UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

RECONOCIMIENTO DE PATRONES EN SIMULACIÓN GEOESTADÍSTICA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

OSCAR FRANCISCO PEREDO ANDRADE

PROFESOR GUÍA:
SR. JULIÁN ORTIZ CABRERA

MIEMBROS DE LA COMISIÓN:
SR. CARLOS HURTADO LARRAÍN
SRA. MARÍA CECILIA RIVARA ZUÑIGA

SANTIAGO DE CHILE
AGOSTO 2008

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN
POR: OSCAR FRANCISCO PEREDO ANDRADE
FECHA: 22/08/2008
PROF. GUÍA: JULIÁN ORTIZ CABRERA

RECONOCIMIENTO DE PATRONES EN SIMULACIÓN GEOESTADÍSTICA

La evaluación de yacimientos mineros tiene por objetivo estimar, con el menor error posible, la calidad y cantidad de un recurso mineral, que tiene potencial de ser explotado. Una de las principales herramientas utilizadas es la *Geoestadística*, la cual pone énfasis en el contexto geológico y la relación espacial entre los datos a estudiar. Los métodos de estimación tradicionales, llamados *Kriging*, calculan el mejor estimador lineal insesgado de una variable en base a los valores de datos vecinos. Para representar de mejor manera la variabilidad espacial de los datos, se recurre a la *Simulación Convencional*, que consiste en añadir un factor aleatorio a *Kriging* para obtener una distribución de escenarios, que sirven como medida de la respuesta ante la incerteza.

El inconveniente de esta técnica es que no permite correlacionar diversos puntos al mismo tiempo, sólo se pueden correlacionar dos puntos a la vez (se basa en la covarianza de 2 puntos). Este hecho representa una potencial pérdida de información en la construcción del modelo que se utilizará para realizar estimaciones o simulaciones. Al correlacionar dos puntos a la vez, se induce una suavización sobre las estimaciones, lo cual puede entregar resultados sobreestimados o subestimados.

Para solucionar el problema anterior, se propone utilizar métodos que contemplan estadísticas de múltiples puntos, representadas por *patrones*. Se espera que la utilización de patrones junto a un método de simulación no convencional llamado Recocido Simulado (*Simulated Annealing*) puedan tener un mejor desempeño al evaluar un proceso que la simulación convencional.

Los principales problemas del método propuesto son el tiempo de cálculo del Recocido Simulado y el manejo de patrones grandes. Se optó por resolver el primero de ellos y dejar abierta la puerta para posibles trabajos a futuro en la optimización de las estructuras de datos para el manejo de patrones grandes.

El tiempo de cálculo del Recocido Simulado aumenta exponencialmente al incrementar el tamaño de los patrones a estudiar. Para solucionar esto, se realizó una implementación utilizando múltiples procesos, configurados según una técnica llamada *Computación Especulativa*, la cual permite realizar varios pasos de la simulación a la vez. Se estudió el speedup teórico de la paralelización y se realizaron pruebas numéricas, las cuales entregaron resultados satisfactorios en tiempo y en calidad de la imagen simulada.

AGRADECIMIENTOS

Agradezco a mi familia y mis seres queridos por el apoyo incondicional y desinteresado. También agradezco a Julián Ortiz, del Depto. de Ing. en Minas, por sus incontables aportes y recomendaciones, además de la gran dedicación que brinda a sus estudiantes. Igualmente agradezco la confianza entregada por el profesor del Depto. de Ing. en Minas Enrique Rubio y al ingeniero Juan Pablo Gil del CEMCC-UFRO por facilitarme las instalaciones y servidores del Laboratorio de Planificación Minera, dependiente del Depto. de Ing. en Minas de la U. de Chile. Finalmente agradezco a todas las personas que me entregaron su apoyo, recomendaciones, indicaciones, etc. entre las cuales hay alumnos (cantina 434 y amigos DCC) y profesores del Departamento de Ciencias de la Computación y Departamento de Ingeniería Matemática, de la U. de Chile.

Esta memoria fue financiada por el Proyecto FONDECYT Regular 1040690, titulado *Evaluación de Yacimientos mediante Simulación Estocástica integrando Estadísticas de Múltiples Puntos*.

Índice general

Índice de figuras

Índice de algoritmos

Capítulo 1

Introducción

1.1. Contexto

La *evaluación de yacimientos* se refiere a la evaluación de recursos y reservas minerales en un yacimiento, para poder tomar decisiones respecto a su explotación y a la rentabilidad del proyecto minero.

Esta área de estudio contempla diversos métodos que tienen por objetivo estimar (con el menor error posible) la calidad (ley), cantidad (tonelaje) y ubicación de un recurso mineral, que tiene potencial de ser explotado, y evaluar su incertidumbre.

Para ello, las empresas mineras invierten grandes sumas de dinero en campañas de sondajes, exploraciones, análisis de muestras y maquinaria, entre otros items. Toda la inversión realizada en la evaluación tiene por objetivo determinar si el yacimiento es rentable, y en caso de serlo, mejorar la fase de planificación (de corto y largo plazo), lo que se traduce en ganancias directas para la empresa, pues se sabe con mayor precisión donde está ubicado el mineral dentro del depósito.

La principal herramienta utilizada en la evaluación de yacimientos es la *Geoestadística*, rama de la Estadística aplicada que pone énfasis en el contexto geológico y la relación espacial entre los datos. Se desarrolla a partir de trabajos de H. Sichel y D. G. Krige en los años 50 y es formalizada por G. Matheron en los años 60 ([Kri51], [Sic52], [Mat62], [Mat63]). Sus orígenes radican en la minería, sin embargo, sus metodologías son usadas en diversas áreas como por ejemplo: petróleo, geotecnia, pesca, silvicultura, oceanografía e hidrología.

La idea básica consiste en estimar el valor de una variable, cuyo verdadero valor se desconoce, en una posición $\mathbf{u}_0 \in \mathbb{R}^2$, la cual se denota $Z(\mathbf{u}_0)$. Ese estimador se denota $Z^*(\mathbf{u}_0)$. Una manera de calcular esa estimación es utilizar observaciones $Z(\mathbf{u}_1), ..., Z(\mathbf{u}_n)$ ubicadas en posiciones *cercanas* (según sea el criterio de cercanía) $\mathbf{u}_1, \ldots, \mathbf{u}_n \in \mathbb{R}^2$.

Una forma de utilizar las observaciones cercanas en la estimación es considerar una ponderación lineal de ellas, es decir

$$Z^*(\mathbf{u}_0) = a + \sum_{i=1}^{n} \lambda_i Z(\mathbf{u}_i) \tag{1.1}$$

donde $a \in \mathbb{R}$ es un coeficiente aditivo y $\{\lambda_i \in \mathbb{R} : i = 1, \ldots, n\}$ son ponderadores. El objetivo de este enfoque es encontrar los mejores ponderadores según sea el método que se escoja para estimar. Para la asignación de los ponderadores se pueden considerar diversos criterios:

- Distancia a la posición que está siendo estimada.
- Redundancia entre los valores de los datos.
- Continuidad o variabilidad espacial.
- Anisotropía (propiedad general de la materia según la cual determinadas propiedades físicas, tales como la elasticidad, temperatura, conductividad, velocidad de propagación de la luz, etc., varían según la dirección en que son examinadas).

Los métodos de estimación tradicionales en Geoestadística, llamados *Kriging* ([Ort07], [Deu02], [Goo97]), calculan el mejor estimador lineal insesgado (*Best Linear Unbiased Estimator*) $Z^*(\mathbf{u}_0)$ del valor $Z(\mathbf{u}_0)$, considerando a las variables $Z(\mathbf{u}_i), i = 0, 1, ..., n$ como variables aleatorias e incluyendo los 4 criterios anteriormente mencionados. Existen diversos tipos de *Kriging*: Simple, Ordinario, con deriva, no lineal, Cokriging, etcétera. El tipo más sencillo de *Kriging* es el *Kriging Simple*, en el cual se consideran las siguientes hipótesis:

1. Se conoce el valor promedio m de la variable Z, es decir, $\mathbb{E}(Z(\mathbf{u}_i)) = m, i = 0, 1, \ldots, n$.
2. Se conoce el *variograma* $2\gamma(\mathbf{h})$ ([Goo97]). El *variograma* permite cuantificar la correlación espacial de la variable Z. Se define de la siguiente manera, para $\mathbf{h} \in \mathbb{R}^3$:

$$2\gamma(\mathbf{h}) = \mathbb{E}\left\{[Z(\mathbf{u}+\mathbf{h}) - Z(\mathbf{u})]^2\right\} \tag{1.2}$$

   con $\mathbf{u} \in \{\mathbf{u}_1, \ldots, \mathbf{u}_n\}$. El variograma es equivalente a la covarianza espacial $\text{Cov}(Z(\mathbf{u+h}), Z(\mathbf{u}))$, en el siguiente sentido:

$$C(\mathbf{h}) = C(\mathbf{0}) - \gamma(\mathbf{h}) \tag{1.3}$$

   donde $C(\mathbf{h}) = \text{Cov}(Z(\mathbf{u+h}), Z(\mathbf{u}))$ y $C(\mathbf{0}) = \text{Var}(Z(\mathbf{u}))$.

Tras las hipótesis, se procede a imponer dos condiciones fundamentales:

- **Condición de Insesgo**:

  Se impone la condición $\mathbb{E}(Z^*(\mathbf{u}_0) - Z(\mathbf{u}_0)) = 0$:

$$\begin{aligned}
\mathbb{E}(Z^*(\mathbf{u}_0) - Z(\mathbf{u}_0)) &= \mathbb{E}(a + \sum_{i=1}^{n} \lambda_i Z(\mathbf{u}_i)) - \mathbb{E}(Z(\mathbf{u}_0)) && (1.4)\\
&= a + \sum_{i=1}^{n} \lambda_i \mathbb{E}(Z(\mathbf{u}_i)) - \mathbb{E}(Z(\mathbf{u}_0)) && (1.5)\\
&= a + \sum_{i=1}^{n} \lambda_i m - m && (1.6)
\end{aligned}$$

  Luego, para que la condición se cumpla se debe cumplir $a = m(1 - \sum_{i=1}^{n} \lambda_i)$.

- **Condición de Mínima Varianza**:

  Se desea encontrar los ponderadores que minimizan la varianza de $Z^*(\mathbf{u}_0) - Z(\mathbf{u}_0)$. Para ello, se deriva la varianza con respecto a los λ_i (derivadas parciales) y se iguala a cero donde corresponda (se considera $a = 0$ para simplificar los cálculos):

$$\begin{aligned}
\text{Var}(Z^*(\mathbf{u}_0) - Z(\mathbf{u}_0)) &= \mathbb{E}((Z^*(\mathbf{u}_0) - Z(\mathbf{u}_0))^2) && (1.7)\\
&= \mathbb{E}((Z^*(\mathbf{u}_0))^2) - 2\mathbb{E}(Z^*(\mathbf{u}_0)Z(\mathbf{u}_0)) + \mathbb{E}((Z(\mathbf{u}_0))^2) && (1.8)\\
&= \sum_{i=1,j=1}^{n} \lambda_i \lambda_j \mathbb{E}(Z(\mathbf{u}_i)Z(\mathbf{u}_j)) && (1.9)\\
&\quad -2\sum_{i=1}^{n} \lambda_i \mathbb{E}(Z(\mathbf{u}_i)Z(\mathbf{u}_0)) + \mathbb{E}((Z(\mathbf{u}_0))^2) && (1.10)\\
&= \sum_{i=1,j=1}^{n} \lambda_i \lambda_j \text{Cov}(Z(\mathbf{u}_i), Z(\mathbf{u}_j)) && (1.11)\\
&\quad -2\sum_{i=1}^{n} \lambda_i \text{Cov}(Z(\mathbf{u}_i), Z(\mathbf{u}_0)) + \text{Var}(Z(\mathbf{u}_0)) && (1.12)
\end{aligned}$$

  Derivando se obtiene

$$\sum_{j=1}^{n} \lambda_j \text{Cov}(Z(\mathbf{u}_i), Z(\mathbf{u}_j)) = \text{Cov}(Z(\mathbf{u}_i), Z(\mathbf{u}_0)), \qquad \forall i = 1, ..., n \qquad (1.13)$$

  Resolviendo este sistema se encuentran los parámetros $\lambda_j, j = 1, \ldots, n$, con los cuales se realiza la estimación.

La covarianza entre un valor estimado por *Kriging* y un valor en algún punto de la grilla se calcula de la siguiente forma (se considera $a = 0$ para simplificar los cálculos):

$$\begin{aligned}
\text{Cov}(Z^*(\mathbf{u}_0), Z(\mathbf{u}_k)) &= \mathbb{E}(Z^*(\mathbf{u}_0)Z(\mathbf{u}_k)) && (1.14)\\
&= \mathbb{E}\left\{\sum_{i=1}^{n}\lambda_i Z(\mathbf{u}_i)Z(\mathbf{u}_k)\right\} && (1.15)\\
&= \sum_{i=1}^{n}\lambda_i \mathbb{E}(Z(\mathbf{u}_i)Z(\mathbf{u}_k)) && (1.16)\\
&= \sum_{i=1}^{n}\lambda_i \text{Cov}(Z(\mathbf{u}_i), Z(\mathbf{u}_k)) && (1.17)\\
&= \text{Cov}(Z(\mathbf{u}_0), Z(\mathbf{u}_k)) && (1.18)
\end{aligned}$$

El paso de (1.17) a (1.18) se debe a la igualdad (1.13). Se puede observar que *Kriging* induce a la covarianza entre los datos estimados y sus vecinos a ser igual a la covarianza entre los valores reales y sus vecinos. El problema ocurre cuando se desea calcular la varianza de los datos estimados (considerando $a = 0$ para simplificar los cálculos):

$$\begin{aligned}
\text{Var}(Z^*(\mathbf{u}_0)) &= \mathbb{E}((Z^*(\mathbf{u}_0))^2) && (1.19)\\
&= \mathbb{E}\left(\left(\sum_{i=1}^{n}\lambda_i Z(\mathbf{u}_i)\right)\left(\sum_{j=1}^{n}\lambda_j Z(\mathbf{u}_j)\right)\right) && (1.20)\\
&= \sum_{i=1,j=1}^{n}\lambda_i\lambda_j \mathbb{E}(Z(\mathbf{u}_i)Z(\mathbf{u}_j)) && (1.21)\\
&= \sum_{i=1,j=1}^{n}\lambda_i\lambda_j \text{Cov}(Z(\mathbf{u}_i), Z(\mathbf{u}_j)) && (1.22)
\end{aligned}$$

Aplicando las ecuaciones (1.11)-(1.12)-(1.13) en (1.22) se tiene que

$$\text{Var}(Z^*(\mathbf{u}_0)) = \text{Var}(Z(\mathbf{u}_0)) - \underbrace{\text{Var}(Z^*(\mathbf{u}_0) - Z(\mathbf{u}_0))}_{\sigma^2_{SK}(\mathbf{u}_0)} \quad (1.23)$$

Se puede observar que existe un factor $\sigma^2_{SK}(\mathbf{u}_0)$ que reduce el valor de la varianza de los valores estimados. La estimación entregada por *Kriging* induce una suavización sobre las estimaciones, lo cual puede entregar resultados sobreestimados o subestimados. Es por esto que esta estimación no debe ser utilizada para representar la variabilidad espacial de una variable (por ejemplo, cuando se quiere realizar un análisis de riesgo o evaluar incertidumbre).

Para solucionar lo anterior se recurre a la *Simulación Convencional* que consiste en añadir a *Kriging* un residuo aleatorio que permite reproducir la variabilidad que tiene la variable original,

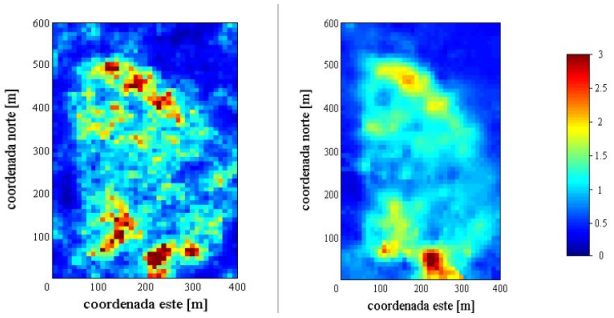


Figura 1.1: Valores reales (izquierda) y estimados vía Kriging (derecha)

por lo que las realizaciones pueden verse como otros escenarios posibles de la realidad:

$$Z_s(\mathbf{u}_0) = Z^*(\mathbf{u}_0) + R(\mathbf{u}_0) \tag{1.24}$$

El residuo aleatorio $R(\mathbf{u}_0)$ tiene por objetivo corregir la diferencia entre las varianzas del valor estimado y real, y también depende de las covarianzas o variograma de los datos. Una de las técnicas más utilizadas asociada a la Simulación Convencional se llama *Simulación Gaussiana Secuencial* ([Deu02],[Goo97],[DJ92]) y se basa en la adopción de variables aleatorias con distribuciones normales o gaussianas.

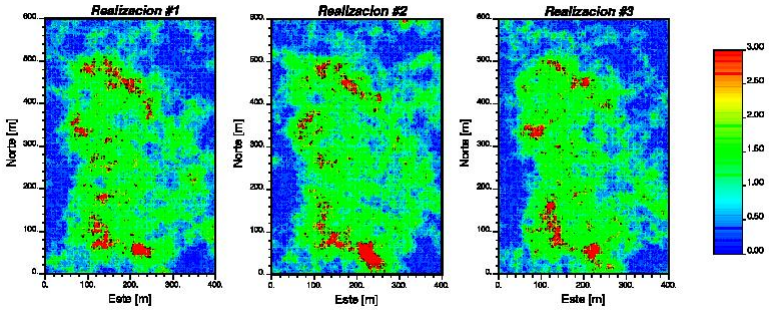


Figura 1.2: Realizaciones obtenidas utilizando Simulación Convencional (Secuencial Gaussiana)

Debido a la dependencia que presenta la Simulación Convencional sobre las covarianzas o variograma, las realizaciones simuladas reproducen la covarianza espacial, pero no reproducen otras estadísticas que consideran más de dos puntos a la vez, por ejemplo, *patrones de datos*.

En resumen, ***el inconveniente de esta técnica es que no permite correlacionar diversos puntos al mismo tiempo, sólo se pueden correlacionar dos puntos a la vez.*** Este hecho representa una potencial pérdida de información en la construcción del modelo que se utilizará para realizar estimaciones o simulaciones.

Para solucionar el problema anterior, se propone utilizar métodos que contemplan estadísticas de múltiples puntos, representadas por patrones. Por ejemplo, si se conocen los valores de las variables $Z(\mathbf{u}_1), ..., Z(\mathbf{u}_n)$, se quiere obtener una estimación de la probabilidad condicional

$$\mathbb{P}(Z(\mathbf{u}_0) = z | Z(\mathbf{u}_1) = z_1, ..., Z(\mathbf{u}_n) = z_n)$$

donde $z_1, ..., z_n$ pueden ser continuas o discretas. Para ello, se calculan las frecuencias de los patrones que aparecen en alguna vecindad definida con anterioridad, luego, se utilizan esas frecuencias para realizar simulaciones estocásticas. En el capítulo Antecedentes se detalla un ejemplo de esta estimación para un patrón de 4 puntos donde se conocen solamente 3 valores.

1.2. Justificación

En esta sección se plantean las áreas de estudio que presenta la implementación de un método que utilice estadísticas de múltiples puntos representadas por patrones. En particular se implementará el método del *Recocido Simulado* o *Simulated Annealing* ([KGV83]) proveniente de la Optimización Combinatorial.

1.2.1. Cálculo de frecuencias de patrones

El primer problema que se presenta es el conteo de apariciones de los patrones contenidos en una vecindad de un punto, debido a la magnitud del conjunto de patrones posibles y a la forma de la vecindad. Este conteo se realiza con el objetivo de inferir las frecuencias de patrones a partir de información de entrenamiento, las cuales se utilizarán en la simulación explicada en el segundo problema.

Si un punto $\mathbf{u}_0$ tiene una vecindad compuesta por N puntos, digamos $\mathbf{u}_1, ..., \mathbf{u}_N$, y si se conoce el valor de $Z(\mathbf{u}_i) \in \{1, \ldots, K\}$ para $i = 1, ..., N$, el número total de patrones que pueden ser generados es K^{N+1}. Claramente esta cantidad es inmanejable cuando K y N son lo suficientemente grandes.

