	0.8746
barham	bar	51.2081	1.1586
canterbury	cant	51.2783	1.0776
chartham	chart	51.255	1.0205
cranbrook	cran	51.0968	0.5354
deal	deal	51.2251	1.3992
dungeness	dung	50.9164	0.9742
dover	dov	51.1295	1.3089
faversham	fav	51.3168	0.89
folkestone	folk	51.0815	1.166
gillingham	gill	51.3839	0.5609
gravesend	graves	51.4419	0.3707
harrietsham	harr	51.244	0.6673
hastings	hast	50.8539	0.5748
herne_bay	hb	51.3735	1.1257
hythe	hy	51.0726	1.0805
maidstone	maid	51.2751	0.5205
margate	mar	51.3893	1.3874
new_romney	nr	50.9859	0.9397
ramsgate	rams	51.3363	1.4211
rye	rye	50.9498	0.7373
sandwich	sand	51.2771	1.337
sheerness	sheer	51.4421	0.756
sittingbourne	sit	51.3378	0.7321
sturry	st	51.3036	1.1211
tenterden	tent	51.0694	0.6891
whitstable	whit	51.3621	1.0223



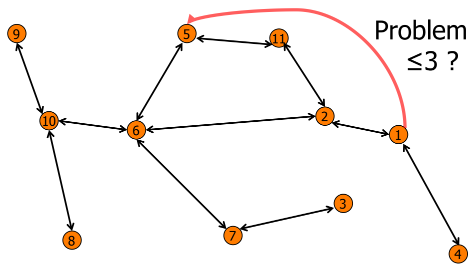
Adjacency list

{ash➔[chart, fav, folk, harr, hy, nr, rye, tent], bar➔[cant, dov, folk], cant➔[bar, chart, fav, sand, st, whit], chart➔[ash, cant, harr], cran➔[hast, maid], deal➔[dov, sand], dov➔[bar, deal, folk, sand], dung➔[], fav➔[ash, cant, whit], folk➔[ash, bar, dov, hy], gill➔[graves, sit], graves➔[gill, maid], harr➔[ash, chart, maid], hast➔[cran, rye, tent], hb➔[mar, st, whit], hy➔[ash, folk, nr], maid➔[cran, graves, harr, sit, tent], mar➔[hb, rams, st], nr➔[ash, hy, rye], rams➔[mar, sand, st], rye➔[ash, hast, nr, tent], sand➔[cant, deal, dov, rams], sheer➔[sit], sit➔[gill, maid, sheer], st➔[cant, hb, mar, rams], tent➔[ash, hast, maid, rye], whit➔[cant, fav, hb]}

Depth-first (limited) search

```
private LinkedList<Town> depthFirst(Town start, Town dest, int depth)
{     if (depth == 0) return null;
      else if (start.equals(dest))
      {
            LinkedList<Town> route = new LinkedList<Town>();
            route.add(dest); // construct singleton route
            return route;    }
      else
      {
            LinkedList<Town> nextTowns = graph.get(start);
            for (Town next:nextTowns)       // search top-down
            {   LinkedList<Town> route = depthFirst(next, dest, depth - 1);
                if (route != null)
                {    route.addFirst(start);
                     return route;    }
            }
            return null;    }
}
```

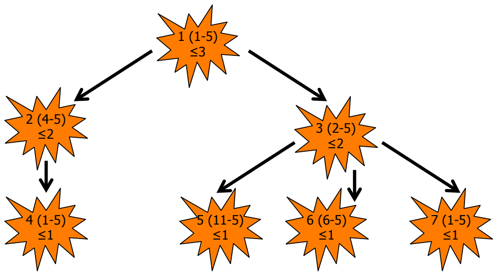
Depth-first (limited to 3)



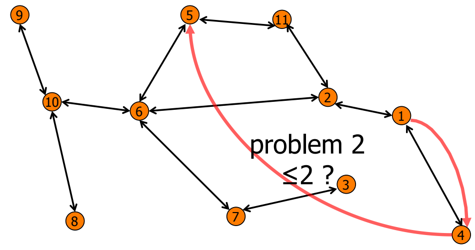
Problem 1

≤3 ?

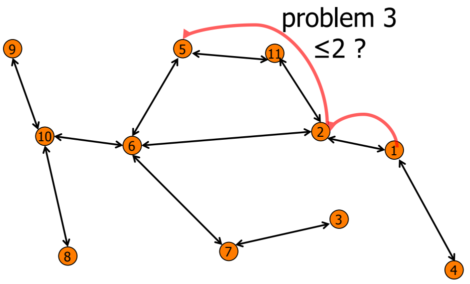
Problem decomposition



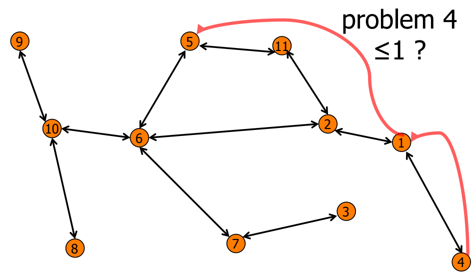
Reducing problem 1 (1-5) to problem 2 (4-5)



