Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXVII, págs. 2967-2978 (artículo completo)
Alberto Cardona, Mario Storti, Carlos Zuppa. (Eds.)
San Luis, Argentina, 10-13 Noviembre 2008

ANÁLISIS DE PERFOMANCE EN EL PROCESAMIENTO PARALELO SOBRE CLUSTERS DEL PROBLEMA DEL PUZZLE N2-1.*

Victoria Sanz[a], Franco Chichizola[b], Armando E. De Giusti[c], Marcelo R. Naiouf[d] and Laura C. De Giusti[e]

Instituto de Investigación en Informática LIDI (III-LIDI), Facultad de Informática, Universidad Nacional de La Plata, 50 y 120 2do piso, La Plata, Argentina, http://lidi.info.unlp.edu.ar.
[a] Ayudante Diplomada Dedicación Semiexclusiva, vsanz@lidi.info.unlp.edu.ar
[b] Profesor Adjunto Dedicación Exclusiva, francoch@lidi.info.unlp.edu.ar
[c] Investigador Principal del CONICET, Profesor Titular Dedicación Exclusiva, degiusti@lidi.info.unlp.edu.ar
[d] Profesor Titular Dedicación Exclusiva, mnaiouf@lidi.info.unlp.edu.ar
[e] Profesor Adjunto Dedicación Exclusiva, ldgiusti@lidi.info.unlp.edu.ar

Palabras clave: Optimización discreta, Algoritmos Paralelos, Procesamiento distribuido, Speedup, Superlinealidad, Eficiencia.

Resumen. En este trabajo se presenta un análisis de la solución paralela del Puzzle N2-1 utilizando clusters. El problema es especialmente interesante por su complejidad y sus posibles aplicaciones, en particular en el campo de la robótica.

Resolver el Puzzle N2-1 es un problema de optimización discreta tipo BFS (Best First Search), cuya complejidad y costo computacional favorecen el desarrollo de soluciones paralelas. Básicamente se trata de evolucionar desde un estado inicial, explorando en paralelo posibles recorridos, hasta alcanzar un estado solución. El número de movimientos requeridos es la función a optimizar.

En el trabajo se desarrolla una variación de la heurística clásica para la distribución de trabajo en el cluster, así como para decidir el avance del procesamiento paralelo en los procesos que se crean dinámicamente.

Posteriormente se presenta la solución paralela sobre cluster y se analiza el speedup alcanzable, en relación con el tiempo del algoritmo secuencial A*.

Un aspecto de interés es la posibilidad de superlinealidad (que surge de la clase de problema paralelo que representan los algoritmos de búsqueda en árboles como A*), así como el análisis de la eficiencia y escalabilidad de la solución distribuida desarrollada.

Se presenta un trabajo experimental con diferentes configuraciones de cluster, así como diferentes dimensiones del problema (variando N) y estados iniciales con distinto grado de desorden (calculado a partir de la distancia de Manhattan modificada).

Por último se discute la generalización del problema, para aplicarlo al movimiento de multi-robots con múltiples objetivos, y se analiza preliminarmente la migración del algoritmo a esquemas multi-cluster o Grid.

* Este trabajo está parcialmente financiado por la CIC, el Proyecto CyTED GRID, y la Fundación YPF

1 INTRODUCCIÓN

Los problemas de optimización discreta abarcan un gran número de áreas (Sergienko & Shylo, 2006) y a menudo son resueltos con métodos de exploración del espacio de estados, buscando un estado "solución" (Lambur & Shaw, 2004).

Estas técnicas de búsqueda normalmente tienen un alto costo computacional, e incluso en muchos casos es imposible el estudio exhaustivo del espacio de soluciones, de modo que debe recurrirse a heurísticas que aproximen una solución óptima (Grama & Kumar, 1999; Parberry, 1995).

La complejidad y el tiempo de cómputo impulsan el desarrollo de algoritmos paralelos para los problemas de optimización discreta y en particular las técnicas de procesamiento de grafos que representen el problema han sido de gran interés (Ferreira & Pardalos, 1996; Reinefeld, 1993).

Este es el caso de los métodos de BFS (Best First Search) que parten de un nodo del grafo que representa el problema y utilizan alguna métrica de estimación del trabajo para llegar a la solución, de modo de evolucionar a partir del estado inicial del grafo hacia el estado "solución óptima".

La paralelización natural de la técnica consiste en iniciar la evolución desde diferentes nodos "posibles" sobre los distintos procesadores de la arquitectura multiprocesador. A medida que el algoritmo progresa es necesario comunicar los procesadores para informar resultados alcanzados o bien descartar soluciones parciales de acuerdo a la métrica elegida (Hanson et al, 1992; Korf & Schultze, 2005).