La forma de la vecindad puede ser considerada irregular, lo cual puede traer dificultades en términos de indexación y búsqueda. En la implementación realizada en este trabajo, sólo se consideraron vecindades con forma de paralelepípedos.

Debido a que el enfoque de este trabajo es la implementación de la simulación usando las frecuencias, se le dará menos importancia a este problema, dejando abierta la posibilidad de resolverlo en el futuro.

Para validar la simulación, se espera implementar un método que entregue de manera rápida las frecuencias para valores pequeños de K y N.

1.2.2. Simulación usando las frecuencias calculadas

El segundo problema consiste en realizar de manera eficiente una simulación usando las frecuencias de los patrones inferidas en el primer problema.

El Kriging entrega estimaciones que minimizan la varianza de un atributo z de manera local, es decir, se minimiza $\text{Var}(Z^*(\mathbf{u}) - Z(\mathbf{u}))$, donde $Z^*(\mathbf{u})$ es el valor estimado. Si bien esta estimación es buena, tiene la desventaja de suavizar detalles que localmente no son suaves. Tipicamente, valores pequeños son subestimados y valores grandes son sobreestimados.

La Simulación Convencional genera una serie de realizaciones $z^{(l)}(\mathbf{u}), l = 1, ..., L$, del atributo z en cada punto $\mathbf{u}$ donde se realiza el estudio. La distribución o histograma de las L respuestas generadas por la simulación proveen una medida de la respuesta ante la incerteza dado que se tiene conocimiento imperfecto del valor real en el espacio del atributo z.

En este tipo de simulaciones, también se utiliza la covarianza espacial de los atributos en pares de puntos, suponiendo que se conocen a priori las leyes espaciales de los atributos z, por ejemplo, que siguen un modelo de función aleatoria Gaussiana Multivariada. Este hecho hace que estas simulaciones no capturen toda la información que puede ser relevante en el proceso.

Debido a lo anterior, existen otras clases de simulaciones basadas en aspectos diferentes a los convencionales (ejemplos específicos se pueden encontrar en [OD04], [Str02], [Deu02] y [Goo97]). Entre esas clases de simulaciones se encuentra el *Simulated Annealing* o *Recocido Simulado* [KGV83].

Uno de los primeros trabajos que relacionaron las estadísticas de múltiples puntos (patrones) y el Recocido Simulado fue realizado por Deutsch [Deu92] en su tésis de doctorado en 1992. La principal ventaja que presenta este método es la flexibilidad que presenta para trabajar con patrones de más de 2 puntos. Esta flexibilidad viene dada por la sencillez del algoritmo y su fácil adaptación a distintos problemas. El principal problema tiene que ver con el tiempo de cálculo requerido para obtener una buena solución, ya que a medida que el tamaño del patrón va aumentando, la cantidad de operaciones requeridas crece de manera exponencial, y más aún, se deben realizar una cantidad no despreciable de iteraciones para llegar a soluciones óptimas (locales).

Se espera que esta clase de simulaciones, junto al cálculo de frecuencias de patrones 2D como información relevante y a la posible disminución del tiempo de cálculo, pueda tener un mejor

desempeño para evaluar un proceso que la Simulación Convencional.

1.3. Objetivos

1.3.1. Objetivo General

El objetivo general de este trabajo consiste en diseñar e implementar un sistema computacional que permita simular mediante Recocido Simulado y en base a estadísticas de patrones 2D de la distribución de leyes en yacimientos mineros, utilizando imágenes de entrenamiento 2D, las cuales se obtienen de modelos conceptuales o luego de extraer muestras del depósito, y están compuestas por datos categóricos $Z(\mathbf{u}) = k$, con $k \in \{k_1, \ldots, k_K\}$.

En base a estas imágenes, se obtienen las frecuencias de aparición de patrones 2D. Estas frecuencias se utilizarán al realizar las simulaciones del depósito, utilizando el Recocido Simulado, explicado en el capítulo siguiente. En cada iteración de la simulación se calculan las frecuencias asociadas y se comparan con las frecuencias obtenidas de la imagen de entrenamiento. Si la comparación entrega un buen resultado, la imagen calculada en esa iteración es una candidata a ser solución del método.

1.3.2. Objetivos Específicos

Como objetivos específicos, se plantean los siguientes puntos:

1. Implementación de la técnica del Recocido Simulado aplicado al problema de la simulación de imágenes de entrenamiento 2D.
2. Mejoramiento de los tiempos de cálculo para el Recocido Simulado utilizando Programación Paralela.
3. Implementación de una interfaz que permita al usuario ingresar los parámetros de la simulación usando el Recocido Simulado y especificar la geometría de los patrones. Ésta interfaz se realizará en el ambiente de trabajo utilizado, en este caso, en los servidores del Laboratorio de Planificación Minera, en forma de scripts documentados.

1.4. Plan de este documento

En el capítulo 2 se describe en detalle la utilización de las frecuencias de los patrones en la simulación, así como el método utilizado para obtenerlas. También se detalla el método de

simulación implementado, el Recocido Simulado, describiendo el algoritmo, su convergencia y su paralelización. Finalmente se describe el *speedup* teórico que se puede llegar a obtener al realizar la paralelización.

En el capítulo 3 se detalla la implementación. Se describen las principales herramientas utilizadas, el ambiente de trabajo y los principales aspectos del código implementado. También se describen las secuencias de comunicación entre los procesos, al realizar la paralelización.

Los resultados de la simulación y los tiempos de ejecución se presentan en el capítulo 4. Se describen las pruebas realizadas con sus respectivos resultados. Se realizan comparaciones de tiempos de ejecución real y teórico.

Finalmente se explican las conclusiones y el trabajo a futuro en el capítulo 5.

Capítulo 2

Antecedentes

2.1. Cálculo de Frecuencias

Se entenderá por **template** a una estructura ordenada de celdas sin información en el espacio. La única información relevante que aporta un template es la posición de cada celda, con respecto a un origen. En la figura 2.1 se observa un template de 4 celdas.

Figura 2.1: Template de 4 celdas

Se entenderá por **patrón** a una estructura ordenada de celdas con información en el espacio. La posición de cada celda está definida por un **template**. Usualmente la información corresponderá a un valor binario asociado a que el valor de una variable en esa celda sobrepase cierto umbral o no, o la presencia/ausencia de una categoría. En la figura 2.2 se observan los patrones asociados al template de 4 celdas de la figura 2.1.

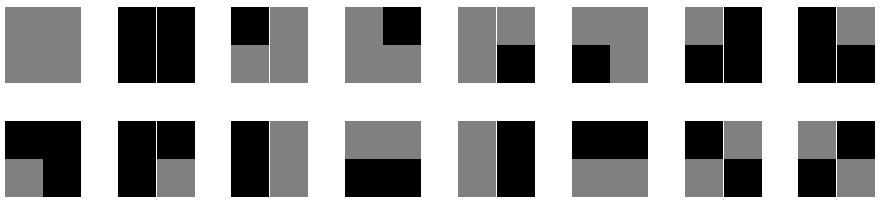


Figura 2.2: Patrones asociados al template de la figura 2.1

El cálculo de frecuencias de los patrones se realiza para tener un estimador de la probabilidad condicional

$$\mathbb{P}(Z(\mathbf{u}_0) = z | Z(\mathbf{u}_1) = z_1, ..., Z(\mathbf{u}_n) = z_n) \tag{2.1}$$

Supongamos que se tiene una configuración como en la figura 2.3 y veamos la utilidad de las frecuencias en la estimación de 2.1.

$\mathbf{u}_1$	$\mathbf{u}_3$
$\mathbf{u}_0$	$\mathbf{u}_2$

Figura 2.3: Configuración de ejemplo

Para un template de 4 celdas como la figura 2.3 y un histograma de frecuencias como se ve en la figura 2.4, el estimador $Z^*(\mathbf{u}_0)$ de una variable $Z(\mathbf{u}_0)$ dado que $Z(\mathbf{u}_1)$ =gris, $Z(\mathbf{u}_2)$ =gris y $Z(\mathbf{u}_3)$ =negro, es de la forma:

$$\begin{aligned} \mathbb{P}(Z(\mathbf{u}_0) = \text{negro} \,|Z(\mathbf{u}_1) = \text{gris} \,, Z(\mathbf{u}_2) = \text{gris} \,, Z(\mathbf{u}_3) = \text{negro}) &= \frac{8}{8+14} \\ \mathbb{P}(Z(\mathbf{u}_0) = \text{gris} \,|Z(\mathbf{u}_1) = \text{gris} \,, Z(\mathbf{u}_2) = \text{gris} \,, Z(\mathbf{u}_3) = \text{negro}) &= \frac{14}{8+14} \end{aligned}$$

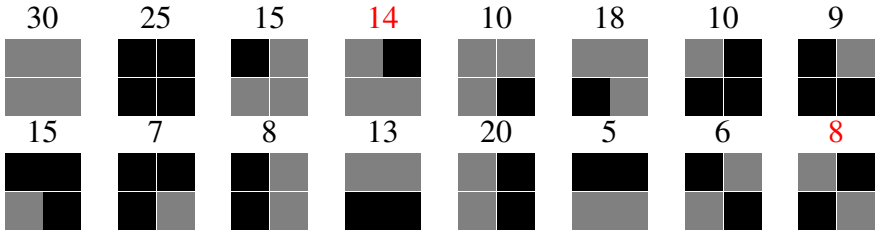


Figura 2.4: Histograma de Frecuencias de Patrones asociados al template de la figura 2.1

De esta manera, si se cuenta con el histograma de frecuencias de una imagen de entrenamiento, es posible estimar las probabilidades condicionales de una variable Z, y el objetivo será minimizar la distancia entre esas probabilidades condicionales y las que se estimarán en otra imagen (la cual representa una realización generada mediante Recocido Simulado). Si esa distancia es cero, los histogramas son idénticos y se tiene la esperanza de que las realizaciones sean parecidas a la imagen de entrenamiento o preserven algunas propiedades.

En esta sección se mencionarán los distintos enfoques abordados para el cálculo de frecuencias de manera eficiente y el enfoque final que se escogió.

2.1.1. Bases de Datos Relacionales

El primer enfoque que se abordó para realizar el cálculo del histograma de frecuencias fue el diseño de una base de datos relacional que entregara de manera simple el conteo de las frecuencias. Para ello se escogió PostgreSQL [pos] , debido a su eficiencia y a que se tenía conocimiento de

una interfaz para administrar y utilizar la base de datos desde C++.

Tempranamente se desechó este enfoque debido a la imposibilidad de realizar consultas geométricas, por ejemplo, para consultar por el número de bloques de color negro cuyo vecino ubicado en el norte tiene color gris y su vecino ubicado en el sur tiene color negro, se debia realizar un triple join de la tabla consigo misma, lo cual era demasiado ineficiente. Se intentó construir un campo donde se almacenaran los id's de los vecinos, para tener una referencia directa, sin embargo, esto limitaba el tamaño del template que se podía utilizar, pues si se aumentaba demasiado el tamaño del template había que recalcular toda esa columna en la tabla, lo cual era demasiado costoso y altamente no mantenible.

2.1.2. Bases de Datos de Objetos

El segundo enfoque que se abordó fue el diseño de una base de datos de objetos, utilizando DB4O [db4], un motor de base de datos para Java o .NET que mapea los objetos definidos en el código directamente en la base de datos, sin dividir al objeto en sus atributos y relaciones. La base de datos se pensó utilizar para almacenar la grilla 2D compuesta por muchos objetos (bloques o celdas) con pocos atributos (posición y dato) y luego realizar consultas acerca de la existencia de patrones potencialmente más complejos e irregulares que los paralelepípedos que se estaban utilizando.

Se decidió no utilizar este enfoque pues si bien el cálculo de frecuencias para patrones irregulares se realizaba de manera exitosa, el tiempo de cálculo no era el esperado y además no era posible integrar DB4O con una plataforma de cálculo paralelo, lo cual era una desventaja muy fuerte frente a otros enfoques.

2.1.3. Conteo simple

Finalmente, se optó por realizar un conteo simple recorriendo la grilla de manera adecuada. Primero, no era necesario recorrer todos los puntos de la grilla, pues bastaba con recorrer aquellos que permitían *mover* el template tomando como soporte fijo a un punto. En la figura 2.5 se puede observar el área que se puede procesar usando un punto como soporte para el template de $2 \times 2 \times 1$ y $3 \times 3 \times 1$ celdas. El algoritmo 1 resume los pasos de la operación.

Esta manera de contar los patrones, además de ser bastante simple, permite dividir la grilla y repartir trabajo a diversos procesos, en caso de ser necesario cálculo paralelo para el conteo. Esta última funcionalidad no se implementó pero en el capítulo Trabajo a Futuro se comenta la manera de incorporarla.

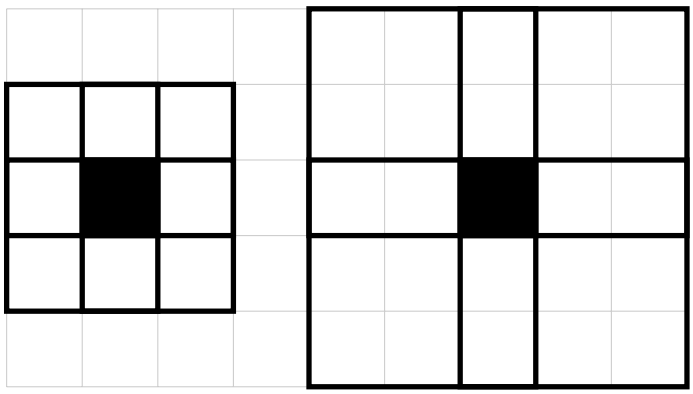


Figura 2.5: Conteo Simple utilizando un punto como soporte, para templates de $2 \times 2 \times 1$ y $3 \times 3 \times 1$ celdas.

Algoritmo 1 Conteo Simple

1: **Para** $i = 1$ **hasta** $imagen.size.x$ **hacer**
2: **Para** $j = 1$ **hasta** $imagen.size.y$ **hacer**
3: **Para** $k = 1$ **hasta** $imagen.size.z$ **hacer**
4: **Si** $i = 0 \,(\text{mod}\, template.size.x)$ && $j = 0 \,(\text{mod}\, template.size.y)$ && $k = 0 \,(\text{mod}\, template.size.z)$ **entonces**
5: Contar patrones asociados a un template T usando como soporte el punto (i, j, k).
6: **Fin (Si)**
7: **Fin (Para)**
8: **Fin (Para)**
9: **Fin (Para)**

2.2. Recocido Simulado

El Recocido Simulado o *Simulated Annealing* ([KGV83]) es una técnica de optimización basada en un fenómeno de la Termodinámica, en el cuál se calienta y enfría controladamente un material para incrementar el tamaño de sus cristales a nivel molecular, y de esa manera disminuir sus defectos. El calor provoca que los átomos se muevan desde sus posiciones iniciales (un mínimo local de la energía interna) y caigan aleatoriamente en estados de mayor energía; el enfriamiento lento les da más oportunidades de encontrar configuraciones con menor energía interna que la inicial.

2.2.1. Algoritmo

Si se desea minimizar una función objetivo O, donde $O = O_i$ es el valor de función objetivo para la configuración inicial i y $O = O_j$ es el valor de función objetivo para una nueva configuración j, que se construye perturbando la configuración i, el Recocido Simulado nos dice que la

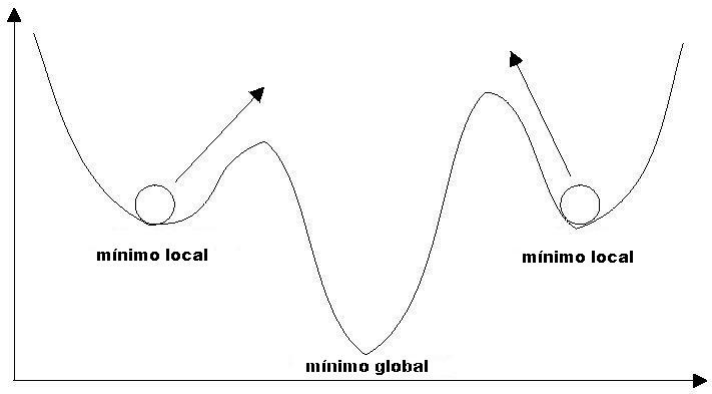


Figura 2.6: Idea central del Recocido Simulado: Permitir el salto desde óptimos locales a nuevos estados

probabilidad de aceptar la perturbación estará dada por:

$$\mathbb{P}(\text{aceptar configuración } j \text{ a partir de } i) = \begin{cases} 1 & O_j \leq O_i \\ e^{(O_i - O_j)/T} & O_j > O_i \end{cases}$$

es decir, todas las configuraciones favorables ($O_j \leq O_i$) son aceptadas con probabilidad 1 y además son aceptados cambios desfavorables ($O_j > O_i$) con probabilidad $\exp((O_i - O_j)/T)$. La función $\exp((O_1 - O_2)/T)$ proviene de la distribución de probabilidad de Boltzmann $\mathbb{P} \sim \exp(-E/(k_b T))$, utilizada en Termodinámica. Como se debe realizar el enfriamiento controlado de la temperatura, ésta se debe actualizar (de manera decreciente) en cada paso del algoritmo siguiendo alguna regla de decisión, por ejemplo, $t_k \leftarrow f(t_k)$. El detalle se puede ver en el algoritmo 2.

Algoritmo 2 Recocido Simulado Secuencial

$k \leftarrow 0$
Repetir
  Repetir
    Perturbar(configuración$_i$ $\rightarrow$ configuración$_j$, ΔO_{ij})
    Si: $\Delta O_{ij} \leq 0$ **entonces**
      Aceptar perturbación a configuración$_j$
      Update(configuración$_j$)
    De lo contrario, Si: $\exp(-\Delta O_{ij}/t_k) > \text{random}[0,1)$ **entonces**
      Aceptar perturbación a configuración$_j$
      Update(configuración$_j$)
    Fin (Si)
  Hasta que: Se alcance un **equilibrio**
  $t_{k+1} \leftarrow f(t_k)$
  $k \leftarrow k + 1$
Hasta que: Se cumpla criterio de parada

Los puntos más complejos en este esquema son:

- **Update de la función objetivo**: **Update**(configuración$_j$)

  El update de la función objetivo se debe realizar en base a una función objetivo definida a priori la cual debe contener las características más relevantes del modelo, y ser consistente para cada característica que se incluya. Al realizar el update, sólo se deben actualizar las partes del modelo que han sido perturbadas, de esa manera se minimiza la utilización de memoria para recalcular el valor.

- **Perturbar la realización**: **Perturbar**(configuración$_i \rightarrow$ configuración$_j, \Delta O_{ij}$)

  La acción de perturbar la realización implica definir una regla o conjunto de reglas que se deben aplicar cada vez que se quiere obtener un nuevo estado, para realizar la comparación entre el valor previo y posterior a la perturbación. El tipo de perturbación que se escoja influirá de manera importante en la convergencia del método, junto al update de la regla de decisión.

- **Update de la regla de decisión**: $t_{k+1} \leftarrow f(t_k)$

  El update de la regla de decisión consiste en actualizar la temperatura t en la probabilidad de aceptación. Para ello, se debe diseñar un ***annealing schedule***, en el cual se especifica de que manera se debe disminuir la temperatura, conservando la convergencia sin tener riesgos de quedar atrapado en mínimos locales. El ***annealing schedule*** puede definirse de la forma más conveniente según sea el contexto del problema.

  La manera de encontrar el ***annealing schedule*** óptimo es por ensayo y error. Se prueban diversos métodos y parámetros y en base a datos de rendimiento se escoge la mejor combinación. Cabe destacar que el cálculo del esquema óptimo está fuera del alcance del trabajo de título, por lo cual se utilizará uno evaluado junto al profesor guía.

2.2.2. Convergencia

La pregunta natural que surge es si este método converge a una solución *aceptable* o no. En [vLA87] se realiza un análisis teórico exhaustivo acerca del comportamiento asintótico del algoritmo en base a Cadenas de Markov, lo cual puede dar algunas luces sobre su comportamiento en general. En esta subsección se revisarán los resultados más importantes sobre la convergencia del método.

