1)

c.
Grumble does not establish conclusively whether the problem is NP-complete. He would need to reduce a known NP-complete problem to the Balrog problem.

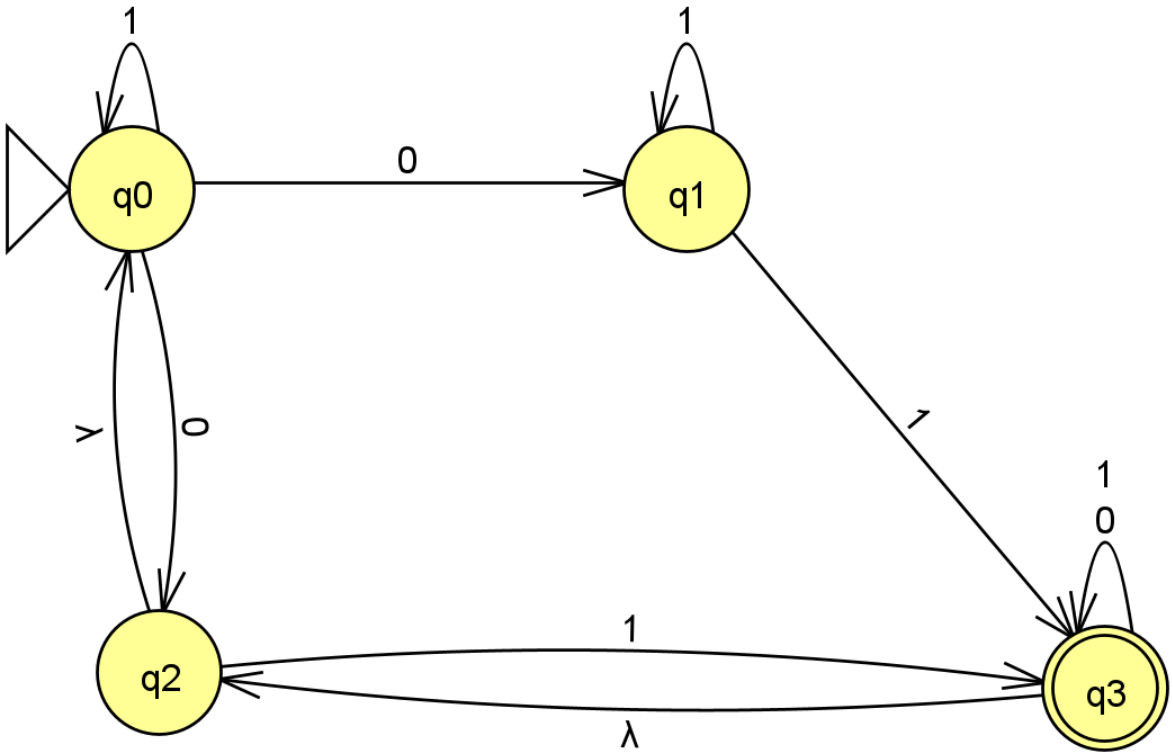
d.

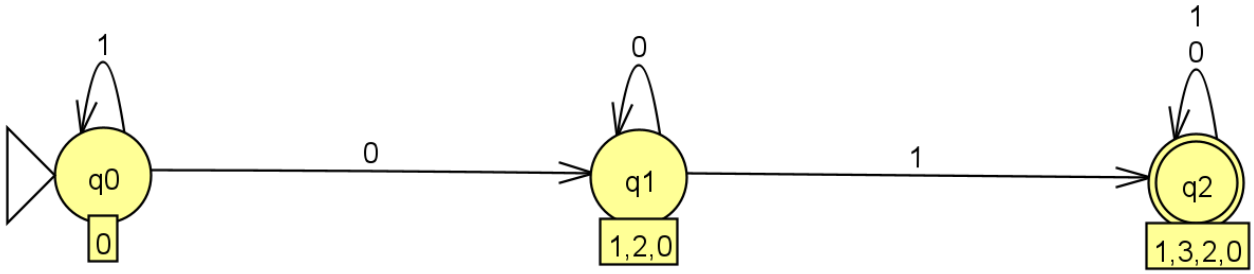
“Algorithms are an expression of knowledge that makes computers work. In order to solve a particular computational problem, such as finding the factors of a given number, or sorting a list of students by their student number, an algorithm must be developed and implemented. It comes as a great surprise to many people that there are some problems for which there is no algorithm possible for all inputs. These are known as undecidable problems and include the Halting problem for Turing machines, the **Hamiltonian Circuit problem**, the busy beaver problem, the **tile problem** and the blank tape problem. Alan Turing was the first to show there was an undecidable problem, and since that time many other problems have been shown also to be undecidable. This is done by reducing a problem (say A) with unknown status to a problem known to be intractable (say B). From this reduction it is simple to show that problem A must be intractable using the **Pumping Lemma**. The existence of undecidable problems means that certain properties of programs cannot be guaranteed. The Church-Turing thesis, which states that any real-world computation can be performed by a Turing machine, indicates that we cannot avoid undecidable problems by changing technologies. **Undecidable problems** are also an issue for nondeterministic pushdown automata and nondeterministic finite state automata, as nondeterminism introduces computations based on chance. Unrestricted grammars do not have any issues with undecidable problems, **as there are by definition unrestricted**. Note though that NP-complete and other intractable problems are all decidable.”