Es interesante reflexionar sobre algunos aspectos que se dan al utilizar arquitecturas paralelas tipo clúster en la resolución de problemas de optimización discreta (Anderson et al, 1995):

- La granularidad de la paralelización es crítica, porque de ella dependerá la mejora en el tiempo de solución y también el overhead de comunicaciones.
- En general el balance de carga tiene que ser dinámico (lo que exige comunicación) ya que el trabajo exploratorio es variable y es muy difícil predecirlo a priori (Bohn & Lamont, 2002).

En general, en procesamiento paralelo el primer punto de interés en la resolución de un algoritmo sobre una arquitectura multiprocesador es el *factor de Speedup* ***Sp.*** Dicho factor es una medida de perfomance relativa definida como la relación entre el tiempo de ejecución del mejor algoritmo secuencial sobre una máquina monoprocesador y el tiempo de ejecución del algoritmo paralelo sobre una máquina multiprocesador (Grama et al, 2003; Leopold, 2001). Si llamamos ***Ts*** al tiempo de ejecución secuencial y ***Tp*** al paralelo, tenemos la relación ***Sp=Ts/Tp*** que normalmente se trata de maximizar en el desarrollo de aplicaciones paralelas. ***Sp*** está limitado por el máximo grado de concurrencia que puede obtenerse de la aplicación, por el inevitable componente secuencial del algoritmo y por el número de procesadores ***N*** disponibles para la ejecución (Qüiin, 1993).

Un segundo parámetro de importancia al analizar aplicaciones paralelas es la *Eficiencia* ***E*** alcanzada. Se define como Eficiencia la relación entre el Speedup y el número de procesadores utilizados para obtenerlo: ***E=Sp/N***. Esta definición pone la Eficiencia entre 0 y 1. Alcanzar valores cercanos a 1 significa que se logra ***Sp*** cercano al óptimo ***N***. ***E*** resulta una métrica de calidad y de costo del algoritmo paralelo que es particularmente importante y no siempre se puede mantener al escalar los problemas, al incrementar el número de procesadores o al portar el algoritmo sobre otra arquitectura multiprocesador (Buyya, 1999).

La *Escalabilidad* es un factor muy importante en las aplicaciones paralelas: normalmente los problemas "escalan", es decir aumenta el volumen de trabajo a realizar, y también las

arquitecturas multiprocesador que utilizamos pueden "escalar" incrementando los procesadores utilizados. Es de interés investigar el efecto de escalar trabajo y/o procesadores sobre el rendimiento de los algoritmos paralelos, considerando ***Sp*** y ***E*** (Hwang, 1993).

El máximo Speedup teórico puede en algunos casos ser mejorado y esto da lugar al concepto de *Superlinealidad* ***Su.*** Es interesante analizar por qué ***Sp*** puede superar ***N***, en particular en los problemas de optimización discreta en paralelo:

- La exploración del espacio total de soluciones posibles puede reducirse al distribuir el trabajo entre N procesadores y poder "cortar" o "finalizar" la búsqueda global al llegar al resultado esperado en cualquiera de ellos (Helmbold & McDowell, 1990; Manquinho & Marques-Silva, 2002). Es decir que en teoría la arquitectura de clúster podrá permitirnos alcanzar superlinealidad, dependiendo del balance de carga, la heterogeneidad de los procesadores y la relación tiempo de procesamiento/tiempo de comunicaciones del algoritmo empleado (Sanz, 2007).
- Si utilizamos arquitecturas distribuidas aún más débilmente acopladas (como multiclusters o Grid), la relación entre tiempo de procesamiento y tiempo de comunicación marcará un límite a la posibilidad de alcanzar superlinealidad (Wilkinson & Allien, 2005).

2 CONTRIBUCIÓN

En este trabajo se estudia el procesamiento paralelo sobre clusters para el caso del Puzzle N^2-1, un problema de optimización discreta NP complejo que tiene especial interés por la posibilidad de obtener superlinealidad (Sanz et al, 2007) y también por su aplicación en problemas de planificación de movimientos de robots (Fitch et al, 2005).

Las contribuciones de este trabajo son:

- Incorporar al algoritmo presentado en (Sanz et al, 2007) la detección "a priori" de la posibilidad de resolución de una configuración inicial para una configuración final dada.
- Analizar la implementación de una variante (DMCL) de la heurística de predicción de trabajo que combina la distancia de Manhattan (DM) con la detección de conflictos lineales, que permite una mejora muy significativa en la respuesta en tiempo del algoritmo, tanto secuencial como paralelo (Hanson et al, 1992).
- Presentar una serie de resultados experimentales sobre tableros con diferente dimensión y sobre clusters de 4, 6, 8, 12 y 16 procesadores, analizando perfomance (speedup, eficiencia, superlinealidad). Además se presenta un parámetro de trabajo local (LW) que puede influir en el balance de carga.
- Por último se discute la generalización del problema para aplicarlo con múltiples objetivos, y se analiza de manera preliminar la migración del algoritmo a esquemas multi-cluster o Grid.