Para fijar conceptos, conviene recordar algunas definiciones y propiedades de la Teoría de Probabilidades y los Procesos Estocásticos [Mar05]:

Definición 2.2.1 (Proceso Estocástico)**.** *Dado un espacio de probabilidad* $(\Omega, \mathcal{F}, \mathbb{P})$*, un Proceso Estocástico con espacio de estados* I *numerable es una colección de variables aleatorias con valores en* I *indexadas por un conjunto* T *"tiempo"(*$T = \{t : t \geq 0\}$ *ó* $T = \mathbb{N}$*). Esto es, un proceso*

estocástico Y es una familia $\{Y(t) : t \in T\}$ donde cada $Y(t)$ es una variable aleatoria con valores en I.

Definición 2.2.2 (Cadena de Markov)**.** *Un Proceso Estocástico se llama Cadena de Markov si se cumple la siguiente propiedad (llamada Propiedad de Markov):*

$$\mathbb{P}(Y(t) = i_t | Y(s_n) = i_{s_n}, \ldots, Y(s_0) = i_{s_0}) = \mathbb{P}(Y(t) = i_t | Y(s_n) = i_{s_n}),$$

$$0 \leq s_0 < \ldots < s_n < t \in T, i_{s_0}, \ldots, i_{s_n}, i_t \in I$$

Definición 2.2.3 (Matriz de Transición)**.** *La matriz de Transición P asociada al proceso estocástico Y se define de la siguiente forma:*

$$P_{ij}(t) = \mathbb{P}(Y(t) = i | Y(0) = j), 0 \leq t \in T$$

Si $T = \mathbb{N}$, $P_{ij}(t)$ representa la probabilidad de transición desde la configuración i a la j en t pasos. La matriz se dice homogénea si

$$\mathbb{P}(Y(t) = i | Y(s) = j) = \mathbb{P}(Y(t-s) = i | Y(0) = j), \forall 0 \leq s < t \in T, i, j \in I$$

es decir, no depende del "tiempo"(o el número de transiciones que han transcurrido). En caso contrario, se dice no-homogénea.

De las definiciones anteriores se deduce que una Cadena de Markov permite modelar el comportamiento de un proceso estocástico sin memoria, pues lo que ocurra en una configuración sólo depende del estado inmediatamente anterior y no de la totalidad de los estados anteriores.

Con esto, la dinámica del Recocido Simulado consta de los siguientes elementos:

- $I = \{i : i = 1, ..., N\}$: conjunto de configuraciones posibles.
- $X(k) \in I$: configuración en la iteración k-ésima
- $O(i)$: valor objetivo en la configuración i
- $G_{ij}(t_k)$: probabilidad de generar la configuración j, desde la configuración i, con temperatura t_k. Se adopta usualmente la siguiente caracterización:

$$G_{ij}(t_k) = G_{ij} = \begin{cases} \frac{1}{|\mathcal{R}_i|} & j \in \mathcal{R}_i \\ 0 & j \notin \mathcal{R}_i \end{cases} \tag{2.2}$$

  donde $\mathcal{R}_i$ es el conjunto de configuraciones alcanzables desde i en un paso.

- $A_{ij}(t_k)$: probabilidad de aceptar la configuración j, desde la configuración i, con temperatura t_k. Se adopta usualmente la siguiente caracterización:

$$A_{ij}(t_k) = \begin{cases} \exp\left(-\frac{O(j)-O(i)}{t_k}\right) & O(j) > O(i) \\ 1 & O(j) \leq O(i) \end{cases} \tag{2.3}$$

Utilizando lo anterior se contruye la matriz de transición de la cadena de Markov no-homogénea (depende de la temperatura t_k en cada transición) asociada:

$$P_{ij}(t_k) = \begin{cases} G_{ij}(t_k)A_{ij}(t_k) & \forall j \neq i \\ 1 - \sum_{s=1, s\neq i}^{N} G_{is}(t_k)A_{is}(t_k) & j = i \end{cases} \tag{2.4}$$

El objetivo de esta contrucción es probar que el Recocido Simulado converge asintóticamente a un elemento del conjunto de soluciones óptimas, es decir,

$$\lim_{k\to\infty} \mathbb{P}(X(k) \in \mathcal{R}_{opt}) = 1$$

Para ello, se deben imponer condiciones sobre la matriz de transición y la temperatura. Lo anterior se resume en el siguiente teorema, desarrollado en [Hay88]:

Teorema 2.2.1. *Si la matriz de transición asociada al proceso estocástico X que representa al Recocido Simulado es de la forma 2.4, con $A_{ij}(t_k)$ y $G_{ij}(t_k)$ de la forma 2.3 y 2.2 respectivamente, y además:*

- $\forall t_k \in T, t_k > 0, \forall i, j \in I, \exists p \geq 1, \exists \lambda_0, \lambda_1, \ldots, \lambda_p \in I, \lambda_0 = i, \lambda_p = j$ *tal que* $G_{\lambda_r \lambda_{r+1}}(t_k) > 0, r = 0, 1, \ldots, p-1$ *(para cualquier temperatura y cualquier par de configuraciones, existe una secuencia de perturbaciones que lleva una configuración en otra).*
- $\forall H \geq 0, \forall i, j \in I$, *$i$ es alcanzable en a lo más una altura H desde j ($\exists p \geq 1, \lambda_0, \ldots, \lambda_p \in I, \lambda_0 = i, \lambda_p = j : G_{\lambda_r \lambda_{r+1}}(t_k) > 0, r = 0, \ldots, p-1$ y $O(\lambda_r) \leq H, r = 0, \ldots, p$) sí y sólo sí j es alcanzable en a lo más una altura H desde i.*
- *La secuencia de temperaturas $\{t_k\}_{k\in\mathbb{N}}$ satisface:*
  - $\lim_{k\to\infty} t_k = 0$
  - $t_k \geq t_{k+1}$ *para todo k*

Entonces

$$\lim_{k\to\infty} \mathbb{P}(X(k) \in \mathcal{R}_{opt}) = 1$$

sí y sólo sí

$$\sum_{k=1}^{\infty} \exp\left(-\frac{D}{t_k}\right) = \infty$$

con D una constante que depende de la estructura del sistema.

Como corolario del teorema anterior, útil para chequear la manera a la que se debe *enfriar* el sistema, se tiene el siguiente resultado:

Corolario 2.2.1 (Test de Convergencia)**.** *Si consideramos t_k de la forma:*

$$\forall k : \; t_k \geq \frac{\Gamma}{\log k}$$

con Γ *constante, el teorema de convergencia implica que*

$$\lim_{k\to\infty} \mathbb{P}(X(k) \in \mathcal{R}_{opt}) = 1$$

se cumple sí y sólo sí

$$\Gamma \geq D$$

La manera de encontrar Γ puede resultar tan difícil como resolver el problema mismo, por lo cual se debe confiar en la intuición y la experiencia o simplemente realizar pruebas y elegir la constante que entregue los resultados más satisfactorios.

El teorema 2.2.1 indica la existencia de una secuencia de tiempo que asegura la convergencia asintótica de la cadena de Markov relacionada con el Recocido Simulado. Si bien esta secuencia existe, el resultado es poco práctico pues como no hay unicidad, experimentalmente se pueden encontrar secuencias que teóricamente aseguran convergencia pero la velocidad es demasiado lenta.

2.3. Programación Paralela

2.3.1. Ideas básicas

Inicialmente puede parecer simple el concepto de *Paralelismo* en Computación. A grandes rasgos, se puede decir que consiste en organizar varios procesos de tal manera que realicen una tarea, cuyo tiempo de ejecución se espera sea menor que si la realizara sólo un proceso. Sin embargo, la situación no es tan sencilla. No debemos olvidar que los procesos son *ignorantes*, es decir, no saben mucho sobre el mundo que los rodea, ni quienes son sus compañeros de trabajo, ni cual es la mejor forma de manejar los datos que se le entregan, etc.

Debido a esto, la implementación en paralelo de una aplicación requiere de 3 grandes tareas:

- Desarrollar algoritmos y estructuras de datos para resolver el problema.
- Dividir el problema en subproblemas o resolver varias instancias del problema en una sola.
- Identificar las comunicaciones que se deben realizar para que los procesos trabajen organizados.

Teniendo estos puntos bien definidos, es posible llegar a un resultado exitoso. En la siguiente subsección se mencionan las principales características del enfoque abordado para paralelizar el Recocido Simulado.

2.3.2. Paralelización del Recocido Simulado

El principal problema que presenta el Recocido Simulado es el tiempo de cálculo que necesita para obtener una solución aceptable. Mientras más iteraciones se realizen, mejor solución se puede obtener, sin embargo, a mayor número de iteraciones, mayor tiempo de cálculo. Es por esto que se estudió la implementación del algoritmo usando multiples procesadores.

En la literatura se mencionan diversos enfoques para realizar una paralelización eficiente del Recocido Simulado. En [RRD90] se describen dos enfoques:

- **Paralelización de la evaluación de cada transición**: Este enfoque intenta reducir el tiempo de cálculo en cada etapa interna del algoritmo (perturbación de la configuración, update de configuración, update de la regla de decisión y cálculo de valor objetivo) en cada paso de la cadena de Markov generada. Como cada paso de la cadena se calcula independientemente, se pueden destinar todos los procesos a la resolución secuencial de las etapas internas.
- **Paralelización de la exploración de la cadena de Markov**: Este enfoque intenta reducir el tiempo de cálculo adelantando trabajo a realizar en futuros pasos de la cadena. Cada paso de la cadena es realizado por un solo proceso y como solamente hay 2 posibles pasos futuros (aceptar la perturbación o rechazarla), se pueden utilizar 2 procesos para evaluar esos pasos por adelantado. La manera en la que se puede explorar la cadena depende de la naturaleza del problema y del comportamiento a medida que disminuye la temperatura. Por ejemplo, se puede escoger explorar sólo los pasos asociados a aceptaciones.

Se decidió utilizar una técnica perteneciente al segundo enfoque, llamada *Computación Especulativa*, descrita en [Bur85], la cual consiste en construir un árbol binario balanceado de decisiones, donde cada nodo representa un paso de la cadena de Markov asociada al problema, con su respectiva configuración actualizada dependiente del paso anterior (nodo padre en el árbol).

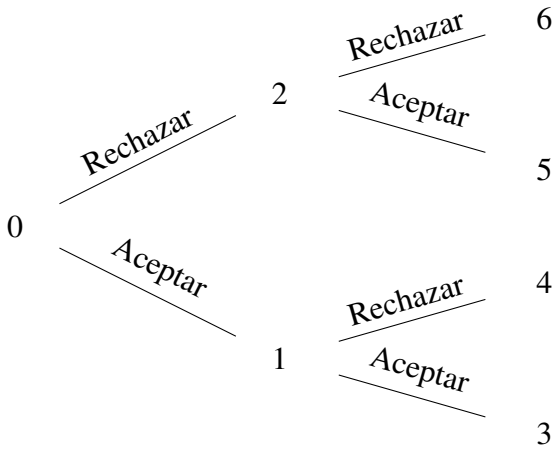


Figura 2.7: Topología de Árbol Binario con 7 procesos

El algoritmo 3 detalla el esquema de la aplicación de la Computación Especulativa al Recocido Simulado.

Algoritmo 3 Recocido Simulado Paralelo

1: $k \leftarrow 0$
2: **Repetir**
3: **Repetir**
4: **Para cada** Proceso P **hacer**
5: Actualizar la configuración actual en P, en base a la configuración actual de $padre(P)$, sin aceptar la última perturbación realizada por el padre.
6: **Si:** P es un proceso aceptador **entonces**
7: **Perturbar**(configuración$_{i(padre(P))} \rightarrow$ configuración$_{j(padre(p))}, \Delta O_{i(padre(p))j(padre(p))}$)
8: Aceptar perturbación a configuración$_{j(padre(P))}$
9: **Update**(configuración$_{j(padre(p))}$)
10: **Fin (Si)**
11: **Perturbar**(configuración$_i \rightarrow$ configuración$_j, \Delta O_{ij}$)
12: **Si:** $\Delta O_{ij} \leq 0$ **entonces**
13: Aceptar perturbación a configuración$_j$
14: **Update**(configuración$_j$)
15: **De lo contrario, Si:** $\exp(-\Delta O_{ij}/t_k) >$ random$[0,1)$ **entonces**
16: Aceptar perturbación a configuración$_j$
17: **Update**(configuración$_j$)
18: **Fin (Si)**
19: **Fin (Para)**
20: **Hasta que:** Se alcance un **equilibrio**
21: $t_{k+1} \leftarrow f(t_k)$
22: $k \leftarrow k+1$
23: **Hasta que:** Se cumpla criterio de parada

2.4. Speedup

Primero, conviene mencionar una definición del speedup, extraída de [Pac96]:

Definición 2.4.1 (Speedup)**.** *El speedup de una aplicación se define como la razón entre el tiempo secuencial que ésta demora y el tiempo en paralelo, utilizando múltiples procesos. Más precisamente, si T_σ es el tiempo secuencial y $r \in [0,1]$ corresponde a la fracción de la aplicación que se puede paralelizar de manera óptima, con lo cual el tiempo en paralelo utilizando P procesos es $T_\pi = (1-r)T_\sigma + \frac{rT_\sigma}{P}$, entonces el speedup utilizando P procesos es*

$$S = \frac{T_\sigma}{T_\pi} \tag{2.5}$$

$$= \frac{T_\sigma}{(1-r)T_\sigma + \frac{rT_\sigma}{P}} \tag{2.6}$$

Estudiar el speedup teórico de una aplicación tiene una importancia crucial debido a que existen aplicaciones que no son susceptibles a ser paralelizadas o que tienen una cota superior en términos del número de procesos que la pueden abordar, y poder detectar a tiempo una aplicación así ayuda a enfocar los esfuerzos en otras alternativas o no destinar más recursos a la paralelización.

2.4.1. Ley de Amdahl

La Ley de Amdahl, según [Pac96], entrega una cota superior para el speedup de una aplicación. Si el speedup (visto como función del número de procesos P) está dado por

$$S(P) = \frac{T_\sigma}{(1-r)T_\sigma + \frac{rT_\sigma}{P}} \tag{2.7}$$

con $r \in [0,1]$, eliminando T_σ queda

$$S(P) = \frac{1}{(1-r) + \frac{r}{P}} \tag{2.8}$$

Derivando con respecto a P se obtiene

$$\frac{dS(P)}{dP} = \frac{r}{((1-r)P + r)^2} \tag{2.9}$$

La fórmula anterior es positiva para todo P, luego $S(P)$ es una función creciente cuyo límite cuando P tiende a ∞ es

$$S(P) \rightarrow \frac{1}{1-r} \tag{2.10}$$

Debido a esto, existe una cota superior para el speedup que se puede obtener, incluso utilizando cientos de miles de procesos. Por ello es importante calcular una cota apriori del speedup para ver si conviene invertir en la incorporación de más procesos en el cálculo de la aplicación.

2.4.2. Caso Árbol Balanceado

En esta subsección se analizará en base a [WCF91] el speedup teórico de la incorporación de computación especulativa al Recocido Simulado.

Para fijar ideas, definamos las siguientes variables:

- τ: temperatura de enfriamiento.
- N: número de temperaturas realizadas.
- I: número de iteraciones realizadas por temperatura.

- $T_\sigma(\tau)$: tiempo de ejecución secuencial con temperatura τ.
- $T_\pi(\tau)$: tiempo de ejecución en paralelo con temperatura τ.
- $S(\tau)$: speedup con temperatura τ.

Primero, se considera que el tiempo de ejecución secuencial se divide en 3 partes (independientes de la temperatura): t_m, tiempo de ejecución que demora la generación de una nueva configuración, t_e, tiempo de ejecución que demora la evaluación de la nueva configuración y t_d el tiempo que demora la ejecución de la decisión. Es decir

$$T_\sigma(\tau) = t_m + t_e + t_d \tag{2.11}$$

El speedup para una temperatura τ se escribe

$$S(\tau) = \frac{T_\sigma(\tau)}{T_\pi(\tau)} \tag{2.12}$$

El speedup para el algoritmo considerando todas las temperaturas involucradas es

$$S = \frac{\sum_\tau T_\sigma(\tau)}{\sum_\tau T_\pi(\tau)} \tag{2.13}$$

Utilizando 2.12 en 2.13 y recordando que N es el número de temperaturas, se tiene

$$\begin{aligned} S &= \frac{\sum_\tau T_\sigma(\tau)}{\sum_\tau T_\pi(\tau)} && (2.14)\\ &= \frac{NI(t_m + t_e + t_d)}{\sum_\tau \frac{T_\sigma(\tau)}{S(\tau)}} && (2.15)\\ &= \frac{NI(t_m + t_e + t_d)}{\sum_\tau \frac{I(t_m+t_e+t_d)}{S(\tau)}} && (2.16)\\ &= \frac{N}{\sum_\tau \frac{1}{S(\tau)}} && (2.17) \end{aligned}$$

De esta manera, la expresión 2.17 representa el speedup del algoritmo general, basado en el speedup obtenido para cada temperatura.
Finalmente, se debe obtener una estimación para el speedup $S(\tau)$. Como se utilizará un árbol binario balanceado, si se utilizan P procesos, el número de niveles será $\log_2(P+1)$. Luego, el speedup promedio es igual a la cantidad de niveles del árbol (independiente de la temperatura)

$$\begin{aligned} S(\tau) &= \frac{T_\sigma(\tau)}{T_\sigma(\tau)/\log_2(P+1)} && (2.18)\\ &= \log_2(P+1) && (2.19) \end{aligned}$$

Usando la estimación 2.19 en 2.17, se obtiene (considerando ahora la suma $\sum_\tau$ de la forma $\sum_{\tau \leq \tau_h}$, donde τ_h es el nivel temperatura máximo)

$$\begin{aligned} S &= \frac{N}{\sum_\tau \frac{1}{S(\tau)}} && (2.20)\\ &= \frac{N}{\sum_{\tau \leq \tau_h} \frac{1}{\log_2(P+1)}} && (2.21) \end{aligned}$$

Considerando $N = \frac{\tau_h}{\Delta\tau}$,

$$S = \frac{\tau_h}{\sum_{\tau \leq \tau_h} \frac{\Delta\tau}{\log_2(P+1)}} \tag{2.22}$$

Si $\Delta\tau$ es suficientemente chico, se puede aproximar la suma por una integral, luego

$$S = \frac{\tau_h}{\int_0^{\tau_h} \frac{d\tau}{\log_2(P+1)}} \tag{2.23}$$

$$S = \frac{\tau_h}{\frac{\tau_h}{\log_2(P+1)}} \tag{2.24}$$

$$S = \log_2(P+1) \tag{2.25}$$

Con esta estimación, utilizando $P = 3$ procesos se debe obtener un speedup aproximado de 2 y utilizando $P = 7$ procesos se debe obtener un speedup aproximado de 3, es decir, ***el tiempo de cálculo se debe reducir a la mitad y a un tercio con respecto al tiempo secuencial***.

2.4.3. Caso Árbol No Balanceado

En [WCF91] se analiza el caso de un árbol no balanceado. Este caso no se abordó en este trabajo de título, pero por completitud se incluyen esos resultados para el speedup en el caso general, teniendo en mente un posible trabajo a futuro o extensión del método.

