AI Programming (CSC416) - Textbook Assignment #2

Dan Wysocki

Chapter 2 deals with uninformed search. Uninformed search is a broad term which defines a process which seeks to find a specific element in an unordered collection. The most primitive form of uninformed search is the brute force method, in which every possibility is considered. Other methods seek to make sense of what is being searched, and impose some sort of order upon the collection while searching.

Questions for discussion

1. **Why is search an important component of an AI system?**

   If an AI system was incapable of search, then it would not be able to make decisions based on existing knowledge, and therefore would not be able to build upon that existing knowledge. If an AI system were capable of search, but all of its data were unordered, such that it had to potentially search through every element in its memory before it could find a piece of information (or find that it was never in its memory), the more memories it accumulated, the slower it would become. Even with organized memory, increasing its quantity would slow down search, so just imagine what would happen if it was not organized. The more efficient the organization and search algorithms an AI system (or any computer system) are, the faster it can make a decision.

2. **What is a state-space graph?**

   To quote the text: "A state-space [graph] for a problem contains all states that a problem could be in, as well as all possible transitions between these states." These are useful, because they allow one to rule out certain states based on which transitions have occured. The text uses the False Coin Problem to demonstrate its usefulness, showing that if presented with a set of 6 coins in which only 1 has a different weight, it is possible to determine which coin is different through only 3 weighings. In this case, the results of the weighings are what determine the transitions.

3. **Describe the generate-and-test paradigm.**

   The generate-and-test paradigm involves two parts, which may occur in separate phases, or may occur in lockstep. The first is the generation phase, in which a set of possible solutions to a problem are proposed. The second is the test phase, in which each proposed solution is either accepted or rejected based on whether it fits within the constraints of the problem.

4. **What properties should a generator posess?**

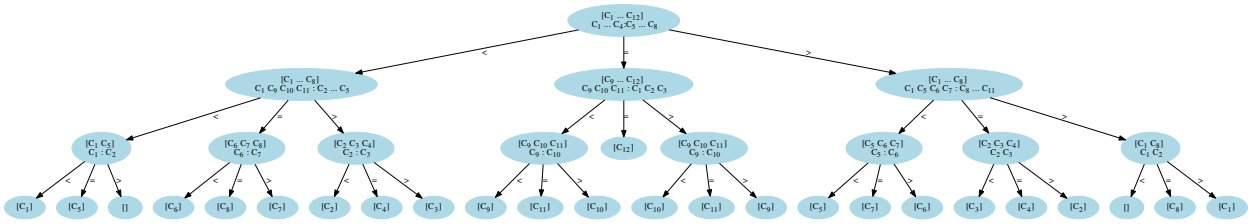
   A reliable generator must propose every possible solution to the problem it hopes to solve. However, being reliable is just the bare minimum requirement to be a good generator. A good generator should also be nonredundant. The more informed a generator is, the more efficient it is in finding solutions, because it allows it to limit the solutions it proposes, and therefore has fewer false proposals to test.

5. **How does backtracking improve on exhaustive enumeration?**

   The advantage of backtracking over exhaustive enumeration is that once it reaches a state which does not satisfy the constraints of the problem, it *backtracks* to the previous state, and continues from there. Exhaustive enumeration does no such thing, and will in fact make every possible transition from a state which has already failed.

Exercises

1. Solve the False Coin Problem for 12 coins. Only three combinations of coins are permitted to be weighed. Recall that a balance scale returns one of three results: equal, left side is lighter, or left side is heavier.



4. **Another generator for the n-Queens Problem is: Place a Queen in row 1. Do not place the second Queen in any square that is attacked by the first Queen. In state i, place a Queen in column i in a square that is not under attack from any of the previous $i-1$ Queens.**

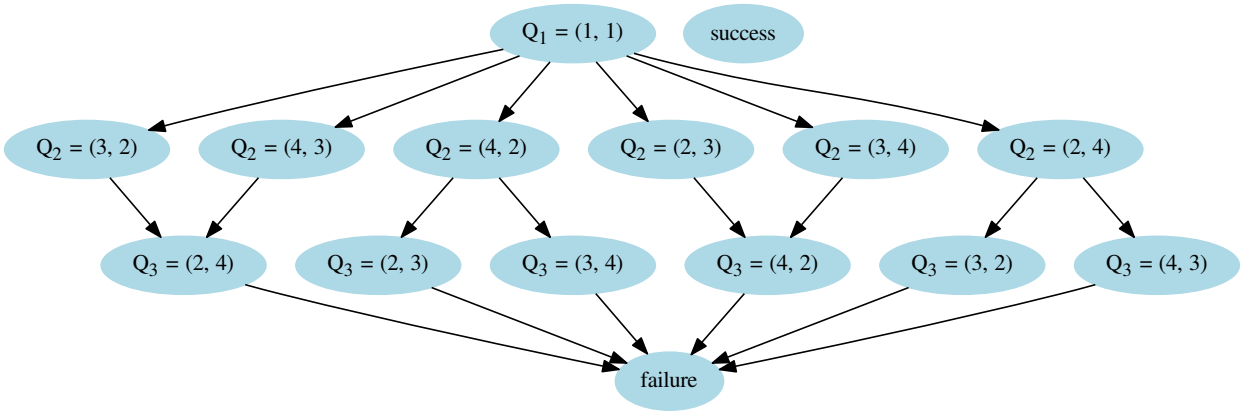
   a. **Solve the 4-Queens problem using this generator.**