3 CARACTERIZACIÓN DEL PROBLEMA DEL PUZZLE N2-1.

El problema del Puzzle N^2-1 es una generalización del Puzzle-15 ideado por Sam Lloyd (Ratner & Warmuth, 1990). Consiste en N^2-1 piezas numeradas de 1 a N^2-1 colocadas en un tablero de tamaño N^2. N^2-1 casilleros del tablero contienen exactamente una pieza, quedando sólo una casilla vacía la cual se denomina "hueco".

El objetivo del Puzzle es repetidamente llenar el hueco con una pieza adyacente a él en sentido horizontal o vertical, hasta alcanzar un tablero donde en la casilla (i,j) se encuentra la

pieza numerada como (i-1)*N + j y en la casilla (N,N) se encuentra el hueco.

La solución al problema planteado tendrá que ser aquella que minimice la cantidad de movimientos que deben realizarse para alcanzar la configuración final desde la configuración inicial dada.

La figura 1.a muestra un Puzzle N^2-1 donde N es 4. La figura 1.b representa el tablero solución utilizado.

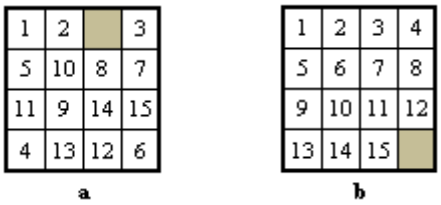

Figura 1: Tableros de Puzzle 15. **a.** Tablero inicial. **b.** Tablero final.

3.1 Casos solubles y no solubles.

No siempre el tablero final puede ser alcanzado a partir de un tablero inicial dado. Esto se debe a que solo la mitad de los estados es alcanzable desde cualquier otro, y por ello el espacio de búsqueda para el Puzzle N^2-1 se reduce a $N^2!/2$.

El procedimiento para verificar si un tablero inicial tiene solución para un tablero final es el siguiente:

- Para cada ficha *i* ($i = 2..N^2-1$), se cuenta la cantidad de piezas con menor número que aparecen después de ella, ya sea en la misma fila a su derecha, o en cualquier fila inferior. Llamaremos a este número "inversión de *i*", y se denotará n_i.
- A continuación se calcula para el tablero inicial y final $NT = n_2 + n_3 + \ldots n_{(N2-1)}$ (la suma de las inversiones de todas las fichas). Si N es par, se le suma a *NT* el número de la fila donde se encuentra el hueco.
- Si la paridad de ambos resultados es la misma, entonces el tablero final es alcanzable desde el tablero inicial. Esto se debe a que (*NT mod 2*) se mantiene invariante con cualquier movimiento legal.

En el ejemplo anterior, para la figura 1.a. *NT*=28 y para la figura 1.b. *NT*=4, por lo tanto el tablero de la figura 1.a tiene solución.

3.2 Distancia de Manhattan (DM)

Supongamos que cada posición del tablero se representa como un par ordenado. La distancia entre las posiciones (i,j) y (k,l) está definida como |i-k|+|j-l|. Dicha distancia recibe el nombre de *Distancia de Manhattan*. La suma de las distancias de Manhattan entre las posiciones del tablero x y el tablero final será un estimador mínimo del número de movimientos requeridos para transformar el tablero *x* en el tablero solución.

3.3 Conflictos lineales

Una heurística más afinada haría que el algoritmo procese menor cantidad de nodos, reduciendo el tiempo total de búsqueda.

Supongamos dos fichas *x* e *y*, llamamos "conflicto lineal" al caso en que están posicionadas en su fila correcta pero invertidas. Una de ellas deberá cambiar de fila para que la otra pueda pasar hasta su posición final y, luego de desplazarse a través de las columnas

como lo indica su DM, la ficha debe retornar a su fila destino. Lo mismo puede aplicarse a las columnas. Así, por cada conflicto lineal, se podría agregar 2 movimientos adicionales a la Distancia de Manhattan del tablero.

Una pieza podrá formar parte de a lo sumo un conflicto lineal por fila y uno por columna. Esta restricción evita que la heurística sobreestime el costo real, dejando así de ser admisible.

La Figura 2 presenta dicha situación en un tablero de *2x2*. Supongamos que $x > y$, y ambas piezas están en su fila correcta. Notar que la distancia de Manhattan de ambas fichas es 1.

