ANÁLISIS COMPARATIVO DE LA APLICACIÓN DE DISTINTOS OPERADORES GENÉTICOS AL PROBLEMA DE JOB SHOP SCHEDULING

Alfonso H., Salto C.
Proyecto UNLPAM – 09/F009
Dep. de Informática – Fac. de Ingeniería - UNLPAM
Calle 110 esq. 9 - (6360) Gral. Pico – La Pampa – Argentina
e-mail: {alfonsoh,saltoc}@ing.unlpam.edu.ar
phone: +54 2302 422780/422372, Ext. 6302

Gallard R.
Proyecto UNSL-338403
Departamento de Informática - UNSL
Ejército de los Andes 950 – Local 106
(5700) – San Luis – Argentina
e.mail: rgallard@unsl.edu.ar
phone: +54 2652 420823

Abstract
Este trabajo describe un estudio de distintas soluciones al problema del job shop scheduling basado en algoritmos genéticos. La característica fundamental es la utilización de decodificadores incorporada a la representación del cromosoma. El objetivo del trabajo es comparar la performance de los distintos tipos de operadores de crossovers que se pueden usar con esta representación.

1. Introducción

El problema de scheduling job shop clásico [2] se puede describir por m máquinas diferentes y n jobs diferentes que se deben procesar para formar un schedule. Cada job está compuesto por un conjunto de operaciones, las cuales tienen un determinado orden sobre las máquinas. Cada operación se caracteriza por la máquina que necesita y por el tiempo de procesamiento en la misma. El problema es determinar la secuencia de operaciones sobre las máquinas con el objetivo de satisfacer ciertas restricciones y optimizar algunos criterios. Uno de esos criterios puede ser minimizar el makespan, es decir, el tiempo requerido para completar todos los jobs.

La especificación de una representación apropiada de un schedule es decisiva para la performance del algoritmo evolutivo. Una de las principales dificultades que se plantea en un problema de scheduling es determinar la representación de cromosoma a utilizar. Hay dos opciones básicas a la hora de elegir la representación del cromosoma para el scheduler [10]. La primera es una lista de tareas a ser asignadas. Una opción alternativa a la de usar una secuencia de tareas es directamente adoptar un schedule como cromosoma. Esto puede parecer una representación pesada; se deben utilizar operadores complicados y específicos, pero que tiene ventajas cuando se tratan problemas del mundo real complicados como es el caso de scheduling.

Es posible clasificar todas las opciones como varios problemas de scheduling sobre la base de la representación del cromosoma. Esas representaciones caen en dos categorías [1]:

- *Representaciones indirectas*: donde una transformación desde la representación del cromosoma a una schedule de producción legal ha de ser realizada por un decodificador especial; sólo entonces la solución individual puede ser evaluada. Luego, esas representaciones puede dividirse en representaciones específicas del problema e independientes al dominio [3].
- *Representaciones directas*: donde se usa el schedule de producción como un cromosoma. Usualmente esta representación requiere algún operador específico del problema.

2. Representación del cromosoma

Dentro de las representaciones indirecta hay una representación en la que se usan decodificadores. Un cromosoma se representa como una lista de n enteros; el i-ésimo componente del vector es un número entero en el rango $[1 .. (n - i + 1)]$. La idea detrás de una representación ordinal es la siguiente: Hay una lista ordenada, la cual sirve como punto de referencia para decodificar el vector al seleccionar el ítem apropiado desde la lista. Por ejemplo, para una lista de ítems L = (1,2,3,4,5,6), el vector $\langle 4,3,4,1,1,1 \rangle$ es decodificado como la siguiente secuencia de ítems $\langle 4,3,6,1, 2,5 \rangle$. La secuencia de ítems

decodificada es una permutación de los elementos de la lista (se obtiene una secuencia de operaciones válidas), a la cual llamaremos vector decodificado.
El algoritmo evolucionario genera el schedule correspondiente a la secuencia de jobs, el primer job en la lista es asignado primero, luego se toma el segundo, y así sucesivamente.
La principal ventaja de la representación con decodificadores es que se puede aplicar el crossover clásico: *one-point* (1-PX). Por consiguiente, se puede extender a *two-point* (2-PX), *three-point* (3-PX) y *uniform* (UX) crossover.
Varios autores ([4], [6],[8]) describieron la construcción de operadores de reordenamiento que combinan características de inversión y crossover en un único operador. Estos operadores son similares, aplicables a representaciones de cromosomas basadas en permutaciones de los jobs. Por consiguiente, son aplicables al vector decodificado. Esos tipos de operadores son:

- *parcial-mapped crossover* (PMX): fue propuesto en Goldberg [6]. Puede ser visto como una extensión del two-point crossover para un string binario a una representación basada en permutaciones. Usa un procedimiento de reparación especial para resolver la ilegitimidad causada por el two-point crossover.
- *order crossover* (OX): propuesto por Davis [4] . Puede verse como una variación de PMX con un procedimiento de reparación diferente.
- *cycle crossover* (CX): propuesto por Oliver, Smith y Holland [8] Es esencialmente una clase de crossover uniforme para una representación basada en permutaciones junto con un procedimiento de reparación. Toma algunos alelos desde uno de los padres y selecciona los alelos restantes desde el otro padre. Los alelos desde el primer padre no son seleccionados en forma aleatoria y sólo se eligen aquellos alelos que definen un ciclo acorde a las correspondientes posiciones entre los padres.
- *one-cut-point crossover (OCPX):* propuesto por Reeves [9]. Es una clase de one-point crossover para una representación basada en permutaciones. Define un punto de corte, toma todos los alelos del primer padre hasta esa posición, luego completa el hijo con los alelos restantes en el orden de aparición en el otro padre.

