0.1 Traveling salesman problem

- *Algorithm:* Christofides Algorithm (algo. 1)
- *Input:* a universe of cities $V = \{1, 2, ..., n\}$, a metric distance function $d : V \times V \to R^+$.
- *Complexity:* $\mathcal{O}(n^2 \log n)$, for approximate algorithm, actually we care more about the accuracy.
- *Data structure compatibility:* graph and matrix
- *Common applications:* supply chain management and chip production

Problem. Traveling salesman problem

We assume a universe of cities $V = \{1, 2, ..., n\}$, the metric distance function is $d : V \times V \to R^+$. We want to find a path(a traverse of the cities) $k(1), k(2), ..., k(n)$, such that the total cost $\sum_{i=1}^{n-1} d_{k(i)k(i+1)} + d_{k(n)k(1)}$ is minimized.

Description

For a simple introduction to approximate algorithm, please refer to problem 71, or read the book above, or take course VG441, which is a course offered by JI. Unlike humanity course like VR477 introduction to humanity project, VG441 is a real CS course.

TSP, shorthand for traveling salesman problem originates from a salesman that wants to travel through every city exactly once using shortest distance. It is a classic problem in the field of operations research and computer science. It is related to problems like traveling purchaser problem. We would like to research on TSP because it has tons of applications in real life. For example supply chain management has to take TSP into account so that the goods are delivered in a fast and cheap way. TSP can also be modified to cope with problems in chip production and DNA measurement. If you are interested in TSP and wants to study further, you may refer to a book(collection of papers TAT) called *the Design of Approximate Algorithms*[3]. You will find variation like asymmetric TSP and euclidean TSP.

Detailed definition of this problem is shown below. Assume we have a universe of cities $V = \{1, 2, ..., n\}$, and the symmetric distance between every city pair is defined as $C = (c_{ij})$. We assume a distance function to be $d : V \times V \to R^+$. The distance can have meanings like physical distance, travel expenses, travel time cost and so on. Notice that the cost should be nonnegative. Since there does not exist an approximation algorithm if we do not restrict the cost to be metric[2]. So we should specify the cost to be metric. here metric means the cost be symmetry and it should satisfy the triangular inequality. Given the cities and the cost function between them, we would like to find a path with minimum cost that visits every city in the universe exactly once and then return to the starting point.

Time complexity analysis. For the MST, we can use the prim's algorithm taught in VE477 and the time complexity should be $\mathcal{O}(E + logV) = \mathcal{O}(n^2)$ in our case[1]. Also $\mathcal{O}(n^2)$ is enough for find the set of odd degree vertices. After that we do the minimum cost matching which is of complexity $\mathcal{O}(nm \log n)$, in our case, m is $\mathcal{O}(n)$. Add edges is linearly cheap, find a walk is also linearly cheap and we can ignore it. Shortcutting in the last step in the algorithm needs only linear time as well. In short, the time complexity of this Christofides algorithm depends on the time complexity of finding minimum cost matching.

Algorithm 1: Christofides

Input : a universe of cities $V = \{1, 2, ..., n\}$, a metric distance function $d : V \times V \to R^+$.
Output: a path that visits every city in the universe exactly once and then return to the starting point

```
1 Compute the minimum spanning tree(MST) M of (V,d)
2 Compute the minimum cost matching K on the vertices of M with odd degree
3 Add edges of K to M and we get an Eulerian graph D'
4 find a walk W' that travels through each edge of D' eactly once
5 shortcut W' by skipping revisited vertices to get a TSP path T'
6 return T'
```

References.

[1] Manuel. *VE477 – Introduction to Algorithms (lecture slides)*. 2020 (cit. on p. 1).

[2] Cong Shi. *VG441 – Supply Chain (lecture slides)*. 2020 (cit. on p. 1).

[3] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011 (cit. on p. 1).

