Miniproyecto 1 - Criptoaritmética

Jessenia Piza Londoño y Laura Alejandra Salazar Pérez

23 de septiembre de 2021

1. Introducción

La inteligencia artificial se ha caracterizado por su capacidad de resolver problemas específicos. Algunos de sus objetivos principales incluyen la deducción y el razonamiento matemático. Un problema que involucra tener ambas aptitudes es la criptoaritmética, la cual podemos considerar como un rompecabezas cuyo propósito es reemplazar las letras del alfabeto por dígitos bajo ciertas restricciones.

Ahora bien, uno de los métodos para resolver este tipo de problemas es el razonamiento lógico, sin embargo, no es sencillo. Esto debido a que cada vez va aumentando su dificultad; ya sea incrementando el número de variables, operadores u operandos.

Otra forma de resolver el problema, es hallar todas las posibles combinaciones para 10 dígitos.

$$\mathrm{Cp} = \frac{10!}{(10 - \mathrm{Ld})!}$$

donde,

Cp = cantidad de combinaciones posibles y Ld = cantidad de letras diferentes.

Note que, en el peor de los casos (Ld = 10) existen 3.620.800 combinaciones posibles y, en ocasiones, puede que exista una única solución, varias o ninguna. Así, reafirmamos la complejidad de resolver este tipo de problemas.

2. Métodos

La solución al problema requiere una definición formal previa a la implementación en `python`, que es la siguiente:

- **Estado inicial:** : Situación del entorno desde el cual comienza el juego. En el caso de la criptoaritmética, ninguna letra ha sido reemplazada por algún dígito.
- **Posibles acciones:** Descripción de las posibles acciones. En este caso, reemplazar todas las veces que aparezca un dígito que no haya aparecido en el problema.
- **Prueba de objetivo(s):** Permite determinar si el juego se termina cuando todas las letras tienen un dígito asociado, cumple con las condiciones aritméticas (la operación correctamente verificada) y la condición adicional de que dos letras no pueden tener el mismo valor numérico (comprendido entre 0 y 9).
- **Función de utilidad(s):** Definida sólo cuando el juego se termina. En este caso sin importar el resultado el costo siempre es 0, dado que todas las soluciones son igual de válidas.

Adicionalmente, se creó un algoritmo llamado `sol_algoritmo()` el cual resuelve el problema tomando todas las permutaciones posibles de este. A continuación se encuentra su pseudocódigo:

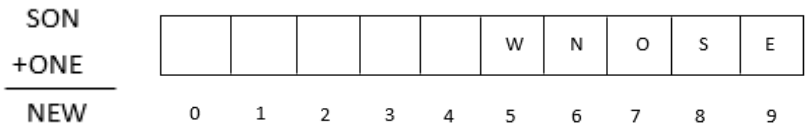
```
FUNCTION sol_algoritmo()
    digitos -> range(10)
    FOR permutacion IN permutations(digitos, len(lista_letras))
       solucion -> dict(zip(lista_letras, permutacion))
       IF test_objetivo(solucion)
           RETURN solucion
    RETURN None


FUNCTION test_objetivo(estado)
    palabras_sol -> []
    FOR palabra IN palabras
        FOR letra IN palabra
            palabra -> palabra.replace(letra, str(estado[letra]))
        TRY
            num_palabra -> int(palabra)
            palabras_sol. append(num_palabra)
```

```
        EXCEPT
            RETURN False
    RETURN sum(palabras_sol[:-1] == palabras_sol[-1]
```

3. Resultados

Dado que en el problema se presentan los dígitos del 0 al 9, se utiliza un diccionario de longitud máximo 10 (este depende de la cantidad de letras que hayan en el sistema a resolver), donde los valores son índices del 0 a 9 para asociarlos a cada una de las letras(llaves). Además, cuando una letra no tiene un valor asignado, este será `None` hasta que se le asigne un dígito. A continuación veremos un ejemplo.



