MCMC, 260. Simulated annealing

Solving a substitution cipher

Alexander Wei

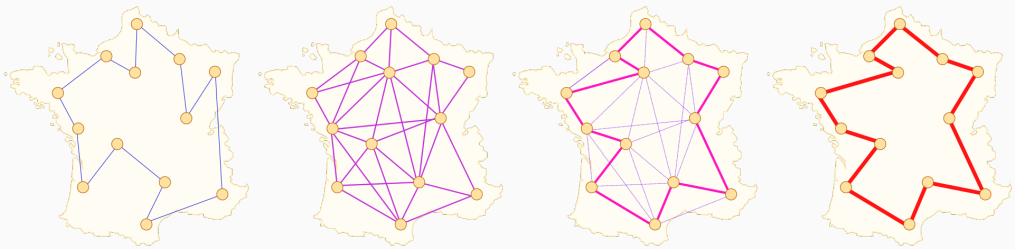
May 7, 2021

Combinatorial Optimization

Set of all solutions

$$\Omega = \{(a_1, a_2, \ldots, a_n)\}$$

Example: traveling salesman problem



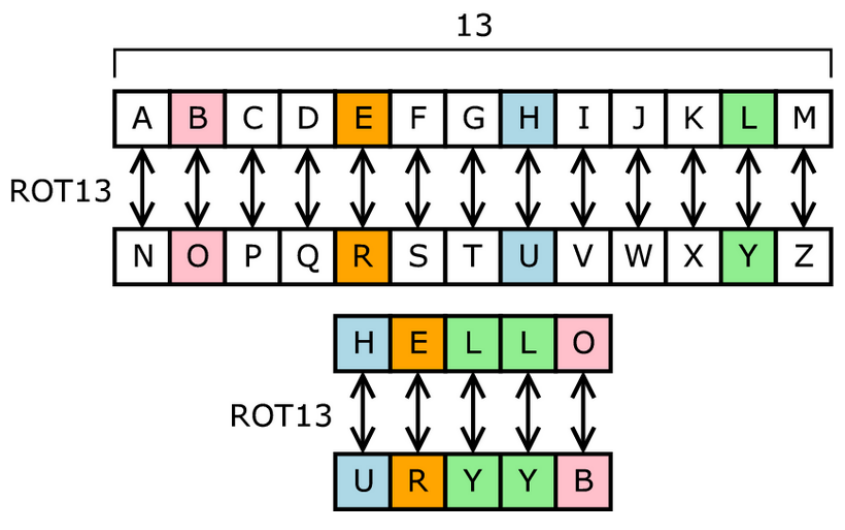
1 **2** **3** **4**

Find a solution ω that optimizes some function,

$$f(\omega) = \min\{f(\Omega)\}\,.$$

Substitution Cipher

Define a cipher C,



If Ω = all possible (English) substitution ciphers,

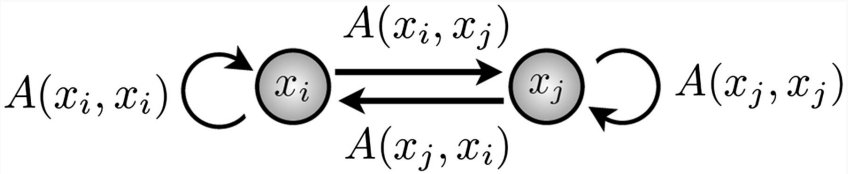
$$|\Omega| = 26!$$

Solution techniques

- Brute forcing all 26! possible keys

The exact algorithms like branch and bound, simplex method, brute force etc methodology is very inefficient for solving combinatorial problem because of their prohibitive complexity (time and memory requirement). The Evolutionary Computation algorithms are employed in an attempt to find an adequate solution to the problem. (Garg 2010)

Evolutionary algorithms



Cipher Key Annealing

Begin with a best guess,

$$G : (a, b, c, \ldots, z) \mapsto (x_1, \ldots, x_{26})$$

Applying $G_0 = (a, b, c, d) \mapsto (h, e, l, o)$ to

$$"abccd"$$

Cipher Key Annealing

Begin with a best guess,

$$G : (a, b, c, \ldots, z) \mapsto (x_1, \ldots, x_{26})$$

Applying $G_0 = (a, b, c, d) \mapsto (h, e, l, o)$ to

$$\text{"}abccd\text{"} \mapsto \text{"}hello\text{"}$$

We got lucky here.

