Capítulo VII

ALGORITMOS EVOLUTIVOS AVANZADOS PARA JOB SHOP SCHEDULING

7.1. INTRODUCCIÓN

En este capítulo se presentan tres distintas implementaciones planteadas para resolver el problema de job shop scheduling usando algoritmos evolutivos. Principalmente, esas distintas opciones difieren en cuanto el tipo de representación de cromosoma utilizado; esto trae como consecuencia la selección de los operadores genéticos factibles de aplicar. En la primer propuesta, se introduce un algoritmo evolutivo con características de multirecombinación y prevención de incesto para la obtención de resultados para esta clase de problema. Otra de las soluciones analiza el comportamiento de una implementación con representación basada en reglas de prioridad en la cual se incorporan semillas generadas por heurísticas a la población inicial. Finalmente, se presenta un algoritmo evolutivo con representación basada en operaciones usando el método de crossover de *parcial exchange*.

7.2. OPCIÓN CON MULTIPLICIDAD Y PREVENCIÓN DE INCESTO

Una solución posible al problema del job shop scheduling es usar conjuntamente, en un algoritmo evolutivo, multiplicidad y prevención de incesto [123].

En un problema de job shop scheduling, una representación basada en jobs consiste de una lista de jobs, donde el schedule se construye acorde a esa secuencia de jobs. En esta representación se consideran permutaciones de jobs, por lo tanto se deben usar operadores genéticos adecuados para una permutación, tal como *partially-mapped crossover*, *order crossover* y *cycle crossover*. Otra forma de encarar el problema cuando hay permutaciones es usando *decodificadores*. Aquí un cromosoma se

Cromosoma	Permutación
1 1 1 1 1 1	1 2 3 4 5 6
4 3 4 1 1 1	4 3 6 1 2 5
4 2 3 2 1 1	4 2 5 3 1 6

Tabla 7.1 Correspondencia entre cromosomas y permutaciones.

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 esta representación 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 actual. Por ejemplo, para una lista de ítems L = (1,2,3,4,5,6), en la tabla 7.1 se observa la correspondencia entre cromosomas y permutaciones.

Un decodificador es una transformación desde el espacio de representaciones a una zona del espacio de soluciones factible, el cual incluye transformaciones entre representaciones que son sometidas al proceso de evolución y representaciones que constituyen la entrada a la función de evaluación. Esto simplifica la implementación y produce hijos factibles bajo ciertos métodos de crossover convencionales, evitando el uso de penalidades o acciones de reparación.

El método de multiplicidad usado en esta opción es MCMP (*múltiples crossovers sobre múltiples padres*), permite la múltiple recombinación de múltiples padres bajo el uso de algunas de las variantes de SX, esperando un adecuado balance entre la explotación y la exploración del espacio del problema.

En cuanto al método de prevención de incesto utilizado es EIP (*prevención de incesto extendida*) en donde se previene el incesto entre individuos que pertenecen a la misma familia. Este método combinado con multiplicidad genera un algoritmo al que denominaremos MCMPIP (MCMP *Incest Prevention*).

En la figura 7.1 se muestra pseudocódigo de un procedimiento que tiene en cuenta multiplicidad (cuando se usa un número de padres $n_2 > 2$) y prevención de incesto entre miembros de una misma generación o una generación consecutiva, es decir entre hermanos ó entre padres e hijos.

7.2.1. DESCRIPCIÓN DEL EXPERIMENTO

Tanto MCMP como MCMPIP fueron contrastados para un conjunto de instancias seleccionadas para el JSSP, con el uso de una representación con decodificadores. Para

```
procedure multiple  parent, multiple crossover, incest prevention
    begin
     int mating_pool[number_of_parents]
        //arreglo para almacenar los padres seleccionado
     int children_pool[number_of_cross]
      //arreglo para almacenar los hijos creados
     for 1 to pop_size
      select indiv-1 C(t)
      mating_pool[1] = indiv-1
      i = 2
      while (i <= nro_de_padres)
        repeat
           select indiv-i C(t)
        until (no_es_pariente(mating_pool, indiv-i))
        //control de ancestros no comunes y de unicidad de los padres en
        //mating_pool
        matting_pool[i] = indiv-i
        i = i + 1
      end while
      recombinar usando MCMP y mutar
      asignar individuos desde mating_pool a children_pool
      selecccionar el mejor individuo de children_pool
      construir C'(t)
     end for
    end procedure
```

Figura 7.1 Procedimiento que une MCMP y EIP

cada instancia, se realizaron 15 corridas diferentes. Los experimentos corresponden a diferentes combinaciones de cantidades de crossovers n_1 y cantidades de padres n_2, diferentes tipos de SX y métodos de selección de los hijos generados. En los algoritmos evolutivos se usó selección proporcional al fitness del individuo para la selección de los individuos que conformarán el matting pool. Los mejores individuos se retuvieron por elitismo. En ambos métodos se fijó los mismos parámetros a excepción del tamaño de la población. En los experimentos se consideró un tamaño de la población de 50

individuos para MCMP y de 120 individuos para MCMPIP. Esta diferencia en el tamaño de la población se basa en el hecho de que en MCMPIP, cada vez que se elige un padre para adicionar al *matting_pool*, se debe controlar que ya no sea parte del mismo y que además no mantenga alguna relación ancestral con los individuos ya incorporados al matting pool, por lo tanto para poder llevar a cabo este control es necesario contar con mayor cantidad de individuos para no caer en un bucle infinito, debido a que ningún individuo cumple con las restricciones impuestas. Se implementaron tres métodos de scanning crossovers y para la inserción en la próxima generación se optó por $n_3 = 1$; se contrastaron dos métodos de selección de un hijo entre los n_1 generados en cada cruzamiento: elegir el mejor hijo y realizar la elección en forma aleatoria. El número de crossovers y padres se definió en $1 \leq n_1 \leq 4$ y $3 \leq n_2 \leq 5$, respectivamente. Se usó la mutación *big-creep* como operador de mutación, la cual consiste en reemplazar el valor de un gen por otro dentro del rango permitido. El número máximo de generaciones se determinó en 500 y las probabilidades de crossover y mutación se fijaron en 0.8 y 0.01, respectivamente. Estos valores se precisaron como la mejor combinación de probabilidades luego de muchos ensayos de prueba y error. Se usaron 6 instancias [97] con óptimos bien conocidos (tabla 7.2).

Las siguientes variables de performance fueron elegidas para contrastar los resultados encontrados por cada algoritmo:

- ✓ ***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 está 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*.