Maybe it’s not really that surprising?
Hamiltonian cycle is NP-complete but not undecidable.
Tile problem- decidable and brute forceable
Technically others have come before him?
Proof by contradiction not Pumping Lemma
Pushdown and NFA can be reduced to DFA
Not because of them being unrestricted, but on the basis of grammars being resolvable by automata

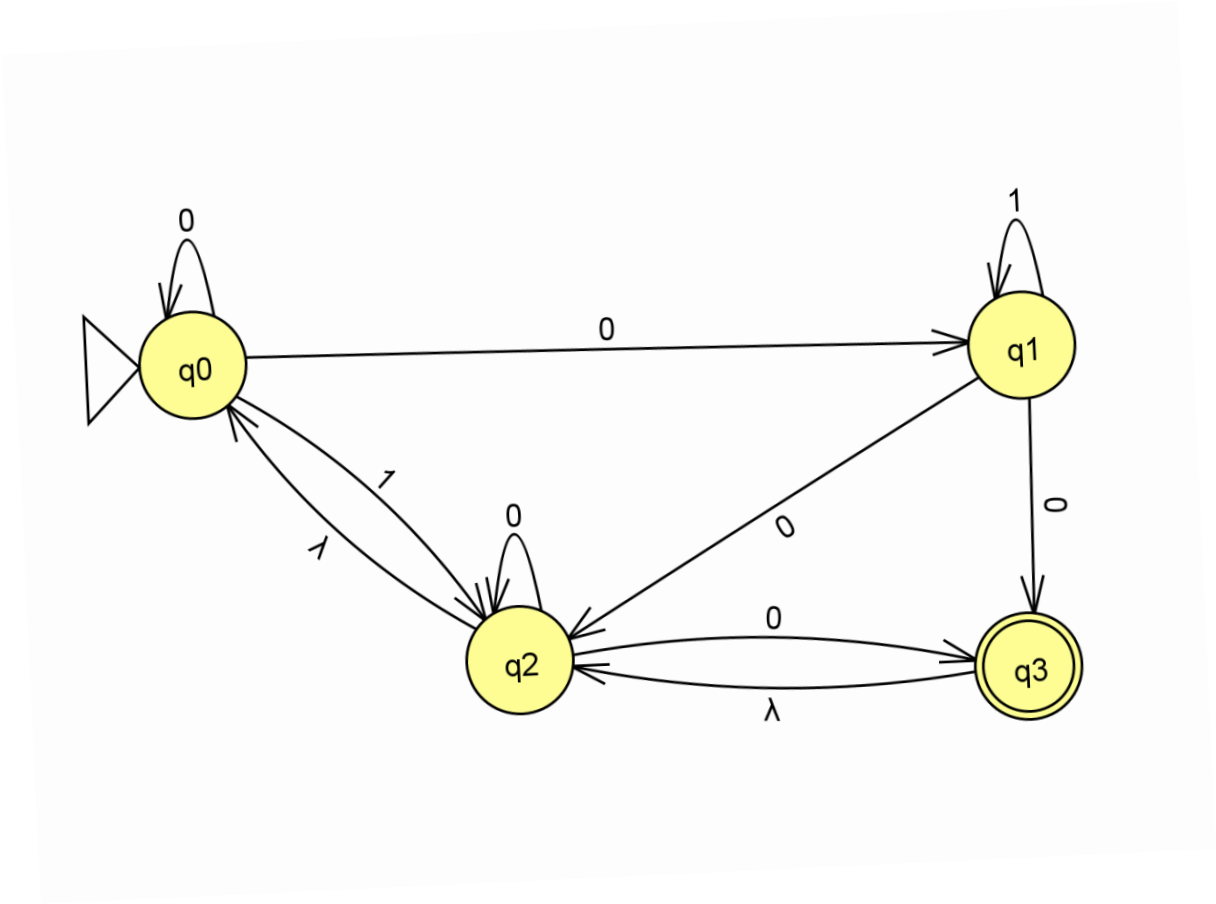
2)

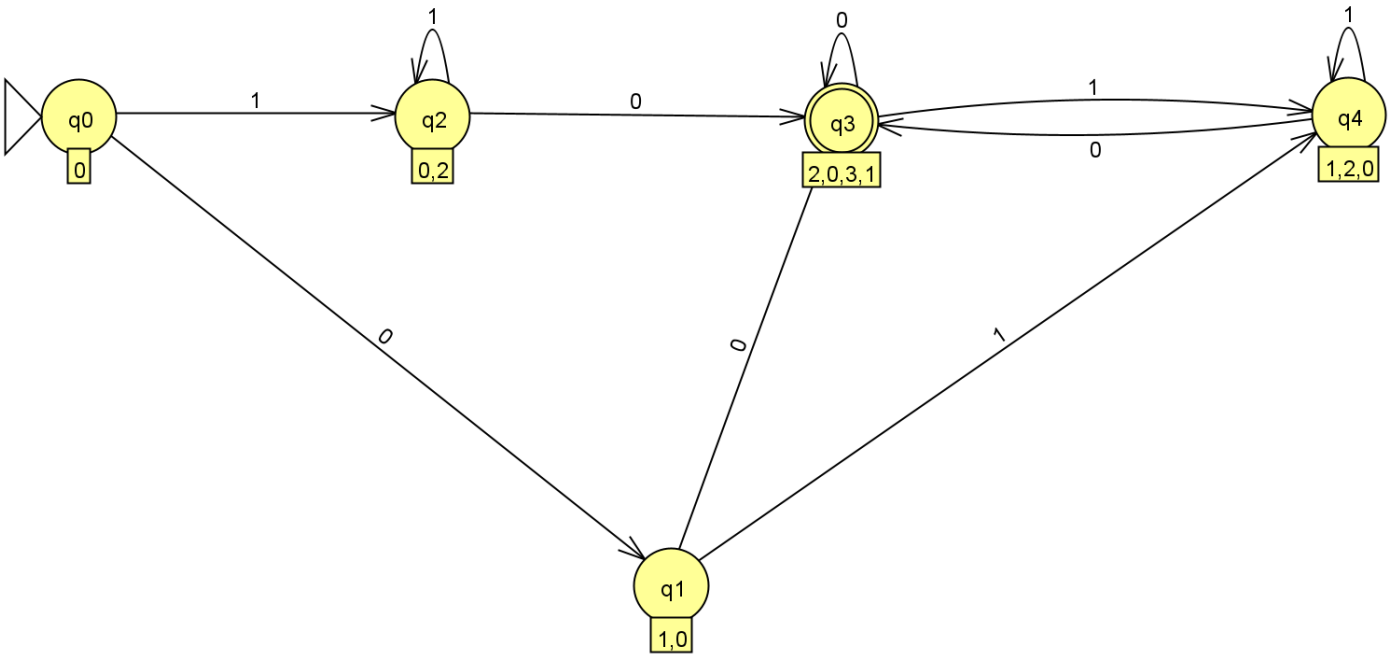
a. I added as many transitions as possible to cover all the cases as this reduces uncertainty and variance, then deleted transitions as required.