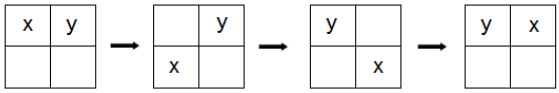

Figura 2: Secuencia de movimientos ante un conflicto lineal en una fila**.**

3.4 Distancia de Manhattan y Detección de Conflictos Lineales (DMCL)

La heurística de predicción de trabajo analizada en este trabajo (DMCL) se basa en lo propuesto por Hanson (Hanson et al, 1992) y combina la distancia de Manhattan y la detección de conflictos lineales, definiéndose con la fórmula 1.

$$DMCL\ (t) = DM(t) + CL(t) \tag{1}$$

Donde *t* representa el tablero actual, DM es la función que calcula la distancia de Manhattan para *t*, y *CL(t)* = (cantidad de conflictos lineales en *t*) * 2.

4 SOLUCIÓN SECUENCIAL UTILIZANDO LAS HEURISTICAS *DM* Y *DMCL*

A* es una de las variante de la técnica de búsqueda Best First Search (Reinefeld, 1993). Cada nodo *n* es valuado de acuerdo con el costo de alcanzar el mismo a partir de la raíz del árbol de búsqueda (***g(n)***) y una heurística que estima el costo para ir de *n* hasta un nodo solución (***h(n)***). Luego la función de costo será $L(x) = g(x) + h(x)$. Está garantizado que el algoritmo A* siempre encontrará la mejor solución.

Este algoritmo mantiene una lista de nodos no explorados (lista abierta) ordenada de acuerdo al valor de la función *L*, y otra lista de nodos ya explorados (lista cerrada) utilizada para evitar ciclos en el grafo de búsqueda. Inicialmente la lista abierta contiene un solo elemento, el nodo inicial, y la lista cerrada está vacía.

En cada paso, el nodo con menor valor *L* (el *mejor nodo*) es removido de la lista abierta y es examinado. Si es el nodo solución, el algoritmo termina. Caso contrario, el nodo es expandido e insertado en la lista cerrada. Cada nodo sucesor es insertado en la lista abierta sólo si no estaba en la lista cerrada, o estaba pero su valor L es menor al anterior.

```
Crear lista abierta.
Crear lista cerrada.
Insertar (lista abierta, x, h(x)).
// h(x) es la función heurística
mientras (lista abierta no vacía) y (no encontré solución)
      // Extraigo el mínimo nodo de la lista abierta, llamémoslo n
      n = EliminarMínimo (lista abierta)
      // Si h(n) = 0  termina el algoritmo, sino se expande n.
      si (EsSolución(n))
            solución = n
      sino
            hijos = Expandir(n)
            Insertar(lista cerrada,n)
              // Un nodo es aceptable si no está en la lista cerrada, o si está  pero con un costo mayor al actual para cada hijo de n
              si (EsAceptable(hijo))
                        Insertar(lista abierta, hijo, h(hijo) + g(n) + 1)
Retornar solución.
```

Figura 3: Algoritmo secuencial.

Se realizaron pruebas experimentales con las dos funciones heurísticas (*DM* y *DMCL*) presentadas anteriormente. Los resultados que se visualizan en la tabla 1, muestran claramente que el empleo de la heurística DMCL permite una sensible mejora en el tiempo de respuesta del algoritmo secuencial A* para encontrar una solución "óptima" al problema.

A partir de estos resultados, se adopta la heurística DMCL en el trabajo experimental de procesamiento paralelo sobre clúster.

Número de Prueba.	Profundidad de la solución.	Tamaño del tablero.	% Reducción de tiempo al usar DMLC.
1	38	4x4	63
2	39	4x4	89
3	40	4x4	80
4	42	4x4	98
5	43	4x4	99
6	34	5x5	84
7	34	5x5	95
8	36	5x5	90
9	38	5x5	82
10	39	5x5	85
11	40	6x6	96
12	42	6x6	99
13	44	6x6	95

Tabla 1: % de reducción en el tiempo de ejecución de algunas de las pruebas con DMLC respecto a DM.

5 ESQUEMA DE LA SOLUCIÓN PARALELA SOBRE CLUSTER.

La estrategia de paralelización consiste en mantener las listas *abierta* y *cerrada* locales en cada procesador. Al principio sólo un procesador (*Worker 0*) tendrá como trabajo el nodo inicial, quien también se encargará de detectar la terminación de la búsqueda. A medida que son generados otros nodos, los procesadores recibirán los mismos para comenzar a trabajar.

