Algoritmo de optimización por cúmulo de partículas para la construcción de redes de ordenamiento rápidas

Blanca López, Nareli Cruz-Cortés

Centro de Investigación en Computación,
Instituto Politécnico Nacional (CIC-IPN), DF, México

blopezb10@sagitario.cic.ipn.mx, nareli@cic.ipn.mx

Resumen. Una Red de Ordenamiento (RO) es un método de ordenamiento *oblivious* que permite ordenar N objetos mediante el uso de un número fijo de comparadores binarios los cuales están distribuidos en un orden predefinido. Por décadas, el diseño de las RO ha capturado la atención de muchos investigadores quienes han dedicado un gran esfuerzo por minimizar el número de comparadores para una entrada dada N. En este trabajo, proponemos una estrategia novedosa y sencilla para diseñar RO que maximicen el número de comparadores en paralelo, lo que es equivalente a encontrar RO rápidas. La heurística bioinspirada llamada Optimización por Cúmulo de partículas (PSO) es empleada para guiar el proceso de búsqueda. Nuestros resultados muestran que el esquema propuesto permite construir RO rápidas y en algunos casos se encuentran RO óptimas por el número de comparadores. Los diseños de las RO construidas cuentan con diferentes configuraciones a las RO rápidas conocidas en la literatura especializada.

Palabras clave: Redes de ordenamiento, optimización por cúmulo de partículas, ordenamiento.

1. Introducción

Diseñar Redes de Ordenamiento (RO) es un problema combinatorio que ha sido estudiado durante décadas [17,8]. Las RO son un ejemplo de algoritmos llamados *oblivious*, esto es, que los pasos necesarios para ordenar no dependen de los anteriores. La operación fundamental en una RO es la llamada comparación-intercambio y se conoce en la literatura especializada como *comparador*. Un conjunto de comparadores conforman una RO. Un comparador C recibe dos datos de entrada (a, b) para ser comparados, si $a > b$ entonces se aplica la operación de intercambio donde $a < b$. Una RO puede ser interpretada como un circuito para ordenar una secuencia de datos [16].

Generalmente las RO son representadas esquemáticamente como en la Figura 1. Cada línea horizontal representa el trayecto de un dato de entrada.

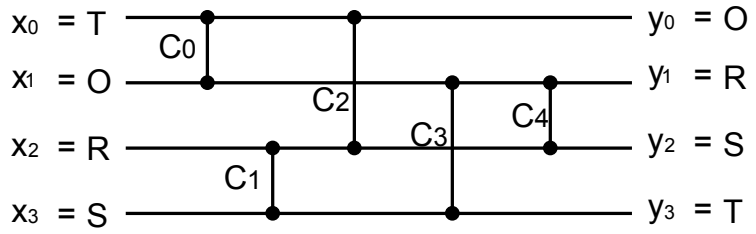

Fig. 1. Ejemplo de una Red de Ordenamiento para 4 datos de entrada.

Cada línea vertical es un comparador que evalúa los elementos entre su extremo superior e inferior. Si el valor del extremo superior es más grande que el extremo inferior entonces intercambia los datos. Por ejemplo, en una RO donde se quieren ordenar 4 datos de entrada, los valores de x_0, x_1, x_2, x_3 son colocados del lado izquierdo de la RO y un dato por cada línea horizontal, como en la Figura 1. Las líneas verticales etiquetadas como C_0, C_1, C_2, C_3, y C_4 son los comparadores. Cada comparador recibe dos valores, esto es, C_0 recibe los valores x_0 y x_1; C_1 recibe los valores de x_2 y x_3, y así sucesivamente. De esta forma, los comparadores son ejecutados de izquierda a derecha según su aparición. En este ejemplo primero se ejecuta C_0, posteriormente C_1, en seguida C_2, entonces C_3 y finalmente C_4. Esto es, C_0 evalúa 'T'>'R' entonces, los valores de x_0 y x_1 son intercambiados. El siguiente comparador es C_1 que evalúa 'O'>'S' mantiene a x_0 y x_1 sin intercambio. En seguida C_3 evalúa 'R'>'O' entonces, los valores x_0 y x_2 son intercambiados. Nótese que es posible ejecutar los comparadores C_0 y C_1 al mismo tiempo. De manera análoga los comparadores C_2 y C_3 podrían aplicarse en forma paralela. Este proceso continúa hasta que todos los comparadores son aplicados. Finalmente, la lista ordenada y_0, y_1, y_2, y_3 se obtiene al lado derecho de la RO y debe cumplir con $y_0 \leq y_1 \leq y_2 \leq y_3$.

