Examinee number: 11070
Name: Gábor Ádám Fehér

Answer to Theme A

From the fundamental concepts related to mathematical informatics we present the **Cooley–Tukey algorithm**. The algorithm allows us to compute the discrete Fourier transform of a n long sequence in $\mathcal{O}(n \log n)$ time. Algorithm with such properties are called fast Fourier transform (FFT) algorithms. Among the many variants of the Cooley–Tukey algorithm, the **radix-2 DIT** is the simplest and most common one, and thus we present it in great detail. Other forms of the algorithm, as well as other types of FFT algorithms, are also mentioned but no rigorous construction is given.

Mathematical overview

Definition 1 (Discrete Fourier transform)**.** The discrete Fourier transform (DFT) over the n dimensional complex field is an invertible linear transformation $\mathcal{F} : \mathbb{C}^n \to \mathbb{C}^n$. The output of the function for $(x_k)_{0 \le k \le n-1}$ is defined by

$$X_k = (\mathcal{F}(x))_k = \sum_{j=0}^{n-1} x_j e^{-\frac{2\pi ikj}{n}} = \sum_{j=0}^{n-1} x_j \left[\cos\left(\frac{2\pi ikj}{n}\right) - i \sin\left(\frac{2\pi ikj}{n}\right)\right], \tag{1}$$

where the right side of the equation is due to Euler's formula and the trigonometric properties of sine and cosine.

We should mention that the sign of the exponential is sometimes taken as positive. The resulting equation is equal to the inverse of the previously defined Fourier transformation multiplied by the constant n, that is

$$x_k = \left(\mathcal{F}^{-1}(X)\right)_k = \frac{1}{n} \sum_{j=0}^{n-1} X_j e^{\frac{2\pi ikj}{n}}. \tag{2}$$

With some minor adjustments the Cooley–Tukey algorithm is capable of computing the inverse-DFT of the n long sequence, so no matter which convention we use, the algorithm remains useful.

We introduce some notations. Given $x = (x_k)_{0 \le 2n-1}$, we define $X^{[0]}$ to be the DFT of the even-indexed terms, while $X^{[1]}$ to be the DFT of the odd-indexed terms in the sequence. That is

$$X_k^{[0]} = \left(X^{[0]}\right)_k = (\mathcal{F}(x_0, x_2, \dots x_{2n-2}))_k = \sum_{j=0}^{n-1} x_{(2j)} e^{-\frac{2\pi ikj}{n}} = \sum_{j=0}^{n-1} x_{(2j)} e^{-\frac{4\pi ikj}{2n}} \tag{3}$$

$$X_k^{[1]} = \left(X^{[1]}\right)_k = (\mathcal{F}(x_1, x_3, \dots x_{2n-1}))_k = \sum_{j=0}^{n-1} x_{(2j+1)} e^{-\frac{2\pi ikj}{n}} = \sum_{j=0}^{n-1} x_{(2j+1)} e^{-\frac{4\pi ikj}{2n}}. \tag{4}$$

The key observation to make in order to verify the validity of the algorithm is

$$X_k = \sum_{j=0}^{2n-1} x_j e^{-\frac{2\pi ikj}{2n}} = \sum_{j=0}^{n-1} x_{(2j)} e^{-\frac{4\pi ikj}{2n}} + e^{-\frac{2\pi ik}{2n}} \sum_{j=0}^{n-1} x_{(2j+1)} e^{-\frac{4\pi ikj}{2n}}. \tag{5}$$

Thus for $0 \le l \le n-1$ we have

$$X_l = X_l^{[0]} + e^{-\frac{2\pi ik}{2n}} X_l^{[1]}, \text{ and } X_{n+l} = X_l^{[0]} - e^{-\frac{2\pi ik}{2n}} X_l^{[1]}, \tag{6}$$

since

$$\sum_{j=0}^{n-1} x_{(2j)} e^{-\frac{4\pi i(n+l)j}{2n}} = \sum_{j=0}^{n-1} x_{(2j)} e^{-\frac{4\pi ilj}{2n}}, \text{ and } e^{-\frac{2\pi i(n+l)}{2n}} = -e^{-\frac{2\pi il}{2n}}. \tag{7}$$