Si ahora consideramos un árbol binario no balanceado, el peor speedup en promedio que se puede obtener ocurre en el caso equilibrado (pues si el árbol crece por un camino, debe disminuir por otro de manera equivalente)

$$S_{\text{peor caso}}(\tau) = \log_2(P+1)$$

y el mejor speedup posible ocurre cuando se realizan P pasos en la cadena de Markov (el árbol es completamente desbalanceado, con un solo camino desde la raíz hasta la hoja)

$$S_{\text{mejor caso}}(\tau) = P$$

En base al comportamiento del Recocido Simulado descrito en [WCF91], se puede suponer que el speedup se comporta de manera lineal entre $\log_2(P+1)$ y P dependiendo de la temperatura, es decir

$$S(\tau) = \frac{\log_2(P+1) - P}{\tau_h}\tau + P \tag{2.26}$$

donde τ_h representa el largo del intervalo donde se encuentra la temperatura.
Usando la estimación 2.26 en 2.17, se obtiene (considerando ahora la suma $\sum_\tau$ de la forma $\sum_{\tau \leq \tau_h}$)

$$S = \frac{N}{\sum_\tau \frac{1}{S(\tau)}} \tag{2.27}$$

$$= \frac{N}{\sum_{\tau \leq \tau_h} \frac{1}{\frac{\log_2(P+1)-P}{\tau_h}\tau + P}} \tag{2.28}$$

Considerando $N = \frac{\tau_h}{\Delta\tau}$,

$$S = \frac{\tau_h}{\sum_{\tau \leq \tau_h} \frac{\Delta\tau}{\frac{\log_2(P+1)-P}{\tau_h}\tau + P}} \tag{2.29}$$

Si $\Delta\tau$ es suficientemente chico, se puede aproximar la suma por una integral, luego

$$S = \frac{\tau_h}{\int_0^{\tau_h} \frac{d\tau}{\frac{\log_2(P+1)-P}{\tau_h}\tau + P}} \tag{2.30}$$

Realizando el cambio de varible $r = \frac{\log_2(P+1)-P}{\tau_h}\tau + P$, $dr = \frac{\log_2(P+1)-P}{\tau_h}d\tau$, se tiene

$$S = \frac{\tau_h}{\int_1^{\log_2(P+1)} \frac{dr}{\frac{\log_2(P+1)-P}{\tau_h}r}} \tag{2.31}$$

$$= \frac{P - \log_2(P+1)}{\ln(P) - \ln(\log_2(P+1))} \tag{2.32}$$

Por lo tanto, para P grande, el speedup se comporta de la forma

$$S = \frac{P}{\ln(P)} \tag{2.33}$$

Capítulo 3

Trabajo Realizado

El principal objetivo de este trabajo fue realizar una implementación del Recocido Simulado utilizando el enfoque entregado por la Computación Especulativa. Para ello, se escogió el lenguaje de programación `C++` y el estándar `MPI` (*Message Passing Interface*) que permite desarrollar programas portables independientes de la arquitectura donde se ejecute.

3.1. Ambiente de Trabajo

El desarrollo se realizó en las instalaciones del Laboratorio de Planificación Minera, dependiente del Departamento de Ingeniería en Minas. Se utilizó un cluster Rocks 4.3 (Mars Hill) con dos servidores, cada uno con la siguiente configuración:

- Modelo Dell PowerEdge 1950.
- Doble procesador Quad Core Xeon E5320 2x4MB Cache, 1.86 GHz, FSB 1066 MHz.
- 4GB RAM 667 MHz (4x1GB).
- Disco duro 80 GB SATA 7200 RPM.
- 2 puertos ethernet Dual Embedded Broadcom NetXtreme II 5708 Gigabit Ethernet.

Cada servidor posee 8 CPUs, conectadas por unas red de 1 Gbps. A su vez, los servidores están conectados por una red de 100Mbps, por lo cual se decidió utilizar solamente los procesadores asociados a 1 servidor, para no obtener resultados afectados por la baja velocidad de comunicación de la red entre servidores.

3.2. Herramientas utilizadas

En esta sección se describen las herramientas utilizadas para el desarrollo de la aplicación.

3.2.1. C++

Se utilizó el lenguaje de programación `C++` con el compilador `gcc 3.4.6`. Se escogió este lenguaje debido a que ya se tenía experiencia desarrollando aplicaciones con la API `MPI` en este lenguaje. Además se cuenta con estructuras de datos pertenencientes a la `Standard Template Library`, en particular la estructura `map<string, int>`, que se utilizó para almacenar las frecuencias de aparición de los patrones.

3.2.2. MPI

El estandar `MPI` [Pac96] consiste en una serie de funciones y especificaciones agrupadas en una librería, disponible para diversos lenguajes de programación, que permite la comunicación entre distintos computadores. Sus ventajas son su alto rendimiento, su escalabilidad y su portabilidad. Incluye paso de mensajes punto a punto o de manera colectiva. Tambien permite diseñar topologías de comunicación entre procesos y grupos de procesos afines. Se escogió ésta librería debido a que ya se tenía experiencia desarrollando aplicaciones junto a `C++` (se dictó un curso electivo llamado *Cálculo de Alto Desempeño* en el Departamento de Ingeniería Matemática el Semestre de Otoño del 2007). Otra razón por la que se escogió esta librería fue la existencia de *Open MPI* [ope] en el cluster del Laboratorio. Como no existía una instalación adecuada para otra alternativa (*PVM* [PVM] o *BSPlib* [HMS+98]), se tomó la decición de utilizar ésta herramienta, pues lo importante era verificar que la técnica de paralelización efectivamente funcionara.

Las principales instrucciones utilizadas en la implementación fueron las siguientes:

- `MPI_Send`:

  Permite enviar un mensaje ubicado en la dirección `buf` de tamaño `count` y de tipo `datatype`, al proceso `dest` a través del comunicador `comm` con la etiqueta `tag`.

```
int MPI_Send( void *buf,
              int count,
              MPI_Datatype datatype,
              int dest,
              int tag,
              MPI_Comm comm );
```

- `MPI_Recv`:

  Permite recibir un mensaje en la dirección `buf` de tamaño `count` y de tipo `datatype` enviado por el proceso `source` a través del comunicador `comm` con la etiqueta `tag` guardando en `status` el estado de la recepción.

```
int MPI_Recv( void *buf,
              int count,
              MPI_Datatype datatype,
              int source,
              int tag,
              MPI_Comm comm,
              MPI_Status *status );
```

- `MPI_Bcast`:

  Permite enviar un mensaje ubicado en la dirección `buffer` de tamaño `count` y tipo `datatype`, desde el proceso `root` hacia el resto de los procesos a través del comunicador `comm`.

```
int MPI_Bcast( void *buffer,
               int count,
               MPI_Datatype datatype,
               int root,
               MPI_Comm comm );
```

- `MPI_Barrier`:

  Bloquea a los procesos a través del comunicador `comm` hasta que todos hayan llegado al punto donde se realiza la llamada a la instrucción.

```
int MPI_Barrier( MPI_Comm comm );
```

3.2.3. GSLIB

`GSLIB` [DJ92] es el acrónimo de *Geostatistical Software LIBrary*. Este nombre fue originalmente usado para definir una colección de programas geoestadísticos desarrollados en lenguaje `Fortran`, en la Universidad de Standord, E.E.U.U., durante los últimos 15 años. La principal rutina que se utilizó se llama `pixelplt` que permite graficar imágenes 2D ingresando la grilla de puntos con su respectivo valor (usualmente 0 o 1) y un archivo con parámetros para la especificación de las dimensiones de la grilla y los colores del gráfico.

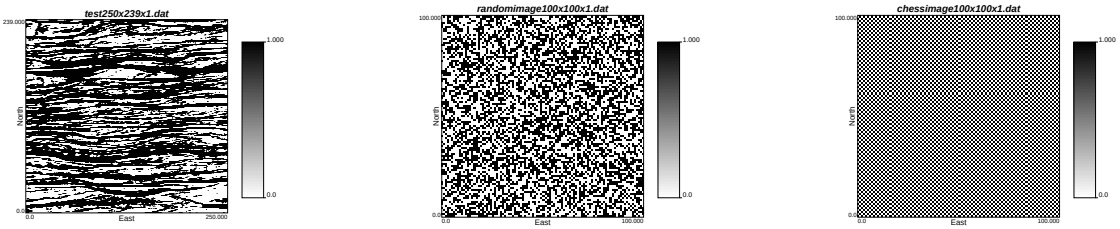


Figura 3.1: Figuras generadas con GSLIB

3.3. Implementación

El detalle de la implementación se puede ver en el Apéndice. En esta sección se revisarán dos utilidades importantes que permiten modificar la aplicación. También se verá el esquema utilizado para el paso de mensajes entre procesos, pues es el esqueleto de la implementación y el ámbito donde se pueden realizar optimizaciones críticas que pueden influir de manera directa en la reducción de los tiempos de cálculo.

3.3.1. Implementación de distintas funciones objetivo

La especificación de la función objetivo se encuentra implementada en evaluate_realization. Actualmente se encuentra implementada la siguiente función:

$$O = \sum_{i \in \mathcal{P}} \frac{\frac{1}{f_i^{TI}+1}}{\sum_{j \in \mathcal{P}} \frac{1}{f_j^{TI}+1}} (f_i^{TI} - f_i^{RE})^2 \tag{3.1}$$

$$= \underbrace{\sum_{i \in \mathcal{P}, f_i^{TI} \neq 0} \frac{\frac{1}{f_i^{TI}+1}}{\sum_{j \in \mathcal{P}} \frac{1}{f_j^{TI}+1}} (f_i^{TI} - f_i^{RE})^2}_{A} + \underbrace{\sum_{i \in \mathcal{P}, f_i^{TI} = 0} \frac{1}{\sum_{j \in \mathcal{P}} \frac{1}{f_j^{TI}+1}} (f_i^{RE})^2}_{B} \tag{3.2}$$

Los términos f^{TI} y f^{RE} representan a los vectores de frecuencias de aparición de los patrones en la imagen de entrenamiento y la realización respectivamente. El código de la función se puede

Código 3.1: Código de función `evaluate_realization` para ecuación 3.2

```
1  double evaluate_realization(  map<string,int>& hash,
2                                map<string,int>& hashrealization,
3                                float prom){
4    float total=0.0;
5    float totalInv=0.0;
6    map<string,int>::const_iterator iterHash;
7    for(  iterHash=hash.begin();
8          iterHash!=hash.end();
9          ++iterHash)
10         {
11     if (hashrealization.count(iterHash->first)>0) {
12       total=total+
13             ((1.0/(iterHash->second+1.0))/prom)*
14             pow(
15               iterHash->second-hashrealization[iterHash->first]
16               ,2
17             );
18     }
19     else{
20       total=total+
21             ((1.0/(iterHash->second+1.0))/prom)*
22             pow(
23               iterHash->second-0
24               ,2
25             );
26     }
27   }
28   map<string,int>::const_iterator iterHashRe2;
29   for(  iterHashRe2=hashrealization.begin();
30         iterHashRe2!=hashrealization.end();
31         ++iterHashRe2)
32         {
33     if (hash.count(iterHashRe2->first)==0) {
34       total=total+
35             (1.0/prom)*
36             pow(
37               0 - iterHashRe2->second
38               ,2
39             );
40     }
41   }
42   return (double)total;
43 }
```

observar en el cuadro 3.1. El término $\sum_{j\in\mathcal{P},f_j^{TI}\neq 0}\frac{1}{f_j^{TI}}$ se almacena en `prom`. En las líneas 7 a 27 se calcula el término A. En las líneas 29 a 41 se calcula el término B.

Si la función objetivo se cambia por la siguiente función (sin *pesos*):

$$O = \sum_{i\in\mathcal{P}}(f_i^{TI}-f_i^{RE})^2 \tag{3.3}$$

$$= \sum_{i\in\mathcal{P},f_i^{TI}\neq 0}(f_i^{TI}-f_i^{RE})^2+\sum_{i\in\mathcal{P},f_i^{TI}=0}(f_i^{RE})^2 \tag{3.4}$$

El código asociado se puede observar en el cuadro 3.2.

Código 3.2: Código de función `evaluate_realization` para ecuación 3.4

```
double evaluate_realization(  map<string,int>& hash,
                              map<string,int>& hashrealization,
                              float prom){
  float total=0.0;
  float totalInv=0.0;
  map<string,int>::const_iterator iterHash;
  for(  iterHash=hash.begin();
        iterHash!=hash.end();
        ++iterHash)
        {
    if (hashrealization.count(iterHash->first)>0) {
      total=total+pow(iterHash->second-hashrealization[iterHash->first],2);
    }
    else{
      total=total+pow(iterHash->second-0,2);
    }
  }
  map<string,int>::const_iterator iterHashRe2;
  for(  iterHashRe2=hashrealization.begin();
        iterHashRe2!=hashrealization.end();
        ++iterHashRe2)
        {
    if (hash.count(iterHashRe2->first)==0) {
      total=total+pow(0 - iterHashRe2->second,2);
    }
  }
  return (double)total;
}
```

Otra posible función objetivo utiliza el módulo en vez de la potencia al cuadrado:

$$O = \sum_{i \in \mathcal{P}} |f_i^{TI} - f_i^{RE}| \tag{3.5}$$

$$= \sum_{i \in \mathcal{P}, f_i^{TI} \neq 0} |f_i^{TI} - f_i^{RE}| + \sum_{i \in \mathcal{P}, f_i^{TI} = 0} |f_i^{RE}| \tag{3.6}$$

El código asociado se puede observar en el cuadro 3.3.

Código 3.3: Código de función `evaluate_realization` para ecuación 3.6

```
1  double evaluate_realization(  map<string,int>& hash,
2                                map<string,int>& hashrealization,
3                                float prom){
4    float total=0.0;
5    float totalInv=0.0;
6    map<string,int>::const_iterator iterHash;
7    for(  iterHash=hash.begin();
8          iterHash!=hash.end();
9          ++iterHash)
10         {
11     if (hashrealization.count(iterHash->first)>0) {
12       total=total+abs(iterHash->second-hashrealization[iterHash->first]);
13     }
14     else{
15       total=total+abs(iterHash->second-0);
16     }
17   }
18   map<string,int>::const_iterator iterHashRe2;
19   for(  iterHashRe2=hashrealization.begin();
20         iterHashRe2!=hashrealization.end();
21         ++iterHashRe2)
22         {
23     if (hash.count(iterHashRe2->first)==0) {
24       total=total+abs(0 - iterHashRe2->second);
25     }
26   }
27   return (double)total;
28 }
```

3.3.2. Implementación de la perturbación a una configuración

La especificación de la función que perturba una configuración se encuentra implementada en perturb_realization.

En el código 3.4 se observa que en las líneas 20 a la 38 se remueve del histograma asociado a la realización todos los patrones que tocan al bloque de coordenadas `coord`. Posteriormente se cambia el valor del dato asociado al bloque en las líneas 39 a 50. Finalmente en las líneas 51 a 67 se actualiza el histograma con los nuevos patrones que aparecen al cambiar el valor del dato en el bloque.

La estructura de datos `map<string,int>` realiza las operaciones `erase` e `insert`, las cuales tienen complejidad logarítmica en el tamaño de la llave. Se deja como trabajo a futuro investigar diferentes estructuras de datos con complejidades diferentes a las de `map<string,int>`, que puedan mejorar el tiempo de cálculo.

Código 3.4: Código de función `perturb_realization`

```
1  void perturb_realization( geometry *tem, geometry* reservoir,
2                            int *coord, map<string,int>& hashrealization,
3                            int my_id, int iterout){
4    char dataaux;
5    string procname;
6    if(my_id==0){
7      procname="MASTER";
8    }
9    else{
10     procname="SLAVE";
11   }
12   if(reservoir->node[coord[0]][coord[1]][coord[2]].data==WHITE){
13     dataaux='W';
14   }
15   else{
16     dataaux='B';
17   }
18   int i,j,k;
19   string code="";
20   for(i=0;i<tem->lengthx;i++){
21     for(j=0;j<tem->lengthy;j++){
22       for(k=0;k<tem->lengthz;k++){
23         reset_template(tem);
24         code=evaluate_template_ret(i,j,k,coord,0,tem,reservoir, hashrealization);
25         if(code.compare("OUT")!=0){
26           map<string,int>::iterator iter = hashrealization.find(code);
27           if (iter!=hashrealization.end()) {
28             if(hashrealization[code]>0){
29               hashrealization[code]--;
30             }
31             if(hashrealization[code]==0){
32               hashrealization.erase(iter);
33             }
34           }
35         }
36       }
37     }
38   }
39   if(reservoir->node[coord[0]][coord[1]][coord[2]].data==WHITE){
40     reservoir->node[coord[0]][coord[1]][coord[2]].data=BLACK;
41   }
42   else{
43     reservoir->node[coord[0]][coord[1]][coord[2]].data=WHITE;
44   }
45   if(reservoir->node[coord[0]][coord[1]][coord[2]].data==WHITE){
46     dataaux='W';
47   }
48   else{
49     dataaux='B';
50   }
51   for(i=0;i<tem->lengthx;i++){
52     for(j=0;j<tem->lengthy;j++){
53       for(k=0;k<tem->lengthz;k++){
54         reset_template(tem);
55         code=evaluate_template_ret(i,j,k,coord,0,tem,reservoir, hashrealization);
56         if(code.compare("OUT")!=0){
57           map<string,int>::iterator iter = hashrealization.find(code);
58           if (iter!=hashrealization.end()) {
59             hashrealization[code]++;
60           }
61           else{
62             hashrealization.insert(make_pair(code,1));
63           }
64         }
65       }
66     }
67   }
68 }
```

3.3.3. Implementación del *annealing schedule*

El *annealing schedule* se encuentra implementado en la función schedule_update.

El código se puede observar en el cuadro 3.5. En la línea 21 se verifica que el reporte se realiza solamente en las iteraciones que son múltiplos de `irepo`. En la línea 33 se chequea si la función objetivo actual es mayor que la mejor función objetivo obtenida en alguna configuración anterior. En caso de ser mayor, se incrementa la variable `*attempt`, y cuando esta alcanza un valor `maxatt`, se realizan las actualizaciones de las líneas 55 a 61. En caso contrario, cuando la función objetivo actual es menor o igual a la mejor función objetivo obtenida hasta el momento, se realizan las actualizaciones de las líneas 37 a 53. De la línea 41 a la 51, se efectúa el chequeo de repeticiones.

Es importante recordar que el *annealing schedule* depende en gran medida de los parámetros `T`, `lambda`, `tred` y `maxrepetitions`, los cuales se ingresan como argumentos al programa principal main. La manera de estimar que parámetros entregan la mejor convergencia es utilizando ensayo y error, por lo cual su determinación queda fuera de los alcances de este trabajo.

Código 3.5: Código de función `schedule_update`

```
1  void schedule_update( int iter,
2                        int irepo,
3                        double startobj,
4                        double* latestobj,
5                        int* attempt,
6                        int maxatt,
7                        double* T,
8                        double lambda,
9                        int* tred,
10                       int ntemp,
11                       int maxrepetitions,
12                       int *repcounter,
13                       map<string,int>& hash,
14                       map<string,int>& hashrealization,
15                       geometry *realization,
16                       char *outfile,
17                       ofstream &file,
18                       char *logfilename,
19                       float prom){
20    double obj=evaluate_realization(hash,hashrealization,prom);
21    if(((int)(iter/irepo))*irepo==iter){
22      cout  << "time: "<<time(NULL)
23            << ", iter: " << iter
24            << ", obj: " << obj
25            << ", %: " << obj/startobj*100
26            << ", latestobj: "<<*latestobj
27            << ", obj-latestobj: "<<obj-*latestobj
28            << ", attempt: " << *attempt
29            << ", tred: " << *tred
30            << ", repcounter: "<<*repcounter
31            << ", T: " << *T
32            << endl;
33      if(*latestobj<obj){
34        (*attempt)++;
35      }
36      else{
37        print_reservoir(realization,outfile);
38        open_logfile(file,logfilename);
39        print_histogram(file,hashrealization);
40        close_logfile(file);
41        if(abs((long)(*latestobj-obj))<=1.0e-7){
42          (*repcounter)++;
43          if(*repcounter>=maxrepetitions){
44            *T=(*T)*lambda;
45            (*tred)++;
46            *repcounter=0;
47          }
48        }
49        else{
50          *repcounter=0;
51        }
52        *latestobj=obj;
53        *attempt=0;
54      }
55      if(*attempt>=maxatt){
56        *T=(*T)*lambda;
57        (*tred)++;
58        cout << "T: " << *T << endl;
59        *attempt=0;
60        *repcounter=0;
61      }
62    }
63  }
```

3.3.4. Esquema de paso de mensajes

Para esquematizar la manera en que se implementó el paso de mensajes, se revisará un ejemplo con 7 procesos en el árbol binario.

La primera etapa consiste en realizar las perturbaciones en cada elemento del árbol, utilizando las perturbaciones aceptadas por los nodos padres. En la figura 3.2 se observa al proceso 0 enviando las coordenadas P_0 al proceso 1 (aceptador). El proceso 1 debe actualizar su configuración inmediatamente recibidas las coordenadas del padre. Luego, el proceso 1 envía las coordenadas P_0 a los procesos 3 y 4, y además envía P_1 al proceso 3. Se debe enviar P_0 al proceso 4 y realizar la perturbación pues tiene como padre al proceso 1, quien anteriormente realizó la perturbación en P_0. Como el proceso 3 acepta las perturbaciones de 0 y 1, debe realizar la perturbación en P_0 y luego en P_1. Una situación similar se puede observar en la otra rama del árbol, correspondiente a los procesos 2, 5 y 6.