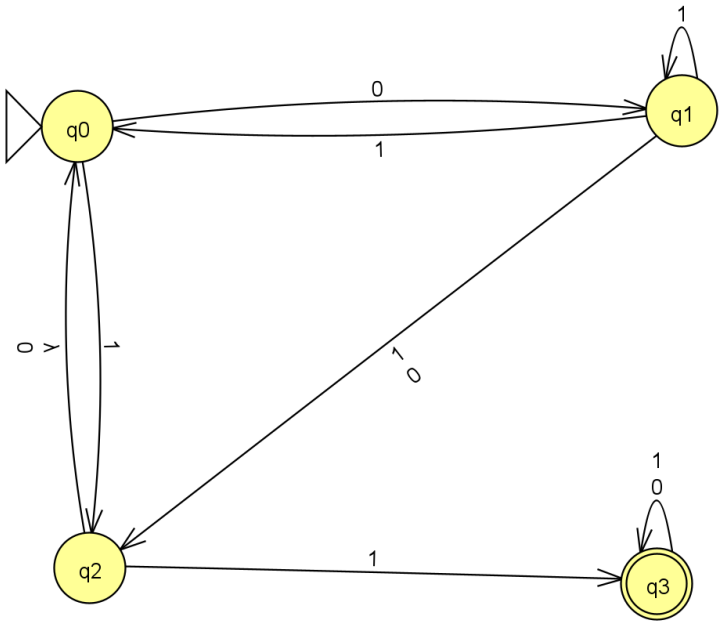


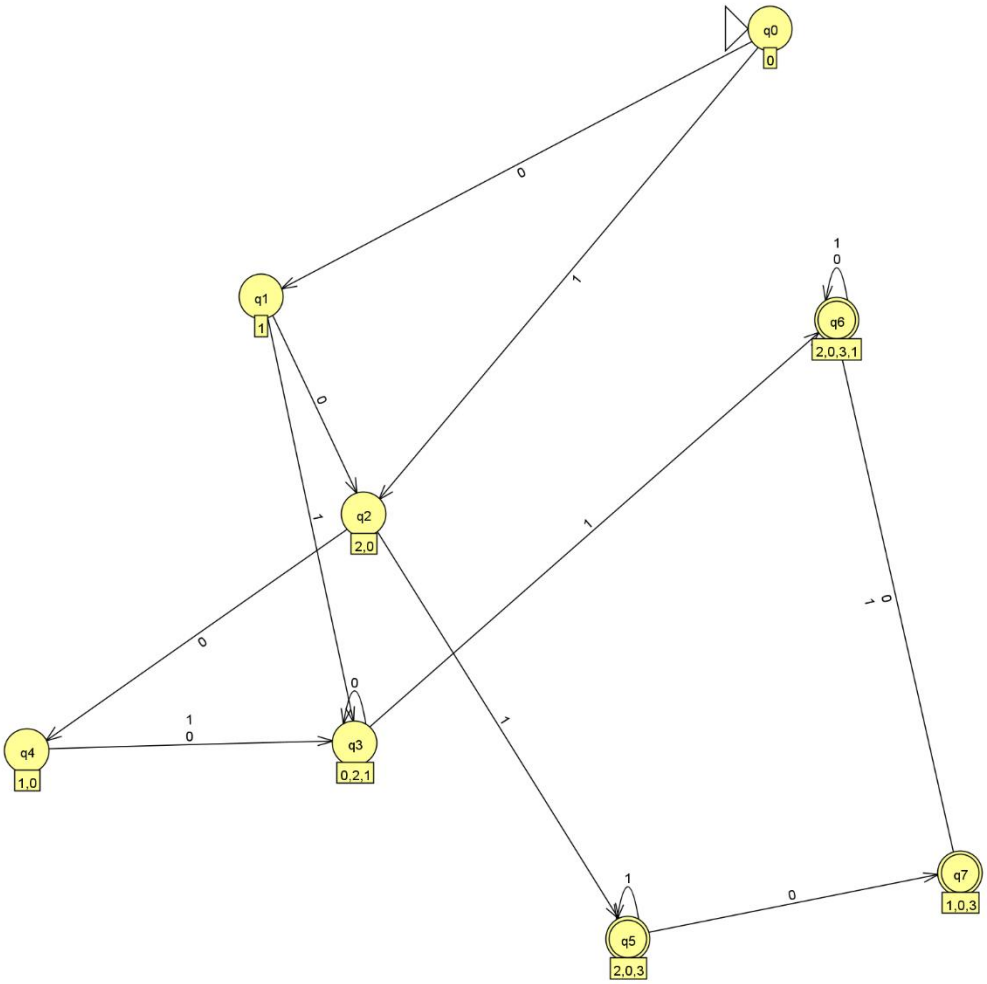
b. I created this by paring down some of the cases in the original and adding more ways to reach the end state.





c. By this point I was really brute forcing it, eight states with a removed lambda was the most I could get.





3)

a.

Assume that L_1 is regular.

According to the Pumping Lemma, there exists an $n \geq 1$ such that for all strings $w \in L$, where $|w| \geq n$, we have $w = xyz$ where $|xy| \leq n$, $y \neq \lambda$ and $xy^iz \in L$ for all $i \geq 0$.
Take the string belonging to L_1 as w. According to the pumping lemma, there exists an |xyz| for all $w \in$ L.
If $|xy| \leq n$, |xy| must contain only *1*s as it is the first part of the string. For x, this amount will be $n > k > 1$, where k is the length of y. The number of *1*s will be *n*, where the length of *x* can be represented as $n - k - r$, the length of *y* as *k* and the remaining amount of *1*s as *r*. Take $i = 3$ as our value of *i*; according to the Pumping Lemma, $xy^3z \in L_1$, as xy^iz must belong to L_1.
The total number of *1*s in the first section of the string is *n-k-r + 3k+r = n+2k*. This string then takes the form of $L = 1^{n+2k}2^n3^{2n}1^n2^n$. $L \notin L_1$.

This is a contradiction, so $L_1$is not regular.

b.

Assume that L_1 is regular.