The algorithm

This basis of all variants of the Cooley–Tukey algorithm is the divide-and-conquer technique. In the radix-2 case the size of the input has to be a power of two. We may add padding zeros to fit the size constraint. From this point on we assume that input vectors are of the right size. Given an $x_{0 \le k \le n}$, where $2 \le n$, equations 3 and 4 indicate that the input vector should be divided into two parts: the even-indexed and the odd-indexed terms. Once the DFTs of the split vector are calculated, via recursive calls, the DFT of the input vector can be calculated by using equation 6. If the input vector is one dimensional, its DFT is itself. The following Python code is a working implementation of the radix-2 case, illustrating the key ideas used, but due to its performance it is not meant to be used in real world applications.

```
1  """A working fft implementation"""
2  from sympy import exp, pi, I
3  
4  def fft(terms):
5      """Implementation of Cooley-Tukey FFT algorithm"""
6      input_length = len(terms)
7      if input_length == 1:
8          return terms
9      minus_nth_root_of_unity = exp(-2*pi*I/input_length)
10     running_root = 1
11     even_indexed_terms = [terms[i] for i in range(0, input_length, 2)]
12     odd_indexed_terms = [terms[i] for i in range(1, input_length, 2)]
13     even_fft = fft(even_indexed_terms)
14     odd_fft = fft(odd_indexed_terms)
15     left_fft, right_fft = [], []
16     for k in range(0, int(input_length/2)):
17         sub_expr = running_root * odd_fft[k]
18         left_fft.append(even_fft[k] + sub_expr)
19         right_fft.append(even_fft[k] - sub_expr)
20         running_root *= minus_nth_root_of_unity
21     return left_fft + right_fft
```

To evaluate the time complexity of the algorithm, apart from the recursive calls in line 13 and 14, the algorithm runs in $\Theta(n)$ time, where n is the length of the input. Using the recurrence relation

$$T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n). \tag{8}$$

With little modification, the Cooley–Tukey algorithm can compute the inverse-DFT of a transformed vector. Similarly to equation 5, we can derive

$$x_k = \sum_{j=0}^{2n-1} X_j e^{\frac{2\pi ikj}{2n}} = \sum_{j=0}^{n-1} X_{(2j)} e^{\frac{4\pi ikj}{2n}} + e^{\frac{2\pi ik}{2n}} \sum_{j=0}^{n-1} X_{(2j+1)} e^{\frac{4\pi ikj}{2n}} \tag{9}$$

so in the 9th line, instead of $e^{-2\pi i/N}$ we have $e^{2\pi i/N}$ for the N long input, and by multiplying each term of the end result by $1/n$ given the initial input had size of n, we obtain the original x, within the same time complexity.

Importance and Usage

The FFT is widely used among different fields of engineering, computer science and mathematics.

Many popular compression methods, such as `jpeg` or `webm`, use discrete cosine transforms (DCT). Algorithms that compute DCTs in $\mathcal{O}(n \log n)$ are known as fast cosine transform (FCT) algorithms. Specialized FCT algorithms exist, that directly compute a vectors DCT, and tough the theoretical running time of such algorithms are better, on modern hardware computing DCTs via FFTs with some $\mathcal{O}(n)$ pre- and post-processing steps are often faster.

When two degree-bound n polynomials are represented in the a "extended" point-value form, over $2n$ distinct points, multiplication can be done in $\mathcal{O}(n)$ time, by simply multiplying the coefficients. Representing such a polynomial over the nth root of unity is the same as having the coefficient vectors transformed. Thus multiplying two polynomials can be done in $\mathcal{O}(n \log n)$ time by using FFTs: first transforming the polynomials to their "extended" point value form in $\mathcal{O}(n \log n)$ time, multiplying them in $\mathcal{O}(n)$, and then transforming back the result to coefficient form. Polynomial representation of the Toeplitz matrices also allow us to multiply them within the same time complexity.