(a) Problema (b) Asignación

Figura 1: Asignación dado un problema

En el ejemplo podemos notar que al momento de cambiar los valores de las letras en el diccionario creado, si la letra se encuentra repetida, el algoritmo la escribe una sola vez. Además, se les asigna un valor, en este caso un dígito, de lo contrario el valor de la letra es `None`.

4. Discusión

Cada uno de los algoritmos de búsqueda permiten que cualquier problema criptoaritmético sea resuelto, sin embargo, el costo computacional que les lleva resolver el problema, es bastante alto.

Ahora bie, podemos decir que el algoritmo de búsqueda `Best_first_search` es uno de los más ágiles (esto debido a que tal como su nombre lo dice, encuentra la primera solución y el algoritmo para de inmediato) y por ende vamos a centrarnos en este para explicar la solución que presenta con el problema planteado.

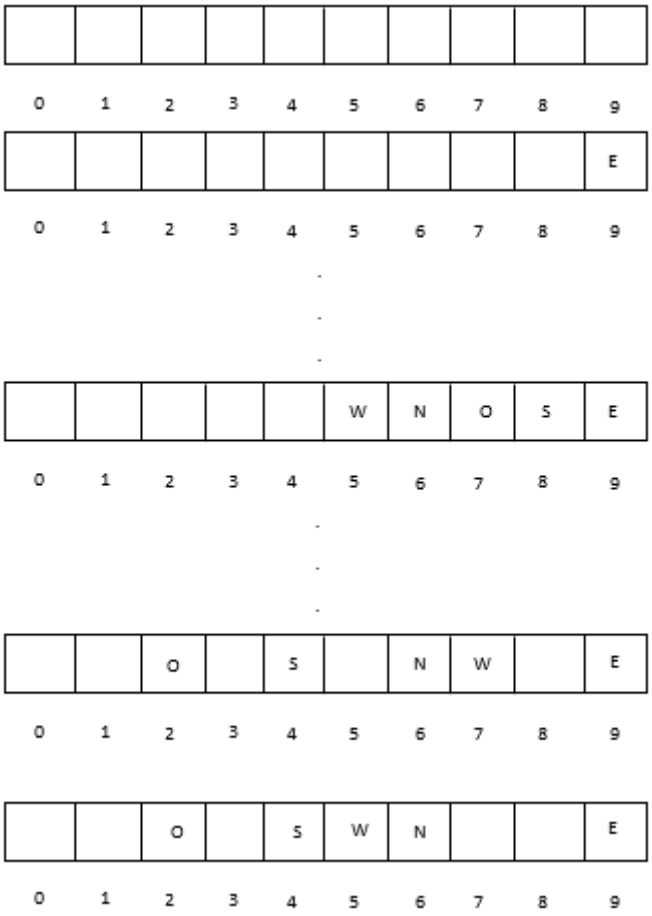


Figura 2: Algoritmo

En la figura anterior, podemos ver como va cambiando el diccionario, conforme aumentan las iteraciones. En un principio, todas las letras tienen su valor igual a `None`, después, se van asignando los dígitos del 0 al 9 de forma aleatoria es decir el valor va cambiando de `None` al dígito otorgado y así sucesivamente hasta que se encuentra la solución en la cual todas las letras(llaves) tienen un valor distinto a `None` y se cumple la condición de suma en este caso.

5. Conclusiones

- Dado que el problema de la criptoaritmética está muy correlacionado con las permutaciones, los algoritmos de búsqueda no logran dar una solución rápida si se habla de un problema criptoaritmético complejo. Esto debido al costo computacional que genera el problema.
- Los resultados obtenidos demuestran que es adecuado utilizar algoritmos de busqueda para

resolver problemas criptoaritméticos, debido a que exploran el espacio de estados convergiendo rápidamente a una de las soluciones del problema, siempre que exista.

- Se implementó de forma satisfactoria un programa de agente para la resolución del problema criptoaritmética que es imbatible.

Referencias

[1] Mounier. M, Aguirre. F, Barbosa. M, *Resolución de Problemas Criptoaritméticos Utilizando Algoritmos Genéticos*