La independencia de los datos de entrada se debe a la configuración interna de las RO. Es por este motivo que las RO se encuentran en la clasificación de algoritmos no adaptativos. Las principales diferencias entre las RO y los algoritmos clásicos de ordenamiento (tales como el *bubblesort*, *sellsort*, *quicksort*) son las siguientes:

1. La RO cuenta con una cantidad fija de comparadores necesaria para ordenar cualquier permutación de datos de entrada;
2. El número de pasos necesarios para ordenar es fijo *a priori*.

En realidad, el diseño de una RO de tamaño mínimo incrementa considerablemente su complejidad cuando el número de entradas es incrementado. Algunas RO que han sido ampliamente estudiadas son $n = 10$, $n = 12$, y $n = 16$. Además en la literatura las publicaciones son orientadas en su mayoría a la construcción de RO minimizando el número de comparadores. Por lo que existe poco trabajo dedicado al diseño de RO rápidas, esto es, RO con máximo paralelismo.

Diferentes trabajos se han encontrado para construir Redes de Ordenamiento rápidas, como en [16,15] donde emplean estrategias empleadas en las técnicas

clásicas de ordenamiento, tal como el *sellsort* y el *mergesort* paralelo respectivamente. Además en [1] fue propuesta una estrategia recursiva. Por otro lado, el trabajo publicado por Batcher en [11,3] es una estrategia eficiente llamada $Odd - Even\ sort$, la cual ayuda a construir RO para $n > 16$ con algunos resultados favorables con respecto a RO rápidas.

En este trabajo, proponemos una técnica para encontrar RO que maximicen el número de comparadores por capa. Para esto proponemos una novedosa y compacta manera de representar la RO, esto ocasiona la construcción de RO rápidas convenientemente. Por otro lado, la tarea de búsqueda la lleva a cabo una técnica bioinspirada llamada algoritmo de Optimización por Cúmulo de Partículas (PSO por sus siglas en inglés), es importante señalar que no se ha estudiado la construcción de RO con PSO. Los resultados presentados muestran que la estrategia es eficiente y capaz de encontrar RO rápidas. Además, la cantidad de comparadores encontrados en las RO resultantes son competitivas con el estado del arte.

2. Algoritmo de optimización por cúmulo de partículas (PSO)

El PSO [10] es una meta-heurística bioinspirada inspirada en el comportamiento social del vuelo de las aves o el movimiento de los bancos de peces. Ésta se ha convertido en un técnica popular para resolver problemas de optimización complejos. El PSO se describe generalmente como una población de vectores (llamadas partículas) cuyas trayectorias oscilan en torno a una región. En el caso más general, teniendo en cuenta la configuración del enjambre (población), hay dos versiones del PSO: globales y locales. En la versión global (*gbest*) la trayectoria de cada partícula está influenciada por el mejor punto encontrado por cualquier miembro de todo el enjambre. En la versión local (*lbest*) la trayectoria de cada partícula puede ser influenciada sólo por el mejor miembro en el vecindario (el mismo enjambre). Además, cada partícula también se ve influida por su mejor éxito previo.

En su forma más general el PSO binario se describe como: Se denota a Np como el número de partículas en la población. Así que $Y_i^t = (y_{i1}^t, y_{i2}^t, \dots, y_{iD}^t)$, $y_{id}^t \in 0, 1$, es una partícula i con D *bits* en la iteración t, donde Y_i es una solución potencial que tiene una tasa de cambio llamada velocidad.

La velocidad es $V_i^t = (v_{i1}^t, v_{i2}^t, ..., v_{iD}^t)$, $v_{id}^t \in R$. Así que $P_i^t = (p_{i1}^t, p_{i2}^t, ..., p_{iD}^t)$ representa la mejor solución que la partícula i ha obtenido hasta la iteración t, y $p_g^t = (p_{g1}^t, p_{g2}^t, ..., p_{gD}^t)$ representa la mejor solución obtenida a partir de P_i^t en la población (*gbest*)o del vecindario local (*lbest*), en la iteración t.

