Diseño y Análisis de Algoritmos

Proyecto # 3: Kevin el encargado

Laura Victoria Riera Pérez
Marié del Valle Reyes

Cuarto año. Ciencias de la Computación.
Facultad de Matemática y Computación, Universidad de La Habana, Cuba

13 de junio de 2023

I. Repositorio del proyecto

https://github.com/computer-science-crows/algorithms-design-and-analysis

II. Definición inicial del problema

Kevin ha sido puesto al frente de la comisión de la facultad que elegirá las fechas de las pruebas de los k cursos que se dan en la facultad.

Cada curso tiene una cantidad de pruebas determinadas que quiere poner, y propone para esto, por ejemplo, los días { 17, 34, 65 y 87 } del curso escolar, si vemos a este como una sucesión de días en los que se imparten clases. Para mostrarse flexibles, los cursos a veces elaboran más de una propuesta incluso.

Por un problema de desorganización las propuestas se regaron y ahora no se sabe que curso propuso que propuesta, pero ya Kevin esta cansado de tanta gestión. Kevin quiere elegir k propuestas que ninguna quiera poner pruebas el mismo día que las otras, así supone que todo el mundo estará contento, ayude a Kevin.

III. Definición en términos matemático - computacionales

I. Problema X

IV. Soluciones implementadas

I. Backtrack

Como primera solución al problema fue implementado un *backtrack*. Esta es una solución correcta, ya que genera todas las combinaciones posibles de k propuestas y verifica si cumplen con la restricción de que no haya dos cursos con exámenes el mismo día. La complejidad temporal de este algoritmo es $O(n^k)$, donde n es el número de propuestas. Dado que n puede ser grande, este algoritmo puede no ser práctico para valores grandes de n.

En una computadora de 32GB de RAM, intel core i7-11na generación, se puede resolver para una cantidad máxima 5 ciudades con 5 aristas. Dicha solución puede ser encontrada en src/solutions/backtrack_solution.py.

II. Metaheurística

En la resolución del problema se utiliza el algortimo genético como metaheurística. La implementación se basa en representar las soluciones como una lista binaria del mismo tamaño que la lista de propuestas de entrada. En esta representación, un valor de 1 en un índice indica que esa propuesta ha sido seleccionada, mientras que un valor de 0 indica que no ha sido seleccionada.

El objetivo del algoritmo genético es encontrar una lista binaria con exactamente k valores de 1, lo que representa la selección de k propuestas sin que ninguna de ellas tenga días comunes. Para lograr esto, se emplean dos operadores principales: mutación y cruzamiento.

La operación de mutación consiste en intercambiar posiciones en la lista binaria. Esto permite explorar nuevas soluciones al cambiar las propuestas seleccionadas. Por ejemplo, si se encuentra un 1 en un índice, se puede intercambiar ese valor con otro índice donde haya un 0 y viceversa.

El operador de cruzamiento, o crossover, se aplica a dos listas binarias. Dado un índice i, se intercambian todos los elementos hasta el índice i de una lista con los elementos de la otra lista. Esto permite combinar características de ambas soluciones y explorar nuevas posibilidades.

La función de evaluación se encarga de determinar la calidad de una solución. En este caso, devuelve un número positivo si la lista binaria es válida, es decir, si cumple con las restricciones de que las propuestas seleccionadas no tengan días comunes. Por el contrario, devuelve un número negativo si la lista binaria no es válida.

El algoritmo genético continúa iterando hasta encontrar una lista binaria válida que cumpla con las restricciones. Se generan nuevas soluciones aplicando mutación y crossover, y se seleccionan las mejores soluciones para la siguiente generación, utilizando la función de evaluación como criterio de selección.

V. Generador de casos de prueba

En *src/app/generator.py* fue implementado un generador, el cual recibe una cantidad *s* de muestras a producir, genera valores random con el formato de entrada de los algoritmos implementados, halla la solución óptima con *backtrack* y las guarda en *json/test_cases.json*. Se generaron 3000 casos de prueba.

VI. Tester

En *src/app/tester.py* fue implementado un tester, que recibe una función y prueba el desempeño de la misma en cuanto a si obtuvo la solución óptima o no, y el tiempo que demoró en hacerlo, comparando con los casos de prueba obtenidos con el generador. Además, estos resultados se guardan en un *.json* con el nombre de la función en la carpeta tests. La solución implementada fue testeada para todos los casos de prueba generados y puede encontrarse en *json/tests/corruption_strategy_solution.json* .

VII. Comparación de soluciones implementadas

Referencias

[1] Cormen, Thomas H. y otros. *Introduction to Algorithms*. The MIT Press. 4ta Edición. Cambridge, Massachusetts. 2022.