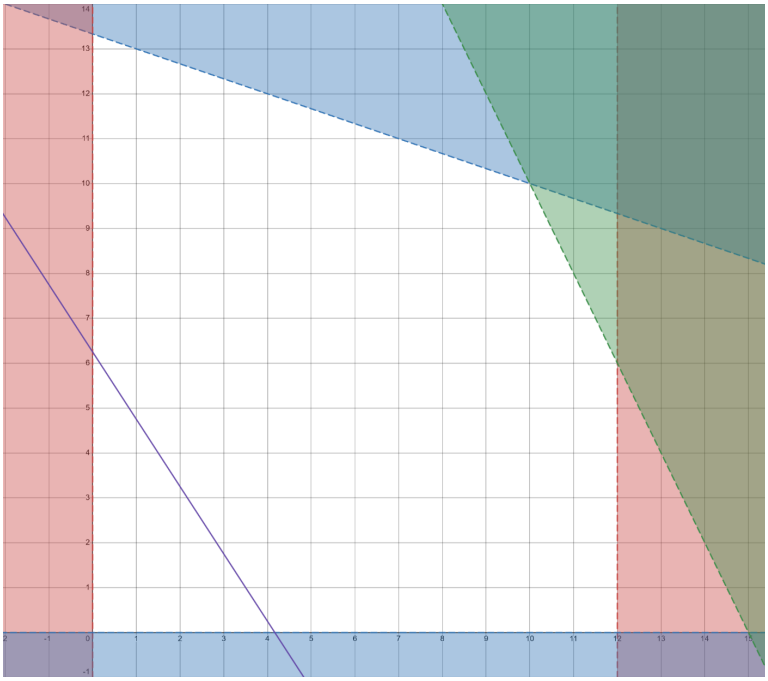
COMP361

Assignment 4

Assorted Algorithms and Problems

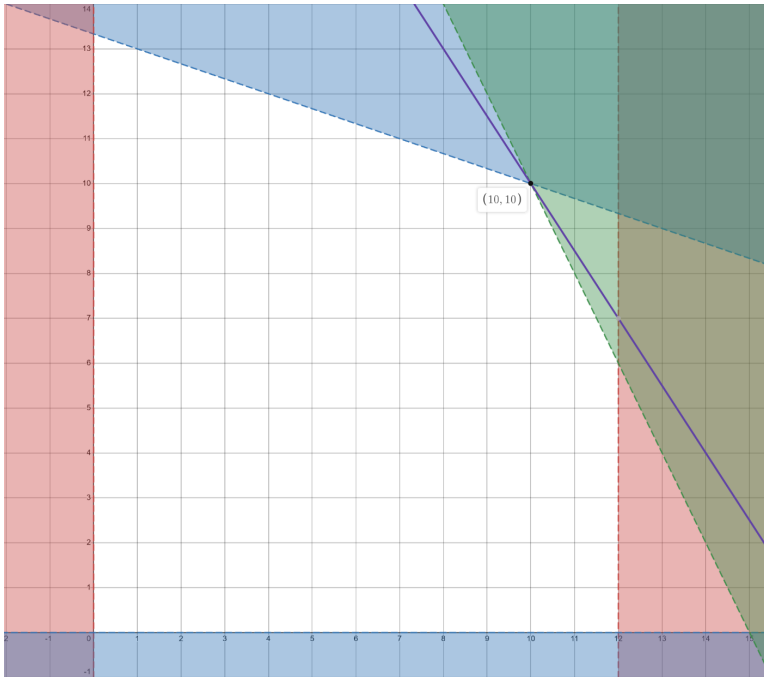
William Kilty
300473541

1.1:



The area not shaded is the solution space.

The purple line is the profit line. The further to the right this bar moves, the higher the profit we have achieved.



Here we have the profit bar as far to the right as possible without leaving the solution space. There is a single point $(10, 10)$ within the solution space. This point is the maximum profit.

$$z = 3x_1 + 2x_2$$
$$= 3 \cdot 10 + 2 \cdot 10$$
$$= 50$$

So the maximum profit possible is 50.

This can be verified by looking at the values at each intersection between two boundaries in the solution space.

We have valid solutions at $(12, 6)$, $(12, 0)$, $(0, 0)$, and $(0, 13\frac{1}{3})$

In order, we have solutions of:

$z = 3 \cdot 12 + 2 \cdot 6 = 48$

$z = 3 \cdot 12 + 2 \cdot 0 = 36$

$z = 3 \cdot 0 + 2 \cdot 0 = 0$

$z = 3 \cdot 0 + 2 \cdot 13.33 = 26.67$

We can see that 50 is larger than all of these values. So 50 is indeed the maximum

1.2:

Row									BV
0	z	$-2x_1$	$-x_2$	$-x_3$				$= 0$	$z = 0$
1		$3x_1$	x_2	x_3	s_1			$= 60$	$s_1 = 60$
2		$2x_1$	x_2	$2x_3$		s_2		$= 20$	$s_2 = 20$
3		$2x_1$	$2x_2$	x_3			s_3	$= 20$	$s_3 = 20$

x_1 is the new BV. Row 1 limit on $x_1 = \frac{60}{3} = 20$. Row 2 limit on $x_1 = \frac{20}{2} = 10$. Row 3 limit on $x_1 = \frac{20}{2} = 10$. The smallest amongst these is Row 3. Row 3 becomes the Pivot Row, making R3x_1 the new NBV.