Todos los procesadores harán una búsqueda en el ámbito local, construyendo su propia lista cerrada, para evitar trabajo repetido localmente, como también su lista abierta. Los

procesadores deben comunicarse los valores mínimos de las soluciones encontradas, a fin de minimizar búsquedas innecesarias.

Para la implementación del algoritmo paralelo se utiliza la técnica de balance de carga distribuida "Asynchronous Round Robin" (ARR) y el algoritmo de "Terminación de Dijkstra" modificado (Dijkstra & Scholten, 1980).

Cada uno de los p procesos "workers" dispondrá un valor indicando el costo de la mejor solución encontrada hasta el momento (CMS), que se utilizará para acotar la búsqueda.

Un proceso que posee trabajo en su lista abierta a lo sumo procesa una cantidad fija de nodos en cada iteración (LW) o procesa nodos hasta encontrar una solución o que se vacíe su lista abierta. A continuación el "worker" recibe, si los hay, costos de "mejores soluciones" encontradas por los demás y a medida que esto ocurre actualiza (si es necesario) su variable CMS. De esta manera los nodos a procesar serán sólo los que tengan costo menor a CMS.

Si el proceso todavía posee trabajo en su lista abierta, verifica si ha recibido pedidos de trabajo de otros procesos, caso en el cual envía nodos del principio y final de su lista abierta al trabajador solicitante ocioso. Luego continúa trabajando sobre sus nodos.

En caso contrario, el proceso está ocioso, por lo que envía un pedido de trabajo a su donador siguiendo el algoritmo ARR. Si había encontrado una nueva solución, envía a todos los demás procesos el costo de la misma. Luego espera los tipos de mensajes enumerados a continuación, los cuales serán atendidos sin prioridad alguna:

- Petición de trabajo: un trabajador ocioso seleccionó a este proceso como su donador.
- Trabajo: el donador envía el trabajo requerido. Ahora el proceso está activo nuevamente.
- Rechazo de pedido de trabajo: el donador seleccionado no tiene trabajo. El proceso debe enviar un mensaje de pedido de trabajo al próximo donador.
- Token: recepción del token para detección de terminación. Si es necesario se actualiza el token y se pasa al siguiente proceso. El proceso 0, al recibir el token, verifica si debe terminar la búsqueda y en ese caso envía un mensaje a los demás procesos avisando el fin del cómputo.
- Nueva solución encontrada: en caso de ser necesario, se actualiza la variable CMS.

Se utiliza el token de terminación para trasladar los movimientos de la solución de costo mínimo hasta el proceso 0, de forma que los mensajes de comunicación de nuevas soluciones encontradas durante el algoritmo sólo posean un valor entero, el costo, evitando así overhead de comunicación.

6 RESULTADOS EXPERIMENTALES CON LA HEURÍSTICA *DMCL*.

Para las pruebas se dispone de un clúster homogéneo compuesto por 20 procesadores Pentium 4 de 2.4GHz y 1GB de memoria RAM.

Para estudiar la performance del algoritmo paralelo desarrollado, se realizaron pruebas para diferentes estados iniciales de tableros de 4x4, 5x5 y 6x6, utilizando en todos los casos $LW = \{250, 500, 750 \text{ y } 1000\}$.

Para analizar el comportamiento de la aplicación al escalar la arquitectura, cada tablero se probó con P procesadores pertenecientes al cluster antes mencionado, donde $P = \{4, 6, 8, 12, 16\}$.

Para observar la forma en que el algoritmo escala al aumentar el tamaño del problema (tableros más grandes), se definieron dos tipos de configuración inicial:

- *Configuración 1*: invertir en el subtablero de 4x4 inferior derecho la tercera columna y luego la tercera fila.
- *Configuración 2*: invertir en el subtablero de 4x4 inferior derecho la segunda columna y luego la segunda fila.

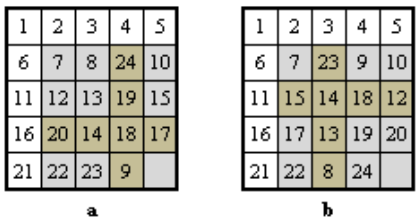

Figura 4: Ejemplos para tableros de 5x5 a. Configuración 1 b. Configuración 2.

Las tablas 2 y 3 muestran los resultados para las pruebas con la configuración 1 y 2 respectivamente, con los diferentes tamaños de tablero y con distinta cantidad de procesadores. En cada caso se muestra el valor de LW que optimiza el tiempo final del algoritmo paralelo.

