Winning World War II with constraint programming

A practical on constraint modeling

Installation

You will use miniCP for this practical (`http://www.minicp.org/`). The code provided contains the complete solver, all the constraints you need are already implemented, but feel free to develop your own constraints if you need to.
We suggest to use an IDE, you will find all recommendation for the installation and the documentation in `http://www.minicp.org/`[1].
At each step, you will have to complete piece of code. Each part of the practical corresponds to one file, and each questions to one method.
The code to be completed is in `code/src/main/java/minicp/enigma`.

During the session, you will need to type passphrases in our Discord server, so join us at `https://discord.gg/mwdthvRuaN`.

You never used Discord before ? Don't worry, you don't need to create an account (press escape when prompted to do so). Just make sure that your username is your complete name (first name + last name). If you already have an account, please rename yourself for this server with the command `/nick first name last name`.
If you need any help, please ask on this server, we are available for questions in the voice and text channels "Général".

---

[1] The Javadoc is not up to date, especially if you need constraints that are not in the documentation take a look at the package `code/src/main/java/minicp/engine/constraints` for the complete list

Warmup (`Warmup.java`)

Let's start with a simple *transposition* code. A transposition code uses a **static** transposition table: a permutation of the alphabet (a character will always be encoded by the same other character). Here we pretend to be spies and we want to use constraint programming to encrypt and decrypt messages.

Exercice 1 `transposeEncodeChar`**:** Write a model with two variables, one standing for a text letter and another standing for the ciphered letter by the given transposition table. The search is already setup to find all feasible combinations.

Exercice 2 `transposeEncodeText`**:** Write a model with one variable per letter in the ciphertext. The solution will correspond to the result of encrypting the plaintext "IAMAREALSPYNOW" with the same transposition table.

Exercice 3 `transposeDecodeText`**:** In an attempt to hoard the rewards for themselves, the enemy has encrypted the next part of the hackathon with a transposition code! Fortunately, in a massive oversight, they didn't change the transposition table...

Write a model with one variable per letter in the plaintext. The solution will correspond to the message that was encrypted. The message is the content of the file `part2_encrypted.html`, and the output will be written in `part2.html`. Run your program and open `part2.html` in your favorite browser to continue the hackathon.

Breaking commercial Enigma (`CommercialEnigmaBreaker.java`)

Now the real fun begins, we are going to break Enigma's code !

The commercial Enigma machine is significantly harder to break. It is based on *rotors* which are rotating transposition tables. An Enigma machine has a *keyboard*, three *rotors*, a *reflector* and a *display*. When a key is pressed ('A' in the example in figure 1), the characters traverses the three rotors once *forward*, then it is transposed to another character by the reflector (a transposition table that is symmetric and irreflexive), then it traverses the three same rotors *backward* to finally appear on the display.

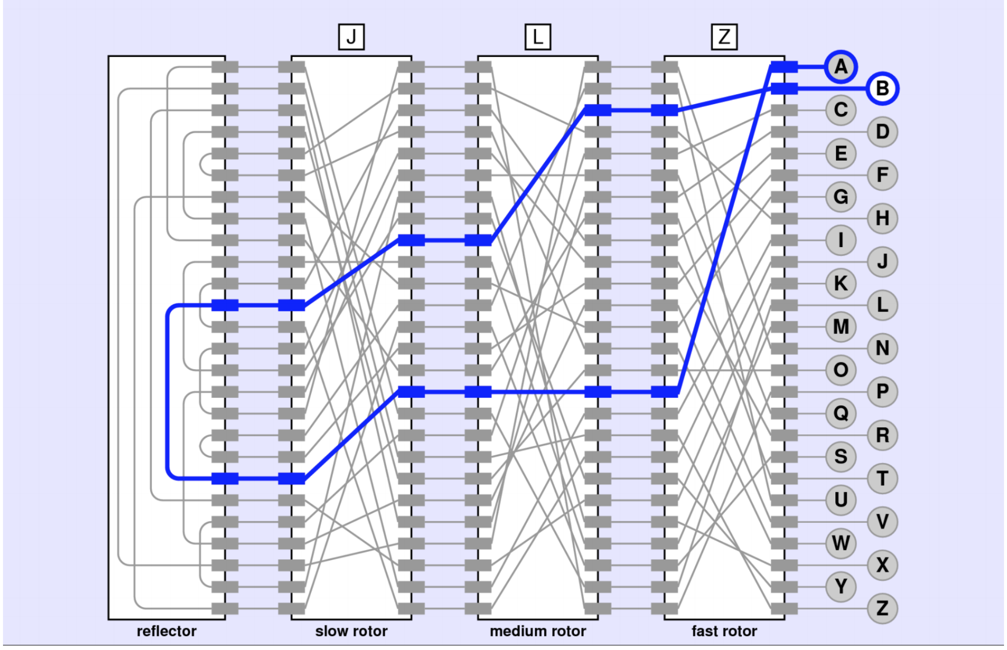

Figure 1: Enigma machine, pressing the key 'A' (credit Eric Roberts)

However, the characteristic that makes Enigma powerful is that after each such encrytption of a character, the first (*fast*) rotor rotates (as the name suggests) by one position, thus changing the transposition table. Moreover, when the fast rotor has completed a whole revolution, the *medium* rotor turns once and when the middle rotor completes a revolution, the *slow* rotor turns as well. As a result, pressing the same character twice does not yield the same output character (see figure 2).