Cada partícula ajusta su velocidad de acuerdo a la siguiente ecuación:

$$v_{id}^t = v_{id}^{t-1} + co_1 r_1 (p_{id}^t - y_{id}^t) + co_2 r_2 (p_{gd}^t - y_{id}^t) \tag{1}$$

donde co_1 es el factor de aprendizaje cognitivo, co_2 es el factor de aprendizaje social, y r_1 y r_2 son números generados en forma aleatoria con distribución

uniforme en $[0, 1]$. Los valores co y r determinan los pesos de dos partes donde la suma de ellos es limitada normalmente a 4 [10].

La velocidad representa la probabilidad de que el valor actual tome el valor de uno. Para mantener el valor en el intervalo $[0, 1]$ se emplea la función sigmoidal:

$$sig(v_{id}^t) = \frac{1}{1 + exp(-v_{id}^t)} \tag{2}$$

Algoritmo de PSO

Algoritmo 1 Red de ordenamiento para $n = 4$

Input: x_0, x_1, x_2, x_3

```
1    begin
2        FOR i = 0 hasta Np
3            STATE Calcular la aptitud por cada p_i
4            IF Y_i^t > P_i^t
5                FOR d = 1 hasta D bits
6                    STATE P_id^t = Y_i^t
7                END FOR
8            END IF
9            g = i
10           FOR j=índices de los vecinos
11               IF P_j^t > P_g^t
12                   STATE g = j
13               END IF
14           END FOR
15           FOR d = 1 hasta D
16               STATE v_id^t = v_id^(t-1) + co1 r1 (P_id^t - Y_id^t) + co2 r2 (P_gd^t - Y_id^t)
17               STATE v_id^t ∈ [-V_max, +V_max]
18               IF Número aleatorio [0,1] < sig(v_id^t)
19               STATE Y_i^(t+1) = 1
20           ELSE
21               STATE Y_i^t = 0
22           END IF
23           END FOR
24       END FOR
```

Output: y_1, y_2, y_3, y_4

3. Redes de Ordenamiento: Conceptos básicos

Las RO son algoritmos que ordenan una lista de datos. Están compuestas por un grupo de comparadores, donde cada comparador ejecuta una acción

comparación-intercambio entre dos elementos (a, b). El elemento a debe ser más pequeño que el elemento b, de lo contrario será intercambiado (b, a). Los datos de salida de las RO están en una lista con un orden monotónicamente creciente.

Durante el proceso de construcción de una RO es necesario determinar si la RO candidata es válida o no. Es posible saberlo si todas las permutaciones de los datos de entrada son correctamente ordenados por la RO, de ser así, es considerada como una RO válida. Por lo tanto, para un tamaño de dato de entrada n es necesario probar $n!$ veces la RO, esto implica que la tarea de la verificación incrementa su complejidad computacional cuando n se incrementa, inclusive puede llegar a ser intratable [13]. Una alternativa es aplicar el principio cero-uno ($TheoremZ$) presentado en el libro de Knuth [11]. Este principio afirma que una RO que ordene todas las 2^n secuencias de ceros y unos en un orden decreciente, ordenará cualquier secuencia arbitraria de n números. Por lo que para $n > 2$, $n! > 2^n$, la aplicación de este principio permite contar con una forma razonable de verificación en las RO.

En general, la eficiencia de una RO es medida con base en dos criterios:

- El número de comparadores implicados; y
- La velocidad máxima que la RO pueda alcanzar, la cual es inversamente proporcional al número de capas.

Desde hace más de 5 décadas, varios investigadores se han dedicado a buscar RO con el mínimo número de comparadores. El hecho de verificar que una secuencia de comparadores es una RO válida, es un problema NP-Completo [1,14,6] y diseñar RO con un mínimo número de comparadores o maximizar el número de capas es un problema de tipo NP-hard [14]. Una ventaja de las RO es que sus comparadores pueden ser ejecutados en paralelo. Usualmente, el tiempo discreto requerido para el ordenamiento se refiere al número de pasos o capas que la RO debería considerar después de verificar la independencia de los comparadores que pueden ser ejecutados en paralelo.

Una RO es *rápida* si tiene un bajo número de capas, donde cada capa está compuesta por un subconjunto de comparadores que puedan realizar sus operaciones concurrentemente sin interferir entre ellas. Por ejemplo, en la Figura 2 se tiene el diagrama de flujo de la RO para $n = 4$ de la Figura 1. Los comparadores C_0 y C_1 pueden ser aplicados al mismo tiempo (independientes), esto permite realizar las operaciones de forma concurrente. Al igual que C_2 y C_3 podrían ejecutarse en paralelo.

