Problem 1. (30%)

Calculation of time complexity (3%)

Given n cities, using exhaust search to see every result takes O(n!).

Calculation of time needed to solve the problem (2%)

40 cities:40! different tours 40 add operations per each tour computing power: 0.5 trillion add operations per second: $40! * 40/(0.5 * 10^{12})$ seconds = 2069800312653215967391200582435.6 years

Problem 2. Birds in a graph (20%)

1. (5%) The corresponding decision problem can be defined as follows: Given an undirected graph $G = (V, E)$ with capacity c_i, $i \in V$, decide whether k birds can live in the given graph without fighting.

   Because the maximum number of birds must be in the $[\max_i c_i, \sum_i c_i]$, we can solve the original optimization problem by binary search on k. That is, find the maximum k such that the decision problem returns true. The binary search requires solving the decision problem $\log \sum_i c_i$ times, which is linear to the input size. If the decision problem belongs to P, there exists an algorithm solving it in polynomial time. By the binary search on k, we also can solve the original optimization problem in polynomial time.

2. (5%) Given any bird allocation as the certificate, we can check that no two adjacent vertices are occupied by birds in $O(|V|^2)$ and that the number of birds $>= k$ in $O(|V|)$. Obviously, the above verification algorithm runs in polynomial time. For a problem instance of which answer is no, it is clear that no such ceritficate exists. Therefore, the decision problem is in NP.

3. (7%) The following is a reduction from 3-CNF-SAT to the decision problem. Given a 3-CNF-SAT instance $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where C_j is the disjunction of 3 variables, drawn form $x_1, x_2, \ldots, x_l$ and $\bar{x_1}, \bar{x_2}, \ldots, \bar{x_l}$. Construct graph G as follows: First, create a vertex with capacity = 1 for each variable in each clause. Second, add an edge between two vertices corresponding the variables from the same clause. Finally, add an edge between every pair of vertices corresponding the variables x_i and $\bar{x_i}$. Now, we use the black box of the decision problem to determine whether m birds can live in G without fighting. If they can, return yes. Otherwise, return no. (We do not require the proof of correctness in this homework problem. The proof is similar to that of the reduction from 3-SAT to the maximum independent set, which is a well-known problem.)

   Let us check the time complexity of the reduction. The first and second steps run in $O(m)$, and the last step runs in $O(m^2)$. Therefore, 3-CNF-SAT is polynomial-time reducible to the decision problem.

4. (3%) By the fact that 3-CNF-SAT is a known NP-complete problem and the polynomial-time reduction from 3-CNF-SAT to the decision problem, the decision problem is in NP-hard. Because the decision problem is in NP and NP-hard, the decision problem is a NP-complete problem.

Problem 3. MCS (15%)

1. (5%) Show that $L \in NP$

   First transform the problem into decision problem that determine whether there is an induced subgraph of both G_1 and G_2 with edges $\geq$ k, a constant value. we can design an algorithm that verify H is the induced subgrapg of G, which can be done in polynomial time ($O(|H| + mapping(V_H \rightarrow V_G)) = O(n)$), so totally it takes polynomial to decide that H has edges $\geq$ k and H is induced subgrapg of oth G_1 and G_2.

   Second we need to find the maximum k to be the Maximum Commom Subgraph, so we just need to do a for loop k = 0:$min(|G_1|, |G_2|)$ and the last iteration that the decision output yes is the maximum k we want, so its still takes polynomial time to find k. Then $L \in NP$.

2. (10%) Show that $L \in NP$-Hard

   To reduce the Maximum Clique into MCS, we just need to build a new complete graph G', with $|V_G|$ vertices (each 2 vertices have edge), then take the input of Maximum Clique, G, and G', as the input of MCS, then we can get the maximal k of 2 graphes, which means that G has the maximum clique with vertices = k. To build G' only takes $O(|V_G|^2)$, so the reduction can be done in polynomial time.

   Since we know that Maximum Clique is in NP-Complete, $L \in NP$-Hard.