En resumen, cada proceso realiza las perturbaciones correspondientes a sus padres y a su condición de aceptador o rechazador, y guarda el valor del punto escogido aleatoriamente, pues será utilizado para tomar la decisión en la proxima etapa.

La segunda etapa consiste en resolver las decisiones en cada proceso, determinando si se acepta o no la perturbación en el respectivo punto aleatorio. Si el proceso 0 toma la decisión de aceptar la perturbación en P_0, se descarta la rama asociada al rechazo, correspondiente a los procesos 2, 5 y 6, enviando una señal STOP a estos procesos, como se observa en la figura 3.3. Se repite el procedimiento en cada nodo hijo, siguiendo el camino de aceptación o rechazo en base a la decisión del nodo padre. El camino que se obtiene en la figura 3.3 es $P_0 \rightarrow \phi \rightarrow P_4$, donde ϕ representa un rechazo (no importa el valor de ese punto, pues se debe rechazar). El siguiente paso es informar a todos los procesos de este camino de perturbaciones. Para ello se procede en 2 etapas, primero se debe enviar el camino de perturbaciones al nodo padre o proceso 0, y luego se envía la información a todos los procesos mediante un mensaje Broadcast (`MPI_Bcast`).

Para enviar el camino de perturbaciones al proceso 0, se recorren de manera inversa todos los procesos a los cuales se les envió una señal GO, en este ejemplo, corresponden a los procesos 4 y 1. Como se observa en la figura 3.4, el proceso 4 envía P_4 al proceso 1, pues tomó la decisión de aceptar su perturbación, luego el proceso 1 envía ϕ y P_4 al proceso 0, donde ϕ es un valor que representa un rechazo. Una vez recibidos ambos valores, el proceso 0 decide si incluye o no al punto P_0 en el camino. Como en este ejemplo se aceptó la perturbación a P_0, el camino de perturbaciones final es $P_0 \rightarrow \phi \rightarrow P_4$.

Finalmente el proceso 0 envía el camino $P_0 \rightarrow \phi \rightarrow P_4$ a todos los procesos mediante un mensaje Broadcast y cada uno de ellos se sincroniza de manera que todos queden con la misma configuración.

Este esquema pretende ser de ayuda en caso de implementar mejoras o modificaciones al código, sin embargo, conviene revisar en detalle la implementación para tener una idea de las variables involucradas en los mensajes, el manejo de la memoria, el uso de buffers de datos, la recepción de los mensajes por parte de los procesos destinatarios, la sincronización temporal en base a `MPI_Barrier` y muchos detalles más, los cuales no se discutirán en este trabajo.

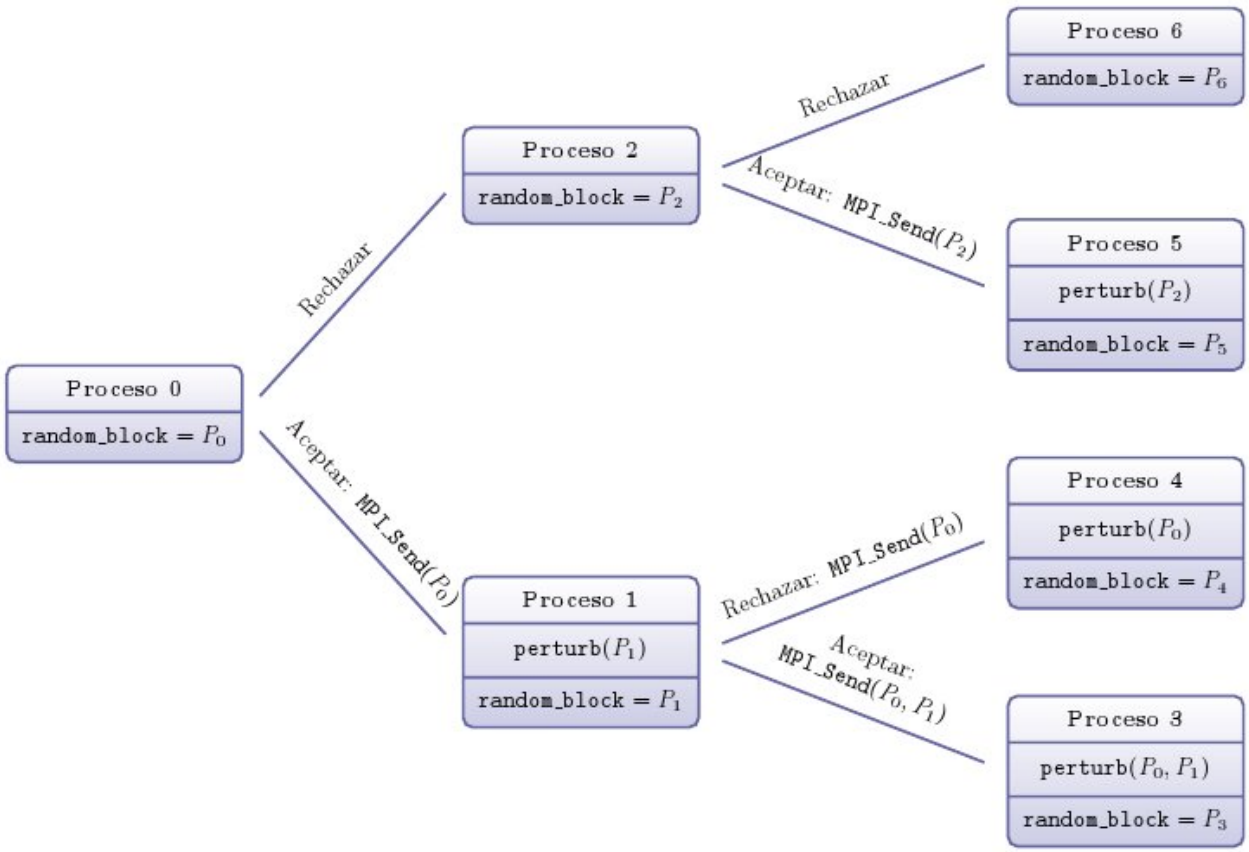


Figura 3.2: Esquema de paso de mensajes: Envío de perturbaciones, 7 procesos

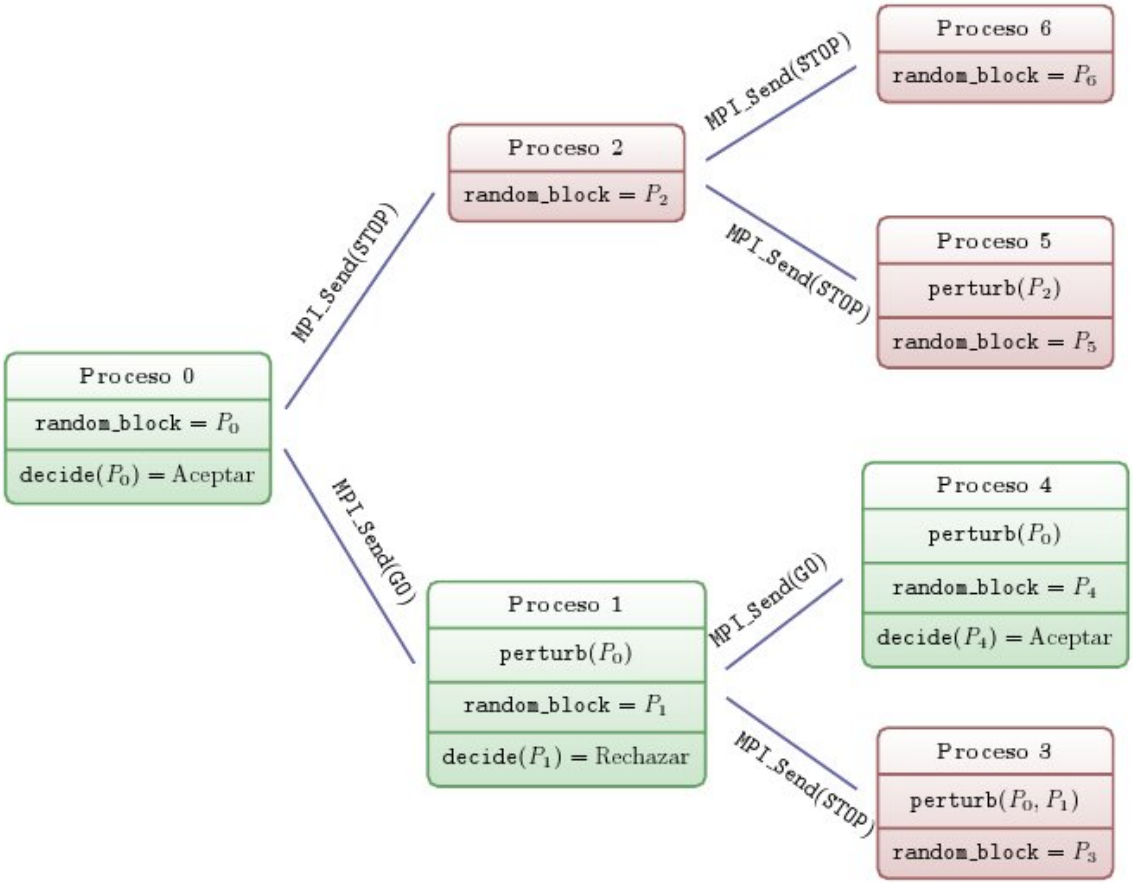


Figura 3.3: Esquema de paso de mensajes: Envío de señales de activación, 7 procesos

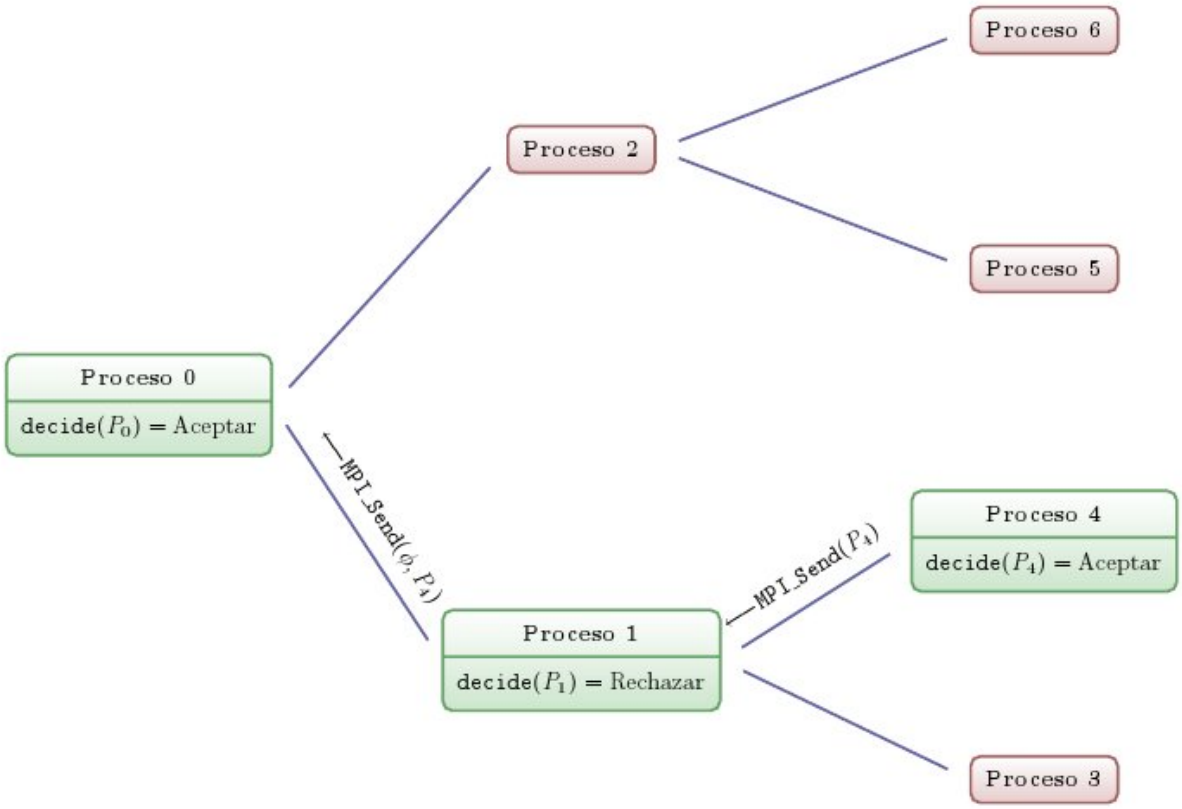


Figura 3.4: Esquema de paso de mensajes: Envío de camino calculado al proceso maestro, 7 procesos

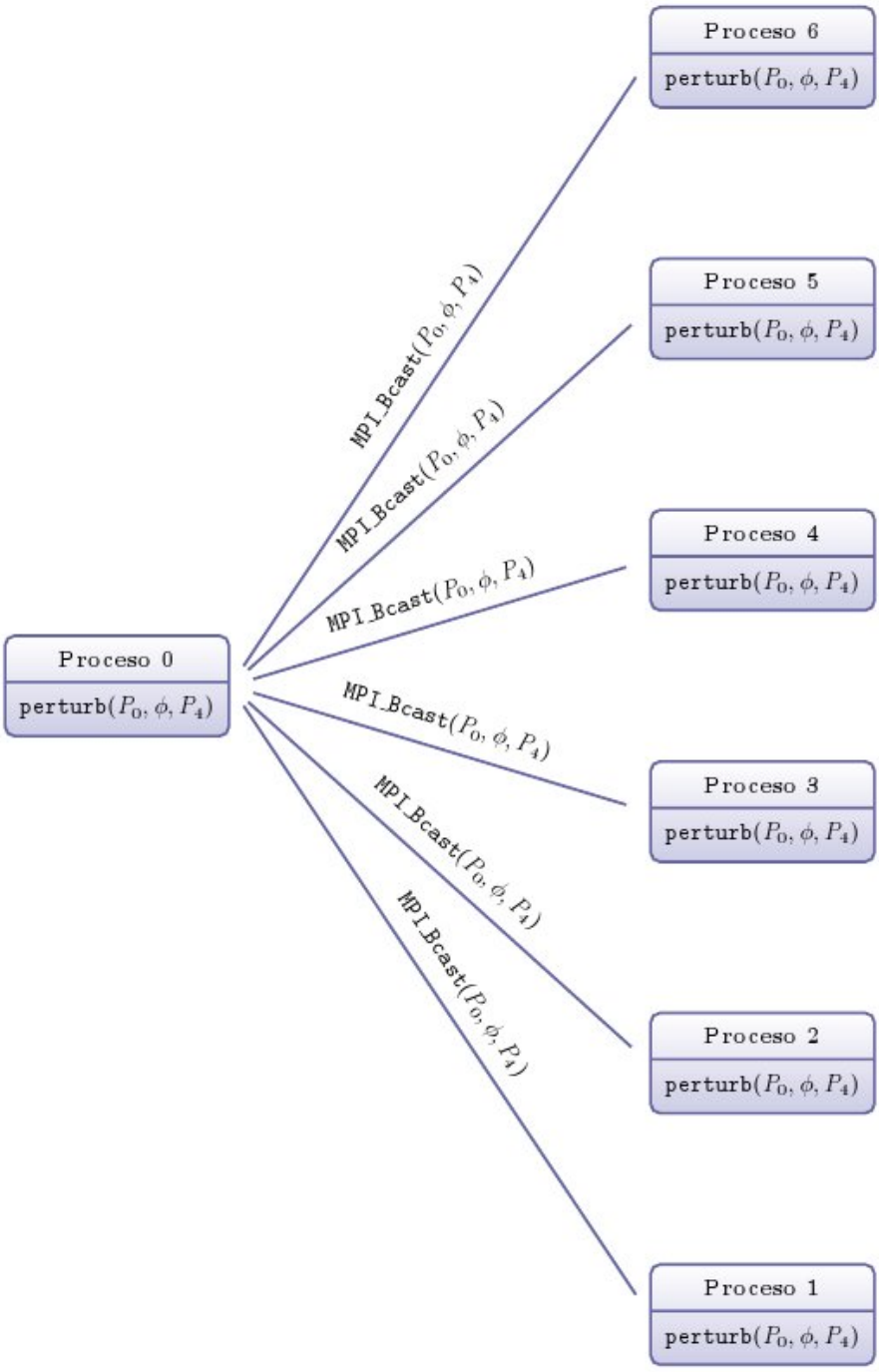


Figura 3.5: Esquema de paso de mensajes: Envío de Broadcast para sincronización de configuraciones, 7 procesos

Capítulo 4

Resultados

La imagen que se desea simular se puede ver en la figura 4.1. Consiste en una grilla de 1's y 0's de dimensiones $100 \times 100 \times 1$. Ella representa la sección 2D de un reservorio fluvial de petróleo. Este tipo de reservorios se caracterizan por la presencia de canales, los cuales se pueden ver claramente definidos, en color negro.

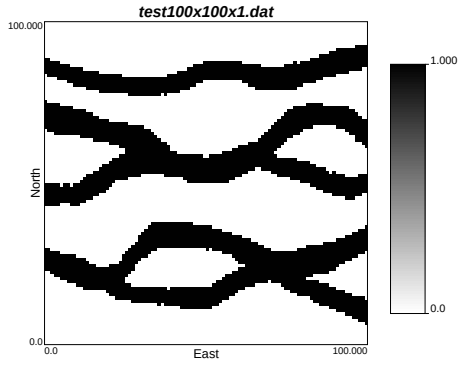


Figura 4.1: Imagen de Entrenamiento

Los dos objetivos que simultáneamente se chequearán son:

- Obtención de simulaciones razonables para patrones de $2 \times 2 \times 1$, $3 \times 3 \times 1$ y $4 \times 4 \times 1$.
- Efectividad de la paralelización comparando con el speedup teórico ($\log_2(P+1)$).

Para uniformar los resultados, se escogieron 3 secuencias aleatorias, que se llamaran *Escenarios A, B y C*. Cada secuencia está compuesta por una serie de 1 millón de números en el intervalo $[0, 1]$ y 1 millón de puntos de la grilla. La razón de utilizar estas secuencias es para chequear que la aplicación secuencial y paralelizada realizan el mismo camino en la cadena de Markov asociada al Recocido Simulado (iguales tomas de decisión e iguales configuraciones). Además todas las pruebas utilizaron el mismo *annealing schedule*, con factor de reducción igual a $0{,}1$ y temperatura inicial igual a 5000.

4.1. Simulaciones

Las simulaciones se realizaron utilizando 2 funciones objetivo diferentes. Ambas son de la forma:

$$O = \|f^{TI} - f^{RE}\|_M^2$$

donde f^{TI} es el vector de frecuencias de aparición de los patrones asociados a un template, el cual puede ser de $2 \times 2 \times 1$, $3 \times 3 \times 1$ o $4 \times 4 \times 1$ celdas.

La primera función objetivo se llamará *Función Objetivo sin Pesos*, pues utiliza $M = I$ (matríz identidad). La segunda función objetivo se llamará *Función Objetivo con Pesos*, y utiliza la siguiente matríz:

$$M_{ij} = \begin{cases} \frac{\frac{1}{f_i^{TI}}}{\sum_{i \in \mathcal{P}} \frac{1}{f_i^{TI}}} & i = j, f_i^{TI} \neq 0 \\ 0 & i = j, f_i^{TI} = 0 \\ 0 & i \neq j \end{cases}$$

Esta matríz representa el inverso de la frecuencia normalizada y tiene por objetivo entregar mayor ponderador a las frecuencias que menos aparecen en la imagen de entrenamiento, y entregar menor ponderador a las que más aparecen. De esta forma, se penalizará en un grado mayor tomar una decisión de aceptar un estado que incremente la frecuencia de los patrones que menos aparecen en la imagen de entrenamiento.

En resumen:

- Sin Pesos asociados a las frecuencias:

$$O = \sum_{i \in \mathcal{P}} (f_i^{TI} - f_i^{RE})^2$$

- Con Pesos asociados a las frecuencias:

$$O = \sum_{i\in\mathcal{P}, f_i^{TI}\neq 0} \frac{\frac{1}{f_i^{TI}}}{\sum_{j\in\mathcal{P}} \frac{1}{f_j^{TI}}} (f_i^{TI} - f_i^{RE})^2 + \sum_{i\in\mathcal{P}, f_i^{TI}=0} (f_i^{RE})^2$$

En las figuras 4.2 y 4.3 se pueden observar los resultados de una realización para cada template, con y sin Peso.