Cipher Key Annealing

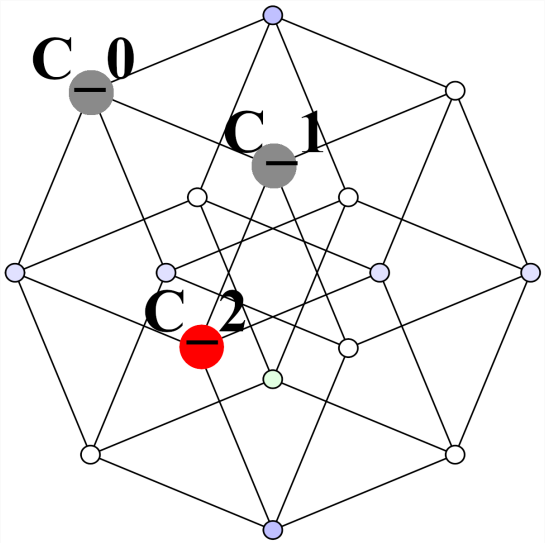
Begin with a best guess,

$$G : (a, b, c, \ldots, z) \mapsto (x_1, \ldots, x_{26})$$

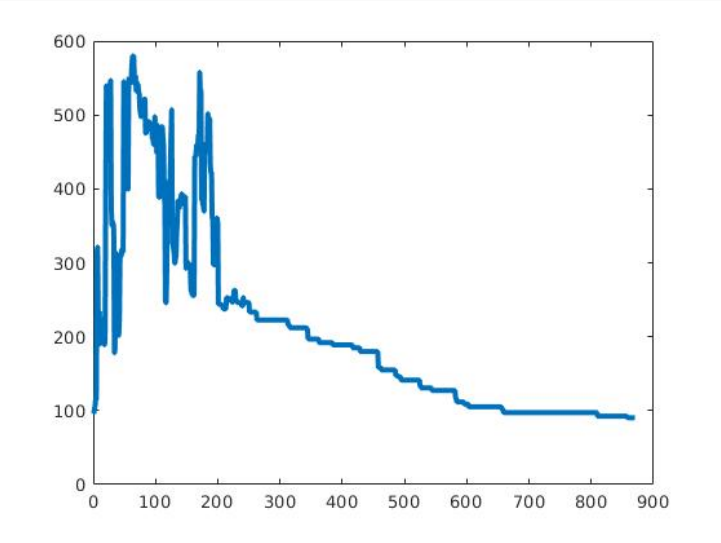
Applying $G_0 = (a, b, c, d) \mapsto (h, e, l, o)$ to

$$"abccd" \mapsto "hello"$$

We got lucky here. In most cases we need a way to evaluate a cipher-key's fitness.



increasing strictness



Going from key to key: how do we judge?