7.2.2. RESULTADOS

Para todas las instancias y en todas las combinaciones posibles de (n_1 ,n_2), cuando se debe decidir por seleccionar un hijo en forma aleatoria o retener el mejor generado de los n_3 generados, la mejor perfomance de los algoritmos se logra al escoger el mejor

Instancia	Tamaño	Óptimo
la06	15×5	926
la12	20×5	1039
la15	20×5	1207
la16	10×10	945
abz6	10×10	943
abz7	20×15	657

Tabla 7.2 Instancias.

hijo obtenido de los n_1 generados en cada cruzamiento. Por lo tanto los resultados que se muestran de aquí en adelante son los obtenidos bajo este criterio de selección.

En las tablas 7.3, 7.4 y 7.5 se detallan los resultados para la instancia *la06*: los valores correspondientes al makespan de los mejores individuos hallados, al *ebest* medio y al *epop* medio, respectivamente. En la tabla 7.3 se puede observar que, para esta instancia, tanto MCMP y MCMPIP hallan el óptimo bajo algún método de SX en el 70% de los experimentos realizados. No se alcanza el óptimo en el caso en el cual se aplica un único crossover. En general, los mejores resultados se obtienen cuando se incrementa el número n_1 de crossovers, independientemente del número n_2 de padres usados en el cruzamiento. En la tabla 7.4 se puede observar que los valores de *ebest*

Mejores Valores de Makespan						
n_2/n_1	MCMP			MCMPIP		
	USX	FBSX	OBSX	USX	FBSX	OBSX
3/1	933	929	930	934	932	934
3/2	926	926	926	926	926	926
3/3	926	926	926	926	926	926
3/4	926	926	934	926	926	926
4/1	928	936	926	934	930	930
4/2	926	926	929	926	926	926
4/3	926	926	926	926	926	926
4/4	926	926	926	926	926	929
5/1	936	930	926	935	934	930
5/2	930	926	931	926	926	926
5/3	926	926	926	926	926	928
5/4	926	926	926	926	926	926
Min	926	926	926	926	926	926
Aver.	927.9	927.4	927.7	928.1	927.5	927.7
Max	936	936	934	935	934	934

Tabla 7.3 Valores de makespan de los mejores individuos hallados bajo cada método para diferentes combinaciones de (n_1, n_2) para la instancia *la06*.

ebest						
n_2/n_1	MCMP			MCMPIP		
	USX	FBSX	OBSX	USX	FBSX	OBSX
3/1	0.76	0.32	0.43	0.86	0.65	0.86
3/2	0.00	0.00	0.00	0.00	0.00	0.00
3/3	0.00	0.00	0.00	0.00	0.00	0.00
3/4	0.00	0.00	0.86	0.00	0.00	0.00
4/1	0.22	1.08	0.00	0.86	0.43	0.43
4/2	0.00	0.00	0.32	0.00	0.00	0.00
4/3	0.00	0.00	0.00	0.00	0.00	0.00
4/4	0.00	0.00	0.00	0.00	0.00	0.32
5/1	1.08	0.43	0.00	0.9719	0.8639	0.43
5/2	0.43	0.00	0.54	0.00	0.00	0.00
5/3	0.00	0.00	0.00	0.00	0.00	0.22
5/4	0.00	0.00	0.00	0.00	0.00	0.00
Min	0.00	0.00	0.00	0.00	0.00	0.00
Aver.	0.21	0.15	0.18	0.22	0.16	0.19
Max	1.08	1.08	0.86	0.97	0.86	0.86

Tabla 7.4 Valores de *ebest* hallados bajo cada método para diferentes combinaciones de (n_1, n_2) para la instancia *la06*.

epop						
n_2/n_1	MCMP			MCMPIP		
	USX	FBSX	OBSX	USX	FBSX	OBSX
3/1	0.13	0.14	0.13	0.14	0.14	0.14
3/2	0.12	0.12	0.12	0.13	0.13	0.12
3/3	0.12	0.11	0.11	0.11	0.12	0.11
3/4	0.11	0.11	0.11	0.11	0.11	0.11
4/1	0.14	0.14	0.13	0.14	0.14	0.13
4/2	0.12	0.12	0.12	0.13	0.12	0.12
4/3	0.12	0.12	0.11	0.12	0.12	0.11
4/4	0.11	0.11	0.11	0.11	0.11	0.11
5/1	0.13	0.14	0.13	0.14	0.14	0.13
5/2	0.12	0.12	0.12	0.12	0.13	0.12
5/3	0.12	0.12	0.11	0.12	0.12	0.11
5/4	0.11	0.11	0.11	0.11	0.11	0.11
Min	0.11	0.11	0.11	0.11	0.11	0.11
Aver.	0.12	0.12	0.12	0.12	0.12	0.12
Max	0.14	0.14	0.13	0.14	0.14	0.14

Tabla 7.5 Valores de *epop* hallados bajo cada método para diferentes combinaciones de (n_1, n_2) para la instancia *la06*.

varían entre 0 y 1.08, la mejor performance es para MCMP con FBSX seguido por MCMPIP con el mismo método de crossover. El peor comportamiento de ambos algoritmos se obtiene bajo el método USX. Los valores de *epop* están dentro del rango 1.48 a 28.31. Los valores mínimos de *epop* varían entre 1.48 a 3.84 y se hallan con $n_1 = 4$ y $n_2 = 5$, esto significa que la población entera está más concentrada alrededor del mejor individuo hallado cuando el n_1 y n_2 son grandes.

En las figuras 7.2 a 7.5 se resume la información sobre las instancias *la06*, *la12*, *la15* y *abz6*. Las abreviaturas M-U, M-F y M-O indican la combinación MCMP con USX, FBSX y OBSX, respectivamente; mientras que MI-U, MI-F y MI-O indican MCMPIP con USX, FBSX y OBSX, respectivamente.

De la observación de las figuras se ve que a medida que las instancias se tornan más complejas se observa una pérdida de performance sobre ambos métodos. Este hecho refleja que a medida que la complejidad del espacio de búsqueda se incrementa es más difícil para ambos métodos hallar schedules cercanos al óptimo. Esto es particularmente cierto cuando se incrementa el número de máquinas, por ejemplo en las instancias *la16*, *abz6* y *abz7*. Este efecto se puede esperar porque el espacio codificado (permutaciones de jobs) corresponde a sólo una parte del espacio de soluciones total y es el precio a ser pagado por producir hijos que sean factibles. A pesar de esto, la performance es mejor que la de usar algoritmos evolutivos simples y se puede mejorar usando representaciones alternativas.

De las figuras 7.2 y 7.3 se observa que los valores de *ebest* varían entre 0.00 a 1.08 para la instancia *la06*, de 0.77 a 4.81 para *la12*, desde 2.07 a 9.78 para *la15* y desde 11.03 a 11.98 para *abz6*. Se detectan pequeñas variaciones a favor de MCMPIP en cuanto a la performance, pero para todos los experimentos los mejores resultados se obtienen bajo ambas opciones cuando se aumenta el número de crossovers para una determinada cantidad de padres.