Tablero	Cantidad de Procesadores	LW	Speedup	Eficiencia
4x4	4	1000	11,44	2,86
	6	1000	12,69	2,12
	8	1000	14,07	1,76
	12	750	25,42	2,12
	16	1000	60,58	3,79
5x5	4	500	15,94	3,99
	6	750	22,97	3,73
	8	750	39,02	4,88
	12	750	51,25	4,27
	16	1000	52,78	3,30
6x6	4	1000	6,70	1,67
	6	1000	13,62	2,27
	8	1000	16,99	2,12
	12	1000	25,10	2,09
	16	1000	32,68	2,04

Tabla 2: Speedup, eficiencia y LW óptimo para las pruebas de la configuración 1.

La figura 5 resume gráficamente la eficiencia logrado en las pruebas, dejando en claro cuando se obtiene superlinealidad.

Tablero	Cantidad de Procesadores	LW	Speedup	Eficiencia
4x4	4	1000	13,49	3,37
	6	1000	25,21	4,20
	8	1000	34,75	4,34
	12	1000	52,82	4,40
	16	500	74,52	4,66
5x5	4	250	18,06	4,52
	6	250	36,44	6,07
	8	250	54,11	6,76
	12	250	81,84	6,82
	16	500	87,20	5,45
6x6	4	500	10,16	2,54
	6	500	16,90	2,82
	8	500	22,26	2,78
	12	500	29,26	2,44
	16	500	41,00	2,56

Tabla 3: Speedup, eficiencia y LW óptimo para las pruebas de la configuración 2.

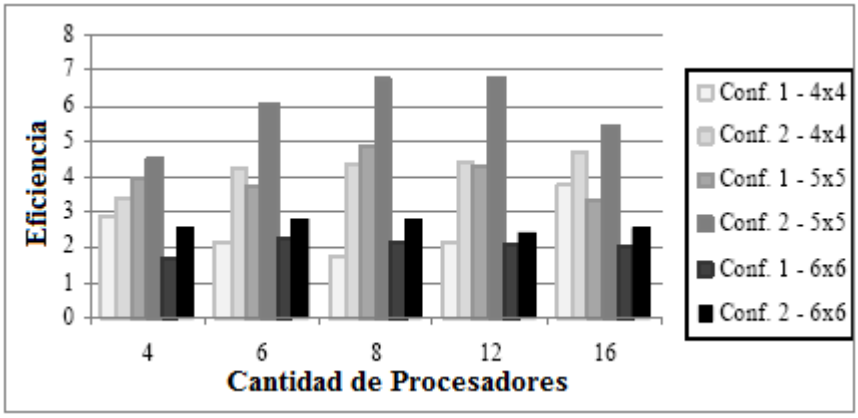

Figura 5: Eficiencia de las pruebas.

Es de interés analizar las causas de la superlinealidad obtenida en la experimentación. En general, en los problemas de búsqueda en grafos es posible obtener superlinealidad. Esto se debe a que el espacio total de búsqueda se distribuye entre los procesadores, permitiendo que los mismos encuentren soluciones sub-óptimas que, al comunicarlas, permiten podar o cortar ramas que llevan a soluciones no óptimas en otros procesadores. Al encontrar la solución en uno de los procesadores, se termina la búsqueda. Así, en la mayoría de los casos, se ha notado una reducción en la cantidad de nodos que procesa el algoritmo paralelo, en comparación con el algoritmo secuencial, acompañado de un aumento en la cantidad de nodos podados.

Otro causante de la superlinealidad es la reducción en cada procesador del tamaño de las listas en la estructura utilizada como lista cerrada. Al distribuir el trabajo entre N procesadores, cada lista cerrada es de menor tamaño y las búsquedas para la eliminación de ciclos son más rápidas, haciendo más probable que quepan en memoria.

Se calculó la cantidad de nodos procesados por el algoritmo secuencial y paralelo para diferentes estados iniciales. Las pruebas realizadas cuyos resultados de speedup fueron superlineales muestran un porcentaje de reducción de nodos procesados por procesador mayor o igual al 52 %. A medida que aumenta la cantidad de procesadores, crece la comunicación,

por lo que el speedup tenderá a decaer. Por lo tanto, se requiere un mayor porcentaje de reducción de nodos procesados para alcanzar superlinealidad a medida que escala la arquitectura.

Se calculó también el tamaño promedio de las listas de la estructura utilizada como lista cerrada, tanto en el algoritmo secuencial como paralelo. Se ha notado una disminución mayor o igual al 41 % en el tamaño de dicha estructura en la ejecución paralela en aquellas pruebas que muestran resultados superlineales.

Al aumentar el tamaño del tablero se observa una disminución del speedup. Se destacan a continuación las posibles causas:

- El espacio de estados crece exponencialmente a medida que N aumenta, por lo que se ha notado un incremento en la cantidad de nodos expandidos por el algoritmo.
- Cada comunicación por petición de trabajo requerirá enviar mayor volumen de datos debido a que los tableros son más grandes.