   The 4-Queens problem cannot be solved using this generator. As shown in the text, if a Queen is placed in (row 1, column 1), there is no solution to the problem.

   b. **Argue that this generator is more informed than either of the two generators used in the text.**

   Despite being an unreliable generator, posessing no solution, it is still more informed than the generators proposed in the book, as it rules out row 1, column 1, and the diagonal stretching from the bottom-left to the top-right corners, instead of just (row 1, column 1).

   c. **Draw the portion of the search tree expanded in the search for a first solution.**



5. **Consider the following generator for the 4-Queens problem: for $i = 1$ to 4, randomly assign Queen i to a row. Is this generator complete? Is it nonredundant? Explain your answer.**

   This generator is complete, as it can assign a Queen to any row (and presumably any column), as it is completely random and unrestricting. In fact, there really are no constraints on the placement of the Queens, and so this is actually exhaustive, and in fact non-terminating. As such, it is redundant, as there was no constraint that a state cannot be repeated.

6. **A number is said to be perfect if it equals the sum of its divisors (excluding itself). For example, 6 is perfect because $6 = 1 + 2 + 3$, where each of the integers 1, 2, and 3 are divisors of 6. Give the most informed generator that you can think of to find all perfect numbers between 1 and 100, inclusive.**

   The Euclid–Euler theorem states that every even perfect number can be represented by the form $2^{n-1}(2^n - 1)$, where $2^n - 1$ is a prime number, which means n is a prime number. No odd perfect number has ever been discovered, so if one exists, it certainly does not lie within 1 and 100, and thus

this definition of a perfect number can be utilized. If we take only the set of the first 3 primes: $\{2, 3, 5\}$. This is enough to obtain the first three perfect numbers, the 3rd of which exceeds 100, and the rest of which are within the range, therefore it gives the only two perfect numbers between 1 and 100: $\{6, 28\}$, with 496 as the next perfect number.

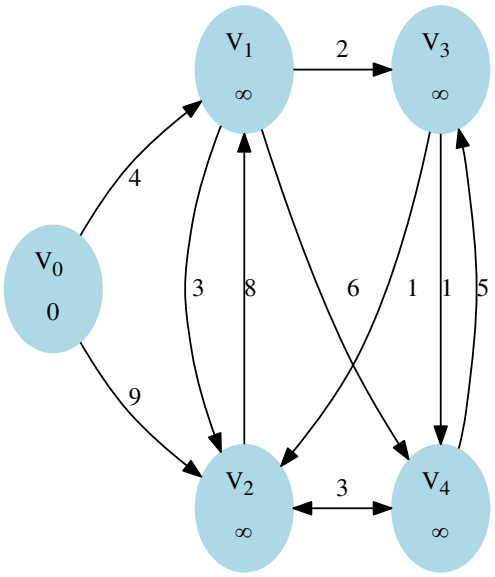
7. Use Dijkstra's Algorithm to find the shortest path from the source vertex V_0 to all other vertices in Figure 2.35.

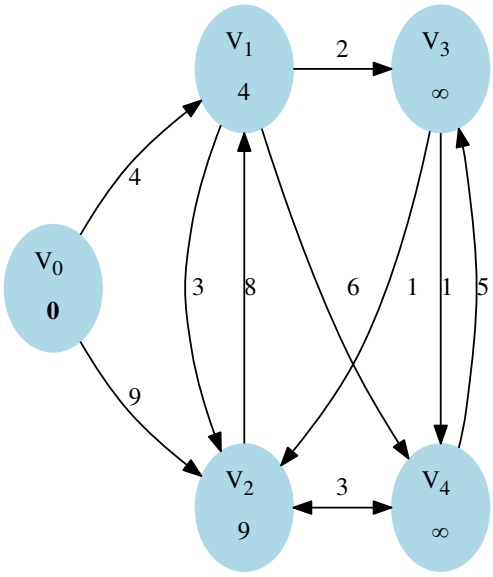
   Using Dijkstra's Algorithm, the shortest paths to each vertex are found to be the following:

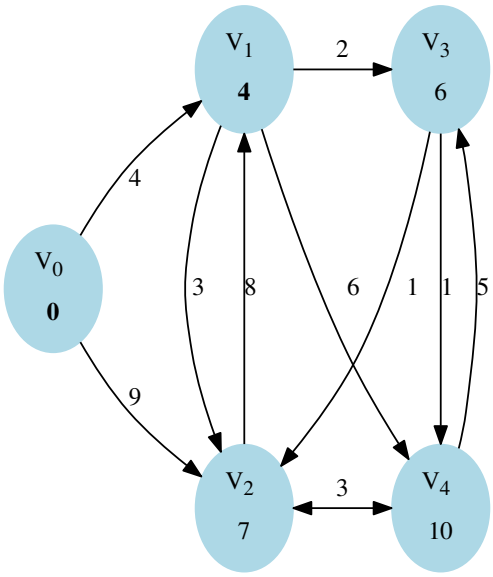
Vertex	Path	Cost
V_0	V_0	0
V_1	$V_0 \to V_1$	4
V_2	$V_0 \to V_1 \to V_2$	7
V_3	$V_0 \to V_1 \to V_3$	6
V_4	$V_0 \to V_1 \to V_3 \to V_4$	7

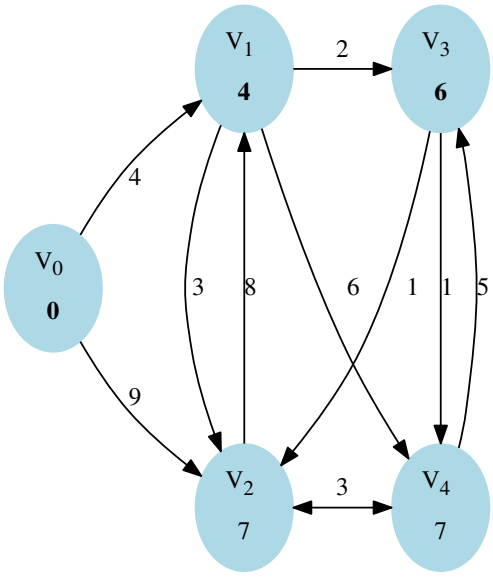
   The steps taken are illustrated in the following graphs. The graphs obey the following notation:

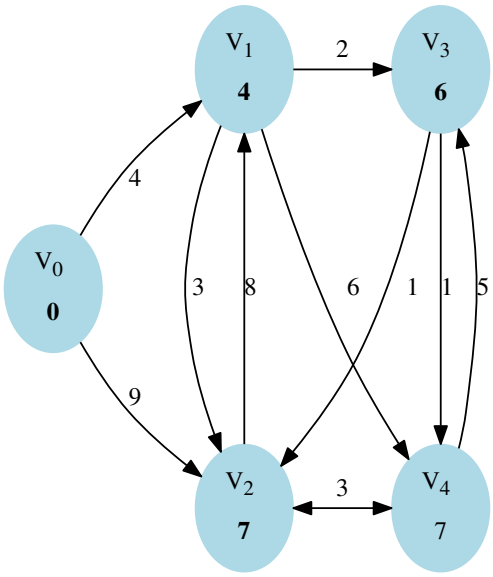
   - The distance from V_0 is denoted by a number below the label of the vertex.
   - If the node has been visited, its distance is written in boldface.

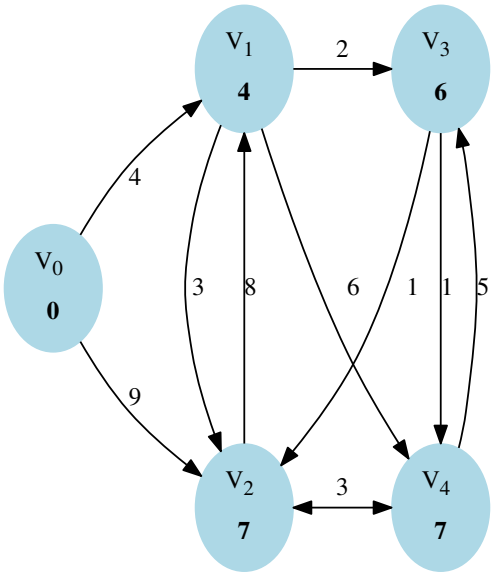



V1
4
2
V3
∞
4
V0
0
3
8
6
1
1
5
9
V2
9
3
V4
∞


V1
4
2
V3
6
4
V0
0
3
8
6
1
1
5
9
V2
7
3
V4
10


V1
4
2
V3
6
4
V0
0
3
8
6
1
1
5
9
V2
7
3
V4
7


V1
4
2
V3
6
4
V0
0
3
8
6
1
1
5
9
V2
7
3
V4
7


V1
4
2
V3
6
4
V0
0
3
8
6
1
1
5
9
V2
7
3
V4
7