Se observa con mayor frecuencia un rango pequeño de valores para los métodos de crossover USX y OBSX independientemente de la opción de recombinación utilizada.

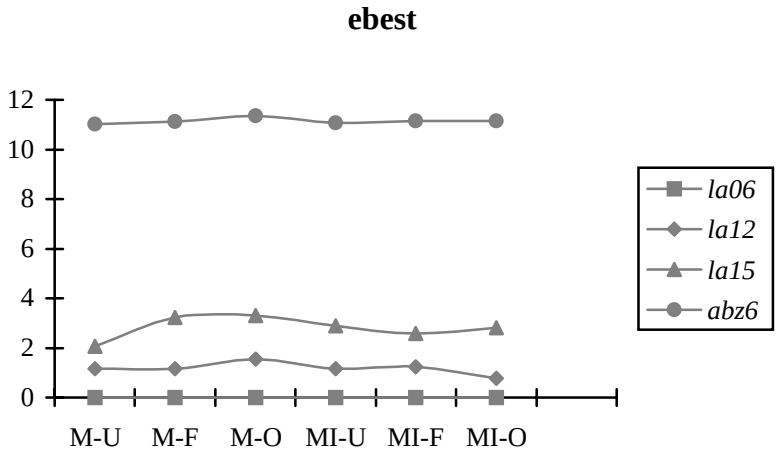


Figura 7.2 Mínimo *ebest* bajo MCMP y MCMPIP.

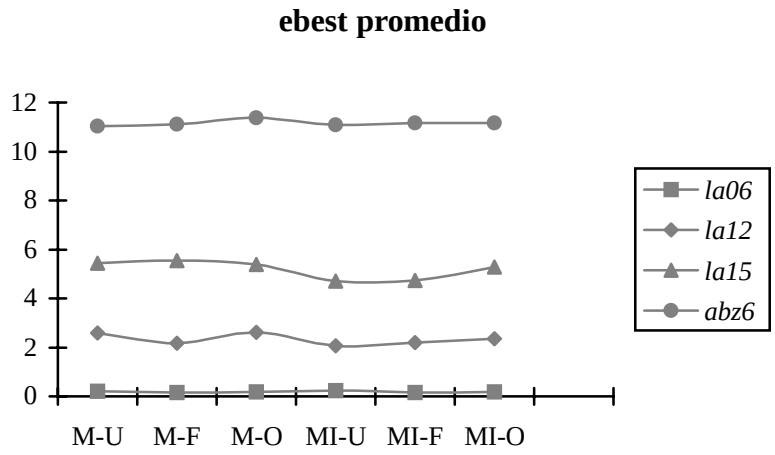


Figura 7.3 *ebest* promedio bajo MCMP y MCMPIP.

Analizando las curvas tanto de *epop* como de *epop* promedio de las figuras 7.4 y 7.5, se observa que estos valores varían desde 1.48 a 28.31 para *la06*, desde 3.85 a 29.92 para *la12*, desde 2.82 a 29.40 para *la15* y desde 14.50 a 47.99 para *abz6*. Nuevamente, los mejores resultados se obtienen bajo ambas combinaciones cuando se aumenta el número de crossovers dado un número determinado de padres. MCMPIP presenta una mejora en los errores poblacionales respecto a MCMP, salvo para la instancia *abz6*.

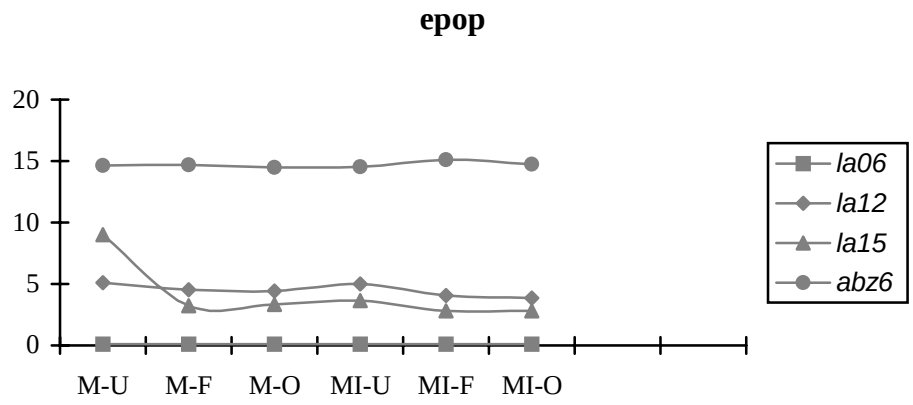


Figura 7.4 Mínimo *epop* bajo MCMP y MCMPIP.

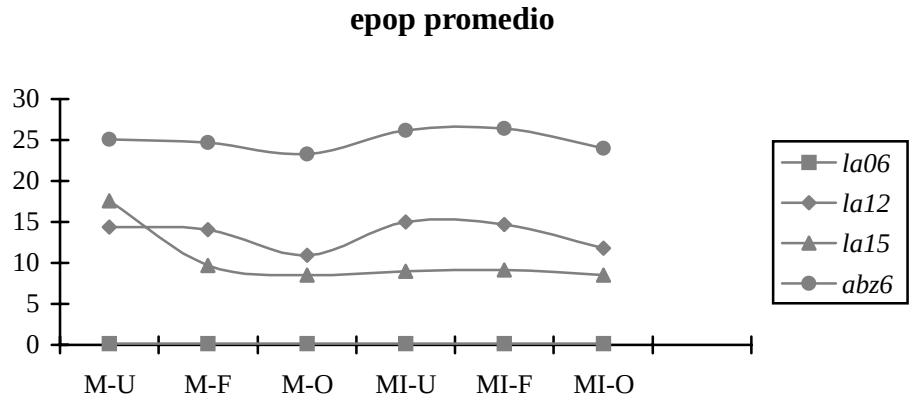


Figura 7.5 *epop* promedio bajo MCMP y MCMPIP.

7.2.3. CONCLUSIONES

La opción presentada introduce una opción con multirecombinación MCMP y su combinación con prevención de incesto MCMPIP, para resolver el problema de job shop scheduling. El uso de decodificadores para problemas con permutaciones permite la implementación directa del crossover convencional evitando la necesidad de penalidades o acciones de reparación. Esto es fundamental para la aplicación de las opciones de multirecombinación ya que el método SX, como inicialmente se define, no es directamente aplicable a representaciones con permutaciones. Luego de una serie de experimentos se puede concluir que:

- ✓ Se obtienen mejores resultados cuando se aumenta el número n_1 de crossovers para un número n_2 de padres. En muchos casos la población final está centrada alrededor

del individuo con fitness más alto, proveyendo un conjunto de schedules alternativos, los cuales pueden ayudar a tratar con sistemas dinámicos.