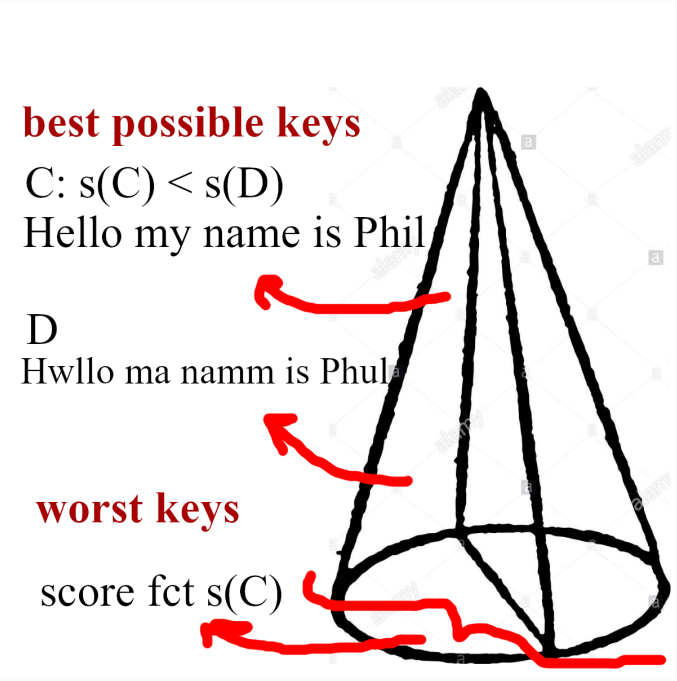
We will discuss how to propose new keys in a later slide.
For now consider Key Scoring.

"Hello, my name is Phil" is an English phrase.
What about "Hwllo, ma namm is Phul?"

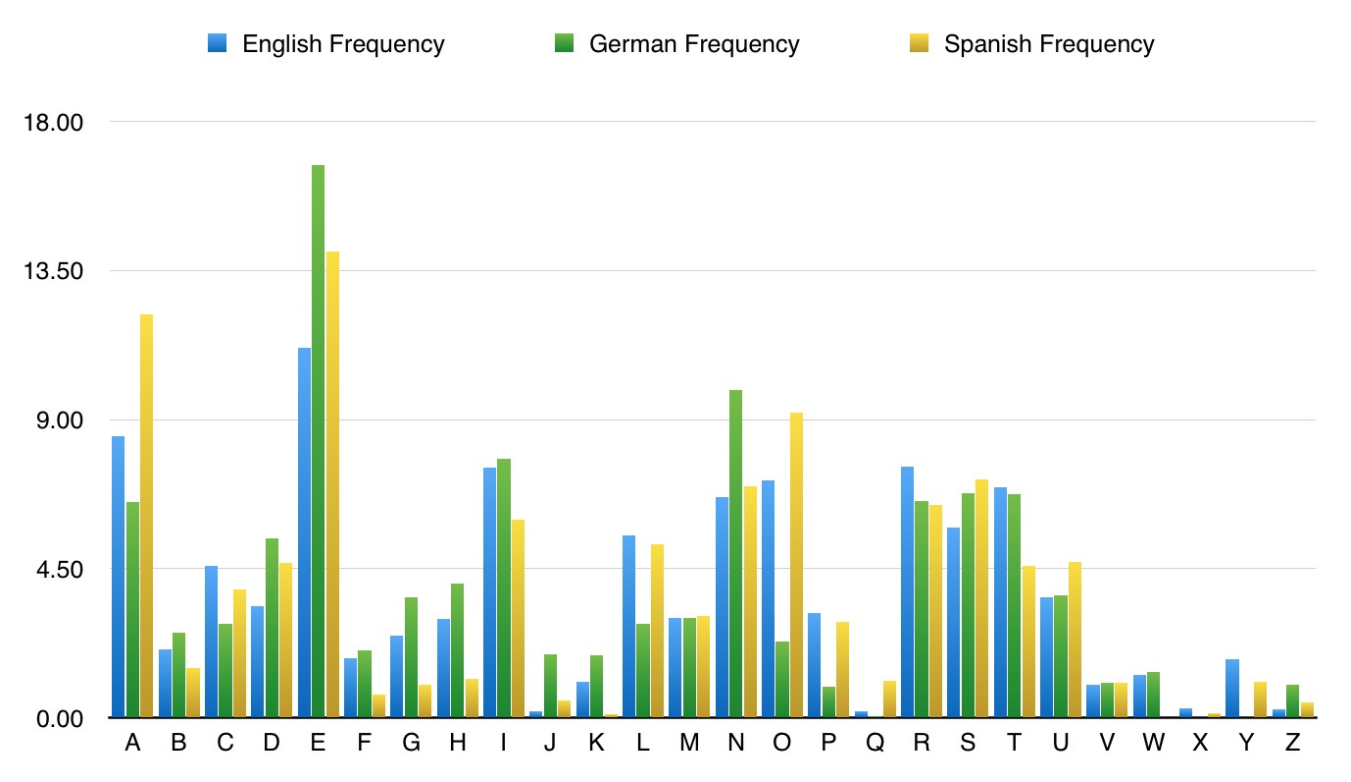
We need to make objective judgements on any text.