According to the Pumping Lemma, there exists an $n \geq 1$ such that for all strings $w \in L$, where $|w| \geq n$, we have $w = xyz$ where $|xy| \leq n$, $y \neq \lambda$ and $xy^iz \in L$ for all $i \geq 0$.
Take the string belonging to L_1 as w. According to the pumping lemma, there exists an |xyz| for all $w \in$ L.
If $|xy| \leq n$, |xy| must contain only 1s as it is the first part of the string. For x, this amount will be $n > k > 1$, where k is the length of y. The number of 1's will be *n*, where the length of *x* can be represented as $n - k - r$, the length of *y* as *k* and the remainder as *r*. Take the string belonging to L_1 as 1^n3^{2n}. Take $i = 0$ as our value of *i*; according to the Pumping Lemma, $xy^0z \in L_1$, as xy^iz must belong to L_1. The total number of *1*s in the first section of the string is *n-k-r + r = n-k*.
This string then takes the form of $L = 1^{n-k}2^n3^{2n}1^n2^n$. $L \notin L_1$.

This is a contradiction, so L_1 is not regular.

c.

Claim: The language $L_2 = \{1^i 2^i \# 3^{2i} @ 2^i 1^i\}$ is not context-free.

Proof: Assume L_2 is context-free. Then the Pumping Lemma applies and so for some $n \geq 1$ such that for all $w \in L$ such that $|w| \geq n$, $w = xyzuv$ where

- $|yzu| \leq n$
- $y \neq \lambda$ or $u \neq \lambda$
- $xy^i zu^i v \in L$ for all $i \geq 0$

Choose $w = 1^n 2^n \# 3^{2n} @ 2^n 1^n$ and so $w \in L$ and $|w| \geq n$. So by the Pumping Lemma $w = xyzuv = 1^n 2^n \# 3^{2n} @ 2^n 1^n$ and $|yzu| \leq n$. This means that y and u can contain either 1 & 2 or 2 & 3. We will refer to the part of the string before # as *zone 1*, the part of the string between # and @ as *zone 2*, and the part of the string after @ as *zone 3*.

Now if y or u contains # or @, then clearly $xy^2 zu^2 v \notin L$, as this would contain two occurrences of either # or @.

Hence neither y nor u contains either # or @.

Now note that both y and u can only cover either zone 1 and zone 2, or zone 2 and zone 3, as $|yzu| \leq n$. This means that $xy^2 zu^2 v \notin L$, as one of the zones will have the same string as w, but the other two will have longer strings, and hence $xy^2 zu^2 v$ is not of the form $1^i 2^i \# 3^{2i} @ 2^i 1^i$.

This is a contradiction, and so L_2 is not context-free.

d.

Assume that L_1 is regular.

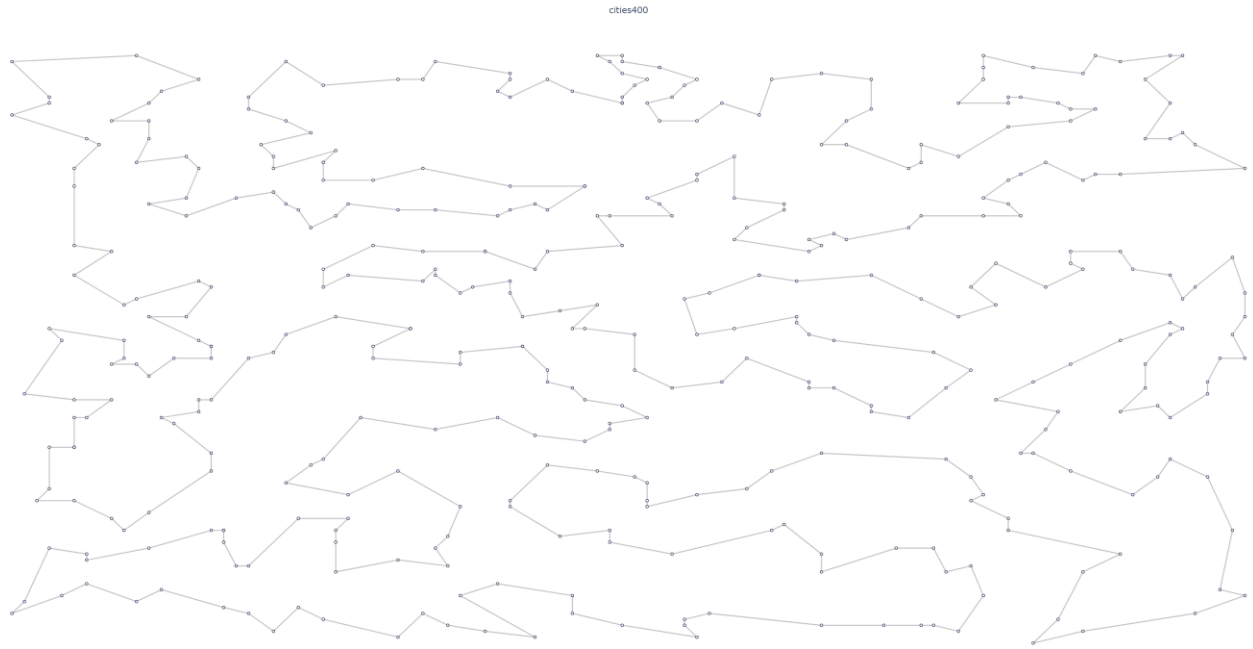
According to the Pumping Lemma, there exists an $n \geq 1$ such that for all strings $w \in L$,$|w| \geq n$, we have $w = xyz$ where $|xy| \leq n$, $y \neq \lambda$ and $xy^i z \in L$ for all $i \geq 0$.
Take a w with length $n + k$ (as w can be larger than n). Now, by the Pumping Lemma, w can be divided into strings xyz where $y \geq 0$ and $xy^i z \in L_1, i \geq 0$. Choose $i = 2$, so the length of w is $|w| + 1$. The length of the new word is the n required by the question, because for every word u in the language L where $u \geq |w|$, the word w is the string $|x| < n$ as required.