FFts are also widely used in theoretical results. For a long time the **Schönhage–Strassen algorithm** was the fastest known algorithm for multiplying two large integers. With running time $\mathcal{O}(n \log n \log\ \log n)$, it is asymptotically faster than other many traditional multiplication methods, like for example the **Karatsuba algorithm**, but it only starts to outperform them for numbers with 10000 to 40000 digits, thus it is rarely used in practice. The algorithm employs modular arithmetic instead of the roots of unity (Fourier transforms can be performed in any algebric ring). In 2019 Harvey and van der Hoeven published a $\mathcal{O}(n \log n)$ algorithm, which also builds upon on FFTs. However this is a galactic algorithm, so it only bears theoretical importance.

Answer to Theme B

When studying stochastic processes, the Markov property is a commonly made assumption. This may limit its applicability, as many real life processes have memory. Excited random walks are a class of non-Markovian stochastic processes that generalize other well understood processes. As a novel field of mathematics, results described here will be theoretical in nature, but we also describe some problems and parallels, where models based on ERWs may be helpful. We are concerned with ERWs on the d dimensional integer lattice, but note that construction of ERWs can be extended to other graphs or continuous time-space processes. As rigorous introduction to the subject is somewhat tedious, we only describe the intuitive meaning behind the ERWs.

Excited random walks

ERWs can be thought of as a generalization of simple random walks (biased or symmetric) or random walks in random environments. On each site of the d dimensional integer lattice, an infinite stack of *cookies* are placed. A cookie is a $2d$ dimensional vector, that describes a probability distribution, meaning its terms are non-negative and sum up to 1. The walker is placed on the node x, and *consumes* the cookie on top, and moves according to the probability distribution described by the cookie. We call a specific configuration of cookies a *cookie-environment*, meaning that on any site of the integer lattice the cookie of a specific location is well defined. We note the set of all cooke-environment with Ω, while a specific cookie environment is usually noted with ω. As per definition, $\omega \in \Omega$. It is easy to see how certain cookie-environment can be equated to the previously given random walks, by putting cookies of the same kind on top of each other on each site. We may assume some probability distribution on the cookie-environments itself. This further generalizes the model, as if we want to investigate a specific environment $\omega \in \Omega$, we may condition that $\mathbb{P}[\omega] = 1$, but also enables us to create convenient assumptions on the model. One such assumption is that the distribution of cookie stacks on each sites are independent of each other, or the weaker assumption that the distribution of cookies are stationary and ergodic with respect to shifts on the lattice. First we present results concerning the more general model, than we focus our attention to specific ERW models.

Results concerning the general case

Answer to Theme C

Mathematical informatics can help immensely to prepare for and to handle infectious diseases. After a brief introduction to the subject, we introduce methodologies to combat each stages of an epidemic. We also take into consideration the recent case of COVID-19, and what are the protective measures that can be implemented and its effects. We should also mention that in the case of an outbreak, the tools of bioinformatics are a useful to analyse the pathogens properties and to develop vaccines but we choose to focus on analytical methods to make precautionary decisions and combat its spread.

Basic concepts

Compartment models are widely used in modeling infectious diseases. In case of an outbreak, each member of the population is assigned with a label, depending on the individuals relation with the infectious agent. An example would be the labels **S** for susceptible, **I** for infectious and **R** for recovered/removed. Individuals with the same label form a compartment. Individuals may also progress between compartments. The underlying social structure may taken to be homogenous, or can me modeled with small-world networks. Models most use differential equations to simulate the movement of people between different compartments, but stochastic frameworks have also been introduced.

Generally epidemic containment efforts have four phases: **preparedness**, **outbreak investigation**, **response** and **evaluation**. These phases contain a wide range of tasks, such as inventory management for essential medical supply and network design for transportation activities during the preparation phase, surveillance system design and implementation during the investigation phase, selection of facilities as PODs (Point of Dispensing) and scheduling of vehicles to be used for transportation during the response phase and identification of bottlenecks during the evaluation phase. Even tough being well prepared is essential, the uncertainties present at an epidemic outbreak dictate the use of real-time solutions. In such an event, decision-makers have to be presented with a wide range of models, that forecast how the epidemic will play out. In the subsequent sections we focus on two key issues they face, the diffusion of the epidemic and the distribution of emergency resources based on demand.