0.2 Square roots mod p (Tonelli-Shanks)

- *Algorithm:* Tonelli–Shanks (algo. 2)
- *Input:* a prime number p and a positive integer n in $\mathbb{Z}/p\mathbb{Z}$.
- *Complexity:* $\mathcal{O}(n^2)$
- *Data structure compatibility:* N/A
- *Common applications:* cryptography research, number theory research

Problem. Square roots mod p (Tonelli-Shanks)

Given a prime number p and a positive integer n in $\mathbb{Z}/p\mathbb{Z}$, we want to find the quadratic residue of n, which is a positive integer r in $\mathbb{Z}/p\mathbb{Z}$, such that $r^2 \equiv n \mod p$.

Description

When designing an algorithm to solve for a computer science problem, we would like to seek for a fast algorithm. If we can get a polynomial time complexity, we will be satisfied and the implementation is workable. But in cryptography study, we do the reverse. The security requires a significantly high level of time complexity if someone else want to recover the secret. However, we should notice that although it should be designed to be hard to recover the secret, using the secret(its inverse) should be fast to process.

In the field of number theory, there is an interesting question that given a large prime number p and a positive interger n in $\mathbb{Z}/p\mathbb{Z}$, we would like to find out an integer r in $\mathbb{Z}/p\mathbb{Z}$ such that $r^2 \equiv n \mod p$. This is also called as finding a quadratic residue. Notice that these calculations are done under modular arithmetic within the field $\mathbb{Z}/p\mathbb{Z}$.

Further, Tonellis's algorithm would be extended for any cyclic group and to the kth root instead of square root. A table can be prepared in advance to facilitate the process. Another extension would be this algorithm can be used on module of p^k instead of p. For this extension, the proof and process is more complicated and can be seen in Dickson's *Theory of Numbers*[1] and is beyond the scope of this course[4]. The usage of this algorithm is mostly in the field of cryptography research like Rabin cryptography system and elliptic curves.

Time complexity analysis is very difficult for this algorithm. According to the paper written by Lindhurst and Scott. The average case time complexity would be $\frac{n^2}{4} + \frac{7n}{4} + 1$ multiplications with the standard deviation of deviation $\sqrt{\frac{n^3}{12} + \frac{3n^2}{8} + \frac{13n}{24} - 1}$[2]. The worst case is easier to analysis, which should be $(n+2)+(n+1)+...+5+4 = \frac{1}{2}\left(n^2 + 5n - 6\right)$ multiplications. Basically, the algorithm time complexity indicates that it is easy for eve to recover the secret. So Alice and Bob should never use a simple prime number p for quadratic residue. Instead, they should use $p \times q$ instead of p, where p and q are both large prime numbers.

Algorithm 2: Tonelli–Shanks

Input : a prime number p and a positive integer n in $\mathbb{Z}/p\mathbb{Z}$
Output: a integer r in $\mathbb{Z}/p\mathbb{Z}$ such that $r^2 \equiv n \mod p$.

```
Factorize the number p - 1 by factor 2 $Q \leftarrow p - 1$ $s \leftarrow 0$ while $2 \mid Q$ do
    $Q \leftarrow Q/2$
    s++
end while
We now get $p - 1 = 2^s \times Q$.
Use Jacobi symbol to rule out the elements in $\mathbb{Z}/p\mathbb{Z}$ that are quadratic non-residues.
$M \leftarrow S$
$c \leftarrow z^Q$
$t \leftarrow n^Q$
$R \leftarrow n^{\frac{Q+1}{2}}$
while $t \neq 0$ and $t \neq 1$ do
    find the least i, $0 < i < M$, such that $t^{2^i} = 1$.
    $b \leftarrow c^{2^{M-i-1}}$
    $M \leftarrow i$
    $c \leftarrow b^2$
    $t \leftarrow tb^2$
    $R \leftarrow Rb$
end while
if $t = 0$ then
    return r = 0
end if
if $t = R$ then
    return r = R
end if
/* notice that if we have the answer $r$, it is obvious that $-r$(or $p - r$) is another answer
for $r^2 \equiv n \mod p$.[3] */
```

References.

[1] Leonard Eugene Dickson. *History of the Theory of Numbers.* Carnegie Institution of Washington, 1919 (cit. on p. 3).

[2] Scott Lindhurst. "An analysis of Shanks's algorithm for computing square roots in finite fields". In: *To appear in the proceedings of the* (1997) (cit. on p. 3).