4. The Travelling Salesperson Problem (TSP, often known as the Travelling Salesman Problem) is a well-known example of an intractable problem. Compare the performance of a complete solution to this problem (i.e. one that uses an algorithm that is guaranteed to always find a minimum) with two more efficient algorithms, which could be either approximation algorithms, heuristics or other means. Your comparison should show the time taken in seconds for all three algorithms to some value of n, which should correspond to your interpretation of some reasonable maximum time for the complete algorithm (say 30 minutes). Whatever your maximum time is, determine the largest value of n which can be solved in this time for each of the more efficient solutions. You should also describe what input data was used and how you obtained it. Some possible efficient algorithms include the Christofides or Christofides-Serdyukov algorithm, the double-tree algorithm, greedy algorithms, the Lin-Kernighan heuristic, and genetic algorithms.

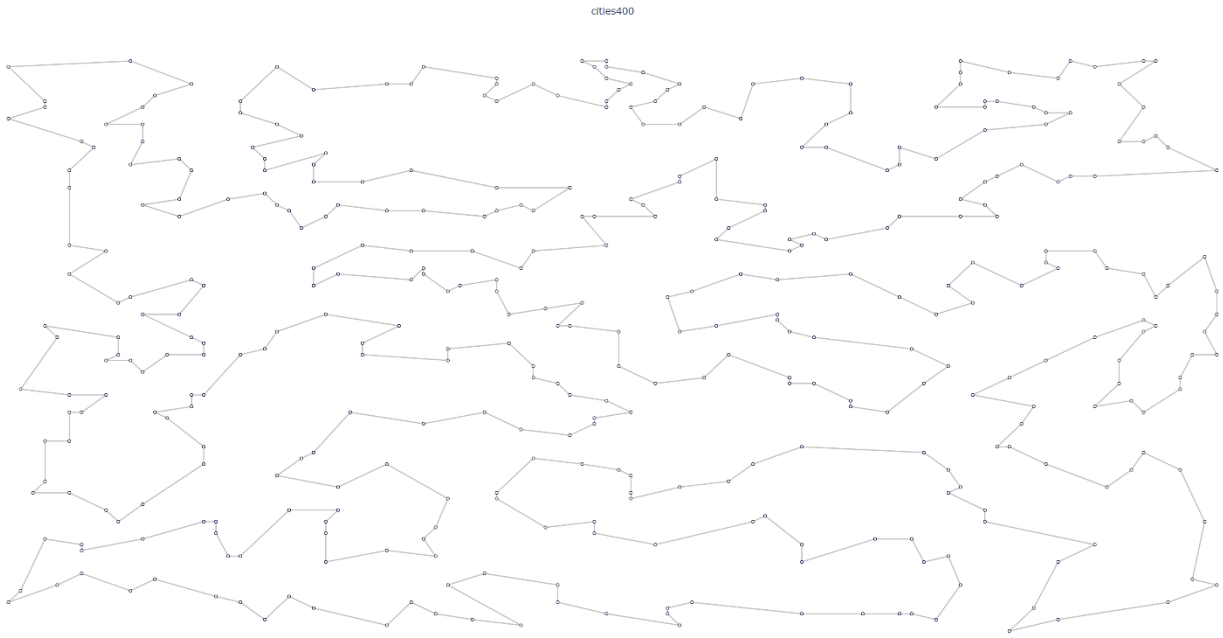
Time to calculate most efficient path for 256 cities (presented as an array). The data was given to the algorithms in the form of a .tsp file, which is a two-dimensional array representing a graph of cities. A graph was plotted based on the results for each algorithm which showed that the solution was complete. The same file was used each run, and was based on a sample of 1024 cities. The code used in the experiment is available here and here.

Algorithm	Time Taken to solve 11 cities (in seconds)	Largest Value of *n* in 4 minutes:
Naïve implementation (brute force)	240 seconds (benchmark)	11
Lin-Kerninghan Heuristic	10 seconds	400
Improved Lin-Kerninghan	< 5 seconds	400

The concept of a naïve implementation for the travelling salesman problem is this- it is simple to program an implementation but hard to run through, as the computational power needed to calculate a minimum for n increases in accordance with a sort of nested factorial- hence the problem is intractable. For consistency, the same n was used in each algorithm. The brute force algorithm was quickly overwhelmed with any number larger than 11, and recording the time taken became unfeasible. The relationship between the size of n and the time taken for the algorithm is not linear. Because of the relationship between n and time likely being exponential, an appropriate value of n was found through selection between upper and lower bounds starting from 1024 down. Both of the more efficient algorithms were capable of solving values of n several hundred times larger than 11 in four minutes, with the Lin-Kerninghan heuristic solving 400 cities in 240 seconds and the improved Lin-Kerninghan solving 400 cities in 240 seconds. The heuristics work remarkably faster than the precise but naïve implementation. Brute force by design checks every single Hamiltonian cycle- therefore in case it finishes in a reasonable time, the solution is guaranteed to be optimal shortest cycle. Practicality restricts the use of this or similar approach. From what is known about complexity, even quantum computing will not be fast enough to resolve this problem, which is NP-complete. The heuristic is almost guaranteed to be sub-optimal.



Lin Kernighan heuristic for 400 cities



Improved Lin-Kerninghan Heuristic implementation for 400 cities

5. I used variations on prompts, making sure to follow the rules. Giving it too many words would confuse the AI, but I also couldn't be too vague. If you input too many words, it generally chose keywords. I found it didn't have a good idea of what a platypus looked like- it always looked more like a nutria. I suspect this is because a platypus is relatively obscure compared to animals like cats and dogs, especially in terms of visual data to model off of. Also, I could confuse it- e.g. asking for a *platypus playing a game against a koala* would generate a platypus and a platypus-derivative type creature, or a hybrid of the two, but not both animals accurately, while a choice of either would yield more accurate results. It generally doesn't interpret *meaning* but reads keywords- for example, asking for 'platypus playing a game of chess against a robot' generated a robot platypus rather than a robot and a platypus. I went with *platypus playing a game of chess* because chess is such a visually distinctive game. The cartoonish style was my favourite as some of the animals it generated were rather uncanny.

6)

While researching ways to expand on my earlier work, I came across many examples of generative art and cellular automata. My original work was basically a straight implementation of the algorithm, overlaying instances of ants with other instances, experimenting with different patterns and some video editing to make the response visually interesting. My reworked implementation makes edits to the code to influence the behaviour of the automaton and see how it would react to boundaries.