- ✓ Para cualquier asociación (n_1, n_2) USX y OBSX proveen las diferencias más pequeñas, entre los menores y mayores valores, cuando se lo compara con FBSX.
- ✓ Para la instancia *la06*, ambos métodos hallan el óptimo usando cualquiera de los métodos SX. Para problemas más grandes, hay un incremento en el error porcentual que siempre permanece debajo del 12%.

7.3. OPCIÓN CON REGLAS DE PRIORIDAD DE DESPACHO

Otra solución posible al problema del job shop scheduling es una implementación de algoritmos evolutivos utilizando reglas de despacho. Una particularidad de este algoritmo es la de utilizar semillas generadas por otro método heurístico, las cuales se insertan en la población inicial, para obtener una mejora en la eficiencia del mismo.

Para un problema clásico de job shop scheduling de *n* jobs y *m* máquinas, un cromosoma codifica una secuencia de reglas de despacho. Se incorpora al algoritmo evolutivo (EA) el algoritmo de Giffler y Thompson, donde el EA se usa para evolucionar una secuencia de reglas de prioridad de despacho y el algoritmo de Giffler y Thompson se usa para deducir un schedule desde la codificación de reglas de prioridad de despacho.

La heurística de reglas de despacho, descripta en 6.1.1, se utiliza para generar individuos, los cuales serán incorporados en la población inicial como semillas, en alguna posición determinada en forma aleatoria. En cada población inicial, se incorpora sólo una semilla. El procedimiento de generación de la población inicial de un EA sufrirá una pequeña modificación (la inclusión de la semilla), el pseudocódigo se presenta en la figura 7.6.

Al momento de determinar como será la conformación de un determinado cromosoma en la población inicial en referencia a las reglas de despacho, se elige, para cada gen del cromosoma una regla de prioridad de despacho en forma aleatoria, dentro del conjunto de reglas de despacho consideradas.

Una consideración adicional para elevar la performance de un algoritmo evolutivo es utilizar una heurística convencional en la generación inicial para producir buenos

```
procedure genera-pop-inicial
begin
 pos = nro_aleatorio(1, pop_size ); //la función aleatorio genera un número entre dos
                                    dados
for i = 1 to pop_size
   begin
   if not( i = pos)
     begin
       generar indiv-i
       insertar indiv-i en C(0)
     end
   else
     crear las semillas  //utiliza la heurística de prioridad de despacho
     incorporar semilla en C(0)
end for
end procedure
```

Figura 7.6 Modificación del proceso de generación de la población inicial

cromosomas que serán usados como semillas. Un cromosoma se genera con la heurística de prioridad de despacho, mientras que los restantes cromosomas se generan en forma aleatoria. A esta opción se la denomina PR-Sem-EA y se compara a esta opción híbrida con una en la cual no incorpora semillas en la población inicial, PR-EA.

7.3.1. DESCRIPCIÓN DEL EXPERIMENTO

En este experimento se contrasta el comportamiento dos algoritmos evolutivos, PR-EA y PR-Sem-EA, ambos utilizando representación basada en reglas de despacho.

Ambos algoritmos se analizaron para un conjunto de instancias seleccionadas para el JSSP. Se realizaron 15 corridas diferentes para cada instancia. Para la selección de los individuos que conformarán el matting pool, en el algoritmo evolutivo se usó selección proporcional al fitness del individuo. El mejor individuo se retuvo por elitismo. En ambos algoritmos se consideraron los mismos valores para los parámetros. En los experimentos el tamaño de la población fue de 50 individuos. El operador de crossover

utilizado es uno de los más sencillos, one-point crossover, el cual para esta representación siempre produce hijos factibles. El operador de mutación fue el *big-creep*, el cual consiste en reemplazar el valor de un gen por otro generado aleatoriamente. El número máximo de generaciones fue fijado en 500 y las probabilidades de crossover y mutación fueron establecidas en 0.65 y 0.01, respectivamente. Se usaron 7 instancias [97] con óptimos bien conocidos (tabla 7.6). Las reglas de despacho consideradas son las que se mencionan en la sección 6.1, a excepción de EDD, la cual no fue implementada.

Las variables de performance elegidas para contrastar los resultados encontrados por cada algoritmo son *ebest* y *epop*, además de incorpora:

- ✓ ***gbest*** = Identifica la generación donde el individuo (retenido por elitismo) fue encontrado.

7.3.2. ANÁLISIS DE LOS RESULTADOS

En la tabla 7.7 se muestran las veces que cada algoritmo ha encontrado el óptimo en cada instancia. En la instancia *ft6* ambos algoritmos encuentran la misma cantidad de veces el óptimo; en *la01*, PR-Sem-EA encuentra en una oportunidad más el óptimo que PR-EA; en *la06*, en el 100% de los casos ambos algoritmos hallan el óptimo; en *la12* en el 87% de los casos PR-EA encuentra el óptimo para esta instancia contra el 93% de PR-Sem-EA. En instancias más complejas como *abz6* y *abz7*, los algoritmos no encuentran en ninguna oportunidad el óptimo correspondiente a la instancia.

Instancia	Tamaño	Óptimo
ft6	6×6	66
la01	10×5	666
la06	15×5	926
la12	20×5	1039
la15	20×5	1207
abz6	10×10	943
abz7	20×15	657

Tabla 7.6 Instancias.

	PR-EA	PR-Sem-EA
ft6	9	9
la01	1	2
la06	15	15
la12	13	14
la15	0	0
abz6	0	0
abz7	0	0

Tabla 7.7 Cantidad de veces que cada algoritmo encuentra los óptimos de cada instancia.

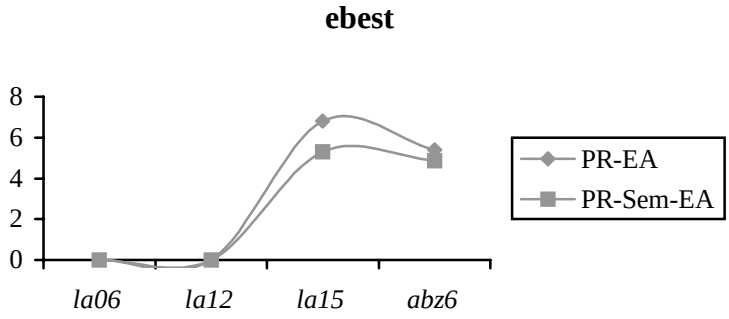


Figura 7.7 Mínimo *ebest* bajo PR-EA y PR-SEM-EA.