Observe that thanks to the reflector, pressing the ciphered character's key when the machine is in the same setting as during encryption yields the original plaintext character! This makes the Enigma machine very convenient to use. The sender and the receiver only need to share the rotors initial positions. The receiver therefore simply types the ciphertext to obtain the plaintext. We call the positions of the rotors (in $\{0, \ldots, 25\}^3$) the *encryption key*, or simply the *key*.

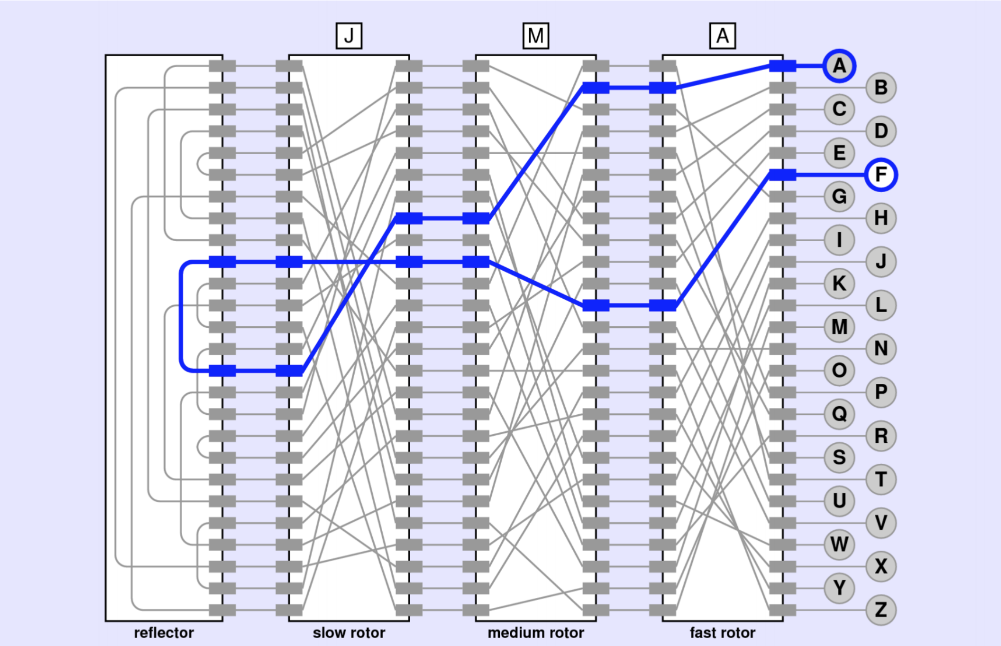

Figure 2: Enigma machine, pressing the key 'A' again (credit Eric Roberts)

Exercice 1 **`singleRotorEmulate`:** Write a constraint program which "emulates" a machine with a single rotor (and no reflector). The array of variables plaintext stands for the message to encrypt, ciphertext for the encrypted message and the variable key stands for the position of the rotor when encrypting the first character.
Hint : One way to do it is to compute all the possible positions for the rotor.

Exercice 2 **`singleRotorBreak`:** We want to decipher the encrypted message "ROOETWKCYDXDKDGOLUIYRVKFLPD" that was obtained by using a single rotor. We have a model of the rotor (`perm[0]`) that was used because our navy found it when boarding a U-pedal-boat, however we do not know what *key* was used to encrypt the message.

To give us the edge, our intelligence agency worked day and night and arrived at the following conclusion: the ennemy is devious, yet always very polite, so the message must start with a greeting ("HI" or "HELLO"). This is a *crib*: we know that the letter 'H', when encoded in first position, corresponds to a 'R'.

Write a constraint program which decipher a message ("ROOETWKCYDXDKDGOLUIYRVKFLPD") encrypted with a single rotor (`perm[0]`), given a crib ("H"). You have to define the branching scheme, either you start branching on the plaintext variables, or you start branching on the key variable. There will be several solutions satisfying all the constraints, however there is only one solution which is plain english.

Exercice 3 `multipleRotorsBreak`**:** We have intercepted a new ciphered message (“HHWBGGLRDUSPEBBINYGKSGEAMQOSMEEHTGJKNTGCQURVMNEUBCTWGMHBK”), however, the ennemy has used not one but two rotors (a fast `perm[0]` and a medium rotor `perm[1]`).