El conjunto de comparadores que se ejecutan concurrentemente se les conoce como capas, denotado por L. Por ejemplo, en la RO de la Figura 3, el rectángulo a la izquierda (línea de puntos separados) representa una capa que es ejecutada en el tiempo $t1$. De tal manera que, la RO puede ser ejecutada en tres pasos. Note que una capa no puede contener más de $\lfloor n/2 \rfloor$ comparadores.

Entre las diferentes formas de construcción, las RO rápidas entre $3 \leq n \leq 16$ con su respectivo número de capas se puede ver en la Tabla 1. La segunda fila presenta el número de capas de las RO rápidas conocidas. En la tercera fila se tienen los límites mínimos de capas que han sido obtenidos de información teórica [14,17].

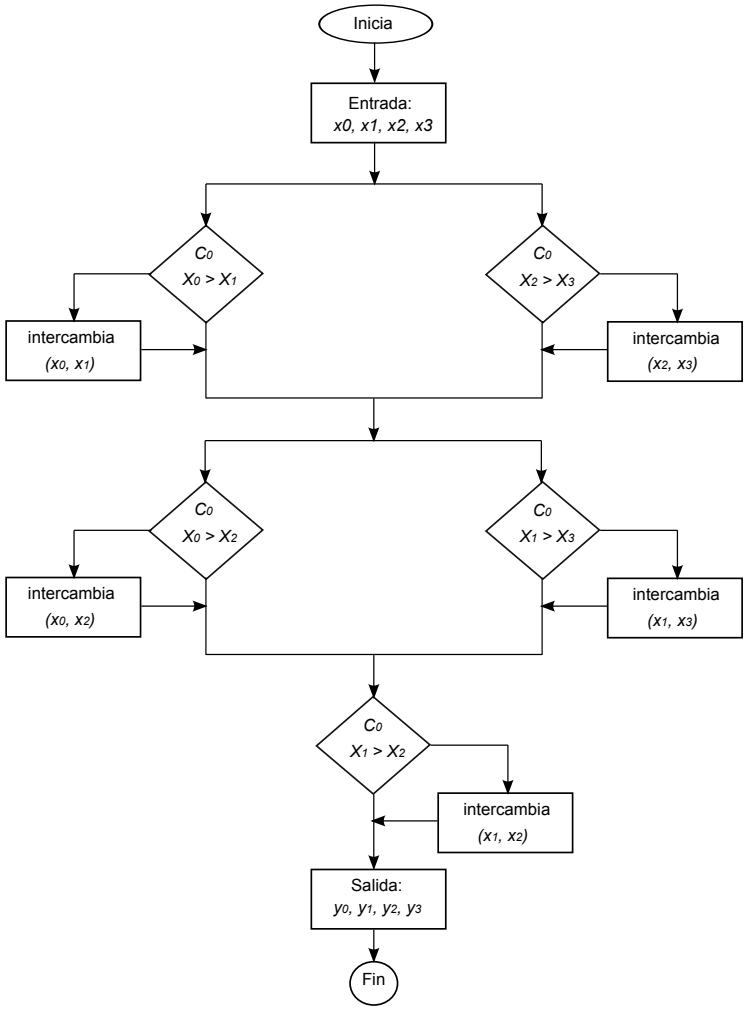

Fig. 2. Diagrama de flujo de la Red de Ordenamiento para 4 datos de entrada.

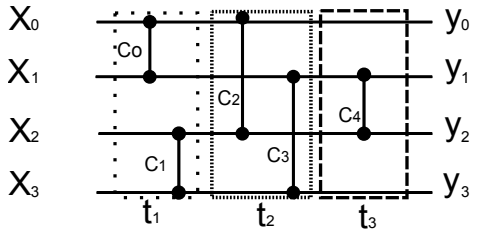

Fig. 3. Ejemplo de una RO para $n = 4$ con 3 capas.

Por otro lado el número de comparadores que tienen las RO rápidas conocidas se encuentran en la Tabla 2.

Tabla 1. RO rápidas conocidas y límites mínimos obtenidos de forma teórica.

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Número de capas	3	3	5	5	6	6	7	7	8	8	9	9	9	9
Límite mínimo	3	3	5	5	6	6	7	7	7	7	7	7	7	7

Tabla 2. Número de comparadores de las RO rápidas conocidas desde $n = 3$ hasta $n = 16$.