3. Descripción del Experimento

Teniendo en cuenta la representación de cromosoma detallada previamente, se generaron dos algoritmos. Dentro de cada uno se tiene el cromosoma con una representación ordinal y su correspondiente vector decodificado. Uno de los algoritmos concebidos, al que llamaremos CODIF, trabaja con los operadores de recombinación (1-PX, 2-PX, 3-PX y UX) aplicados al cromosoma en forma directa mientras que el otro, DECODIF, toma aquellos operadores de recombinación apropiados para emplear con el vector decodificado (PMX, OX, CX y OCPX). La característica que los diferencia es la aplicación de los operadores de recombinación adecuados para cada uno de los casos.
Los algoritmos se corren en forma independiente considerando cada tipo de crossover a la vez, según corresponda. Además, se analiza el comportamiento de los algoritmos si se selecciona en forma aleatoria el tipo de operador de crossover a utilizar cuando se deba cruzar los padres para generar los hijos. Cabe destacar que la población inicial para cada experimento ha sido la misma.
Como cada algoritmo aplica los operadores genéticos sobre distintas representaciones se tuvieron en cuenta distintos tipos de operadores de mutación. Cuando se trabaja con el cromosoma codificado en CODIF se utiliza un operador de mutación que consiste en seleccionar una posición del cromosoma en forma aleatoria y cambiar ese alelo con el valor generado dentro del rango permitido. En DECODIF, al usar el vector decodificado se utiliza una mutación bit swap, se seleccionan en forma aleatoria dos posiciones dentro del cromosoma y se intercambian los valores de los alelos, de este modo el schedule resultante es válido.
Ambos algoritmos, CODIG y DECODIF, se contrastaron por un conjunto de instancias seleccionadas para el problema de job shop scheduling (JSSP). Se utilizaron 6 instancias, de las cuales se conocen sus makespan óptimos:

Instancia	la01	la06	la012	la015	la016	abz6	abz7
Tamaño	10 x 5	15 x 5	20 x 5	20 x 5	10 x 10	10 x 10	20 x 15
Óptimo	666	926	1039	1207	945	943	657

Las variables de performance elegidas son:

- **Ebest** = (Abs(*opt_val* - best value)/*opt_val*)100
  Error porcentual de los mejores individuos hallados cuando se comparan con el valor óptimo de makespan conocido o estimado, *opt_val*,. Da una medida de cuán lejano es el fitness del mejor individuo al compararlo con *opt_val*.
- **Epop** = (Abs(*opt_val*- pop mean fitness)/*opt_val*)100
  Error porcentual del fitness medio de la población cuando se compara con *opt_val*. Da una medida de cuán lejos está el fitness medio de *opt_val*.

4. Conclusiones

Aunque los resultados obtenidos por el algoritmo CODIF son muy similares al evaluar el ebest promedio, se puede observar una tendencia a favor de one-point crossover en algunas instancias y de three-point crossover en otras. Para la mayoría de las instancias, uniform crossover presenta los peores resultados. No se observa una mejora considerable al hacer la elección arbitraria de los distintos tipos de crossover aplicados.

Al analizar los resultados encontrados con DECODIF, en el caso de optimización de schedule, la combinación de los cuatro posibles crossover a aplicar da los mejores resultados en la mayoría de las instancias, seguido por CX y OPCX, siendo PMX el que arroja los peores resultados. Esto puede ser explicado al examinar cómo esos operadores preservan el orden de la información.

Al realizar un análisis comparativo entre los mínimos ebest promedio de cada algoritmo, se observa que los mejores resultados son obtenidos cuando se trabaja con el vector decodificado, esto se debe a las características de los operadores aplicados.

5. Bibliografía

1. Bagchi, S., Uckun, S., Miyabe, Y., Dawamura, K., Exploring Problem-Specific Recombination Operators for Job Shop Scheduling, in Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kauffman Publishers, San Mateo, CA 1991 pp. 10-17
2. Bruns, R., Scheduling. In Th. Bäck, D. B. Fogel, and Z. Michalewicz, editors. Handbook of Evolutionary Computation, Oxford University Press, New York, and Institute of Physics Publishing, Bristol, chapter F1.5, pp F1.5:1-F1.5:9, 1997.
3. Bruns, R., Direct chromosome representation and advanced genetic operators for production scheduling. In Forrest ICGA93 , pp. 352-359.
4. Davis, L., Applying Adaptive Algorithms to Epistatic Domains, In Proceedings of the International Joint Conference on Artificial Interlligence, pp. 162-164, 1985.
5. Gen, M., Cheng, R., Genetic Algorithms & Engineering Design, Wiley Series in Engineering Design an Automation, John Wiley & Sons, Inc.1997.
6. Goldberg, D. y Lingle R., Alleles, loci and traveling salesman problem, in Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, Mj, 1985
7. Husbands, P., Mill, F., y Warrington, W., Genetic algorithms, production plan optimisation, and scheduling. In Schwefel and Maenner PPSN91 , pp 80-84.
8. Oliver, I., Smith, D., y Holland, J., A study of permutation crossover operators on the traveling salesman problem, in Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, Mj, 1985 pp. 224-230
9. Reeves C., A Genetic Algorithms for Flow Shop sequencing, en Computer and Operations Research, 1995, Vol. 22, pp, 5-13.
10. Syswerda, G., Schedule Optimization Using Genetic Algorithms, en Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991, pp, 332-349.