After row manipulation dividing all elements of R3 by 2, and then removing multiples of R3 from each other row to make $x_1 = 0$ in every row but R3, we end up with:

Row								BV	
0	z		x_2				s_3	$= 20$	$z = 20$
1			$-2x_2$	$-\frac{1}{2}x_3$	s_1		$-\frac{3}{2}s_3$	$= 30$	$s_1 = 30$
2			$-x_2$	x_3		s_2	$-s_3$	$= 0$	$s_2 = 0$
3		x_1	x_2	$\frac{1}{2}x_3$			$\frac{1}{2}s_3$	$= 10$	$x_1 = 10$

We no longer have any negative values on Row 0. This means we've found an answer. Our answer can be found at $(10, 0, 0)$ which is:

$$z = 2x_1 + x_2 + x_3$$

$$= 2 \cdot 10 + 0 + 0$$

$$= 20$$

Exactly as the table suggests.

1.3:

$$\begin{aligned} \text{Max} \quad & z = 3x_1 + 2x_2 + 4x_3 + 3x_4 \\ \textit{s.t.} \quad & 10x_1 + 5x_2 + 12x_3 + 9x_4 \leq 15 \\ & x_1, x_2, x_3, x_4 \leq 2 \\ & x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

2.2:

```
Simulated_Annealing(S_0) {
      S = S_0;
      while(iteration < 3000) {
            newS = S;
            newS.swap(random, random); //ensure can't swap pt with itself
            if (E(newS) <= E(S)
            || Math.random() < Math.exp(-(eNew-eCur)/(k*coolTemp()))) {
                  S = newS;
            }
            iteration++;
      }
}
```

The method will loop until stopping criteria is met, picking neighbouring solutions to look at based on whether the neighbour is a better solution or if the temperature is high enough to allow a worse solution to be picked. The final chosen solution is then the one we run with.

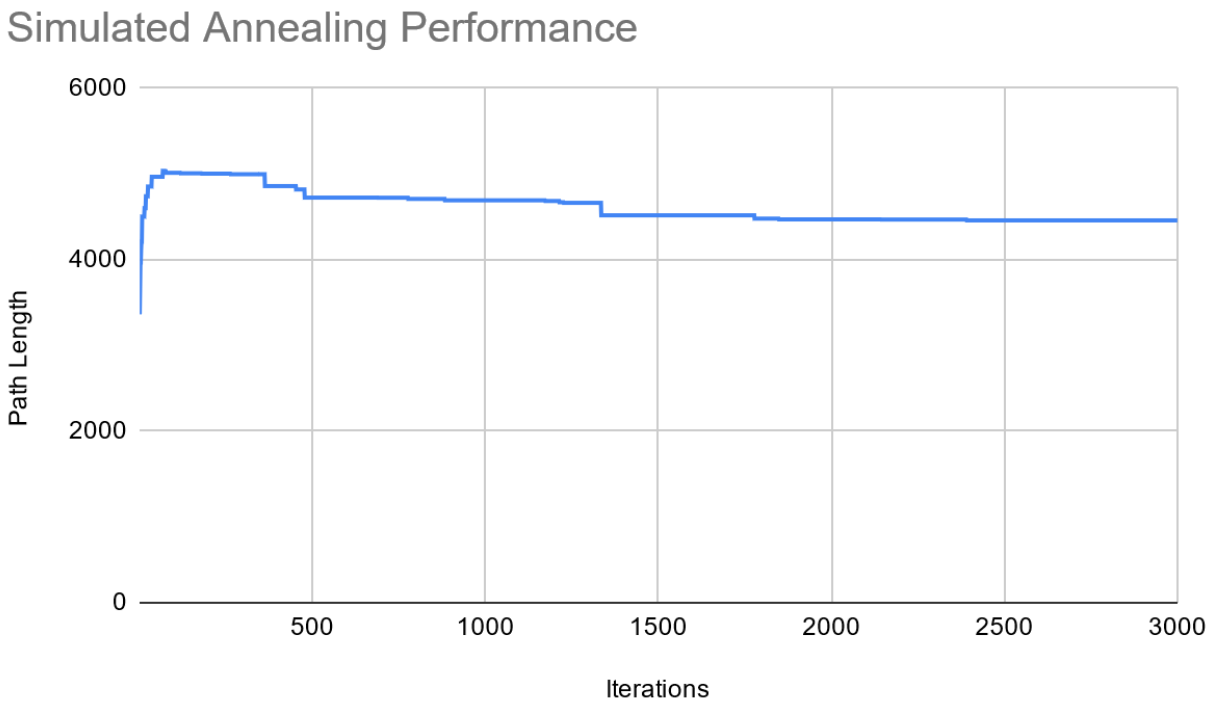
The initial solution is gathered by simply taking a walk through the points in the order they were read in from the file.