$$C(\text{"}\mathit{djff}\ldots\text{"}) = \text{Hello my name is Phil}$$

$$D(\text{"}\mathit{djff}\ldots\text{"}) = \text{Hwllo ma namw us Phul}$$



Key scoring via Frequency Analysis



For a score function s, (lower is better),

$$s(\text{"}hello\text{"}) < s(\text{"}xyzlo\text{"})$$

Efficient scoring

Interested in entire phrases, not just sets of letters

For a single-letter frequency score s_1,

$$s_1(\text{"}helo\text{"}) = s_1(\text{"}ehlo\text{"}) = s_1(\text{"}ehol\text{"}) = \cdots = s_1(S_4 \times \text{"}helo\text{"})$$

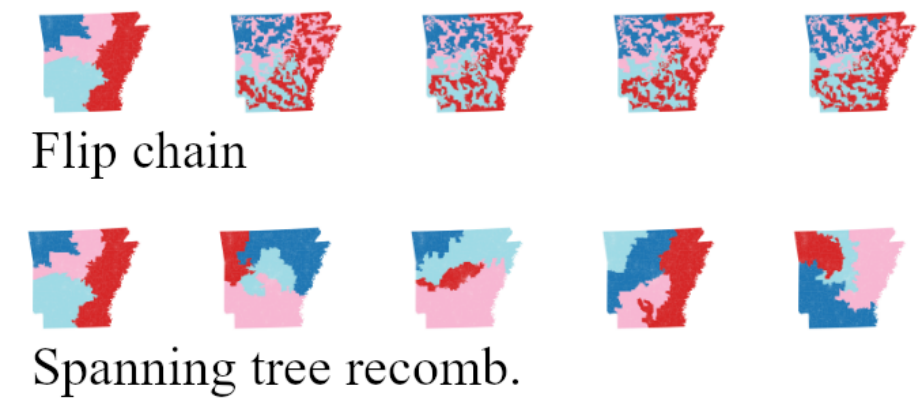
s_1 is vulnerable to a bottleneck:

$$C_1 \to \cdots \to C_k \to C_{k+1}$$

$$\text{"}xkgo\text{"}_{C_1} \to \cdots \text{"}ehlo\text{"}_{C_k} \to \{\text{"}helo\text{"}, \text{"}heol\text{"}, \ldots\}_{C_{k+1}}$$

We will benefit from a more efficient scoring method.