Debido a que una de las características más importantes que se busca en las simulaciones es la conectividad de las estructuras, se puede inferir que las simulaciones con Peso tendrán un mejor resultado.

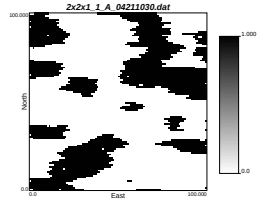
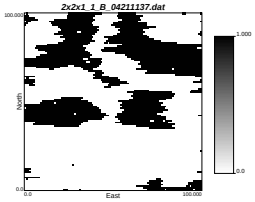
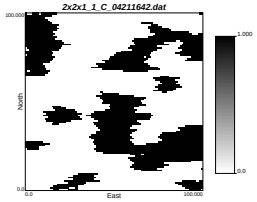
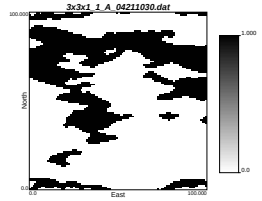
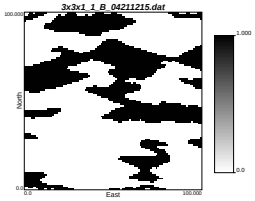
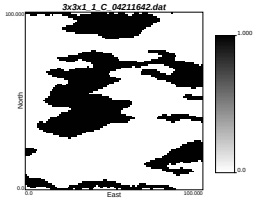
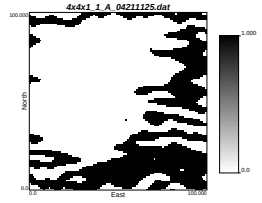
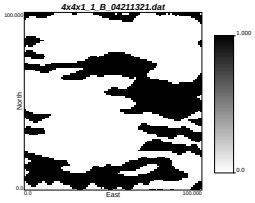
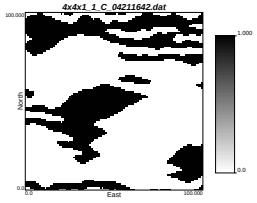
	Escenario A	Escenario B	Escenario C
$2 \times 2 \times 1$			
$3 \times 3 \times 1$			
$4 \times 4 \times 1$			

Figura 4.2: Resultados sin utilizar Pesos en la función objetivo

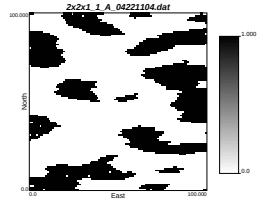
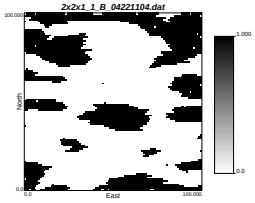
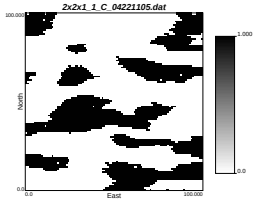
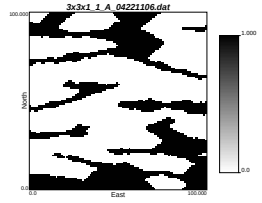
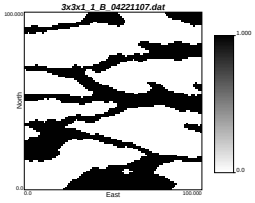
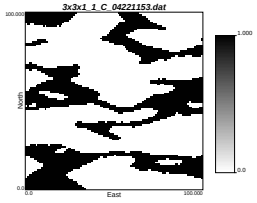
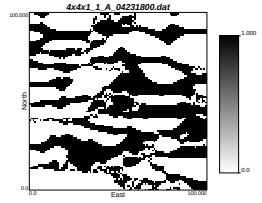
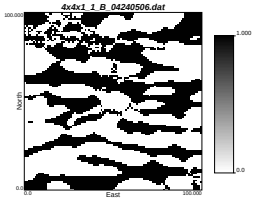
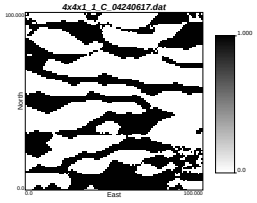
	Escenario A	Escenario B	Escenario C
$2 \times 2 \times 1$			
$3 \times 3 \times 1$			
$4 \times 4 \times 1$			

Figura 4.3: Resultados utilizando Pesos en la función objetivo

Una manera de medir visualmente el resultado de una realización es comparar los histogramas de frecuencias de la imagen de entrenamiento y la realización utilizando la línea recta como objetivo en el gráfico de los pares ordenados (f_i^{TI}, f_i^{RE}), en escala logaritmica. La manera de realizar la medición es verificar que todos los pares ordenados caen en la línea recta central, que une a $(0, 0)$ y $(1, 1)$. Si esto llega a ocurrir, se tiene una reproducción perfecta entre los histogramas de patrones. Si los pares ordenados están demasiado alejados de la recta central, la realización es de mala calidad y no debe considerarse.

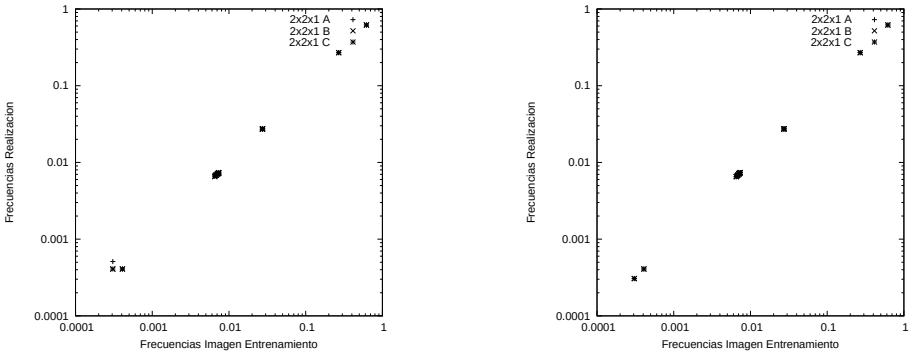


Figura 4.4: Comparación de resultados utilizando histogramas de frecuencia para template de $2 \times 2 \times 1$ celdas, sin (izq.) y con (der.) pesos

En la figura 4.4 se observa el resultado de las realizaciones para los escenarios A, B y C, sin y con pesos en la función objetivo, para un template de $2 \times 2 \times 1$ celdas. En ambos casos se obtiene una alineación con la recta central, por lo tanto se consideran aceptables las realizaciones.

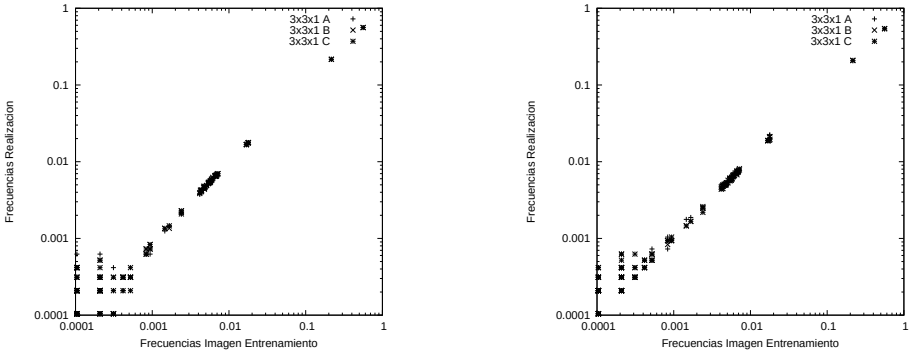


Figura 4.5: Comparación de resultados utilizando histogramas de frecuencia para template de $3 \times 3 \times 1$ celdas, sin (izq.) y con (der.) pesos

En la figura 4.5 se observa el resultado de las realizaciones para los escenarios A, B y C, sin y con pesos en la función objetivo, para un template de $3 \times 3 \times 1$ celdas. En este caso, se observa una mayor dispersión de los pares ordenados en la zona más cercana al cero para la función objetivo sin pesos. A simple vista no se aprecian grandes diferencias, por lo tanto aún se pueden considerar aceptables las realizaciones.

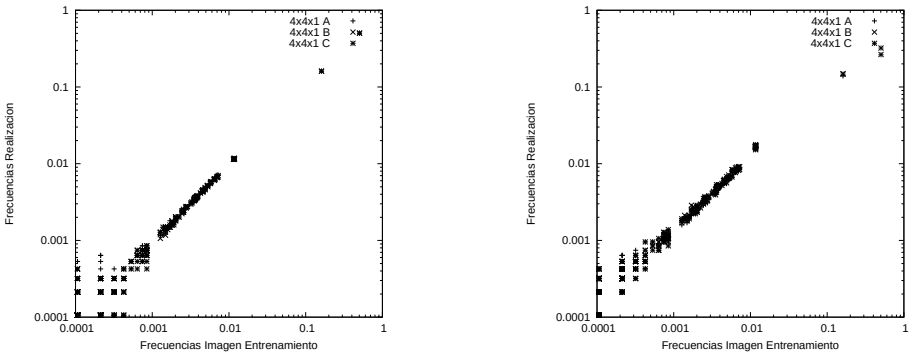


Figura 4.6: Comparación de resultados utilizando histogramas de frecuencia para template de $4 \times 4 \times 1$ celdas, sin (izq.) y con (der.) pesos

En la figura 4.6 se observa el resultado de las realizaciones para los escenarios A, B y C, sin y con pesos en la función objetivo, para un template de $4 \times 4 \times 1$ celdas. En este caso, se observa una mayor dispersión de los pares ordenados en la zona más cercana al cero para la función objetivo sin pesos. Ahora si se aprecian diferencias considerables y si bien ambos resultados no están demasiado cercanos a la recta central, se observa que la realización con pesos se acerca en mayor medida al centro, por lo cual se considera más aceptable que la realización sin pesos.

En la figura 4.7, se puede observar la manera en la que el algoritmo va ajustando la realización. Lo anterior puede dar indicios del camino que está tomando en la cadena de Markov. Por ejemplo, si vemos que la imagen permanece estática durante la mayoría de las iteraciones, esto indica que se están realizando demasiados rechazos, lo que obliga a cambiar el *annealing schedule*.

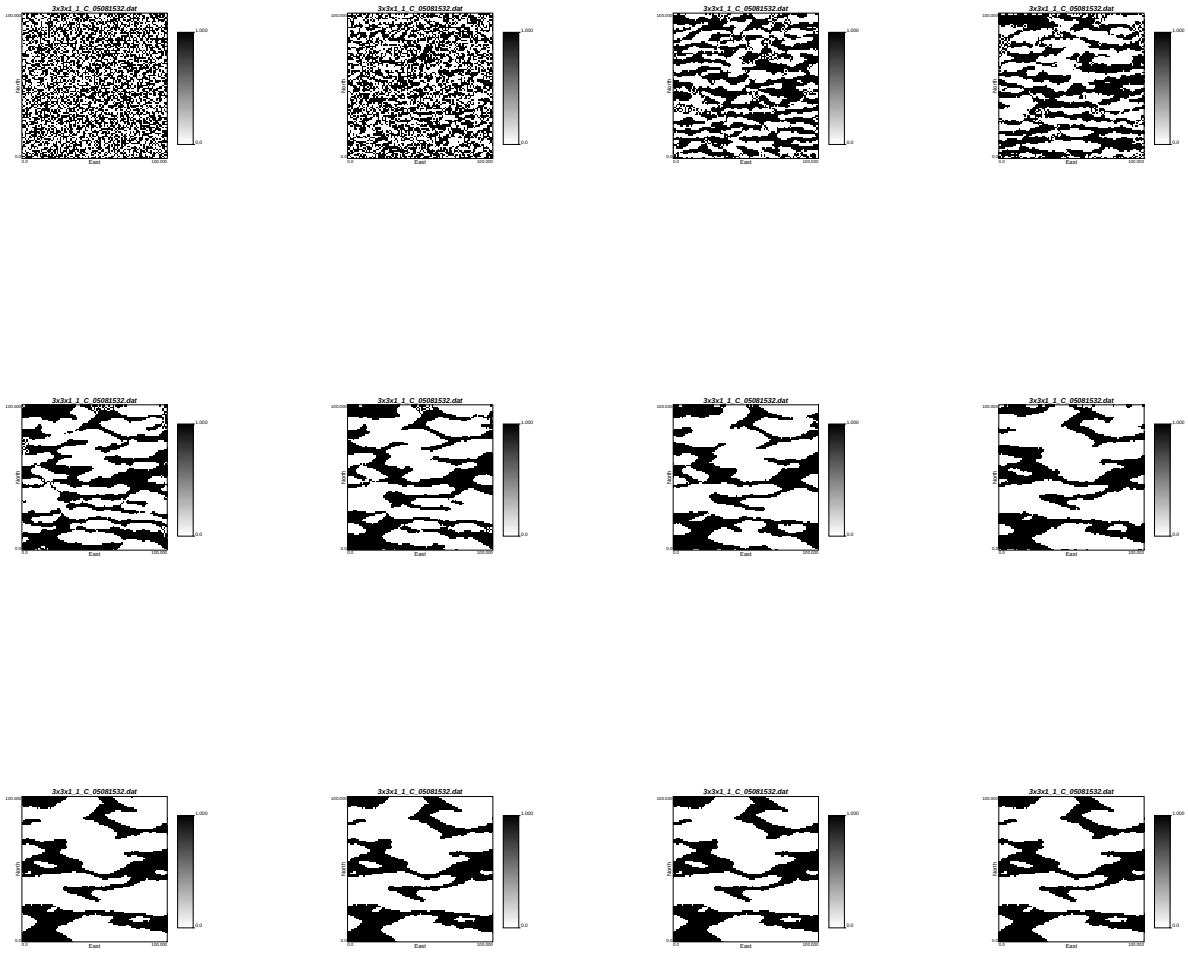


Figura 4.7: Evolución de la realización

Otro resultado interesante se puede ver en las figuras 4.8, 4.9, 4.10, 4.11, 4.12 y 4.13. En ellas, el gráfico de la izquierda representa la evolución del valor de la función objetivo versus el número de iteraciones, en escala logarítmica. Según la literatura y las indicaciones del profesor guía, la evolución esperada se debía parecer a una escalera (con mayor o menor curvatura, según el *annealing schedule* y la escala con la que se grafique). Se obtuvieron los resultados esperados para la mayoría de las realizaciones, sin embargo, se aprecia que para el template $4 \times 4 \times 1$, el descenso no fue razonable, lo cual da indicios de que se debe cambiar el *annealing schedule* para ese template. En los gráficos del lado derecho se representa el descenso porcentual con respecto al primer valor objetivo obtenido. Esto representa una medida de la calidad del descenso, pues si se obtienen 2 realizaciones, y en la primera se observa un descenso porcentual más brusco que en la otra, probablemente la primera está estancada en un mínimo local peor que el de la segunda.

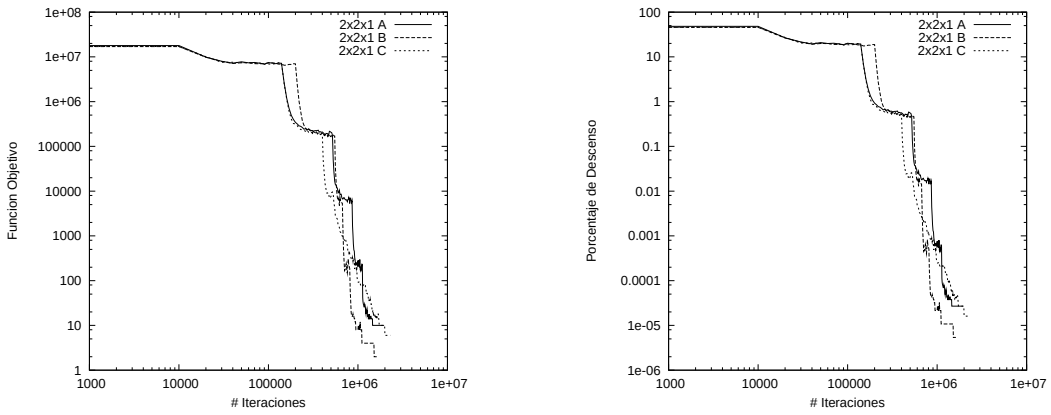


Figura 4.8: # Iteraciones vs. Función Objetivo (izquierda) y # Iteraciones vs. % de descenso (derecha) sin Pesos en la función objetivo, para template $2 \times 2 \times 1$

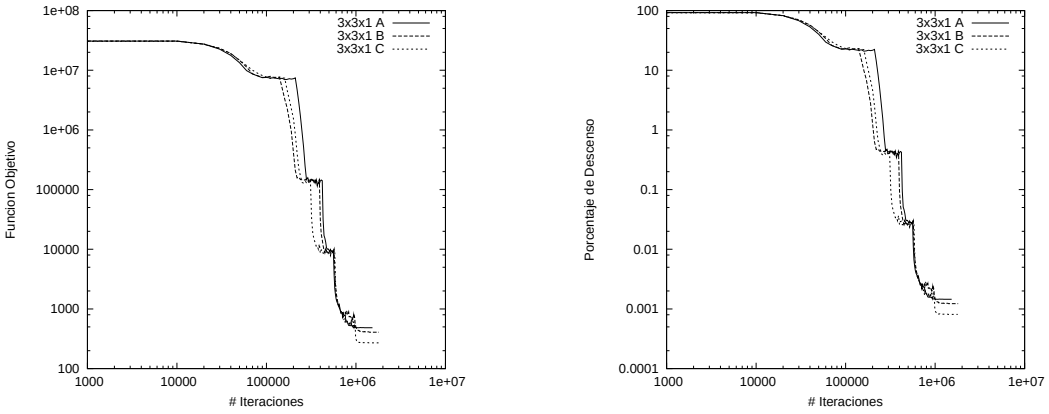


Figura 4.9: # Iteraciones vs. Función Objetivo (izquierda) y # Iteraciones vs. % de descenso (derecha) sin Pesos en la función objetivo, para template $3 \times 3 \times 1$

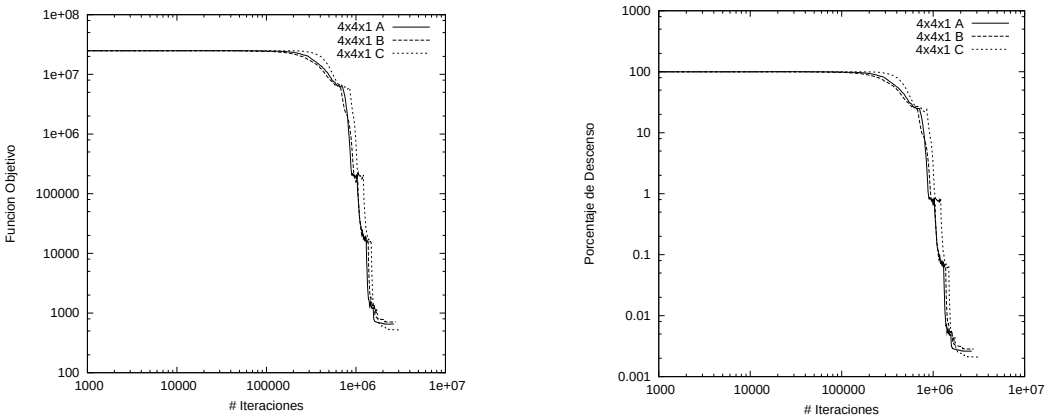


Figura 4.10: # Iteraciones vs. Función Objetivo (izquierda) y # Iteraciones vs. % de descenso (derecha) sin Pesos en la función objetivo, para template $4 \times 4 \times 1$