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Número de capas	3	3	5	5	6	6	7	7	8	8	10	10	10	9
Número de comparadores	3	5	9	12	31	19	25	31	35	40	45	53	56	61

4. Trabajo previo en Redes de Ordenamiento

Uno de las primeras investigaciones para diseñar RO fue presentado por P. N. Armstrong, R. J. Nelson y D. G. O'Connor en 1945 [2]. Allí, los autores diseñaron RO para $n = 5$, $n = 6$, $n = 7$ y $n = 8$ con 5, 9, 12 y 19 comparadores respectivamente. Posteriormente, Knuth [11] demostró que éstas son las RO más eficientes posibles (con el menor número de comparadores).

En los años 60's los investigadores trabajaron intensamente sobre la RO para $n = 16$ datos de entrada. Los resultados más relevantes fueron los que a continuación se mencionan: R. C. Bose y Nelson en 1962 [4] encontraron una RO con 65 comparadores. En 1964 K. E. Batcher y Knuth [3,16] presentaron una RO con 63 comparadores, dos años más tarde G. Shapiro obtuvo una RO con 62 comparadores. En el mismo año (1969), M. W. Green logró el diseño de una RO con sólo 60 comparadores, que sigue siendo hasta el momento la RO más eficiente para $n = 16$. Un logro en la misma década fue para la RO de $n = 9$ con 25 comparadores diseñada por Floyd en 1964 [7]. Mientras Waksman en 1969 [18] daba a conocer que hasta el momento es la RO más eficiente conocida y estudiaba la simetría que presentaban los comparadores, con este estudio diseñó una RO de $n = 10$ con 29 comparadores.

Los algoritmos presentados por Batcher en [3] fueron empleados por Shapiro y Green para trabajar con las RO para $n = 12$ donde se encontraron 39 comparadores.

Desde los años 90's, surgen nuevas propuestas. En 1995 Hugues Juillé [9], utilizó una estrategia co-evolutiva y empleó un gran número de procesadores

para encontrar la RO para $n = 13$ con sólo 45 comparadores, que es la mejor RO conocida.

Recientes estrategias orientadas a reducir la complejidad del algoritmo y el tiempo de procesamiento se han publicado desde entonces, sin embargo no han mejorado el número de comparadores en los resultados encontrados.

Las trabajos más relevantes se presentan a continuación. En 1999 J. Koza [12] diseñó la RO para $n = 7$ con Programación Genética. En 2005, Choi y Moon [5] publicaron un Algoritmo Genético para diseñar una RO para tamaños de entrada $n = 10$, $n = 12$, y $n = 16$. Los autores argumentaron que su trabajo disminuye el tiempo de procesamiento de una manera impactante para encontrar las RO mejores conocidas en segundos con un simple procesador. Sin embargo, su técnica disminuye drásticamente el espacio de búsqueda debido a que copia la mitad de los comparadores de una RO óptima (RO de Green). Por ejemplo, para $n = 12$, su técnica añade los primeros 18 comparadores, mientras que para $n = 16$, toma 32 comparadores con los que inicializa su algoritmo.

5. Propuesta

5.1. Planteamiento del problema

El problema que aborda este trabajo es el diseño de RO rápidas para tamaño de los datos de entrada de $n = 10$, $n = 12$, y $n = 14$.

5.2. Metodología

La idea general es encontrar RO con un mínimo número de capas. El tamaño de capas estará limitado por el tamaño de capas que tiene una RO válida, entonces nuestra técnica procede a llenar cada una de estas capas con una *adecuada* selección de los comparadores. En este caso una *adecuada* selección de los comparadores significa que éstos garantizarán una RO válida.

Se establece que para una entrada de datos de tamaño n, la siguiente información es conocida:

- El máximo número de comparadores en una capa es $\lfloor n/2 \rfloor$,
- El número mínimo de capas en la RO rápida (de acuerdo a la Tabla 1).

Por lo tanto, en este escenario, el problema es determinar qué comparadores debería poblar en cada una de las capas. Debido a la gran cantidad de posibilidades, se decidió utilizar una heurística para encontrar tales RO. Para este caso se presenta la versión binaria del algoritmo PSO que se presentó en la Sección 2.

La representación de las partículas, la función de aptitud, y la descripción general del algoritmo se presentan a continuación.