[3] Manuel. *VE475 – Introduction to Cryptography (lecture slides).* 2020 (cit. on p. 3).

[4] Wikipedia contributors. *Tonelli–Shanks algorithm.* [Online; accessed 6-October-2020]. 2020. URL: #https://en.wikipedia.org/wiki/Tonelli%E2%80%93Shanks_algorithm#cite_note-dickson-3 (cit. on p. 3).

0.3 Set cover

- *Algorithm:* Greedy Approach (algo. 3)
- *Input:* A universe of elements $V = \{e_1, ..., e_n\}$, and a list of sets $\{S_i \subset V\}_{i=1}^m$
- *Complexity:* $\mathcal{O}(mn \cdot \min(n, m))$
- *Data structure compatibility:* data structure set can be used with the algorithm
- *Common applications:* facility location selection in industry

Problem. Set cover

Give A universe of elements $V = \{e_1, ..., e_n\}$, and a list of sets $\{S_i \subset V\}_{i=1}^m$, we want to cover all the elements in universe V with minimal number of sets from the set list.

Description

Detailed definition of the problem. Assume we have a universe of elements $V = \{e_1, ..., e_n\}$, we assume there are a list of non-empty sets $\{S_i \subset V\}_{i=1}^m$, we would like to find out a number of sets $\{T_i\}$ out of the universe V, such that the universe V is equal to the union of them, namely $V = \cup_{i=1}^j T_i$. There are many variations of this problem, we can also set cost to each set(a specific number rather than 1). However, the idea does not change much, we only need to modify the greedy algorithm a little bit. Set cover problem is also related to max coverage and vertex coverage problems.

Set cover is a classic NP problem in the field of mathematics and operations research. Unless we can prove $N = NP$, we can not derive an algorithm that can find the exact solution to the problem within polynomial time[1]. So a natural trade-off will lie between time complexity and accuracy. In our case, the time complexity is more critical. To deal with the time complexity, we have to sacrifice the accuracy and use approximate algorithm.

Another perspective is the usage of greedy algorithm. In algorithm design, the big picture is that greedy algorithm, local search, and LP-relaxation are three main strategies that deal with most NP-hard problems[2]. So brute force these three methods before using fancy technique. To enjoy more about approximate algorithms, read *the Design of Approximate Algorithms*[3] and you will gain a lot.

The key idea in this algorithm is to be greedy, trying to choose the set that covers largest number of elements. Here we assume $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$, and we can tell $H_n \approx \ln n$, which is taught in basic calculus course. Our algorithm is an H_n- approximate algorithm that solves the set cover problem. The detailed proof is beyond the scope of this course(sadly engineers does not share the beauty of mathematics).

Also we have to consider the time complexity for our algorithm. For this simple version of greedy algorithm. We can eliminate at least one element from the set U or we can take it as one set from the set list $\{S_i \subset V\}_{i=1}^m$, so there are at most $\min(n, m)$ loops for the while(not less because of the case that each set S_i has exactly one element). Within each loop, there are $\mathcal{O}(mn)$ operations for the argmax operation. So the total time complexity is $\mathcal{O}(mn \cdot \min(n, m))$. In the exercise problem in Introduction to Algorithm, we can modify the algorithm to make it linear for argmax.

Algorithm 3: Greedy Approach

```
Input  : A universe of elements $V = \{e_1, ..., e_n\}$, a list of sets $\{S_i \subset V\}_{i=1}^m$
Output: minimal number of sets $\{T_j\}$
1 Assume U is the set of uncovered elements.
2 Set $U = \{e_1, ..., e_n\}$.
3 while $U \neq \emptyset$ do
4     pick $T_j = argmax_{j=1,...,m} \mid S_j \cap U \mid$
5     U $\leftarrow U \setminus T_j$
6 end while
7 return
```

References.

[1] Manuel. *VE477 – Introduction to Algorithms (lecture slides)*. 2020 (cit. on p. 4).

[2] Cong Shi. *VG441 – Supply Chain (lecture slides)*. 2020 (cit. on p. 4).