The neighbours to a solution are those which have two points in the walk swapped. A neighbour is selected at random to evaluate each step.

Cooling uses the Linear Cooling method from the lecture slides:

$T(t) = \frac{T_0}{1+\alpha t}$, $where\ \alpha = 0.5$.

The stopping criteria are when 3,000 iterations have occurred. This gives the algorithm time to settle into a local optima after cooling down enough not to leave it again.



This convergence curve shows the improvement of the solutions over time. The initial solution happens to be a fairly good estimate, but with a high initial temperature, the answer gets way worse rapidly. This allows the algorithm to find a different locale, with hopefully a better optima. However, at least in this case, the improvements in the new locale settle into a local optimum which is significantly worse than the starting point.

This is a risk with Simulated Annealing, as it is not guaranteed you will find a good solution.

The final result of this run is 4459.1. The optimal solution to this TSP is 2579. The optimal answer is significantly better than the Simulated Annealing result. This is due to the algorithm settling in a poor local optimum. Running the algorithm for longer or using a different 'k' or 'α' value will change the results, possibly for better, or possibly for worse.

Here's an example of my program running. It draws the nodes on the screen, and displays the current walk on the screen. It outputs the evaluation of the current solution to the terminal. Once the program has finished, it will output the contents of the terminal to a file called $output.txt$.

2.3:

Genetic Algorithm

3.1:

Follow W and sum the distance travelled. If we encounter every node in G exactly once, then W is a valid solution. If the total distance travelled is L, then W is an optimal solution. Exploring every node, n, in W is O(n) complexity. This solution takes polynomial time.

3.2:

Given a graph G, find the shortest Hamiltonian cycle C. C is the optimal solution to the symmetric Travelling Salesman Problem on G.

To find C, we need to find every Hamiltonian cycle, and take the smallest of them. This is a polynomial scaling of the Hamiltonian cycle problem.

3.3:

From Lectures:

If Y is a NPC problem, and X is a problem in NP with the property that Y $\leq^P$ X , then X is NPC.

Given $HAM \in NPC$, $sTSP \in NP$, and $HAM \leq^P sTSP$, then from the theorem above, $sTSP \in NPC$

4.1:

Select each vertex in a random order:

> look at its neighbours and tally up the number of each of the three colours - if the neighbour is coloured already.
>
> Set the colour of the initial selected vertex to the least populous colour of the neighbours. - random if there's a tie.

This guarantees each vertex will share a colour with at most ⅓ of its neighbours. (worst case, three way tie, thus sharing colour with ⅓ of the neighbours.)

In other words, the 3-colouring will be at least ⅔ as good as the optimal 3-colouring every time.

This algorithm is worst-case $O(n^2)$, as we cycle through every (n) vertex, and all of its neighbours, which is at most n-1, if each vertex is connected to each vertex. This means we perform operations on neighbours n*n-1 times at worst-case - or $O(n^2)$. Polynomial time.

4.2:

A solution S is represented by a 3-colouring of a graph G, where each node n_i is represented by a colour n_i^r, n_i^g, or n_i^b.

A candidate for neighbouring solutions is a solution S' which is identical to S, but for a single node n_j using a different colour.

The fitness of a solution is determined by the number of satisfied edges in G.

The Tabu list is a set of solutions visited previously. Each time a solution is visited, it is added to the Tabu list.

The function works as follows:

```
tabuSearch(S_0) {
      TL = new List;
      S = S_0;
      bestS = S;
      bestFitness = S.satisfiedEdges();
      TL.add(S);
      while(true) {
            nextS = null;
            nextFitness = -1;
            for (n : S) {
                  for (int i = 0; i < 3; i++) {
                        newS = S.copyWithNewN(n, i);
                        if TL.contains(newS) continue;
                        newFitness = newS.satisfiedEdges();
                        if newFitness > bestFitness {
                              bestS = newS;
                              bestFitness = newFitness;
                        }
                        if newFitness > nextFitness {
                              nextS = newS;
                              nextFitness = newFitness;
                        }
                  }
            }
            If (nextFitness == -1 || bestFitness == S_0.edgeNum()) {
                  break;
            }
            S = nextS;
      }
      return bestS;
}
```

5.1:

Sort list of jobs in descending order.

Submit jobs to the machine with the lowest pre-existing makespan T_i, adding to it the time of the job j with said machine's speed. (t_j for slow machines, $\frac{t_j}{3}$ for fast machines). This is a simple modification of the second algorithm introduced in class, taking into account the benefits of the fast machines.

5.2:

If there are more than m jobs, then $T^* \geq \frac{2}{3}t_{m+1}$.

Assume machine i* has the maximum load, and job j* is the last job assigned to it during the algorithm. Obviously, j* ≥ m + 1 and $t_j^* \leq t_{m+1} \leq \frac{3}{2}T^*$.

$(T_i^* - t_j^*) * m < \sum_i T_i = \sum_j t_j,$

$T_i^* < \frac{1}{m}\sum_j t_j + t_j^* \leq T^* + \frac{2}{3}T^* = \frac{5}{3}T^*$

$\frac{5}{3} < 4$. So our algorithm is sufficient.