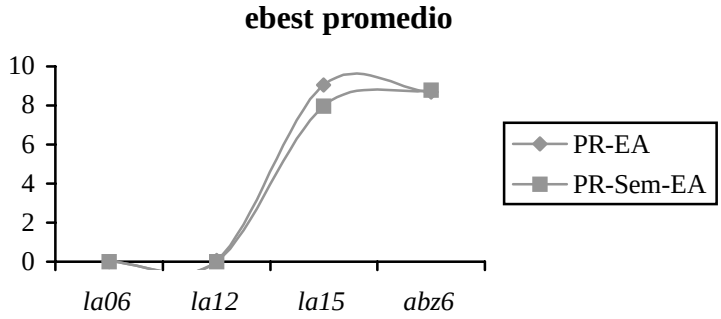


Figura 7.8 *ebest* promedio bajo PR-EA y PR-Sem-EA.

De las figuras 7.7 y 7.8 se pueden analizar las curvas correspondientes al *ebest* y *ebest* promedio tanto para PR-EA como para PR-Sem-EA. Para las instancias *ft6, la01, la06* y *la12*, el *ebest* es del 0%, para ambos algoritmos; lo cual es lógico observando la tabla correspondiente a la cantidad de veces que un algoritmo encuentra el óptimo: en todas estas instancias en alguna oportunidad se ha encontrado el óptimo. Para las restantes instancias, a medida que aumenta la complejidad de las mismas, el *ebest* se

hace más considerable; llegando a un 22.9833% para la instancia *abz7*. Para la instancia *la15* se observa una diferencia a favor de PR-Sem-EA, con un valor de 5.3024 contra 6.7937 de PR-EA, esta misma observación se puede hacer sobre la instancia *abz6* (4.8780 contra 5.4083). En cuanto al *ebest* promedio, para la instancia *la06* es de 0%, tal valor corrobora el porcentaje de éxito para esta instancia, independientemente del hecho de haber incorporado semilla en la población inicial o no. En las restantes instancias, el comportamiento de PR-Sem-EA ha sido levemente mejor que PR-EA; por ejemplo en la instancia *la12* el *ebest* promedio para PR-EA es de 0.1476 y para PR-Sem-EA es de 0.0706 y en la01 el *ebest* promedio para PR-EA es de 2.3123 y para PR-Sem-EA es de 2.2823.

Si se consideran en forma conjunta las curvas de *ebest* y *ebest* promedio, se obtiene una superposición de las mismas cuando se consideran las primeras instancias, lo que significa que los restantes mejores individuos hallados en cada corrida no están tan alejados, en promedio, al óptimo de cada instancia. En tanto que para las instancias *la15, abz6* y *abz7* las diferencias entre *ebest* y *ebest* promedio son un poco más significativas, por ejemplo, bajo la opción PR-Sem-EA el *ebest* para *abz7* es de 22.9833 mientras que el *ebest* promedio es 25.9361.

Analizando las curvas tanto de *epop* como de *epop* promedio de las figuras 7.9 y 7.10, nuevamente, se obtienen resultados bastantes aproximados entre ambas opciones, pero notándose una diferencia a favor de PR-Sem-EA.

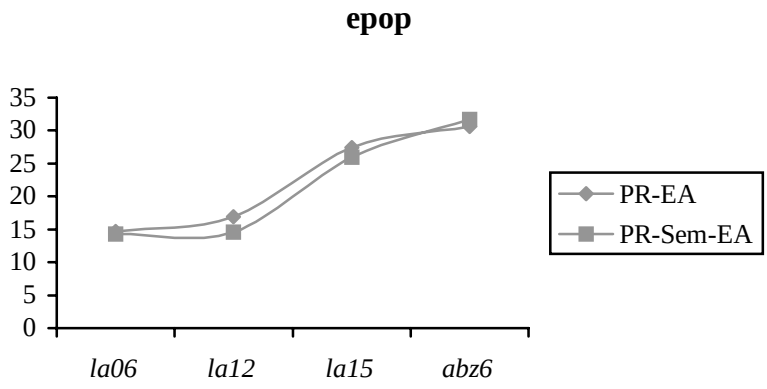


Figura 7.9 Mínimo *epop* bajo PR-EA y PR-SEM-EA.

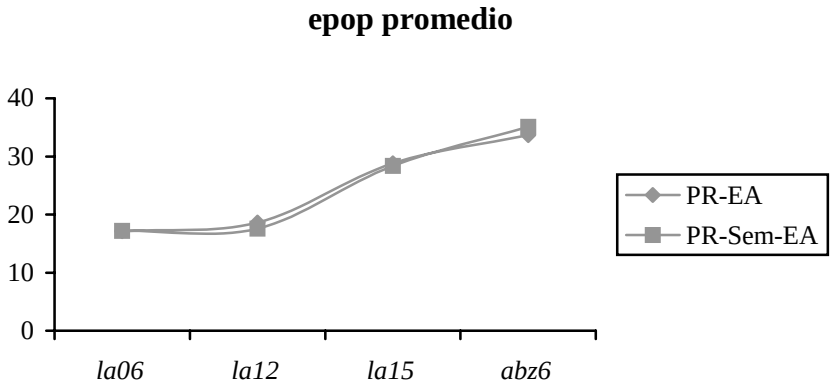


Figura 7.10 *epop* promedio bajo PR-EA y PR-SEM-EA.

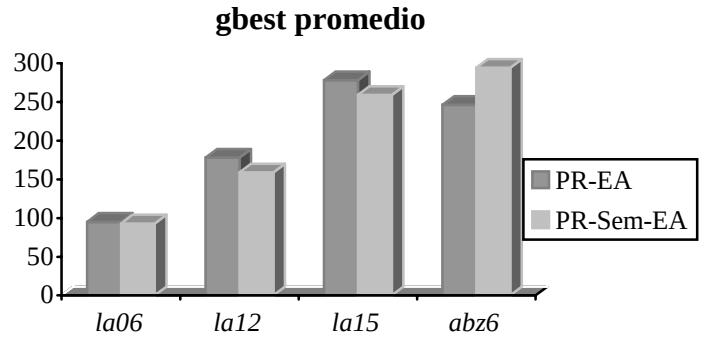


Figura 7.11 *gbest* promedio para todas las instancias.

En la figura 7.11 se observa que para las instancias donde se encuentra los óptimos (*ft6*, *la01*, *la06* y *la12*) e incluyendo a *la15*, PR-Sem-EA llega a encontrar los mejores valores en generaciones más tempranas que PR-EA (para la instancia *ft6* se necesitan 208 generaciones en el caso de PR-EA y 182 para PR-Sem-EA, *la01* tarda 315 generaciones para el caso de PR-EA y 216 para PR-Sem-EA). En el resto de las instancias, la población en PR-Sem-EA necesita evolucionar por más generaciones que en PR-EA para llegar a un valor aproximado al óptimo.