[3] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011 (cit. on p. 4).

0.4 Naive Bayes Classifier

- *Algorithm:* Naive Bayes Classifier(algo. 4)
- *Input:* The training set $T = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$; $x_j^{(i)}$ is the j^{th} feature of the i^{th} sample;a_{jl} is the l possible values of the j^{th} feature
- *Complexity:* $\mathcal{O}(nk)$
- *Data structure compatibility:* N/A
- *Common applications:* Artificial intelligence

Problem. Naive Bayes Classifier

Naive Bayesian Classification is a classification method based on Bayesian Theorem and Conditional Independence Assumption.

Description

Bayesian Theorem

Bayes's theorem is stated as[1]

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{0.4.1}$$

where A and B are events and $P(B) \neq 0$.

Naive Bayes Classifier

Assume input space $X \in \mathbb{R}^n$ is a n dimension vector and the label set of input $Y = \{c_1, c_2, \cdots, c_K\}$. The training set can be denoted as

$$T = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\} \tag{0.4.2}$$

which is generated by $P(X, Y)$ independently.

The goal of Naive Bayes is to learn a joint distribution $P(X, Y)$. Specially, it should learn prior distribution and conditional distribution. The prior distribution is

$$P(Y = c_k), \quad k = 1, 2, \cdots, K \tag{0.4.3}$$

The conditional distribution is

$$P(X = x|Y = c_k) = P(X_1 = x_1, \cdots X_n = x_n|Y = c_k) \tag{0.4.4}$$

Naive Bayes makes a conditional independence assumption, which is

$$P(X = x|Y = c_k) = \prod_{j=1}^{n} P(X_= x_j|Y = c_k) \tag{0.4.5}$$

Naive Bayes will use input x and output the class by the learned largest posterior distribution $P(Y = c_k|X = x)$. The posterior can be calculated by using Bayesian Theorem and equation. 0.4.5

$$P(Y = c_k|X = x) = \frac{P(X = x|Y = c_k)P(Y = c_k)}{\sum_k P(X = x|Y = c_k)P(Y = c_k)} \tag{0.4.6}$$

$$= \frac{P(Y = c_k)\prod_{j=1}^{n} P(X_= x_j|Y = c_k)}{\sum_k P(Y = c_k)\prod_{j=1}^{n} P(X_= x_j|Y = c_k)}, \quad k = 1, 2, \cdots, K \tag{0.4.7}$$

Then, the Naive Bayes can be represented as

$$y = \arg\max_{c_k} \frac{P(Y = c_k)\prod_{j=1}^{n} P(X_= x_j|Y = c_k)}{\sum_k P(Y = c_k)\prod_{j=1}^{n} P(X_= x_j|Y = c_k)} \tag{0.4.8}$$

$$= \arg\max_{c_k} P(Y = c_k)\prod_{j=1}^{n} P(X_= x_j|Y = c_k) \tag{0.4.9}$$

Maximum Likelihood Estimation(MLE)

We can use MLE to estimate prior probability $P(Y = c_k)$.

$$P(Y = c_k) = \frac{\sum_{i=1}^{N} I(y_i = c_k)}{N}, \quad k = 1, 2, \cdots, K \tag{0.4.10}$$

Assume possible value set of the j^{th} feature x_j is $\{a_{j1}, a_{j2}, \cdots, a_{jS_j}\}$. The estimation of conditional probability $P(X_j = a_{jl}|Y = c_k)$ is

$$P(X_j = a_{jl}|Y = c_k) = \frac{\sum_{i=1}^{N} I(x_j^{(i)} = a_{jl}, y_i = c_k)}{\sum_{i=1}^{N} I(y_i = c_k)} \tag{0.4.11}$$

$$j = 1, 2, \cdots, n; \quad l = 1, 2, \cdots, S_j; \quad k = 1, 2, \cdots, K \tag{0.4.12}$$

where $x_j^{(i)}$ is the j^{th} feature of the i^{th} sample; a_{jl} is the l possible values of the j^{th} feature.

References.