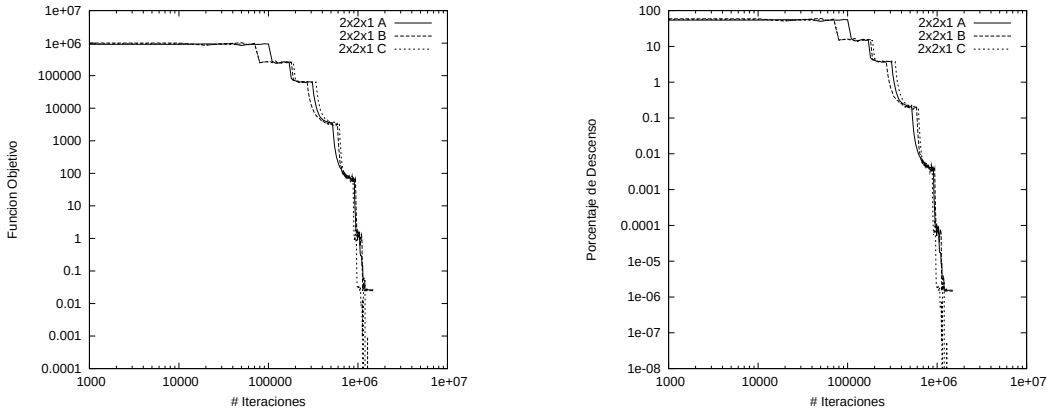


Figura 4.11: # Iteraciones vs. Función Objetivo (izquierda) y # Iteraciones vs. % de descenso (derecha) con Pesos en la función objetivo, para template $2 \times 2 \times 1$

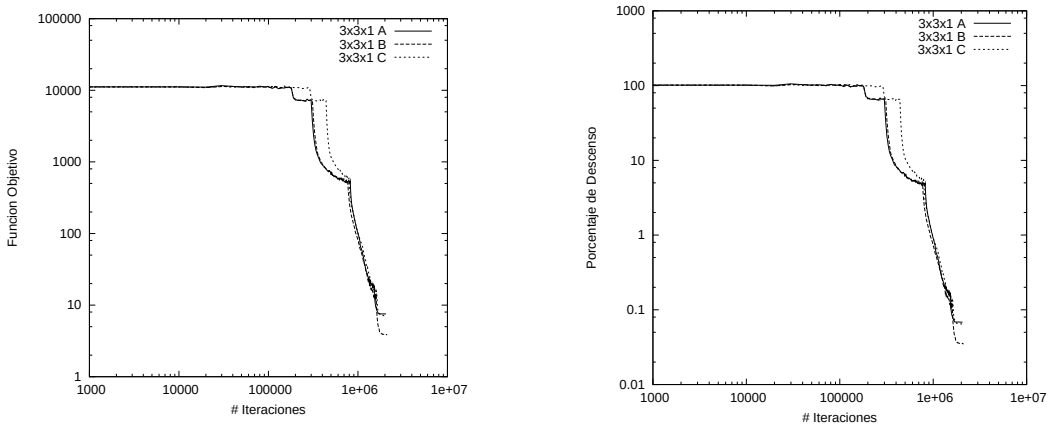


Figura 4.12: # Iteraciones vs. Función Objetivo (izquierda) y # Iteraciones vs. % de descenso (derecha) con Pesos en la función objetivo, para template $3 \times 3 \times 1$

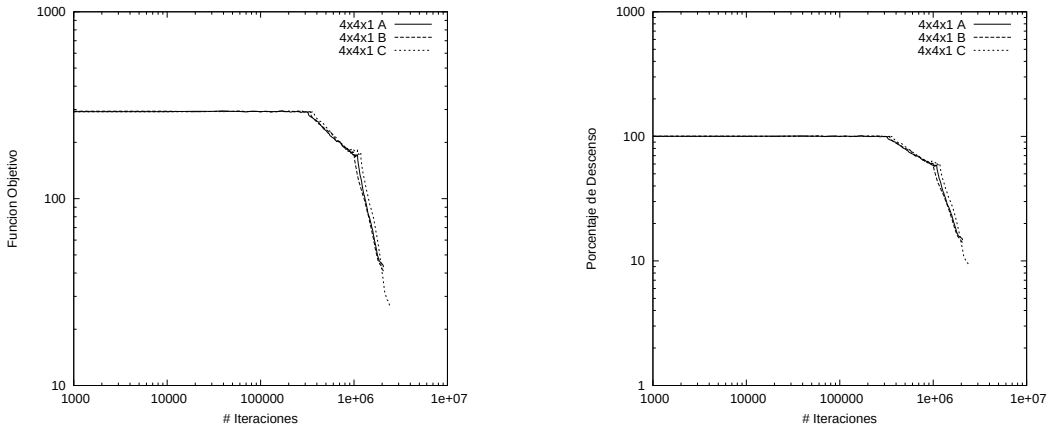


Figura 4.13: # Iteraciones vs. Función Objetivo (izquierda) y # Iteraciones vs. % de descenso (derecha) con Pesos en la función objetivo, para template $4 \times 4 \times 1$

4.2. Comparación con Speedup Teórico

El speedup teórico calculado en el capítulo 2 fue $\log_2(P+1)$. Realizando una observación simple a las figuras 4.14 y 4.15, se puede ver que la cota superior en el speedup de $\log_2(P+1)$ tiene sentido y en algunos casos se puede estar cerca de alcanzarla.

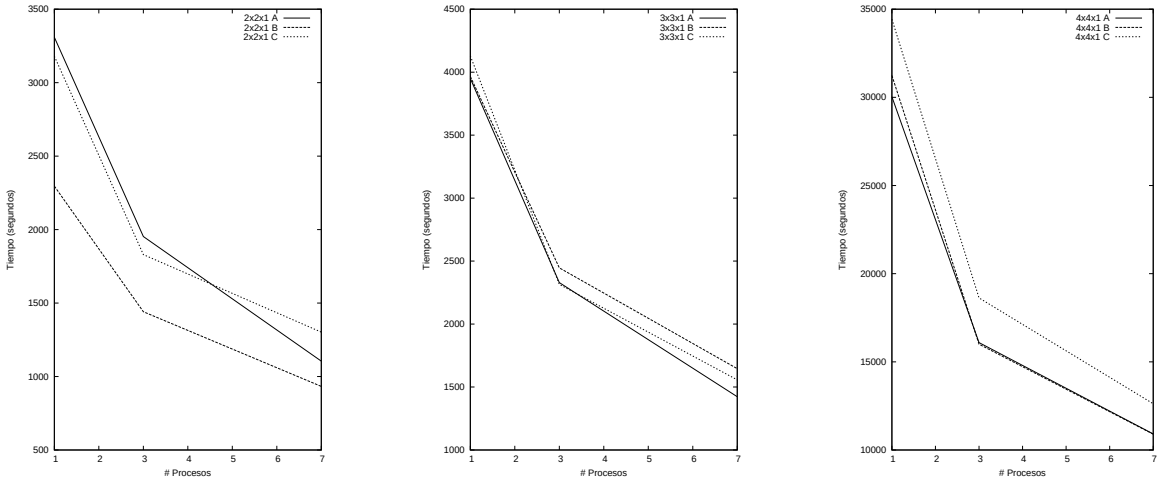


Figura 4.14: # Procesos vs. Tiempo (segundos), sin Pesos en la función objetivo

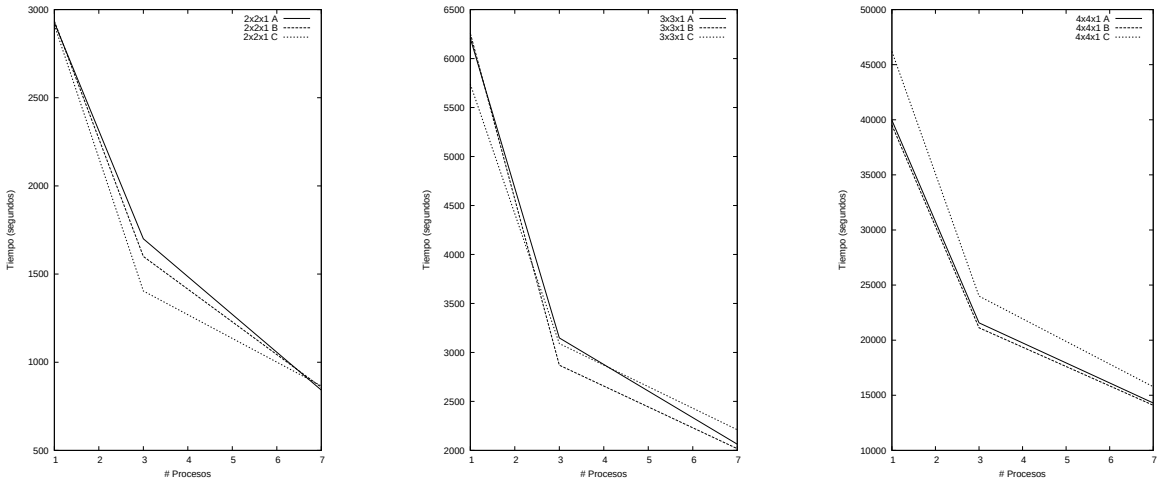


Figura 4.15: # Procesos vs. Tiempo (segundos), con Pesos en la función objetivo

Para la función objetivo sin Pesos, se obtuvieron los resultados de las tablas en la figura 4.16. El speedup se puede ver en la figura 4.17.

Escenario A	$2 \times 2 \times 1$	$3 \times 3 \times 1$	$4 \times 4 \times 1$
1	3309	3950	30037
3	1953	2329	16094
7	1105	1423	10911

Escenario B	$2 \times 2 \times 1$	$3 \times 3 \times 1$	$4 \times 4 \times 1$
1	2296	3966	31240
3	1441	2446	16000
7	933	1646	10897

Escenario C	$2 \times 2 \times 1$	$3 \times 3 \times 1$	$4 \times 4 \times 1$
1	3180	4129	34432
3	1830	2315	18624
7	1302	1557	12625

Figura 4.16: Tablas de tiempos para función objetivo sin Pesos

Escenario A	$2 \times 2 \times 1$	$3 \times 3 \times 1$	$4 \times 4 \times 1$
3	1,694	1,696	1,866
7	2,994	2,775	2,752

Escenario B	$2 \times 2 \times 1$	$3 \times 3 \times 1$	$4 \times 4 \times 1$
3	1,593	1,621	1,952
7	2,460	2,409	2,866

Escenario C	$2 \times 2 \times 1$	$3 \times 3 \times 1$	$4 \times 4 \times 1$
3	1,737	1,783	1,848
7	2,442	2,651	2,727

Figura 4.17: Tablas de Speedup para la función objetivo sin Pesos

Para la función objetivo con Pesos, se obtuvieron los resultados de las tablas en la figura 4.18. El speedup se puede ver en la figura 4.19.

Escenario A	$2 \times 2 \times 1$	$3 \times 3 \times 1$	$4 \times 4 \times 1$
1	2926	6207	39960
3	1700	3148	21572
7	842	2063	14304

Escenario B	$2 \times 2 \times 1$	$3 \times 3 \times 1$	$4 \times 4 \times 1$
1	2935	6257	39525
3	1600	2870	21125
7	858	2018	14094

Escenario C	$2 \times 2 \times 1$	$3 \times 3 \times 1$	$4 \times 4 \times 1$
1	2913	5739	46201
3	1404	3090	23999
7	866	2211	15778

Figura 4.18: Tablas de tiempos para función objetivo con Pesos

Escenario A	$2 \times 2 \times 1$	$3 \times 3 \times 1$	$4 \times 4 \times 1$
3	1,721	1,971	1,851
7	3,475	3,008	2,793

Escenario B	$2 \times 2 \times 1$	$3 \times 3 \times 1$	$4 \times 4 \times 1$
3	1,834	2,180	1,871
7	3,420	3,100	2,804

Escenario C	$2 \times 2 \times 1$	$3 \times 3 \times 1$	$4 \times 4 \times 1$
3	2,074	1,857	1,925
7	3,363	2,595	2,928

Figura 4.19: Tablas de Speedup para la función objetivo con Pesos

Los resultados confirman la veracidad del speedup teórico, pues en la mayoría de los casos se obtienen valores cercanos a 2 cuando se utilizan 3 procesos y cercanos a 3 cuando se utilizan 7 procesos. Se observa que la utilización de pesos en la función objetivo entrega resultados más cercanos al speedup teórico.

No se realizaron pruebas con más procesos debido a las limitaciones físicas del Laboratorio de Planificación Minera, sin embargo en el futuro se puede realizar una prueba con más procesos sin necesidad de modificar la implementación.

Capítulo 5

Conclusiones y Trabajo a futuro

5.1. Conclusiones

En base a los resultados obtenidos en el capítulo anterior, se puede concluir lo siguiente:

- La incorporación de múltiples procesadores, configurados según un enfoque especulativo, efectivamente reduce el tiempo de cálculo con un speedup de $\log_2(P+1)$, lo cual es un avance importante en el desarrollo de técnicas que utilicen estadísticas de múltiples puntos y simulaciones de tipo Recocido Simulado.
- La utilización de pesos en la función objetivo influye levemente en la obtención de mejores realizaciones. Sin embargo, tiene como trade-off un mayor tiempo de cálculo a medida que crece el tamaño del template.
- El speedup teórico es alcanzado con mayor presición cuando se utiliza una función objetivo con pesos.

Sin duda, el resultado más importante obtenido en este trabajo es la verificación de la obtención experimental del speedup teórico $\log_2(P+1)$. La importancia radica en la naturaleza intrínsecamente secuencial del Recocido Simulado, lo que hace muy difícil abordarlo usando un enfoque paralelo. A pesar de ello, se logró realizar exitosamente la paralelización y se validaron los resultados obtenidos con varios procesadores usando el resultado secuencial, utilizando los mismos caminos de perturbaciones y decisiones.

Esta implementación abre paso a desarrollos futuros que se realizarán en el Departamento de Ingeniería en Minas, pues logra demostrar que un cluster de procesadores puede ser utilizado con mayor provecho al incorporar desarrollos que utilicen el paso de mensajes entre procesos como eje principal, en particular la librería MPI. Además sirve como ejemplo de la utilización que se

le puede dar al cluster que actualmente existe en el Departamento, y a los que se instalarán en el futuro.

Con respecto a la comparación entre la Simulación Convencional y el Recocido Simulado junto a estadísticas de múltiples puntos, basandonos en este trabajo, se puede concluir lo siguiente:

- Aún no se puede realizar una afirmación sobre la calidad de los resultados al utilizar patrones de mayor tamaño, pues debido al excesivo tiempo de cálculo no se pudieron realizar más pruebas. Sin embargo, con los resultados obtenidos para patrones de $3 \times 3 \times 1$, se puede inferir un futuro exitoso en la mejoría de la calidad de resultados. Hay que considerar que todas las pruebas realizadas fueron iniciadas con una imagen de entrenamiento absolutamente aleatoria, por lo tanto, si se fijan los datos conocidos en la imagen inicial, se induce una mayor presición en la información y un mejor punto de partida para la simulación.
- El tiempo de cálculo depende absolutamente de la cantidad de procesos que se utilicen. Debido a esto, para obtener resultados de mejor calidad en tiempo y resultados se debe invertir dinero en adquisición de tecnología (servidores, clusters, capacitación, etcétera).
- Por motivos de tiempo, no se logró realizar comparaciones empíricas con resultados obtenidos mediante Simulación Convencional, sin embargo, queda abierta la posibilidad de realizar estas comparaciones de manera formal en el futuro no lejano. El profesor guía consideró que este trabajo constituye el primer intento documentado a nivel mundial que involucra la utilización de cálculo paralelo para realizar una simulación no convencional en base a estadísticas de múltiples puntos en el contexto espacial, por lo tanto hay muchos temas que se pueden investigar y afinar para llegar eventualmente a obtener mejores resultados que la Simulación Convencional.

5.2. Trabajo a futuro

El trabajo realizado dejó muchos caminos abiertos por los cuales se puede continuar el desarrollo. Los más importantes son los siguientes:

- Optimización de las estructuras de datos utilizadas para manejar los patrones. Debido a que el tiempo de cálculo crece de manera exponencial con el tamaño del template, es el trabajo a futuro más importante que se puede realizar para optimizar el tiempo de cálculo y permitir la inclusión de patrones más grandes.
- Realización de conteo simple utilizando múltiples procesos. La implementación actual se realizó de manera que en el futuro sea relativamente simple incorporar cálculo paralelo. Se puede utilizar una división uniforme de la grilla y asignar a cada proceso un área de conteo y finalmente consolidar las frecuencias.
- Incorporación de árboles no balanceados en la configuración especulativa. Como se menciona en el capítulo Antecedentes, es posible dotar al árbol binario de una topología dinámica

que cambie utilizando a la temperatura como parámetro. Cuando la temperatura es alta, ocurren más rechazos, por lo tanto el árbol debería estar compuesto por más nodos de rechazo. Cuando la temperatura es baja, ocurren más aceptaciones, y el árbol debería estar compuesto por más nodos de aceptación. Todo esto mejoraría aún más el speedup obtenido.

- Incorporación de información de mega-bloques. Si se utilizan estadísticas que consideran mega-bloques compuestos por grupos de bloques, se puede concluir información con mayor veracidad.

- Utilización de templates no regulares. En este desarrollo sólo se utilizaron templates con forma de paralelepípedos. Un desafío importante es considerar templates con formas no triviales, por ejemplo estrellas, cruces, líneas, círculos, figuras no conexas, etc.

- Realización de pruebas utilizando una mayor cantidad de procesos. Debido a limitaciones físicas no se pudieron realizar pruebas con más de 7 procesadores, lo cual dejó una incógnita acerca del comportamiento de la implementación cuando el número de procesadores crece de manera considerable.

- Implementación del Recocido Simulado con múltiples procesos utilizando el lenguaje Fortran. Si bien se obtuvieron resultados razonables, Fortran sigue siendo hasta el día de hoy uno de los lenguajes de mayor eficiencia y por lo tanto de mayor utilidad para implementar aplicaciones en Geoestadística. La incorporación de múltiples procesos a aplicaciones implementadas en este lenguaje debería dar como resultado mejoras considerables en los tiempos de cálculo de las simulaciones.

Apéndice A

Documentación de las clases

A.1. Referencia de la Estructura block

Estructura que almacena la información de un punto de la grilla. Cuando se hable de bloque o punto, se entenderá por un elemento asociado a esta estructura.

Atributos públicos

- int **x**
- int **y**
- int **z**
- char **data**

A.1.1. Descripción detallada

Estructura que almacena la información de un punto de la grilla. Cuando se hable de bloque o punto, se entenderá por un elemento asociado a esta estructura.

Definición en la línea 130 del archivo mps-anneal.c.

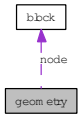
La documentación para esta estructura fue generada a partir del siguiente fichero:

- src/mps-anneal.c

A.2. Referencia de la Estructura geometry

Estructura que almacena la información de la grilla de puntos. También se utiliza para almacenar el template que generará los patrones en la simulación.

Diagrama de colaboración para geometry:



Atributos públicos

- int **lengthx**
- int **lengthy**
- int **lengthz**
- block *** **node**

A.2.1. Descripción detallada

Estructura que almacena la información de la grilla de puntos. También se utiliza para almacenar el template que generará los patrones en la simulación.

Definición en la línea 142 del archivo mps-anneal.c.

La documentación para esta estructura fue generada a partir del siguiente fichero:

- src/mps-anneal.c

Apéndice B

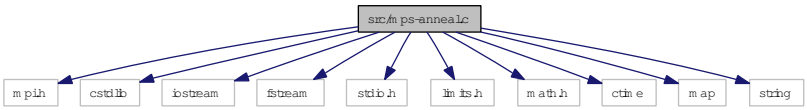
Documentación de archivos

B.1. Referencia del Archivo src/mps-anneal.c

Programa que realiza el Recocido Simulado (Simulated Annealing) para la estimación de estadísticas de múltiples puntos (MPS) utilizando Computación Paralela, bajo el formato del estándar MPI.

```
#include "mpi.h"

#include <cstdlib>

#include <iostream>

#include <fstream>

#include <stdio.h>

#include <limits.h>

#include <math.h>

#include <ctime>

#include <map>

#include <string>
```

Dependencia gráfica adjunta para mps-anneal.c:



Clases

- struct block

  Estructura que almacena la información de un punto de la grilla. Cuando se hable de bloque o punto, se entenderá por un elemento asociado a esta estructura.

- struct geometry

  Estructura que almacena la información de la grilla de puntos. También se utiliza para almacenar el template que generará los patrones en la simulacion.

Definiciones

- #define **WHITE** '/'
- #define **BLACK** 'P'
- #define **GREY** '\000'
- #define **RELEVANT** 'b'
- #define **NOT_RELEVANT** 'N'
- #define **DEBUG** 0
- #define **DEBUG_AC** 0
- #define **DEBUG_RE** 0
- #define **DEBUG_OUT** 0
- #define **DEBUG_PRINT** 0
- #define **DEBUG_END** 1

Funciones

- long filesize (char *filename)

  Calcula el largo en filas de un archivo.