Cabe destacar también el rol que juega el parámetro LW presentado en la sección 5. Dicha medida es la cantidad de nodos que cada procesador trabaja por iteración del algoritmo, para luego después hacer el chequeo de peticiones de trabajo hechas por otros procesadores y también examinar soluciones óptimas encontradas por otros procesos.

Un valor grande de LW podría hacer que los procesadores ociosos a espera de trabajo permanezcan en dicho estado por un largo tiempo. También, al no recibir los costos de soluciones sub-óptimas encontradas por otros, podrían procesar nodos innecesarios.

Un valor muy chico de LW permitiría chequeos más frecuentes, pero podría causar un aumento en los tiempos debido a las constantes verificaciones.

Esto significa que el valor óptimo de LW dependerá de la configuración inicial y del grafo que de esta se genere, y tendrá efectos sobre el balance de carga.

Se realizaron distintas pruebas para diferentes estados iniciales y se observó un comportamiento similar al observado en la Tabla 2, Tabla 3 y en la Figura 5. Se encuentra un conjunto completo de resultados en (Sanz, 2007).

7 GENERALIZACIÓN DEL PROBLEMA

Actualmente se está investigando una generalización multi-objetivo del problema, que hemos denominado 4-Puzzle N^2-1 con las siguientes características:

- El problema consiste de una sucesión de M tableros generados al azar que admiten 4 soluciones posibles, cada una de las cuales es un "tablero objetivo" T_i con $1 <= i <= 4$.

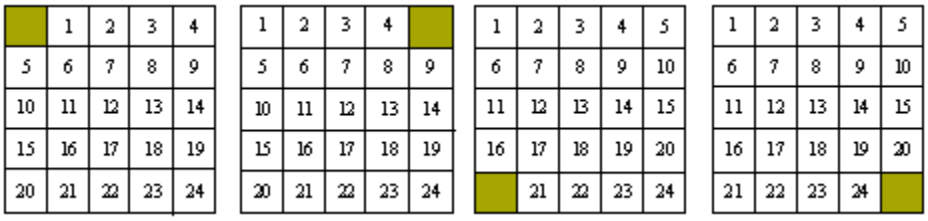

Figura 6: Tableros solución para el problema general.

- El análisis de los casos solubles y el espacio de soluciones posibles indicado en el punto 3.1 sigue siendo válido para cualquiera de los 4 "tableros objetivo" definidos. El análisis detallado de los "tableros objetivos" alcanzables es diferente para *N* par o

impar, pero responde a lo expuesto en 3.1, lo que permite la distribución de tareas entre procesadores.

- Si se utiliza la métrica DMCL definida en 3.4 como un estimador del objetivo más fácilmente alcanzable, estadísticamente para M creciente se balancea la distribución de trabajo entre los 4 T_i propuestos.
- La paralelización del problema se puede resolver en dos niveles: en principio una distribución funcional según el T_i objetivo y luego, la paralelización del algoritmo descripta en la sección 5.
- Interesa estudiar la resolución de este problema sobre clusters, determinando la bondad de la métrica DMLC como estimador del objetivo más probable de alcanzar en menor número de movimientos y también el paradigma de programación paralela a utilizar para maximizar speedup y eficiencia.

8 CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURO

Se ha presentado un análisis de la solución paralela para el problema del Puzzle N^2-1 sobre clusters, incorporando en la implementación la heurística DMCL. La misma considera la distancia de Manhattan y los conflictos lineales para la estimación del trabajo a realizar para alcanzar la solución desde un nodo del grafo, de modo de descartar alternativas que no puedan competir en la función a optimizar (número de pasos para alcanzar la solución).

Se ha analizado la ventaja de la heurística DMCL respecto de la distancia de Manhattan, tanto para el algoritmo secuencial como para el paralelo, y se ha estudiado speedup, eficiencia y superlinealidad para diferentes configuraciones de la arquitectura de cluster y diferentes dimensiones y estados iniciales del problema.

Se observa que no existe un valor de *LW* óptimo para todas las pruebas, sino que éste depende de diferentes factores. Una línea de investigación futura es el estudio de la relación entre estos factores: tamaño del tablero, desorden inicial y cantidad de procesadores; para poder estimar a priori el valor óptimo de *LW* para cada prueba en particular.

Actualmente se está estudiando la heurística propuesta en (Korf & Taylor, 1996), basada en DMCL, que permite mejorar la elección del tablero solución más cercana al tablero inicial en el problema propuesto en 7. También se está estudiando la aplicación de la generalización del problema propuesto en la sección 7.a casos de movimiento de robots, en particular multirobots con múltiples objetivos.