[1] M. G. Kendall, A. Stuart, and J. K. Ord. *Kendall's Advanced Theory of Statistics*. USA: Oxford University Press, Inc., 1987. ISBN: 0195205618 (cit. on p. 5).

Algorithm 4: Naive Bayes Classifier

```
Input  : The training set $T = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$; $x_j^{(i)}$ is the $j^{th}$ feature of the $i^{th}$ sample;$a_{jl}$ is
         the $l$ possible values of the $j^{th}$ feature
Output: class of the sample $x$: $y$
1 Calculate prior distribution by equation. 0.4.10 and conditional distribution by equation. 0.4.12.
2 Calculate posterior distribution with sample $x = (x_1, x_2, \cdots, x_n)^T$ and equation. 0.4.7.
3 Find the class $y$ of $x$ with equation. 0.4.9.
4 return $y$
```

0.5 Neural Network

- *Algorithm:* Neural Network(algo. 5)
- *Input:* A tensor
- *Complexity:* For a n-layer neural network,of which each layer has $n_i(i \in 1, 2, \cdots, n)$, the complexity is $\mathcal{O}(\sum_{i=1}^{n-1} n_i n_{i+1})$.
- *Data structure compatibility:* Matrix
- *Common applications:* Artificial intelligence

Problem. Neural Network

Neural network is a kind of mathematical model imitating the mechanism of biological neural network.

Description

Linear classification machine

In the early development of artificial intelligence, people want to use a simple predicting machine to implement intelligence. The simple predicting machine is actually a linear classification machine.

$$y = Ax + B \tag{0.5.1}$$

where x is input and A, B are linear parameter. However, this machine cannot solve non-linear problem like **XOR**[1] problem(Fig. 1). After studying biological neural network, scientist find the combination of activation function and connection model can solve non-linear problem. That is modern neural network.

Forward propagation

In convenience, we will use a 3-layer(Fig. 2) to demonstrate the theory. The first layer is an input layer, which has the same dimension as the input tensor. The second layer is a hidden layer, whose dimension is decided by user. The last one is an output layer, which has the same dimension as the number of categories. We use I, H, O to denote values of input, hidden and output layer. The feed forward process can be calculated as following equation. For example, H_1 can be calculated as