$L \in NP$ and $L \in NP$-Hard $\rightarrow L \in NP$-complete.

Problem 4. (15%)

Show that any language in NP can be decided by an algorithm running in time $2^{O(n^k)}$ for some constant k.

By the definition of the class NP:

A language L belongs to NP if and only if there exist a two-input polynomial-time algorithm A and a constant c such that

L=$\{x \in \{0,1\}^* :$ there exists a certificate y with $|y| = O(|x|^c)$ such that $A(x,y) = 1\}$

Then algorithm A verifies L in polynomial time.

Therefore, for a language L in NP, we can design an algorithm A_2 that decides L in time $2^{O(n^k)}$ for some constant k.

A_2

for all possible y

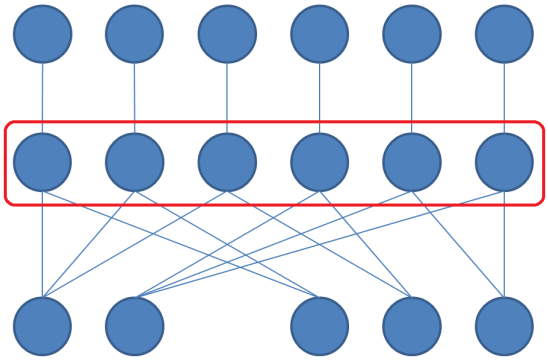
if $A(x,y) == 1$: return 1

return 0

There are $2^{|y|}$ possiblities of y. $|y| = O(|x|^c)$, therefore there are $2^{O(|x|^c)}$ possibilities of y. And for every y, the algorithm runs A(x,y) once, in polynomial time, $O(|x|^t)$ for some constant t. So totally the runtime of the algorithm is $2^{O(|x|^c)}O(|x|^t) = 2^{O(n^c)}(c_2\ n^t) = 2^{O(n^c)}(2^{log_2 c_2 + t log_2 n}) = 2^{O(n^k)}$

Problem 5. Vertex Cover(20%)

1. (4%) Find a minimum vertex cover of the graph in Figure 1.



2. No, there is no such constant k.

   For $n \in \mathbb{N}$, define $G_n = (V_n, E_n)$.
   $V_n = \bigcup_{i=0}^{n} V_{n,i}$
   There are $n!$ vertices in $V_{n,0}$.
   For $i = 1, 2, 3, ..., n$, there are $\frac{n!}{i}$ vertices in $V_{n,i}$.
   Each vertex in $V_{n,i}$ are connected to i vertices in $V_{n,0}$ without duplication, so they are of degree i.
   Therefore, every vertex in $V_{n,0}$ is of degree n.
   The figure of problem 5.1 is an example of G_3. The top 6 points are in $V_{3,1}$. The 6 points in the middle are in $V_{3,0}$. The bottom left 2 points are in $V_{3,3}$. The bottom right 3 points are in $V_{3,2}$.

   Since our algorithm arbitrarily pick one of the vertices that has the greatest degree, it might choose vertices in $V_{n,n}$ over $V_{n,0}$, and $V_{n,n-1}$ over $V_{n,0}$, and so on. Our algorithm may choose all vertices in $V_{n,n}, V_{n,n-1}, V_{n,n-2}, ..., V_{n,1}$ and none in $V_{n,0}$. Therefore, we would have a vertex cover of size $\sum_{i=1}^{n} \frac{n!}{i}$ while the minimum vertex cover is $V_{n,0}$.
   $k * (n!) \geq \sum_{i=1}^{n} \frac{n!}{i} \rightarrow k \geq \sum_{i=1}^{n} \frac{1}{i}$ which would approach ∞ as n grows.

3. Yes. $k = 2$.

   By the definition of vertex cover, if $(u, v) \in E$, at least one of u and v would be in V_C^*.
   For every edge (u, v) we picked during the process, we add both u and v into C and at least one of them would be in V_C^*. Therefore, $|C| \leq 2 * |V_C^*|$.