Being grossly arrogant, there are good reasons to believe that the ennemy sent a message starting with a demonstrative pronoum (“THE”, “THAT”, “THOSE”, etc.), so the crib is “TH”. Write a constraint program to decipher it. Your program should work with k rotors.

Exercice 4 `commercialEnigmaEncode`**:** It appears that the machine that our navy found on the earlier U-pedal-boat, and which our expert analyst first classified as a “minitel” (whatever that is) was in fact an Enigma machine. With the knowledge of its mechanism (given by `rotors` and `reflector`), we will be able to break the ennemy code! First we should be able to emulate it. Write a constraint program which, given a message `plaintext` and an array of (3) rotor positions `key`, computes the encrypted message in the array of variables `ciphertext`.

Exercice 5 `commercialEnigmaBreak`**:** Alright, now we can eventually break the ennemy code! We managed to lure an ennemy homming pigeon by skilfully disguising a Sussex cottage into a Bavarian castle. Here’s the message that was tied to its ankle: “NVGSEZEUFBEVTWCLPZARJKLFKNPFMBZYCZSLCKQKOUUZIHFHVAFDHPZOLIQTFSLJROUNGDXYLONQBLWKXZGRAH”. Our expert psychoanalysts tell us that the ennemy is so self-centered[2] that the message is likely to start with “WE”. Write a constraint program to decipher it.

In the meantime, our spies discovered that an enemy’s special agent has infiltrated our Discord server: the Enigma Bot is working for them ! Our agents thinks that the message you just decoded is in fact a password for it !

Send the password to the bot in a private message[3] to see its reaction !

---

[2] the subsequent dispute about the root cause being mother or father-related hasn’t stopped yet, but that’s beside the point

[3] right-click on its name to send it a private message

Breaking military Enigma[4] (`MilitaryEnigmaBreaker.java`)

Congratulation, you went through the first step of this practical session ! Now let's get down to business...

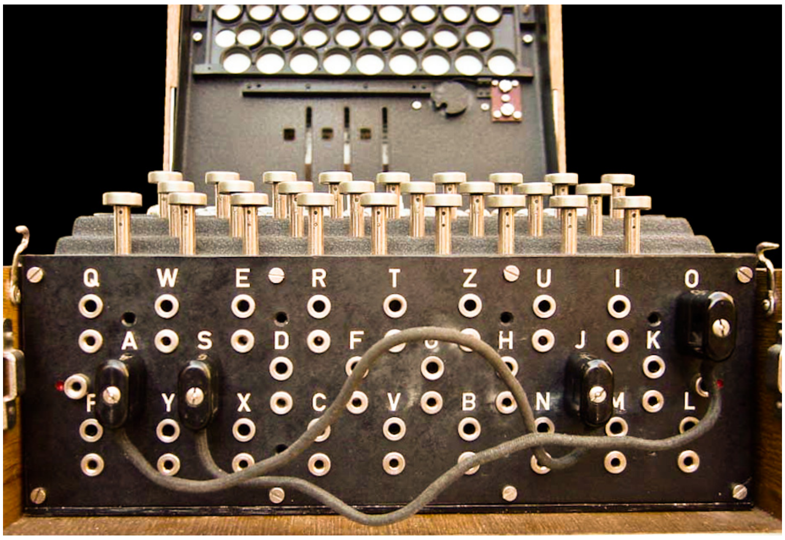

Figure 3: Enigma machine's plugboard (Wikimedia commons)

Unfortunately, the Enigma machine in its military version was upgraded by the addition of a *plug board* (see figure 3). If we refer to Figure 1 or 2, the plug board can be seen as a new device between the keyboard and the first rotor. Before and after going through the 3 rotors, the character signal is transposed using the chosen configuration of the plug board, which is a symmetric permutation. Only ten pairs of characters can be transposed, the remaining six characters are transposed to themselves. With this device, the encryption key now becomes the rotors' positions *plus* the plugboard pairings. This increases the number of potential keys from $26^3 = 17576$ to $150,738,274,937,250$...

Exercice 1 **`militaryEnigmaBreak`:** We will try to generalise the model used to break the commercial version by adding variables for the plug board. However, since the problem is much harder we shall retrict the alphabet to 16 characters ("ACDEGHILMNORSTUW") and also restrict the plugboard so that it is only capable of transposing two pairs of characters (the remaining 12 characters are transposed to themselves). The message is [too long to state in the text] and the crib is "CONGRATULATION". Write a constraint program to break it!

The goal here is to come up with the most efficient constraint program as possible. One way is to add *implied constraints.*

---

[4] or trying to...

An observation that might be useful is that the plugboard is static, it always transposes the same pairs of characters no matter where they are in the message.

With the proper additional constraints and the correct branching scheme, you should be able to decode the message in a few minutes.

Once again, you will have to send the decoded message by a private message to Enigma Bot in the Discord server.