Observando la tabla 7.7, las gráficas correspondientes a *ebest* y la figura 7.11, se puede deducir que aquella población en la cual se introducen semillas parece arribar a sus soluciones finales más rápidamente, sin observar una disminución en la calidad de las soluciones.

7.3.3. CONCLUSIONES

En este caso se presenta otra solución posible al problema del job shop scheduling haciendo uso de algoritmos evolutivos el cual emplea reglas de despacho. El uso de estas últimas en la codificación de un cromosoma permite la aplicación del crossover más sencillo como lo es el one-point crossover, con lo cual siempre se obtendrán hijos factibles sin necesidad de incluir técnicas de penalidades o de reparación. Una particularidad de este algoritmo es la de utilizar semillas generadas por otro método heurístico convencional, las cuales se insertan en la población inicial, para obtener una mejora en la eficiencia del mismo.

Del análisis realizado sobre los resultados obtenidos bajo esta opción se puede resaltar:

- ✓ Que la incorporación de semillas en la población inicial aporta por lo general buenos resultados, permitiendo al algoritmo encontrar individuos más cercanos al óptimo y con poblaciones que permanecen más centradas respecto de ese individuo que en el caso de no incorporar semillas a la población inicial.
- ✓ Además para algunas de las instancias se encuentra el óptimo en más oportunidades cuando se incorporan individuos más adaptados a la población inicial que en el caso de no hacerlo. Además, si se analizan las generaciones promedio en las cuales cada algoritmo encuentra los mejores individuos para cada instancia y la cantidad de veces que para esa instancia se halló en óptimo, se puede concluir que el algoritmo que introduce semillas halla los mejores individuos en forma más rápida, sin que esto implique la pérdida en la calidad de soluciones y la diversidad genética.

7.4. OPCIÓN CON EL MÉTODO DE PARCIAL EXCHANGE

Gen, Tsujimura, y Kubota propusieron una implementación de un algoritmo genético para resolver un problema de job shop scheduling [70]. Usaron una representación del cromosoma basada en operaciones y diseñaron un operador de crossover *partial schedule exchange* para esta representación. El nuevo operador considera schedules parciales como bloques de construcción natural, y tal crossover se utiliza para mantener los bloques en los hijos. El schedule parcial se identifica con el

mismo job en la primer y última posición del schedule parcial. Por ejemplo, si tenemos dos cromosomas padres p_1 y p_2 [69]:

p_1 [3 2 1 2 3 4 1 2 4 4 1 3 4 1 2 3]

p_2 [4 1 3 1 1 3 4 1 2 3 4 2 2 2 3 4]

Los schedules parciales se eligen aleatoriamente de la siguiente manera:

(a) Elegir una posición en el padre p_1 aleatoriamente. Supongamos que es la 6ta posición en la cuál está ubicado el job 4.

(b) Hallar el job 4 más próximo en el mismo padre p_1, es cual está en la posición 9. Ahora tomar el *schedule parcial 1* como (4 1 2 4).

(c) El próximo schedule parcial no es generado aleatoriamente en el padre p_2. Este deberá comenzar y finalizar con el job 4. Note que los dos jobs 4 son el primer y segundo job 4 en el padre p_1. Así el *schedule parcial 2* está formado por los genes entre el primer y segundo job para el job 4 en el padre p_2, (4 1 3 1 1 3 4).

Una ejemplificación gráfica del procedimiento de selección de schedules parciales se muestra a continuación:

schedule parcial 1

p_1 [3 2 1 2 3 4 1 2 4 4 1 3 4 1 2 3]

p_2 [4 1 3 1 1 3 4 1 2 3 4 2 2 2 3 4]

schedule parcial 2

A continuación se muestra el intercambio de schedules parciales:

schedule parcial 2

o_1 [3 2 1 2 3 4 1 3 1 1 3 4 4 1 3 4 1 2 3]

o_2 [4 1 2 4 1 2 3 4 2 2 2 3 4]

schedule parcial 1

Por lo general, los schedules parciales están formados por distintas cantidades de genes, lo que puede provocar la ilegalidad de los hijos que resultan del intercambio. Los genes que se pierden y los que están en exceso en los hijos se pueden determinar de la siguiente manera:

genes perdidos y en exceso para o_1

schedule parcial 1 4 1 2 4 genes perdidos 2

schedule parcial 2 4 1 3 1 1 3 4 genes en exceso 3 1 1 3

genes perdidos y en exceso para o_2

schedule parcial 2 4 1 3 1 1 3 4 genes perdidos 3 1 1 3

schedule parcial 1 4 1 2 4 genes en exceso 2

En el próximo paso se legalizan los hijos al suprimir los genes en exceso y adicionar aquellos genes perdidos. Para el hijo o_1, sus genes en exceso son (3 1 1 3). El hijo o_1, recibe el *schedule parcial 2* desde el padre p_2, pero deberá borrar los jobs 3 y 1 antes del schedule parcial en o_1, para conseguir el mismo orden que en el padre p_2. La razón de lo anterior es que los genes en la representación basada en operaciones se refieren a operaciones las cuales son dependientes del contexto. Si los genes del *schedule parcial 2* en el hijo o_1 toman el mismo orden que en el padre p_2, esos genes se deberán referir a las mismas operaciones en ambos casos; en caso contrario se referirán a operaciones diferentes. Se espera que el hijo o_1 herede el mismo schedule parcial identificado por *schedule parcial 2* en el padre p_2. Debemos dejar los genes del *schedule parcial 2* en el mismo orden tanto en o_1 como en p_2. Para el hijo o_1, el gen que se pierde es 2. Como no hay ningún job 2 en *schedule parcial 2*, se inserta el job 2 en algún lugar (excepto dentro del *schedule parcial* 2), con lo cual no se cambia el orden de los genes en el *schedule parcial 2*. El proceso de legalización del hijo o_1 es el siguiente:

(1) *Borra los genes en exceso* 3 1 1 3

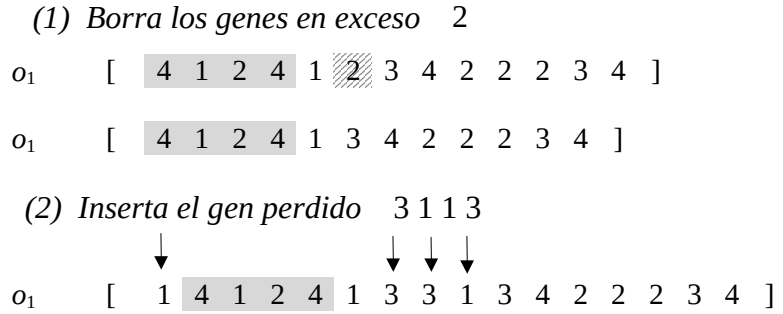
o_1 [3 2 1 2 3 4 1 3 1 1 3 4 4 1 3 4 1 2 3]

o_1 [2 2 4 1 3 1 1 3 4 4 3 4 1 2 3]

