Análisis Artículo Científico

Marco Antonio Arcidiacono Alepuz

July 28, 2023

1 Introducción

En este documento se va a discutir sobre el artículo "A Simple and Effective Iterated Greedy Algorithm for the Permutation Flowshop Scheduling Problem" de Rubén Ruiz del Departamento de Estadística e Investigación de la Universidad Politécnica de Valencia y Thomas Stüzle del Department of Computer Science de Darmstadt University of Technology.

En dicho artículo se habla de un algoritmo voraz e iterativo para resolver el problema de Permutation Flowshop Scheduling (PFSP) donde n trabajos independientes tienen que ser procesados en m máquinas en el menor tiempo posible. Cada trabajo requiere ser procesado en cada máquina durante un tiempo determinado en el mismo orden. Por lo tanto, el objetivo es encontrar una planificación para procesar los trabajos para que se optimice según un criterio (en este caso, el tiempo).

2 El Algoritmo

El algoritmo se basa en la búsqueda local y en la voracidad de ir paso a paso escogiendo la mejor solución vecina posible hasta que se generen vecinos que no mejoren la solución actual.

Concretamente, lo que se hace es calcular los tiempos de cada trabajo en ser procesados en todas las máquinas y se ordenan en orden descendiente de tiempo generando una secuencia de trabajos. Después, se hace una fase de destrucción, es decir, se eligen trabajos de manera aleatoria, se borran de la secuencia y se evaluan las mejores posiciones para estos trabajos en la secuencia. Ahora hay que construir de nuevo, es decir, se introducen esos trabajos en la secuencia formando una nueva. Si esta nueva secuencia mejora el tiempo de la anterior, se vuelve a empezar con la nueva secuencia, pero si no hay mejora se termina y nos quedamos con esa secuencia como solución al problema.

Este algoritmo tiene una complejidad de $O(n^3m)$ siendo n el número de trabajos y m el número de máquinas, es decir, es de complejidad cúbica. Esto se debe a que hay que recorrer la lista de trabajos, evaluando a su vez que trabajos eliminar y después hay que recorrer la lista de nuevo para insertarlos, todo esto hay que repetirlo hasta encontrar una solución.

3 Ventajas

Este algoritmo parte de una solución trivial como es ordenar los trabajos por el tiempo que tardan en ser procesados por las máquinas, lo cual hace que otras soluciones que son claramente peores vayan a ser descartadas nada más empezar haciendo que sea seguro que se reduzca el tiempo de ejecución.

También, al ser una solución trivial, es fácil observar que es mejorable y que hay soluciones vecinas con pocos cambios que son mejores, lo cual hace que, de nuevo, no se exploren todas las soluciones y se ahorre en tiempo de ejecución.

4 Desventajas

La principal desventaja es que no se puede asegurar llegar a una solución óptima ya que hay soluciones que son evitadas desde el principio y solo se miran soluciones vecinas.

Otra desventaja es que no se puede escapar de máximos locales, una vez se llega a una solución cuyas soluciones vecinas exploradas no son mejores, el algoritmo se termina sin explorar más soluciones sin saber a ciencia cierta si se puede llegar a otro máximo.

Por último, en caso de que haya muchos trabajos, el tiempo de ejecución se puede elevar demasiado como para poder ejecutar este algoritmo.

5 Posibles alternativas

Una posible alternativa muy buena para problemas como estos, podrían ser los algoritmos genéticos, pues es fácil pensar en cómo representar el problema, cómo cruzar distintas secuencias de trabajos de manera aleatoria pero que tengan sentido, cómo se deberían mutar dichas secuencias y qué secuencias son las mejores y deberían continuar en el proceso. Esto implicaría en mayor uso de memoria para poder hacer un multiarranque y así tener una población, cosa que podría funcionar con el algoritmo del artículo.

Como se acaba de mencionar, otra alternativa es simplemente hacer este algoritmo multiarranque, creando soluciones iniciales con distintos criterios para explorar más soluciones en un tiempo parecido. Además, esto permitiría evitar máximos locales.

6 Conclusión

En conclusión, este algoritmo funciona correctamente por lo que muestran los autores en la evaluación experimental y encuentra buenas soluciones, aunque no garantice encontrar las óptimas.