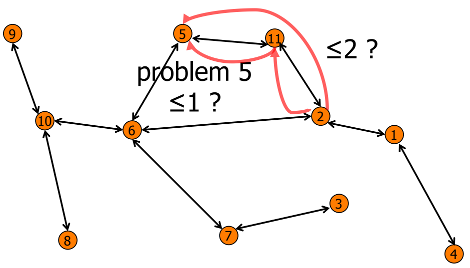
Reducing problem 1 (1-5) to problem 3 (2-5)



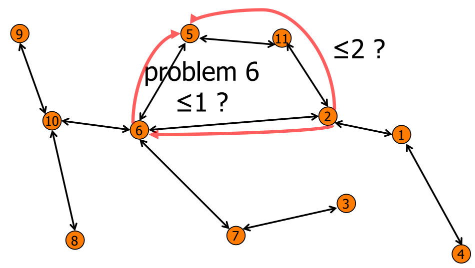
Reducing problem 2 (4-5) to problem 4 (1-5)



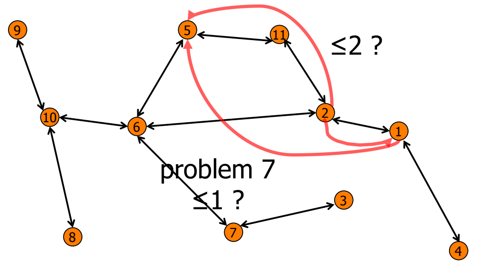
Reducing problem 3 (2-5) to problem 5 (11-5)



Reducing problem 3 (2-5) to problem 6 (6-5)



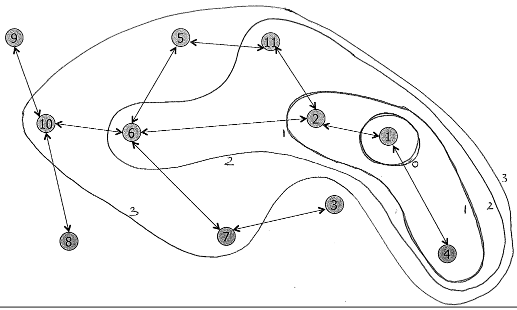
Reducing problem 3 (2-5) to problem 7 (1-5)



Iterative deepening

```
private LinkedList<Town> iterativeDeepening(Town start, Town dest)
{
    for (int depth = 1; true; depth++)        // doubtful termination
    {
        LinkedList<Town> route = depthFirst(start, dest, depth);
        if (route != null) return route;       // fast exit
    }
}
```

Contour map for Iterative deepening



Completeness/incompleteness

- A search algorithm is said to be ***complete*** if, given enough resource, it will either:
  - find a route between two points
  - find that no route exists between two points
- A search algorithm that is not complete is said to be ***incomplete***
- Note that an incomplete algorithm may not terminate for some problems!

Iterative deepening versus depth-first (limited) search

- Iterative deepening is incomplete since:
  - if no route exists, then deepening will continue *ad infinitum*
- Depth-limited search is complete iff the limit is greater or equal to the length of shortest route between any two points:
  - if no route exists, then search will always detect failure (case 2)
  - if a route exists, then a shortest route exists, thus a route exists who length is smaller or equal to the limit, in which case such a route will be found (case 1)

Shortest path cannot exceed the number of configurations

- Recall that depth-first limited search is complete if the depth limit exceeds the length of the shortest route between any two points
- Any shortest path cannot contain two points configurations twice, otherwise the path can be shortened:
  - $[c_1, c_2, c_3, c_4, c_3, c_5] \rightarrow [c_1, c_2, c_3, c_5]$
- Thus the length of a shortest path cannot exceed the total number of ***different*** configurations
- Number of configurations is an upper bound on the length of any shortest path

Optimality/sub-optimality

- A search algorithm is said to be ***optimal*** if, given enough resource, it will:
  - always find a shortest route between two points if a route exists between those points
- A search algorithm that is not optimal is said to be ***sub-optimal***
- Note that there may not be a unique shortest route between two configurations

Iterative deepening versus depth-first (limited) search

- Suppose the optimality criteria is route length
- Iterative deepening is optimal since:
  - it will only consider a route of length k+1 when all routes of length k have already been considered
- Depth-first limited search is sub-optimal because:
  - the only guarantee is that the length of the route will not exceed the limit

Depth-limited without repetition (1 of 2)

```
private LinkedList<Town> depthFirstDevaVu(LinkedList<Town> route,
                                          Town dest, int depth)
{
    if (depth == 0) return null;

    Town last = route.getLast();

    if (last.equals(dest)) return route;
    else
    {
           LinkedList<Town> nextTowns = graph.get(last);
           for (Town next:nextTowns)
                ....

    }
}
```

Depth-limited without repetition (2 of 2)

```
for (Town next:nextTowns)
{
    if (!route.contains(next))
    {
        LinkedList<Town> nextRoute = (LinkedList<Town>) route.clone();
        nextRoute.add(next);
        LinkedList<Town> wholeRoute =
             depthFirstDevaVu(nextRoute, dest, depth - 1);
        if (wholeRoute != null) return wholeRoute;
    }
}
```