$$h_1' = I_1 \cdot w_{1,1} + I_2 \cdot w_{2,1} + I_3 \cdot w_{3,1} \tag{0.5.2}$$

$$h_1 = A(h_1) \tag{0.5.3}$$

where $A(x)$ means activation function. Some common activation functions are

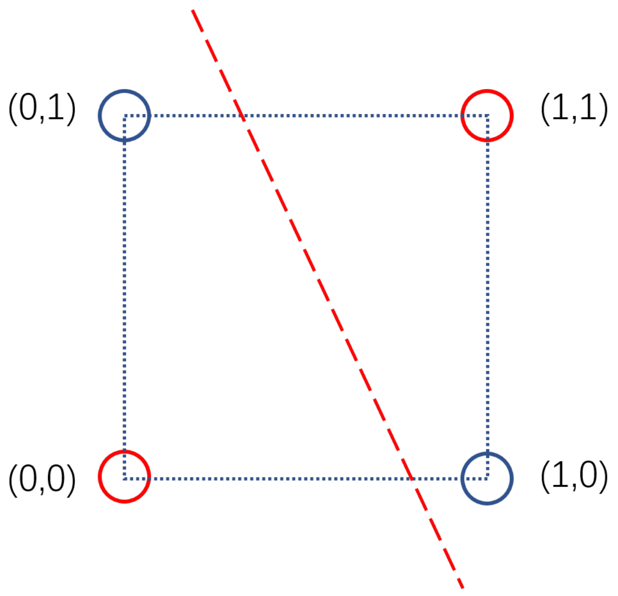

Figure 1: XOR problem

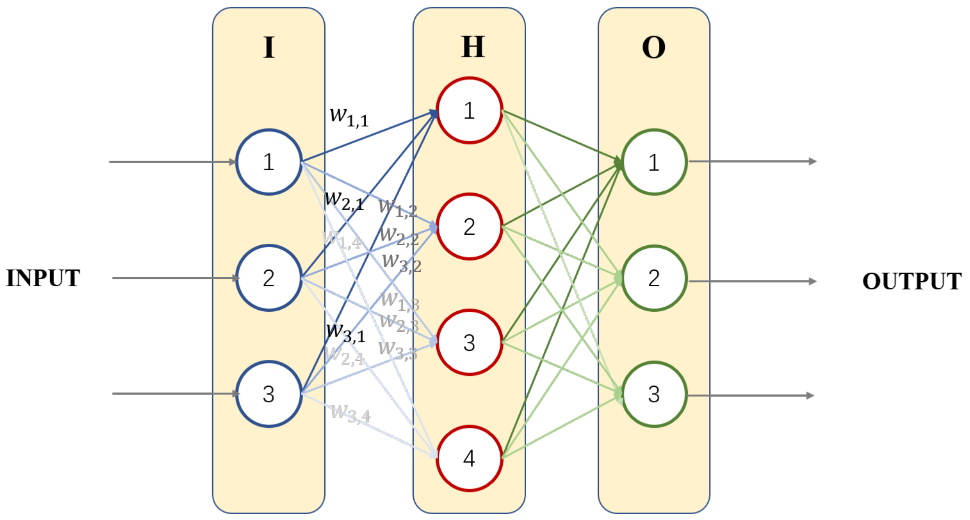

Figure 2: Structure of a 3-layer neural network

1. sigmoid

$$y = \frac{1}{1 + x^{-x}} \tag{0.5.4}$$

2. tanh

$$y = tanh(x) \tag{0.5.5}$$

3. Relu

$$y = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{0.5.6}$$

We can also use matrix to simplify our expression. The input matrix is

$$I = \begin{pmatrix} i_1 \\ i_2 \\ i_3 \end{pmatrix} \tag{0.5.7}$$

The hidden and output matrix are similar. The weight matrix is

$$W_{I,H} = \begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{pmatrix} \tag{0.5.8}$$

So the forward propagation of input and hidden layer can be represent as

$$H = A(W_{I,H}^{\mathsf{T}} I) \tag{0.5.9}$$

In the same way the hidden and output layer is

$$O = A(W_{H,O}^{\mathsf{T}} H) \tag{0.5.10}$$

Back propagation

We first define loss function L to represent the error between ground truth and the output. The most commonly used is square error.

$$L = \sum_n (t_n - o_n)^2 \tag{0.5.11}$$

To update weight, we must calculate

$$\frac{\partial L}{\partial w_{j,k}} = \frac{\partial}{\partial w_{j,k}} (T_k - O_k)^2 \tag{0.5.12}$$

We then use chain rule. The activation function is sigmoid here.

$$\begin{aligned}
\frac{\partial L}{\partial w_{j,k}} &= \frac{\partial L}{\partial O_k} \cdot \frac{\partial O_k}{\partial w_{j,k}} \\
&= -2(t_k - o_k) \cdot \frac{\partial O_k}{\partial w_{j,k}} \\
&= -2(t_k - o_k) \cdot \frac{\partial}{\partial w_{j,k}} \sigma(\sum_j w_{j,k} \cdot o_j) \\
&= -2(t_k - o_k) \cdot \sigma(\sum_j w_{j,k} \cdot o_j)(1 - \sigma(\sum_j w_{j,k} \cdot o_j)) \cdot o_j
\end{aligned}$$

The update function is

$$w_{j,k}^{(new)} = w_{j,k}^{(old)} - \alpha \cdot \frac{\partial L}{\partial w_{j,k}} \tag{0.5.13}$$

α is learning rate which should be tuned.

Algorithm 5: Neural network

```
Input  : Training set $\{(I_i, T_i)\}, i = 1, 2 \cdots N$, Rounds $R$
Output: Output vector $O$ of sample $I$
1 for $r = 1$ to $R$ do
2     for $i = 1$ to $N$ do
3         Forward propagation
4         Calculate loss function $L$
5         Back propagation
6     end for
7 end for
8 return $O$
```

References.

[1] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry.* Cambridge, MA, USA: MIT Press, 1969 (cit. on p. 7).