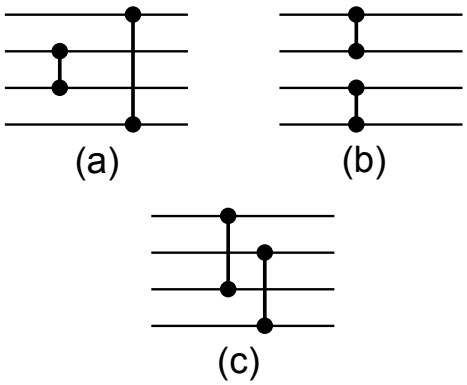

Fig. 4. Total de posibles configuraciones de una capa para una RO de $n = 4$. En (a) se muestra la configuración $\{1, 2\}\{0, 3\}$, en (b) la configuración $\{0, 1\}\{2, 3\}$, y en (c) la configuración $\{0, 2\}\{1, 3\}$.

5.3. Representación de la partícula

Una partícula representa una RO candidata compuesta por l capas (donde l es la más rápida). Cada capa se considera como una variable de decisión.

Recordemos que para que una determinada RO con entrada n, se conoce su número mínimo de capas (Tabla 1), por lo que es, precisamente, el número de capas representado en una partícula.

Para seleccionar los comparadores que poblarán una capa, todas las posibles configuraciones de capas son generadas y enumeradas, y sus números (utilizando codificación binaria) funcionará como identificadores.

Por ejemplo, supongamos que estamos buscando la RO para $n = 4$. El mínimo número de capas conocido es 3 (es el más rápido, ver Tabla 1 tercera fila, tercera columna). Por lo tanto, una partícula P se compone de 3 variables de decisión que es equivalente a 3 capas: $P = (L1, L2, L3)$

Ahora, dentro de cada capa L podemos colocar 2 comparadores paralelos (con base en que $n/2 = 2$). Es evidente que sólo hay tres posibles maneras de poner 2 comparadores paralelos, que son los siguientes (se ilustra en la Figura 4):

- Configuración 1: $\{1, 2\}\{0, 3\}$
- Configuración 2: $\{0, 1\}\{2, 3\}$
- Configuración 3: $\{0, 2\}\{1, 3\}$

La primera configuración es etiquetada con el número 1, la segunda con el 2 y la tercera con el 3. Por lo tanto, las variables de decisión pueden tomar uno de estos tres valores. Continuando con el ejemplo, la partícula

$$P = (1, 3, 1)$$

representaría a la RO

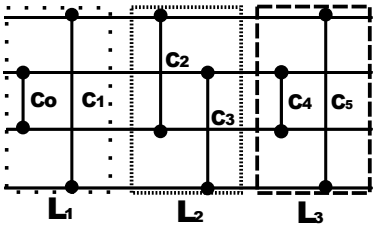

Fig. 5. Ejemplo de la representación de una partícula para $n = 4$

$$L1 = (\{1,2\}\{0,3\}), (L2 = \{0,2\}\{1,3\}), (L3 = \{1,2\}\{0,3\})$$

que se puede ver en la Figura 5. En este trabajo los valores son codificados con representación binaria.

5.4. Función de aptitud

La RO representada por una partícula *debe* ordenar todas las permutaciones de datos de entrada $x = (x_0, x_1, ..., x_{n-1})$ para reconocer su validez.

Esto es equivalente a ordenar todas las 2^n secuencias binarias (Principio cero-uno mencionado en la Sección 3). De acuerdo con este principio, se quiere minimizar el número de secuencias binarias desordenadas que deja una partícula. Así que, el mínimo valor esperado es cero.

La *aptitud* de una partícula es el número de secuencias binarias que no es capaz de ordenar. Las secuencias binarias de entrada son representadas por B_n y el conjunto de secuencias binarias ordenadas, lo llamaremos S_n.

Retomando el ejemplo con la RO para $n = 4$, los datos de entrada son $B_4 = 0000, 0001, 0010...1111$ y, S_4 es el conjunto de secuencias binarias ordenadas, así que $S_4 = 0000, 0001, 0011, 0111, 1111$. El objetivo es minimizar el conjunto de B_4 hasta que todas las secuencias estén ordenadas.

La partícula presentada en la Figura 5 tiene un valor de aptitud igual a 2 porque existen dos secuencias binarias $0010, 1011$ en B_4, que la RO no puede ordenar.

La única manera de evaluar sí una RO ordena un conjunto de secuencias, es aplicando todos sus comparadores a cada secuencia de B_n y supervisando que la salida sea correctamente ordenada.

El PSO presentado en la Sección 2 es utilizado para encontrar los comparadores que conforman cada capa L y minimizar el tamaño de B_n.