Otra línea actual de trabajo es la migración del algoritmo a plataformas multi-cluster y grid. Una estrategia posible en ese sentido es tomar en cuenta la potencia de cálculo de los procesadores/clusters, resolviendo los problemas más simples en los menos potentes y los más complejos en los de mayor potencia; en cada cluster la resolución puede realizarse utilizando la paralelización descripta en la sección 5.

REFERENCIAS

Anderson T., Culler D., Patterson D. A Case for NOW (Networks of Workstations). *IEEE Micro 1995;* 15(1): pp. 54-64.

Bohn C., Lamont G. *Load Balancing for Heterogeneous Clusters of PCs. Future Generation Computer Systems*, 2002; 18(3): 389-400.

Buyya R. *High Performance Cluster Computing: Architectures and Systems*. Prentice-Hall; 1999.

Dijkstra E., Scholten C. *Termination detection for diffusing computations. Information Processing Letters* 1980; 11(1):1-4.

Ferreira A., Pardalos P. *Solving Combinatorial Optimization Problems in Parallel: Methods and Techniques.* New York: Springer; 1996.

Fitch R., Butler Z., Rus D. Reconfiguration Planning Among Obstacles for Heterogeneous Self-Reconfiguring Robots. *In: Proc. of the IEEE International Conference on Robotics and Automation*; 2005. p. 117-124.

Grama A., Gupta A., Karypis G., Kumar V. *An Introduction to Parallel Computing. Design and Analysis of Algorithms*. Pearson Addison Wesley; 2003.

Grama A., Kumar V. State of the art in parallel search techniques for discrete optimisation problems. *IEEE Trans. on Knowledge and Data Engineering*, 1999.

Hanson O., Mayer A., Yung M. Criticizing Solutions to Relaxed Model Yields Powerful Admissible Heuristics. *Information Sciences 1992*; 63(3): 207-227.

Helmbold D., McDowell C. Modeling speedup(n) greater than n. *IEEE Trans. on Parallel and Distributed Systems*, 1990; 1(2): 250-256.

Hwang K. *Advanced Computer Architecture. Parallelism, Scalability, Programmability*. McGraw Hill; 1993.

Korf R. E., Schultze P. Large-scale parallel breadth-first search. *In: Proc. of the 20th National Conference on Artificial Intelligence; Pittsburgh*, USA; 2005. p. 1380-1385.

Korf R. E., Taylor R. "Finding Optimal Solutions to the Twenty-Four Puzzle", 1996. *In: Proc. of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*

Lambur H, Shaw B. *Parallel State Space Searching Algorithms*. 2004. www.metablake.com/parallel_search_project

Leopold C. Parallel and distributed computing. *A survey of models, paradigms, and approaches.* New York: Wiley; 2001.

Manquinho V., Marques-Silva J. Search Pruning Techniques in SAT-Branch-and-Bound Algorithms for Binate Covering Problem. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* 2002; 21(5): 505-516.

Papadimitriou C., Raghavan P., Sudan M., Tamaki H. Motion Planning on a Graph. *In: Proc. of the 35th Annual Symposium on Foundations of Computer Science*; 1994.

Parberry I. *A Real Time Algorithm for the (n2-1) Puzzle. Information Processing Letters* 1995;56(1): 23-28.

Quiin M. J. *Parallel Computing: Theory and Practice*. McGraw-Hill Companies; 1993.

Rao V. N., Kumar V. On the efficiency of parallel backtracking. *IEEE Trans. on Parallel and Distributed System*, 1993; 4(4): 427-437.

Ratner D., Warmuth M. The (n2−1)-puzzle and related relocation problems. *Journal for Symbolic Computation* 1990; 10(2):11–137.

Reinefeld A. Complete Solution of the Eight-Puzzle and the Benefit of Node Ordering in IDA*. *In: Proc. of the 13th International Joint Conference on Artificial Intelligence*; Chambery Savoi, France; 1993. p. 248-253.

Sanz V. *Paralelización de N-Puzzle*. Technical Report 2007.

Sanz V., Chichizola F., Naiouf M., De Giusti L.,De Giusti A. Superlinealidad sobre Clusters. Análisis experimental en el problema del Puzzle N2-1. *In: Proc. of the XIII Congreso Argentino de Ciencias de la Computación*; Chaco-Corrientes, Argentina; 2007. p. 1300-1309.

Sergienko I., Shylo V. *Problems of discrete optimization: Challenges and main approaches to solve them*. New York: Springer; 2006; 42(4):465-482.

Wilkinson B., Allien M. *Parallel Programming: Techniques and Applications Using Network Workstation and Parallel Computers*. Pearson Prentice Hall; 2005.