Recall districting



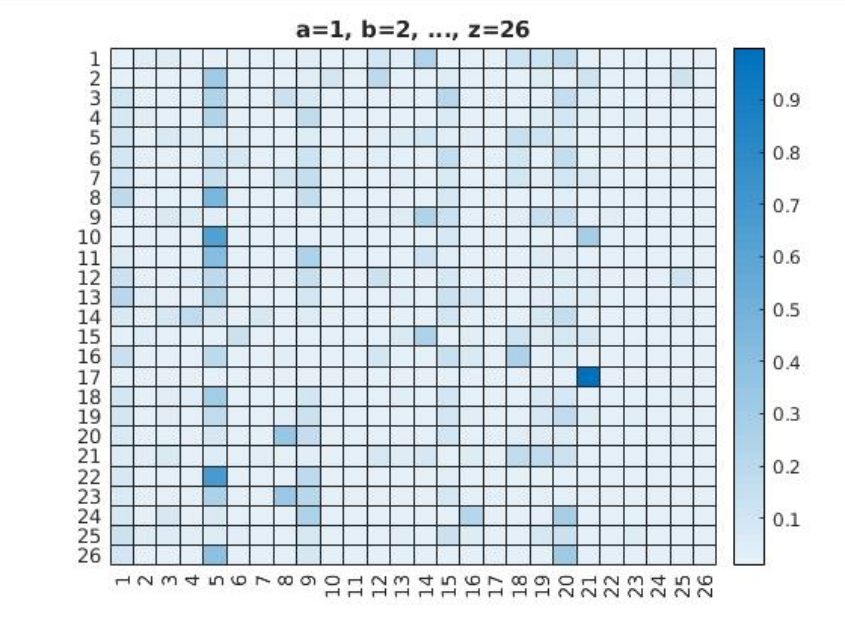
"ReCom samples preferentially from fairly compact districting plans [and] the tendency of the spanning tree process will be to produce districts without skinny necks or tentacles." (Duchin p. 16)

Forming more natural cipher guesses

- We know that *e* is the most common letter in the language
- But then should *eeeee* be a fairly common phrase?

The letter-letter (sound-sound) flow of natural language induces a canonical score: is it readable?
Considering digram frequencies takes care of both troublespots we've seen:

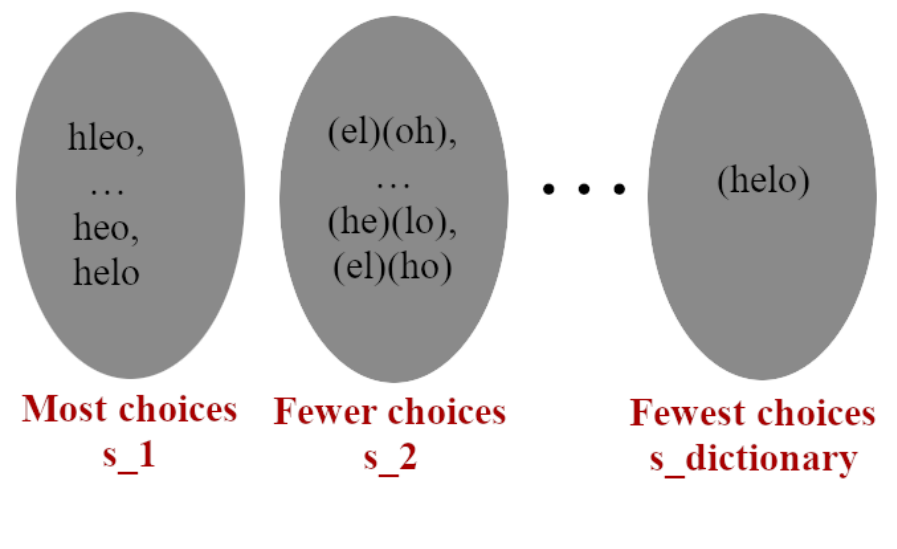
- $s_2(\text{"}helo\text{"}) < s_2(\text{"}eeee\text{"})$
- and the bottleneck
  $s_2(\text{"}helo\text{"}) < s_2(\text{"}ehlo\text{"})$



Key scoring via Frequency Analysis

Decomposition into digrams (first order transition comparison)

Hello my name is Phil $\rightarrow (h,e),(e,l),(l,o),\ldots,(h,i),(i,l)$



$\#s_1$–deciphers observing single letter freq

$> \#s_2$–deciphers observing digram freq

$> \#s_{dict}$–deciphers observing dictionary frequencies

Metropolis Hastings

Shortfalls

Demo

Voynich

Works Cited