Some of the most interesting examples I found were patterns generated by multiple ants and patterns generated when the ants were ‘competing’ for resources. While this is more of a visual art approach to the concept of cellular automata, it was still interesting to see. If I were to start over from scratch, I would edit the ruleset of the algorithm immediately because this gives the most interesting results. I would probably add different ant that follow their own sets of rules and are capable of interacting with one another.

The original code was edited to “throw” the ants out of the boundaries of an exclusion zone. Some patterns displayed interesting behaviour with the addition of the exclusion zone, behaving similarly to a space-filling algorithm. Generally any implementation of Langton’s Ant will fill up the entire space if left running for long enough. It also appeared to speed up with the loss of space on the torus. I modified the original code to move the ants out of the zone if a cell would ‘land’ in the zone. One zone was rectangular, and one zone was circular. Interestingly, the circular exclusion zone was more of a challenge for the automaton to build around as it built far slower, while the rectangular exclusion zone was rather quickly avoided.

What I learned from this task and really the whole assignment is that relatively simple rules or numbers can generate complex systems with emergent behaviour in unexpected ways. The common thread in this assignment and this subject in general is that systems with deceptively simple rules (automata, platypus game, cellular automata, algorithms) are very complex with non-obvious solutions. Small changes or increases by one can result in staggering complexity, and to call its rate of growth ‘fast’ would be an understatement. With regards to analysis or construction, these systems must be treated with respect, and in general computing has only scratched the surface of what is possible.

See the video here

7)

a.

Class	Number	Percentage
Reachable	1511	75.55%
Unreachable	177	8.85%
None	312	15.6%

Tournament	Time Taken	Wins	Draws	Winless Machines
Reachable only	44 minutes	1130150	870850	None
All 2000 machines	108 minutes	1979456	21541	None
None + unreachable	10 minutes	117698	1883302	None
2000 machines + extra	109 minutes	2005978	15077	None

Reachable (cell = machines that were multiple top ten winners)

Number	Wins	For	Against	Ratio
2878152	1340	86465	14403	6.003263
39736309	1326	84948	18926	4.488429
111035385	1242	82960	18535	4.475856
10420116	1331	85565	19341	4.424022
2082194	1323	85297	19601	4.351666
220129174	1324	86825	20359	4.264699
141949606	1311	83568	20815	4.014797
34991586	1335	87475	21806	4.011511
140248029	1327	87532	23151	3.780917
194848664	1351	88988	23902	3.723036

All reachable machines- top ten

(2878152,[t(y,k,y,e,wa),t(g,k,y,e,gg),t(y,e,y,p,gg),t(g,e,y,p,gg),t(y,w,y,e,wa),t(g,w,y,e,wa),t(y,p,g,e,wa)]).

(39736309,[t(y,k,y,p,wa),t(g,k,y,p,wa),t(y,e,g,k,wa),t(g,e,y,w,gg),t(y,w,y,w,wa),t(g,w,g,e,gg),t(y,p,y,p,gg)])

(111035385,[t(y,k,y,p,wa),t(g,k,g,p,wa),t(y,e,g,k,wa),t(g,e,y,w,wa),t(y,w,g,k,gg),t(g,w,y,e,gg),t(y,p,y,p,gg)])

(10420116,[t(y,k,y,p,gg),t(g,k,y,p,gg),t(y,e,y,e,wa),t(g,e,y,p,gg),t(y,w,g,p,gg),t(g,w,y,p,wa),t(y,p,y,p,wa)])

(2082194,[t(y,k,y,p,gg),t(g,k,y,p,gg),t(y,e,y,p,wa),t(g,e,y,w,gg),t(y,w,y,k,gg),t(g,w,y,w,gg),t(y,p,y,p,wa)]).

(220129174,[t(y,k,g,p,gg),t(g,k,g,p,gg),t(y,e,y,e,wa),t(g,e,g,p,gg),t(y,w,y,k,wa),t(g,w,y,p,gg),t(y,p,y,p,wa)]).

(141949606,[t(y,k,g,w,gg),t(g,k,y,w,wa),t(y,e,y,p,gg),t(g,e,y,p,gg),t(y,w,y,p,wa),t(g,w,g,e,gg),t(y,p,g,e,wa)]).

(34991586,[t(y,k,y,w,wa),t(g,k,y,w,wa),t(y,e,g,p,gg),t(g,e,y,p,gg),t(y,w,y,e,gg),t(g,w,y,w,gg),t(y,p,y,p,wa)]).

(140248029,[t(y,k,g,p,wa),t(g,k,y,w,gg),t(y,e,y,k,wa),t(g,e,y,k,wa),t(y,w,y,k,wa),t(g,w,g,e,gg),t(y,p,y,p,gg)]).

(194848664,[t(y,k,g,p,gg),t(g,k,y,w,gg),t(y,e,g,w,wa),t(g,e,g,e,gg),t(y,w,g,k,wa),t(g,w,y,p,wa),t(y,p,y,p,wa)]).

All 2000 machines (cell = machines that were multiple top ten winners)

Number	Wins	For	Against	Ratio
2878152	1824	116626	17077	6.82942
111035385	1736	112725	21673	5.201172
39736309	1812	115215	22837	5.045102
10420116	1813	115343	22992	5.016658
2082194	1814	115406	23244	4.96498
141949606	1812	114802	24136	4.756463
220129174	1807	116890	24908	4.69287
211278245	1778	111847	24767	4.515969
34991586	1823	118448	26491	4.471254
16748215	1702	110149	24821	4.437734

(2878152,[t(y,k,y,e,wa),t(g,k,y,e,gg),t(y,e,y,p,gg),t(g,e,y,p,gg),t(y,w,y,e,wa),t(g,w,y,e,wa),t(y,p,g,e,wa)]).

(111035385,[t(y,k,y,p,wa),t(g,k,g,p,wa),t(y,e,g,k,wa),t(g,e,y,w,wa),t(y,w,g,k,gg),t(g,w,y,e,gg),t(y,p,y,p,gg)])

(39736309,[t(y,k,y,p,wa),t(g,k,y,p,wa),t(y,e,g,k,wa),t(g,e,y,w,gg),t(y,w,y,w,wa),t(g,w,g,e,gg),t(y,p,y,p,gg)])

(10420116,[t(y,k,y,p,gg),t(g,k,y,p,gg),t(y,e,y,e,wa),t(g,e,y,p,gg),t(y,w,g,p,gg),t(g,w,y,p,wa),t(y,p,y,p,wa)])

(2082194,[t(y,k,y,p,gg),t(g,k,y,p,gg),t(y,e,y,p,wa),t(g,e,y,w,gg),t(y,w,y,k,gg),t(g,w,y,w,gg),t(y,p,y,p,wa)]).