6. Experimentos y Resultados

Nuestros experimentos fueron dirigidos a diseñar RO rápidas para $n = 10$, $n = 12$ y $n = 14$ mediante la ayuda del PSO. Fueron aplicadas las dos versiones: local *lbest* y global, para cada n. La versión local del PSO (*lbest*), los vecindarios

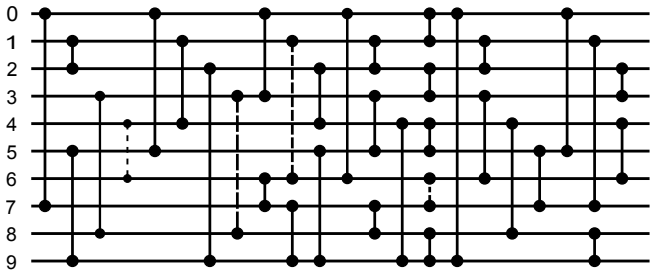

Fig. 6. Nuevo diseño para $n = 10$ encontrada por el algoritmo con el mínimo número de capas conocido igual a 7.

fueron ajustados por 10 partículas seleccionadas aleatoriamente con una configuración estática.

Todos los experimentos fueron ejecutados en un procesador Core i5-430M (2,26 GHz). Para cada caso se realizaron 20 ejecuciones independientes.

Para el diseño de $n = 10$, se llevaron a cabo experimentos con diferente número de partículas. Se encontró una RO con el PSO versión local. Ésta tiene 7 capas y 31 comparadores, resultado que coincide con la RO rápida y óptima en el número de comparadores. Además, esta RO cuenta con una configuración diferente a las conocidas (nuevo diseño). Su esquema es presentado en la Figura 6.

Nuestra técnica diseña una RO con 35 comparadores en 7 capas, de las cuales 4 comparadores son redundantes, estos comparadores no aplican la operación de intercambio durante el desempeño de la RO. Por lo tanto se consideran inútiles y son ignorados.

Es importante señalar que la técnica no recibió comparadores iniciales de ayuda, esto para disminuir el número de secuencias binarias, como se hizo en [5]. Los parámetros para el experimento fueron:

- Número de partículas 100
- Iteraciones 1000
- $V_{max} = 4.0$

Para el caso de $n = 12$, los resultados encontrados se dieron con los siguientes parámetros:

- Local PSO.
- Número de partículas 100
- Iteraciones 1000 hasta 3000
- $V_{max} = 4.0$

La técnica encuentra una RO con 51 comparadores y 9 capas. El esquema de esta RO se encuentra en la Figura 7. Inicialmente con el total de comparadores 51, aunque 7 comparadores son redundantes.

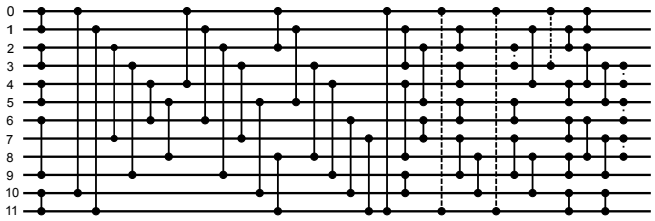

Fig. 7. La mejor RO rápida encontrada por el PSO con 51 comparadores y 9 capas. Las líneas punteadas son los comparadores redundantes.

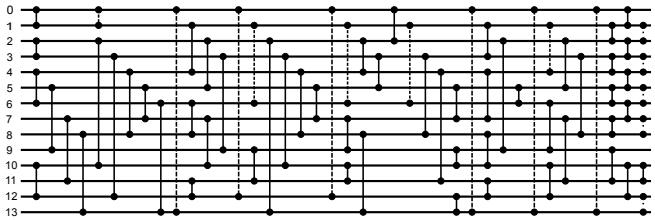

Fig. 8. La mejor RO para $n = 14$ encontrada por el PSO con 74 comparadores. Las líneas punteadas son comparadores redundantes (12). El resultado de la RO es 11 capas y 62 comparadores

.

Las capas inicialmente se fijaron con 8 pasos y al no encontrar RO válidas después de las $300,000$ evaluaciones se agregó una capa más. En la Figura 7 son identificados los comparadores redundantes con lineas punteadas. [1]

Con respecto a los experimentos para la RO con $n = 14$, la heurística encontró una RO con 11 capas y 62 comparadores. Esta solución se encuentra en la Figura 8. Esta RO es un nuevo diseño. La técnica presenta 74 comparadores, de los cuales 12 son redundantes. Estos comparadores se distinguen como líneas de punteadas.