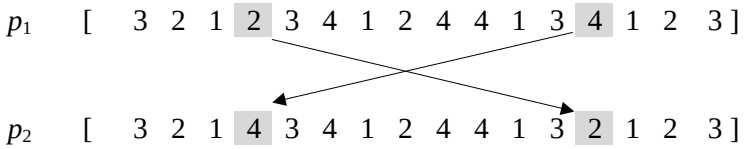
(2) *Inserta el gen perdido* 2

o_1 [2 2 4 1 3 1 1 3 4 2 4 3 4 1 2 3]

Para el hijo o_2, el gen en exceso es el job 2. Se puede borrar algún job 2 en el hijo o_2 excepto dentro del *schedule parcial 1*. Sus genes perdidos son (3 1 1 3). Se debe insertar un job 1 antes del *schedule parcial 1* y un job 1 luego del *schedule parcial 1* en el hijo o_2 de modo de conseguir que el job 1 del *schedule parcial 1* sea el segundo job 1 en el hijo o_2. Como no hay job 3 en el *schedule parcial 1*, se inserta un job 3 en algún lugar (excepto dentro del *schedule parcial 1*). Esto se puede apreciar en el siguiente esquema:



Gen Tsujimura, y Kubota usaron una mutación de intercambio de pares de jobs; es decir, elegir aleatoriamente dos jobs distintos y entonces intercambiar sus posiciones:



Para la representación basada en operación, la relación de mapping entre cromosomas y schedule es muchos a uno, y un hijo obtenido al intercambiar dos jobs cercanos puede traer el mismo schedule de los padres. De este modo la mayor separación entre dos jobs distintos es lo mejor.

7.4.1. DESCRIPCIÓN DEL EXPERIMENTO

En este experimento se contrasta el comportamiento de dos algoritmos evolutivos:

- ✓ **PR-EA** que utiliza la representación basada en reglas de prioridad, descripta en la sección anterior, y sobre la cual se aplica el one-point crossover y la mutación big-creep.
- ✓ **OR-EA** que utiliza la representación basada en operaciones, donde el operador de crossover utilizado es el *parcial schedule exchange crossover* y la mutación consiste en el intercambio de pares de jobs.

Ambos algoritmos se analizaron para un conjunto de instancias seleccionadas para el JSSP. Se realizaron 15 corridas diferentes para cada instancia. Para la selección de los individuos que conformarán el matting pool, en el algoritmo evolutivo se usó selección proporcional al fitness del individuo. Los mejores individuos se retuvieron por elitismo. En los experimentos el tamaño de la población fue de 50 individuos. El número máximo de generaciones fue fijado en 500 y las probabilidades de crossover y mutación fueron

establecidas en 0.65 y 0.01, respectivamente. Se usaron 7 instancias [97] con óptimos bien conocidos (tabla 7.6).

Las variables de performance elegidas para contrastar los resultados encontrados por cada algoritmo son *ebest*, *epop* y *gbest*.

7.4.2. ANÁLISIS DE LOS RESULTADOS

En la tabla 7.8 se muestran las veces que cada algoritmo ha encontrado el óptimo en cada instancia. En la instancia *la06*, en el 100% de los casos ambos algoritmos hallan el óptimo; en la instancia *ft6*, PR-EA encuentra en el 60% de los casos el óptimo contra el 100% de OR-EA; en *la01*, PR-EA encuentra en una única oportunidad el óptimo mientras que OR-EA lo hace en 9 de las 15 ejecuciones; en *la12* en el 87% de los casos PR-EA encuentra el óptimo para esta instancia contra el 100% de OR-EA. En instancias más complejas como *la15*, *abz6* y *abz7*, los algoritmos no encuentran en ninguna oportunidad el óptimo correspondiente a la instancia.

	PR-EA	OR-EA
ft6	9	15
la01	1	9
la06	15	15
la12	13	15
la15	0	0
abz6	0	0
abz7	0	0

Tabla 7.8 Cantidad de veces que cada algoritmo encuentra los óptimos de cada instancia.

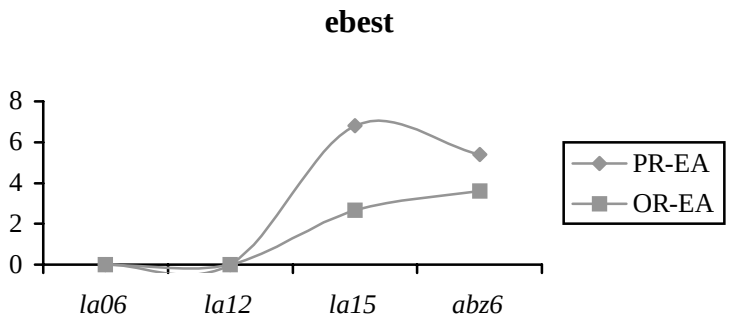


Figura 7.12 *ebest* para PR-EA y OR-EA.

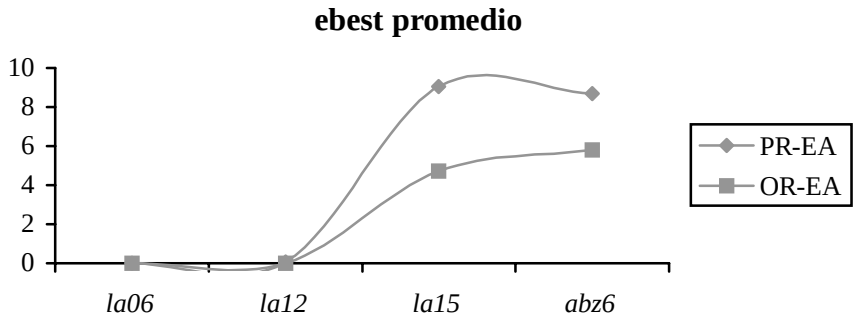


Figura 7.13 *ebest* promedio para PR-EA y OR-EA.

Analizando la figura 7.12 se puede observar que para las instancias *ft6*, *la01*, *la06* y *la12* ambos algoritmos logran un *ebest* del 0%, significando que en al menos una oportunidad se halla el valor óptimo. Para el resto de las instancias consideradas, el *ebest* en el que incurre OR-EA es considerablemente menor que PR-EA, por ejemplo para la instancia *la15*, el *ebest* de OR-EA es 2.6512% y el de PR-EA es 6.7937%.