(141949606,[t(y,k,g,w,gg),t(g,k,y,w,wa),t(y,e,y,p,gg),t(g,e,y,p,gg),t(y,w,y,p,wa),t(g,w,g,e,gg),t(y,p,g,e,wa)]).

(220129174,[t(y,k,g,p,gg),t(g,k,g,p,gg),t(y,e,y,e,wa),t(g,e,g,p,gg),t(y,w,y,k,wa),t(g,w,y,p,gg),t(y,p,y,p,wa)]).

(211278245,[t(y,k,g,w,gg),t(g,k,g,p,wa),t(y,e,y,p,gg),t(g,e,y,w,gg),t(y,w,g,p,wa),t(g,w,y,k,gg),t(y,p,y,p,gg)]).

(34991586,[t(y,k,y,w,wa),t(g,k,y,w,wa),t(y,e,g,p,gg),t(g,e,y,p,gg),t(y,w,y,e,gg),t(g,w,y,w,gg),t(y,p,y,p,wa)]).

(16748215,[t(y,k,y,p,gg),t(g,k,y,p,wa),t(y,e,y,p,wa),t(g,e,y,k,gg),t(y,w,g,e,wa),t(g,w,g,p,wa),t(y,p,g,e,gg)]).

Number	Wins	For	Against	Ratio
177259467	447	72990	45246	1.613181
147857609	430	70632	44168	1.599167
112528525	408	70991	44889	1.581479
178917518	467	77610	49258	1.575582
243974883	452	69572	44249	1.572284
44379577	362	83536	54426	1.534855
162541802	421	69765	45685	1.527088
44247974	357	85443	56587	1.50994
172142007	458	75545	50657	1.491304
176999249	446	71880	48494	1.482245

None + unreachable machines: top ten

(177259467,[t(y,k,g,w,wa),t(g,k,y,k,gg),t(y,e,g,e,gg),t(g,e,y,w,wa),t(y,w,g,k,gg),t(g,w,y,e,wa),t(y,p,y,p,gg)]).
(147857609,[t(y,k,g,w,wa),t(g,k,y,k,gg),t(y,e,y,k,gg),t(g,e,y,e,wa),t(y,w,g,k,gg),t(g,w,g,k,gg),t(y,p,y,e,gg)]).

(112528525,[t(y,k,y,w,gg),t(g,k,g,w,gg),t(y,e,g,w,gg),t(g,e,y,k,wa),t(y,w,g,e,wa),t(g,w,y,w,gg),t(y,p,g,e,gg)]).

(178917518,[t(y,k,g,e,gg),t(g,k,y,e,gg),t(y,e,g,k,gg),t(g,e,y,k,wa),t(y,w,g,w,wa),t(g,w,y,k,gg),t(y,p,g,e,wa)]).

(243974883,[t(y,k,g,e,wa),t(g,k,g,e,wa),t(y,e,g,e,gg),t(g,e,y,w,gg),t(y,w,g,k,gg),t(g,w,y,e,wa),t(y,p,y,e,gg)]).

(44379577,[t(y,k,y,k,gg),t(g,k,y,w,wa),t(y,e,g,w,wa),t(g,e,y,e,wa),t(y,w,g,e,gg),t(g,w,y,w,gg),t(y,p,g,p,gg)]).

(162541802,[t(y,k,g,w,wa),t(g,k,y,k,wa),t(y,e,y,k,gg),t(g,e,g,e,wa),t(y,w,g,w,gg),t(g,w,y,k,gg),t(y,p,g,e,wa)]).

(44247974,[t(y,k,y,k,gg),t(g,k,y,e,wa),t(y,e,g,w,gg),t(g,e,y,e,gg),t(y,w,g,e,wa),t(g,w,y,e,gg),t(y,p,g,p,wa)]).

(172142007,[t(y,k,g,k,gg),t(g,k,y,e,wa),t(y,e,g,e,wa),t(g,e,y,e,gg),t(y,w,y,e,wa),t(g,w,g,w,wa),t(y,p,y,p,gg)]).

(176999249,[t(y,k,g,e,wa),t(g,k,y,w,gg),t(y,e,g,e,wa),t(g,e,y,w,gg),t(y,w,g,e,gg),t(g,w,y,e,gg),t(y,p,y,w,gg)]).

(177259467,[t(y,k,g,w,wa),t(g,k,y,k,gg),t(y,e,g,e,gg),t(g,e,y,w,wa),t(y,w,g,k,gg),t(g,w,y,e,wa),t(y,p,y,p,gg)]).

All 2000 machines + extra

Number	Wins	For	Against	Ratio
2878152	1839	117540	16689	7.042962
111035385	1729	112915	22184	5.08993
39736309	1824	115565	22713	5.088055
2082194	1828	116122	22918	5.066847
10420116	1827	116186	23321	4.982033
220129174	1821	117694	24920	4.722873
141949606	1814	115071	24504	4.696009
211278245	1792	112339	24899	4.511788
188704895	1919	115609	26043	4.439158
34991586	1826	118795	26838	4.426373

(2878152,[t(y,k,y,e,wa),t(g,k,y,e,gg),t(y,e,y,p,gg),t(g,e,y,p,gg),t(y,w,y,e,wa),t(g,w,y,e,wa),t(y,p,g,e,wa)]).

(111035385,[t(y,k,y,p,wa),t(g,k,g,p,wa),t(y,e,g,k,wa),t(g,e,y,w,wa),t(y,w,g,k,gg),t(g,w,y,e,gg),t(y,p,y,p,gg)])

(39736309,[t(y,k,y,p,wa),t(g,k,y,p,wa),t(y,e,g,k,wa),t(g,e,y,w,gg),t(y,w,y,w,wa),t(g,w,g,e,gg),t(y,p,y,p,gg)])

(10420116,[t(y,k,y,p,gg),t(g,k,y,p,gg),t(y,e,y,e,wa),t(g,e,y,p,gg),t(y,w,g,p,gg),t(g,w,y,p,wa),t(y,p,y,p,wa)])

(2082194,[t(y,k,y,p,gg),t(g,k,y,p,gg),t(y,e,y,p,wa),t(g,e,y,w,gg),t(y,w,y,k,gg),t(g,w,y,w,gg),t(y,p,y,p,wa)]).