Epidemic modelling

Different scenarios dictate the use of different models to forecast the diffusion of the infectious agent. Properties of the model depends on the nature of the infectious agent and the policies being implemented. For example if the agent has incubation time, the appropriate compartment needs to be included. The use of quarantine also creates an additional compartment in the model, but this aspect of the model can be controlled by the acting officials. Travel and migration is another aspect which can be controlled. In most countries, the recent COVID-19 outbreak divided people into 5 groups: susceptible people (S), people during incubation period (E), infectious people (I), quarantined people (Q) and recovered people(R). The name of this model is **SEIQRS model**. We describe a version of this, based on differential equations, and taking the underlying social structure into consideration. The following figure shows the arrangement of the compartments, and how people progress between stages.

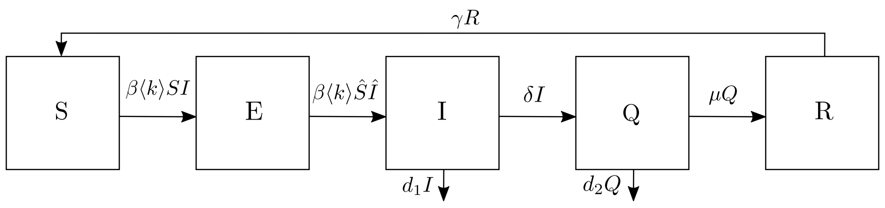
γR
S
$\beta\langle k\rangle SI$
E
$\beta\langle k\rangle \hat{S}\hat{I}$
I
δI
Q
μQ
R
$d_1 I$
$d_2 Q$

Figure 1: The SEIQRS model

We base the model on small-world networks. The numbers of susceptible people at time t is $S(t)$, and the number of people in each compartment is defined analogously. On the above figure, $\langle k\rangle$ is the average degree distribution of the network, $\hat{S}$ and $\hat{I}$ are shorthands for $S(t-\tau)$ and $I(t-\tau)$ respectively, where τ is incubation period. The rate of change, with which the number of susceptible progresses is

$$\frac{\mathrm{d}S}{\mathrm{d}t} = -\beta\langle k\rangle S(t)I(t) + \gamma R(t). \tag{10}$$

All the other differential equations can be derived analogously, by looking at figure 1.Depending on given constants, the model shows, that the described systems of differential equations has a disease free stable equilibrium at point

$P = (s, i, q) = (1 - e, 0, 0)$ if $\beta < \frac{d_1 + \delta}{\langle k \rangle (1-e)}$, where e is a number between 0 and 1, and the lowercase letters of each compartment denotes the proportion of people in the compartment with respect the whole population. Otherwise if $\beta > \frac{d_1 + \delta}{\langle k \rangle (1-e)}$,no disease free stable equilibrium exists. This leads us to an important discovery: the stability of the epidemic relies on parameters $\langle k \rangle, \delta, d_1$, and on the number of exposed people. Of these parameters, active measures can affect $\langle k \rangle$ and δ. We will refer to these as key parameters. If the reader wants to be reassured about the validity the above model (particularly about the usage of $\langle k \rangle$), and the analytical results, he/she we refer to the results of mean-field theory and Routh-Hurwiz stability criterion. Obtaining the analytical solution for the compartments remains difficult, and thus we have to rely on computation experiments to extract more meaningful data from model. Running `SciPy`, with parameters **TODO**, we obtain the following figure.

During an actual outbreak, the most important task decision-makers have, is to keep the number of infected people to the minimum. With that in mind, leaving all the other parameters fixed, we run sensitivity tests on the key parameters to obtain the following figures.

With regards to the recent COVID-19 pandemic, these figures justify the decisions of authorities that implemented strict measures. Declaring curfew and promoting self isolation are some of the most efficient ways to minimize $\langle k \rangle$, and as a consequence the number of infected people. In Wuhan, following the outbreak, authorities were quick to declare emergency and declared curfew subsequently, which lead to (compared to other countries) to the sharp decline of new cases. More efficient quarantining of individuals who came into contact with the virus also helps immensely. To achieve such a task, large scale testings have to be organized. Following the outbreak, South Korea was quick to ramp up the production of testing kits and has provided drive-trough testing facilities to maximize the number of test conducted, and minimize social contact. These measures also resulted in a sharp decline of new COVID-19 cases on the peninsula. On the other hand, countries that failed to put in place protective measures, have seen a steep upsurge of infected people. The United States government were slow to act, and as a result have had the most cases of COVID-19 infections. Governmental communication were, in our opinion, harmful. The US president repeatedly downplayed the significance of the virus, and has advocated for slowing down testing.