7. Conclusiones

En general, existen dos formas como medida de eficiencia de una RO. La primera es el número de comparadores, y la segunda es el número de pasos o capas para identificar qué tan rápida es la RO, éstos se refiere a la máxima paralelización de comparadores en la RO. Considerando este último punto, se presenta una estrategia para diseñar RO de máxima paralelización de comparadores. La idea principal es fijar el tamaño de capas igual a la mínima conocida. De esta forma, los comparadores por cada capa son buscados y agregados a la

[1] Esto se debe a que la estrategia propuesta inserta el número máximo de comparadores por capa, lo que ocasiona que en algunos casos los comparadores lleguen a ser redundantes.

partícula, la cual usa una representación compacta. La técnica de búsqueda es adaptada al PSO. La RO para $n = 10$, $n = 12$ y $n = 14$ con ayuda de la heurística se encontraron nuevos diseños, unos resultados fueron muy cercanos a las RO rápidas y para la $n = 10$ además de una nueva configuración alcanza los mínimos conocidos en ambos objetivos.

Como trabajo a futuro es necesario diseñar una estrategia local para definir los resultados, especialmente durante el final del proceso. Además se puede cambiar la heurística para diseñar una estrategia local.

Agradecimientos. Las autoras agradecen el apoyo parcial de CONACYT a través del proyecto 132073. Además a la Secretaría de Investigación y Posgrado del IPN por el apoyo mediante el proyecto 20140147. De igual forma agradecemos al Instituto Politécnico Nacional.

Referencias

1. Ajtai, M., Komlós, J., Szemerédi, E.: An 0(n log n) sorting network. In: Proceedings of the fifteenth annual ACM symposium on Theory of computing. pp. 1–9. STOC '83, ACM, New York, NY, USA (1983)
2. Armstrong, P.: Sorting system for multiple bit binary records (Aug 27 1968), uS Patent 3,399,383
3. Batcher, K.E.: Sorting networks and their applications. In: Proceedings of the April 30–May 2, 1968, spring joint computer conference. pp. 307–314. AFIPS '68 (Spring), ACM, New York, NY, USA (1968)
4. Bose, R.C., Nelson, R.J.: A sorting problem. J. ACM 9(2), 282–296 (1962)
5. Choi, S.S., Moon, B.R.: A graph-based lamarckian-baldwinian hybrid for the sorting network problem. IEEE Trans. Evolutionary Computation 9(1), 105–114 (2005)
6. Chung, M.J., Ravikumar, B.: Bounds on the size of test sets for sorting and related networks. In: ICPP. pp. 745–751 (1987)
7. Floyd, R.W.: Algorithm 245: Treesort. Commun. ACM 7(12), 701– (Dec 1964)
8. Graham, L., Oppacher, F.: Symmetric comparator pairs in the initialization of genetic algorithm populations for sorting networks. In: Evolutionary Computation, 2006. CEC 2006. IEEE Congress on. pp. 2845 –2850 (0-0 2006)
9. Juille, H.: Evolution of non-deterministic incremental algorithms as a new approach for search in state spaces (1995)
10. Kennedy, J., Eberhart, R.: Particle swarm optimization. vol. 4, pp. 1942–1948 vol.4 (1995)
11. Knuth, D.E.: The Art of Computer Programming, Volume III: Sorting and Searching, 2nd Edition. Addison-Wesley (1998)
12. Koza, J.R.: Genetic programming: On the programming of computers by means of natural selection. Statistics and Computing 4(2) (1994)
13. Parberry, I.: Single-exception sorting networks and the computational complexity of optimal sorting network verification. Mathematical Systems Theory 23(2), 81–93 (1990)
14. Parberry, I.: Parallel Complexity and Neural Networks. MIT Press (1994)
15. R., S.: Analysis of shellsort and related algorithms. In: ESA '96: Fourth Annual European Symposium on Algorithms. pp. 25–27. Springer (1996)

16. Sherenaz W. Al-Haj Baddar, K.E.B.: Designing Sorting Networks: A new Paradigm. Springer (2011)
17. van Voorhis, D.C.: Toward a lower bound for sorting networks. In: Complexity of Computer Computations. pp. 119–129 (1972)
18. Waksman, A.: A permutation network. J. ACM 15(1), 159–163 (Jan 1968)