Como se puede observar en el gráfico correspondiente a *ebest* promedio (figura 7.13), OR-EA encuentra en promedio individuos más cercanos al óptimo que PR-EA. En las instancias *ft6*, *la06* y *la12* se puede observar que el error producido por OR-EA es del 0%, lo cual indica que en todas las ejecuciones el algoritmo ha encontrado el valor óptimo correspondiente a cada instancia; en tanto que para la instancia *la01* el *ebest* promedio para este mismo algoritmo es cercano a 0 (0.5%). En el caso de PR-EA, los errores promedios para las mismas instancias *ft6*, *la01* y *la12* oscilan entre 2.3% (en el caso de *la01*) y 0.07% (para *la12*). Cabe destacar que ambos algoritmos obtienen en todas las ejecuciones el valor óptimo de la instancia *la06*. En los casos de las instancias *la15*, *abz6* y *abz7* se producen errores promedios más altos pero siempre se observa una mejor performance para OR-EA.

Al analizar la figura 7.14, se puede observar que la población final a la que arriba el algoritmo OR-EA está más centrada alrededor del valor óptimo que en el caso de PR-EA; en particular en las instancias *ft6* y *la01*, donde los errores cometidos son menores al 0.8%. En estas mismas instancias y en *abz6* es donde se observan las mayores diferencias entre los *epop* que presentan ambos algoritmos, por ejemplo en la instancia *ft6* el *epop* producido por PR-EA es 24.7637% mientras que OR-EA produce un *epop* de 0.001%.

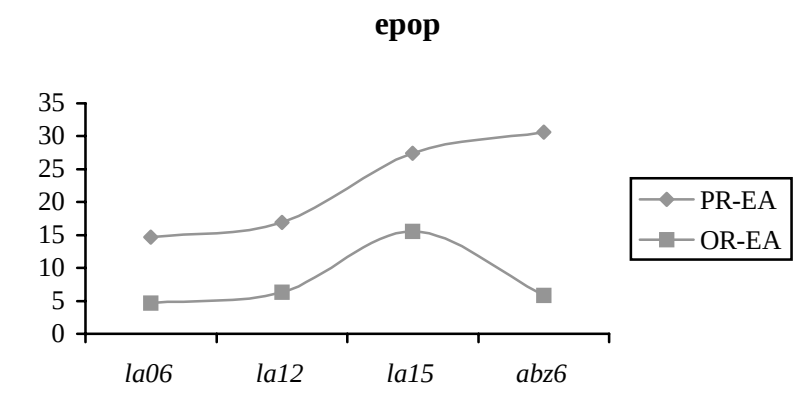


Figura 7.14 *epop* para PR-EA y OR-EA.

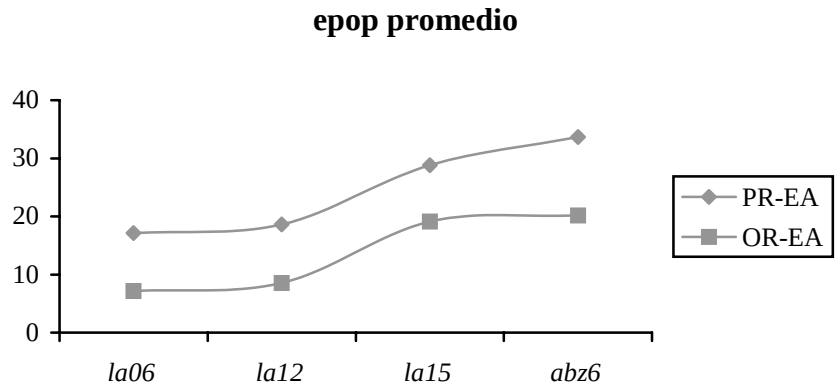


Figura 7.15 *epop* promedio para PR-EA y OR-EA.

En la figura 7.16, se puede observar que para la instancia *la06* donde ambos algoritmos encuentran en el 100% de las ejecuciones el valor óptimo, OR-EA lo hace usando una notable menor cantidad de generaciones (5 generaciones en promedio para OR-EA y 95 para PR-EA). En las instancias *ft6*, *la01* y *la12*, donde el *ebest* es muy cercano a cero, la cantidad de generaciones que OR-EA necesita para llegar a un valor óptimo o muy cernano a él está muy por debajo de las requeridas por PR-EA. Cuando la complejidad de las instancias, medida en la cantidad de máquinas empleadas, se incrementa repercute en la cantidad de generaciones que OR-EA emplea para hallar el valor más cercano al óptimo, en particular para *abz6* y *abz7* necesitan aproximandamente 50 generaciones más que PR-EA, pero de todos modos los errores tanto poblacionales como los de los mejores individuos de OR-EA son siempre inferiores a PR-EA.

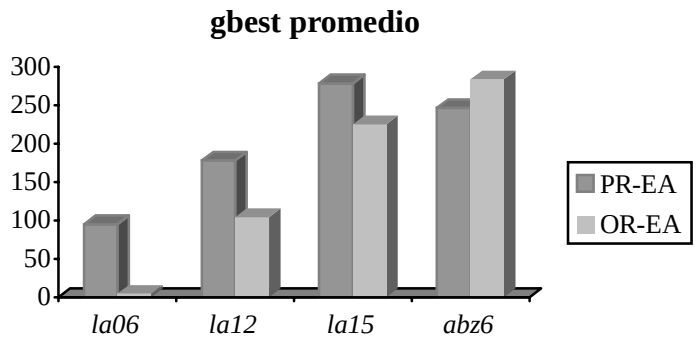


Figura 7.16 Gbest para PR-EA y OR-EA

7.4.3. CONCLUSIONES

En este caso se presenta otra alternativa para la solución del problema de job shop scheduling usando un algoritmo evolutivo, pero aquí con una representación basada en operaciones e incorporando un operador de crossover adecuado para tal representación como lo es el parcial schedule exchange crossover.

De los análisis previos realizados sobre los resultados obtenidos se puede concluir que con un algoritmo evolutivo usando la representación basada en operaciones, la performance es considerablemente mejor que cuando se usa una representación con reglas de despacho; esa performance se mide ya sea en la calidad de soluciones halladas en forma individual como en el error de población final respecto del valor óptimo para cada instancia. Además una característica importante a remarcar es que las poblaciones finales a las que arriba OR-EA permanecen más centradas respecto del mejor individuo hallado que en el caso de PR-EA.

Es importante destacar que en las instancias *la01*, *la06* y *la12*, OR-EA ha encontrado en todas las pruebas realizadas el valor óptimo de cada instancia; a medida que la complejidad de las instancias se incrementa esta cantidad disminuye pero de todos modos el error promedio porcentual cometido por los mejores individuos hallados no es signifivo.