Emergency resource distribution

Once an epidemic outbreak has been confirmed, decision-makers face the important issue of ensuring that the appropriate amount of medical supply is provided to health-care facilities. The objective is to periodically replenish such facilities with equipment usually stored in designated stockpiles. As such, the problem relies on techniques from operations research. In this section, we will refer to stockpiles as depots, and facilities in need as demand nodes. Demand nodes are served by vehicles. There may be restrictions imposed by the models on the number of vehicles housed in each depots and on the capacity of each vehicles.

The epidemic diffusion models helps us to predict the emergency demand of each demand node during an epidemic. A specific node has demand at time t denoted by d_t, and we formulate it as

$$d_t = \langle k \rangle I(t) + Q(t) \tag{11}$$

with respect to the notation used in previous section. To supply demand, multiple distribution methods have been proposed, of which we cover two, the point-to-point distribution model (PTP mode) and the multi-depots multiple travelling salesmen model (MMTS mode). These two are considered as pure models, and optimize replenishment efforts according to different criterion.

PTP mode

In this mode, there is no restrictions on the number of vehicles in the depots, however each depots have finite capacity. Indirect supplies are note allowed, meaning one may only serve one demand node. The distribution network is described with the graph G, O, E, ω, where O represents the depots, E represents the demand nodes, E represents the the edge set, and ω represents the distance between nodes. We assume no constraints on the number of vehicles in the depots or on the capacity of each vehicle. The objective function to minimize is given by

$$\min \sum_{i \in O} \sum_{j \in V} 2\omega_{ij} z_{ij} \tag{12}$$

where z_{ij} is the decision variable taking values on $0, 1$ representing whether demand node j is served by depot i. Other appropriate restrictions, such as each demand node gets sufficiently supplied and depots serve nodes with no more resources than their capacity, are also assumed. The model described here is a simple case of integer programming, and as such classical algorithms, like the **cutting-plane method**, can be used to find the optimal solutions.

The PTP mode is a model, that is guaranteed to find the find the best possible solution in terms of delivery efficiency, which means that it gives us a lower bound on how fast demand nodes can be served. In reality however, due to the various constraints overlooked by the model, its implementation in real emergency situations is not possible. Another thing to consider, is that while the emergency nodes will be supplied quickly, the cost of the replenishment, which is proportional to the total distance travelled by the sum of all vehicles, is not taken into consideration. The next model optimizes just that.

MMTS mode

As opposed to the previous model, the MMTS mode imposes restrictions on the number of vehicles in each depot, as well as on the capacity of the vehicles. We also allow indirect supply of the demand nodes. Using the notation from the previous section, we formulate the objective function as

$$\min \sum_{i \in O \cup V} \sum_{j \in O \cup V} \omega_{ij} z_{ij}, \tag{13}$$

and we also impose other appropriate restrictions, such as the ones from the previous model and constraints on the number of vehicles leaving each depot. The model described is a case of multi-depot, multiple traveling salesmen problem, which has no known efficient solution method. Given the scale of real world emergencies, and the pressure to take immediate action, **genetic algorithms** have been introduced to obtain approximate solutions. Genetic algorithms are also used to obtain approximate solutions, when the objective function is defined in a different manner, or constraints are defined differently.

The MMTS mode is a model, that minimizes the total distance of travelled by the total number of vehicles, but does not take the timeliness of deliveries into consideration. In an emergency situation quick replenishment of the emergency nodes are essential. The use of this version of the MMTS model is not appropriate. To resolve this problem, alternative versions of the MMTS modes have been described. One such examples tries to optimizes to balance between the cost and the timeliness of the delivery, using the solution of the PTP mode as a benchmark. The alternative objective function may be formulated as

$$\min \sum_{i \in O} \Phi_i, \tag{14}$$

where Φ_i denotes the relative timeliness of delivery time compared to the PTP model, and we also impose further timeliness constraints, like that each delivery has to be between the delivery times obtained by the previous to methods.