(141949606,[t(y,k,g,w,gg),t(g,k,y,w,wa),t(y,e,y,p,gg),t(g,e,y,p,gg),t(y,w,y,p,wa),t(g,w,g,e,gg),t(y,p,g,e,wa)]).

(220129174,[t(y,k,g,p,gg),t(g,k,g,p,gg),t(y,e,y,e,wa),t(g,e,g,p,gg),t(y,w,y,k,wa),t(g,w,y,p,gg),t(y,p,y,p,wa)]).

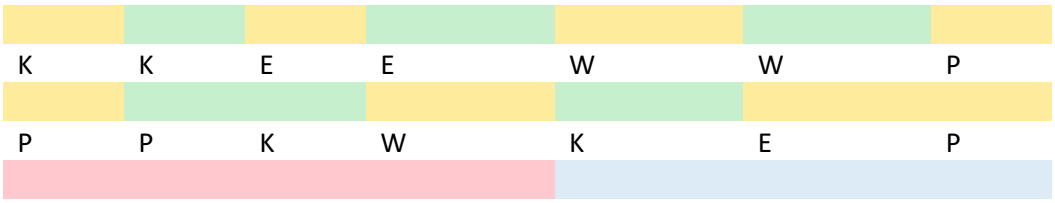
(211278245,[t(y,k,g,w,gg),t(g,k,g,p,wa),t(y,e,y,p,gg),t(g,e,y,w,gg),t(y,w,g,p,wa),t(g,w,y,k,gg),t(y,p,y,p,gg)]).

(188704895,[t(y,k,g,p,wa),t(g,k,y,p,wa),t(y,e,g,w,wa),t(g,e,g,p,wa),t(y,w,y,e,wa),t(g,w,y,k,wa),t(y,p,g,k,gg)]).

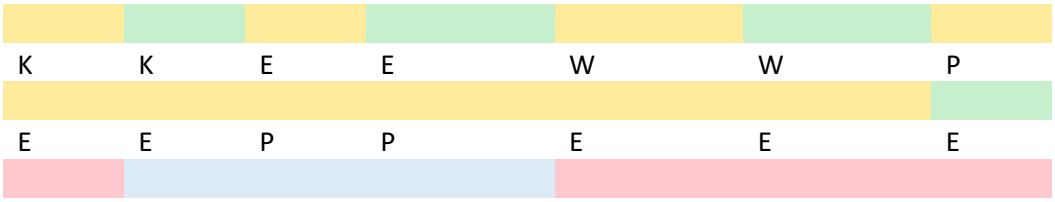
(34991586,[t(y,k,y,w,wa),t(g,k,y,w,wa),t(y,e,g,p,gg),t(g,e,y,p,gg),t(y,w,y,e,gg),t(g,w,y,w,gg),t(y,p,y,p,wa)]).

Some high preforming machines

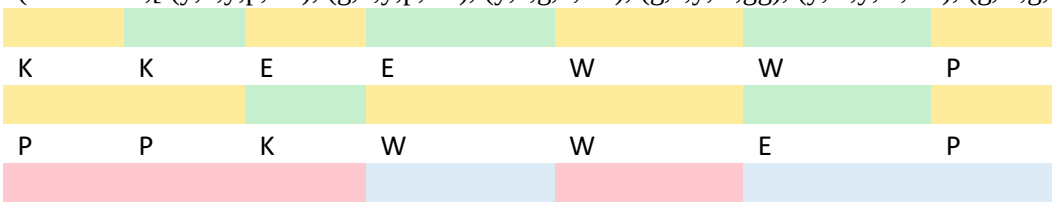
[t(y,k,y,p,wa),t(g,k,g,p,wa),t(y,e,g,k,wa),t(g,e,y,w,wa),t(y,w,g,k,gg),t(g,w,y,e,gg),t(y,p,y,p,gg)])



,[t(y,k,y,e,wa),t(g,k,y,e,gg),t(y,e,y,p,gg),t(g,e,y,p,gg),t(y,w,y,e,wa),t(g,w,y,e,wa),t(y,p,g,e,wa)]).



(39736309,[t(y,k,y,p,wa),t(g,k,y,p,wa),t(y,e,g,k,wa),t(g,e,y,w,gg),t(y,w,y,w,wa),t(g,w,g,e,gg),t(y,p,y,p,gg)])



b.

i. Eight of these machines were randomly generated by a script I wrote so they don't have a strategic advantage. Two of them I wrote myself to try and maximize success based on what worked in the previous tournament. I would be interested to see how the machines we were given were generated. Only one of these machines ended up in the top 30 when ranked in football order, most of these machines were in the bottom half of the results.

ii. I wasn't surprised that my machines didn't perform well as they were entirely randomly generated and the generation could have been more random, it was very simplistic. I was surprised that the 'variant' machines didn't perform very well in the tournament, it's harder to write a well-performing platypus machine ahead of time than it seems. The tournament seems like a method to select well-performing machines. Generating more machines would probably increase the chances of well-performing machines to emerge. I would choose machine 44444444 as it ended up in the top 30 in the tournament based on football ranking and seems to perform well. It has a good number of transitions.

c.
Tournament Rematch: After the tournament is run, the participants could have a chance to analyse their top performers and make changes to their machines to try and increase their chance of winning in the rematch.

Strategic Sabotage: Students could eliminate up to one machine per other student that struck them as a strong competitor at the start of the tournament in order to maximize their chances. This could also be combined with stealth changing rules in the tournament, where the winning machine is not immediately obvious.

Chaos Mode: Special machines are added to the tournament with the intention of causing as much chaos as possible, such as interrupting transitions, forcing the tape to go back or forward, restricting the amount of transitions randomly and so on. where the winner is the machine which can 'withstand' large amounts of variance.