- int level (int my_id)

  Calcula el nivel dentro del árbol binario de un proceso.

- void load_template (geometry *tem)

  Cargar template con dimensiones entregadas como parámetros.

- void free_template (geometry *tem)

  Liberar la memoria utilizada para almacenar el template.

- void reset_template (geometry *tem)

  Resetear posiciones del template.

- void load_associatedPatterns (geometry *tem, geometry *reservoir, unsigned int HASH_SIZE, map< string, int > &hash, ofstream &file)

  Cargar patrones asociados a un template, desde una grilla hacia un map, guardando las frecuencias de aparición.

- void evaluate_template (int despx, int despy, int despz, block b, int numPattern, geometry *tem, geometry *reservoir, unsigned int HASH_SIZE, map< string, int > &hash, ofstream &file)

  Evaluar los patrones que aparecen asociados a un punto en la grilla, para un template dado, guardando las frecuencias en un map.

- string evaluate_template_ret (int despx, int despy, int despz, int *coord, int numPattern, geometry *tem, geometry *reservoir, map< string, int > &hash)

  Función análoga a evaluate_template() pero que retorna un string.

- void load_reservoir (geometry *reservoir, geometry *tem, char *filename, double proportion)

  Cargar la grilla con los datos con dimensiones entregadas como parámetros en una estructura geometry.

- void copy_reservoir (geometry *dest, geometry *orig)

  Copiar una grilla almacenada en una estructura geometry.

- void print_reservoir (geometry *reservoir, char *filename)

  Imprimir una grilla en un archivo.

- void generate_reservoir (geometry *reservoir, double proportion)

  Generar grilla aleatoria.

- void free_reservoir (geometry *reservoir)

  Liberar memoria utilizada para almacenar la grilla.

- void free_reservoir_nodes (geometry *reservoir)

  No se utliza.

- void open_logfile (ofstream &file, char *logfilename)

  Abrir archivo log, donde quedará registrado el histograma de frecuencias en cada actualización.

- void print_to_log (ofstream &file, char *line)

  Imprimir una línea en el archivo log.

- void close_logfile (ofstream &file)

  Cerrar archivo log, donde quedará registrado el histograma de frecuencias en cada actualización.

- int * load_topology_dims (char *filename)

  Carga las dimensiones del árbol binario que define la topología.

- void load_topology (char *filename, int *index, int *edges)

  Carga la topología del árbol binario.

- void load_random (char *filename, float *randomarray)

  Cargar camino aleatorio de números entre 0 y 1. Se utiliza para realizar una simulación con un camino predeterminado.

- void load_random_3cols (char *filename, int *rand1, int *rand2, int *rand3)

  Cargar camino aleatorio de puntos 3D en la grilla de datos. Se utiliza para realizar una simulación con un camino predeterminado.

- void print_histogram (ofstream &file, map< string, int > &hash)

  Imprimir el histograma de frecuencias almacenado en un map en un archivo.

- void schedule_update (int iter, int irepo, double startobj, double *latestobj, int *attempt, int maxatt, double *T, double lambda, int *tred, int ntemp, int maxrepetitions, int *repcounter, map< string, int > &hash, map< string, int > &hashrealization, geometry *realization, char *outfile, ofstream &file, char *logfilename, float prom)

  Realiza la actualización del schedule, actualizando la temperatura, el número de intentos, el número de repeticiones y el factor de reducción lambda.

- void print_schedule (double latestobj, int attempt, double T, int tred, char *filename)

  Imprime el schedule actual.

- void load_schedule (double *latestobj, int *attempt, double *T, int *tred, char *filename)

  Carga el schedule desde un archivo, guardando los datos en variables.

- float total_weight (map< string, int > &hash, float c)

  Peso total de los inversos de las frecuencias. Se utiliza cuando se setea la función objetivo con Pesos.

- int * random_block (geometry *reservoir, int proc, int iter, int *rand1, int *rand2, int *rand3)

  Calcular un bloque o punto aleatorio en la grilla utilizando el camino aleatorio especificado por rand1, rand2 y rand3.

- double evaluate_realization (map< string, int > &hash, map< string, int > &hashrealization, float prom)

  Evaluar la función objetivo en un estado.

- void perturb_realization (geometry *tem, geometry *reservoir, int *coord, map< string, int > &hashrealization, int my_id, int iterout)

  Perturbar un estado, cambiando de color un punto de la grilla, tras lo cual se actualiza el map asociado a la grilla, donde se guarda el histograma de frecuencias.

- int decide_perturbation (double old_value, double new_value, double temp, int proc, int iter, int num_procs, float *randarray)

  Decidir si se acepta o no una perturbación.

- int main (int argc, char **argv)

  Función principal. Realiza la lectura de los argumentos y la iteración principal del Recocido Simulado en Paralelo.

B.1.1. Descripción detallada

Programa que realiza el Recocido Simulado (Simulated Annealing) para la estimación de estadísticas de múltiples puntos (MPS) utilizando Computación Paralela, bajo el formato del estandar MPI.

Autor:

Oscar Francisco Peredo Andrade operedo@gmail.com

Fecha:

30 de Mayo del 2008

Versión:

1.0

El desarrollo de esta implementación se enmarca en el proyecto FONDECYT-REGULAR 1061260 titulado "Evaluación de yacimientos mediante simulación estocástica integrando estadísticas de múltiples puntos". La implementación se probó en el Laboratorio de Planificación Minera, el cual cuenta con un cluster Rocks, que permite ingresar trabajos a un cola para ser procesados en algún momento por las máquinas.

Para mayores detalles contactar a Oscar Peredo (operedo@gmail.com) o Julián Ortiz (jortiz@ing.uchile.cl).

Para compilar este archivo se debe ejecutar, desde el directorio mps-anneal:

```
$ mpic++ -g -o bin/mps-anneal src/mps-anneal.c
```

Se incluye un Makefile con el cual se puede setear el directorio de salida (bin) y entrada (src), además de otras librerías que se deseen utilizar:

```
$ make
```

Para ejecutar, se utiliza el siguiente script, llamado execute_mps-anneal.sh:

```
#!/bin/bash
#$ -cwd
#$ -j y
#$ -m abes
#$ -N mps-anneal
#$ -notify
#$ -pe mpi 7
#$ -S /bin/bash
#$ -now no
set -- $(date "+%Y %m %d %H %M %S %Z")
year=$1
month=$2
day=$3
hour=$4
min=$5
sec=$6
tz=$7
timestamp="${month}${day}${hour}${min}"
nx=100
ny=100
nz=1
tnx=2
tny=2
tnz=1
scenario="B"
resources="/home/operedo/mps-anneal/resources"
dataTI="${resources}/test${nx}x${ny}x${nz}.dat"
dataRE="${resources}/randomimage${nx}x${ny}x${nz}.dat"
```

```
logTI="${resources}/logTI_${nx}x${ny}x${nz}_${tnx}x${tny}x${tnz}_${NSLOTS}_\,
                                                ${scenario}_${timestamp}.txt"
logRE="${resources}/logRE_${nx}x${ny}x${nz}_${tnx}x${tny}x${tnz}_${NSLOTS}_\,
                                                ${scenario}_${timestamp}.txt"
topo="${resources}/topology${NSLOTS}.dat"
out="${resources}/${tnx}x${tny}x${tnz}_${NSLOTS}_${scenario}_${timestamp}.dat"
sched="${resources}/schedule${nx}x${ny}x${nz}_${tnx}x${tny}x${tnz}_${NSLOTS}_\,
                                                ${scenario}_${timestamp}.dat"
rand="${resources}/randomfile${nx}x${ny}x${nz}_${scenario}.dat"
rand2="${resources}/randomfile${nx}x${ny}x${nz}_${scenario}_2.dat"
hashsize=400000
t0=5000
lambda=0.1
npert=$[1000 * $nx * $ny *$nz]
ntemp=10
maxatt=5
irepo=$[1*$nx*$ny*$nz]
advance="salida${tnx}x${tny}x${tnz}_${NSLOTS}_${timestamp}_${scenario}_${t0}.dat"
mpirun -np $NSLOTS bin/mps-anneal      $dataTI $dataRE $logTI $logRE $topo $out \,
                                       $sched $rand $rand2 $nx $ny $nz $tnx $tny\,
                                       $tnz $hashsize $t0 $lambda $npert $ntemp \,
                                       $maxatt $irepo > salidas/$advance
exit 0
```

Este script se ejecuta de la siguiente manera:

```
$ qsub execute_mps-anneal.sh
Your job 227 ("mps-anneal") has been submitted
```

Para revisar la cola de trabajos ingresados al cluster:

```
$ qstat
job-ID prior   name       user    state submit/start at     queue slots ja-task-ID
-----------------------------------------------------------------------------------
227    0.00000 mps-anneal operedo qw    06/03/2008 06:02:16       7
```

Para eliminar un trabajo de la cola:

```
$ qdel 227
```

La línea que indica cuantos procesos se utilizarán es la siguiente:

```
#$ -pe mpi 7
```

El resto de las líneas describen el valor y la ubicación de los parámetros que se entregan al programa.

Definición en el archivo mps-anneal.c.

B.1.2. Documentación de las funciónes

B.1.2.1. void close_logfile (ofstream & *file*)

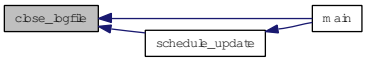
Cerrar archivo log, donde quedará registrado el histograma de frecuencias en cada actualización.

Parámetros:

file

Definición en la línea 1523 del archivo mps-anneal.c.

Gráfico de llamadas a esta función



B.1.2.2. void copy_reservoir (geometry * *dest*, geometry * *orig*)

Copiar una grilla almacenada en una estructura geometry.

Parámetros:

dest

orig

Definición en la línea 1205 del archivo mps-anneal.c.

B.1.2.3. int decide_perturbation (double *old_value*, double *new_value*, double *temp*, int *proc*, int *iter*, int *num_procs*, float * *randarray*)

Decidir si se acepta o no una perturbación.

Parámetros:

old_value Valor de la función objetivo antes de la perturbación

new_value Valor de la función objetivo despues de la perturbación

temp Temperatura

proc Id del proceso

iter Número de iteración

num_procs Número de procesos

randarray Arreglo con números aleatorios entre 0 y 1

La regla de decisión es la siguiente: $P(\text{aceptar estado } j \text{ desde estado } i) = \exp\left(-\frac{O_j - O_i}{T}\right)$ si $O_j < O_i$ ó $P(\text{aceptar estado } j \text{ desde estado } i) = 0$ si $_j \geq O_i$. Se utiliza el camino aleatorio de números entre 0 y 1 almacenado en randarray.

Definición en la línea 1745 del archivo mps-anneal.c.

Gráfico de llamadas a esta función

B.1.2.4. double evaluate_realization (map< string, int > & *hash*, map< string, int > & *hashrealization*, float *prom*)

Evaluar la función objetivo en un estado.

Parámetros:

hash Map con histograma de frecuencias de imagen de entrenamiento

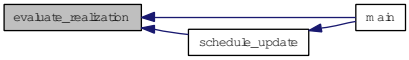
hashrealization Map con histograma de frecuencias de realización

prom Valor promedio de frecuencias inversas para imagen de entrenamiento

Actualmente se encuentra implementada la siguiente función objetivo: $\frac{\frac{1}{f_i^{TI}}}{\sum_{j \in \mathcal{P}} \frac{1}{f_j^{TI}}} \| f_i^{TI} - f_i^{RE} \|^2$. Para ver más detalles sobre la inclusión de una nueva función objetivo, revisar el trabajo de título, capítulo Implementación.

Definición en la línea 1578 del archivo mps-anneal.c.

Gráfico de llamadas a esta función



B.1.2.5. void evaluate_template (int *despx*, int *despy*, int *despz*, block *b*, int *numPattern*, geometry * *tem*, geometry * *reservoir*, unsigned int *HASH_SIZE*, map< string, int > & *hash*, ofstream & *file*)

Evaluar los patrones que aparecen asociados a un punto en la grilla, para un template dado, guardando las frecuencias en un map.

Parámetros:

despx Desplazamiento del template en el eje x con respecto al punto soporte b

despy Desplazamiento del template en el eje x con respecto al punto soporte b

despz Desplazamiento del template en el eje x con respecto al punto soporte b

b Punto soporte en la grilla.

numPattern Variable de uso interno

tem Template utilizado

reservoir Grilla 3D

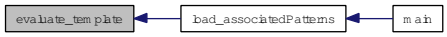
HASH_SIZE Tamaño máximo que el map puede alcanzar. Uso interno

hash En esta estructura se almacenan y actualizan las frecuencias.

file Stream del archivo donde se respaldan las estadísticas en caso que se alcance el tamaño máximo del map. Uso interno

Definición en la línea 1381 del archivo mps-anneal.c.

Gráfico de llamadas a esta función



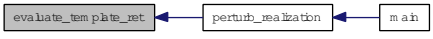
B.1.2.6. string evaluate_template_ret (int *despx*, int *despy*, int *despz*, int * *coord*, int *numPattern*, geometry * *tem*, geometry * *reservoir*, map< string, int > & *hash*)

Función análoga a evaluate_template() pero que retorna un string.

El string retornado puede ser "OUT" u otro patrón distinto. Si es "OUT", el patrón detectado no es ingresado en las estadísticas, pues utiliza puntos fuera de la grilla.

Definición en la línea 1456 del archivo mps-anneal.c.

Gráfico de llamadas a esta función



B.1.2.7. long filesize (char * *filename*)

Calcula el largo en filas de un archivo.

Parámetros:

filename

Definición en la línea 951 del archivo mps-anneal.c.

B.1.2.8. void free_reservoir (geometry * *reservoir*)

Liberar memoria utilizada para almacenar la grilla.

Parámetros:

reservoir

Definición en la línea 1292 del archivo mps-anneal.c.

Gráfico de llamadas a esta función

B.1.2.9. void free_template (geometry * *tem*)

Liberar la memoria utilizada para almacenar el template.

Parámetros:

tem

Definición en la línea 1131 del archivo mps-anneal.c.

Gráfico de llamadas a esta función

B.1.2.10. void generate_reservoir (geometry * *reservoir*, double *proportion*)

Generar grilla aleatoria.

Parámetros:

reservoir

proportion

Definición en la línea 1256 del archivo mps-anneal.c.

B.1.2.11. int level (int *my_id*)

Calcula el nivel dentro del árbol binario de un proceso.

Parámetros:

my_id Id del proceso.

El proceso 0 está en el nivel 1. Los procesos 1 y 2 están en el nivel 2. Los procesos 3, 4, 5 y 6 están en el nivel 3. Asi sucesivamente.

Definición en la línea 936 del archivo mps-anneal.c.

Gráfico de llamadas a esta función

B.1.2.12. void load_associatedPatterns (geometry * *tem*, geometry * *reservoir*, unsigned int *HASH_SIZE*, map< string, int > & *hash*, ofstream & *file*)

Cargar patrones asociados a un template, desde una grilla hacia un map, guardando las frecuencias de aparición.

Parámetros:

tem Template utilizado

reservoir Grilla 3D

HASH_SIZE Tamaño máximo del map

hash En esta estructura se almacenan las frecuencias detectadas.

file Stream del archivo donde se guardarán las estadísticas (respaldo)

La manera en que se recorre la grilla corresponde los puntos que satisfacen $i = 0 \pmod{tem.size.x}$ $j = 0 \pmod{tem.size.y}$ $k = 0 \pmod{tem.size.z}$ Para cada uno de esos puntos, se utiliza la función evaluate_template() para revisar todos los patrones asociados que utilizan como soporte al punto. Se deben resetear las posiciones de la ventana para cada nuevo punto, para ello se utiliza la función reset_template().

Definición en la línea 1326 del archivo mps-anneal.c.

Gráfico de llamadas para esta función

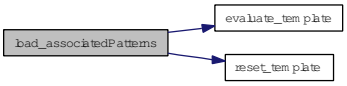


Gráfico de llamadas a esta función

B.1.2.13. void load_random (char * *filename*, float * *randomarray*)

Cargar camino aleatorio de números entre 0 y 1. Se utiliza para realizar una simulación con un camino predeterminado.

Parámetros:

filename Archivo donde están guardados los números aleatorios.

randomarray Arreglo donde se almacenarán en memoria.

Definición en la línea 1048 del archivo mps-anneal.c.

Gráfico de llamadas a esta función

B.1.2.14. void load_random_3cols (char * *filename*, int * *rand1*, int * *rand2*, int * *rand3*)

Cargar camino aleatorio de puntos 3D en la grilla de datos. Se utiliza para realizar una simulación con un camino predeterminado.

Parámetros:

filename Archivo donde están guardados los puntos aleatorios de la grilla.

rand1 Arreglo donde se almacenarán en memoria las primeras coordenadas.

rand2 Arreglo donde se almacenarán en memoria las primeras coordenadas.

rand3 Arreglo donde se almacenarán en memoria las primeras coordenadas.

Definición en la línea 1071 del archivo mps-anneal.c.

Gráfico de llamadas a esta función

B.1.2.15. void load_reservoir (geometry * *reservoir*, geometry * *tem*, char * *filename*, double *proportion*)

Cargar la grilla con los datos con dimensiones entregadas como parámetros en una estructura geometry.

Parámetros:

reservoir Grilla 3D

tem Template utilizado

filename Nombre del archivo con los datos de la grilla

proportion Umbral para los datos. Número entre 0 y 1, si data<proportion entonces WHITE, de lo contrario BLACK

El formato del archivo de entrada debe ser igual al utilizado por pixelplt (ver GSLIB) para leer datos, pero sin las primeras líneas donde se escribe el título, y algunos datos adicionales. Solo deben ir los datos, en una sola columna.

Definición en la línea 1157 del archivo mps-anneal.c.

Gráfico de llamadas a esta función

B.1.2.16. void load_schedule (double * *latestobj*, int * *attempt*, double * *T*, int * *tred*, char * *filename*)

Carga el schedule desde un archivo, guardando los datos en variables.

Parámetros:

latestobj Revisar schedule_update().

attempt Revisar schedule_update().

T Revisar schedule_update().

tred Revisar schedule_update().

filename Archivo que guarda el schedule asociado.

Definición en la línea 905 del archivo mps-anneal.c.

Gráfico de llamadas a esta función

B.1.2.17. void load_template (geometry * *tem*)

Cargar template con dimensiones entregadas como parámetros.

Parámetros:

tem

Definición en la línea 1093 del archivo mps-anneal.c.

Gráfico de llamadas a esta función

B.1.2.18. void load_topology (char * *filename*, int * *index*, int * *edges*)

Carga la topología del árbol binario.

Parámetros:

filename Archivo que contiene la especificación de la topología.

index Arreglo con el número de vecinos de cada nodo, de forma acumulativa.

edges Arreglo con los vecinos de los nodos.

El formato del archivo que contiene la especificación de la topología es de la siguiente forma:

```
3
4
2  //n\'umero de vecinos del nodo 0
3  //n\'umero de vecinos de los nodos 0 y 1
4  //n\'umero de vecinos de los nodos 0,1 y 2
1  //primer vecino del nodo 0
2  //segundo vecino del nodo 0
0  //primer vecino del nodo 1
0  //primer vecino del nodo 2
```

Este ejemplo corresponde a la topología de 3 procesos. El nodo 0 esta conectado con los nodos 1 y 2 y a su vez, los nodos 1 y 2 estan conectados con el nodo 0. La primera línea indica el largo del arreglo index. En este caso 3. La segunda línea indica el largo del arreglo edges. Desde la línea 3 hasta la línea 3+(index-1), se indica el número de vecinos. Desde la línea 3+index hasta la línea 3+index+(edges-1), se indican los vecinos de cada nodo. Para ver más ejemplos, revisar el directorio resources.

Definición en la línea 1012 del archivo mps-anneal.c.

Gráfico de llamadas a esta función

B.1.2.19. int * load_topology_dims (char * *filename*)

Carga las dimensiones del árbol binario que define la topología.

Parámetros:

filename Revisar load_topology().

Definición en la línea 968 del archivo mps-anneal.c.

Gráfico de llamadas a esta función

B.1.2.20. int main (int *argc*, char ** *argv*)

Función principal. Realiza la lectura de los argumentos y la iteración principal del Recocido Simulado en Paralelo.

Parámetros:

argc Número de argumentos. Deben ser exactamente 23.

argv Los argumentos son los siguientes:

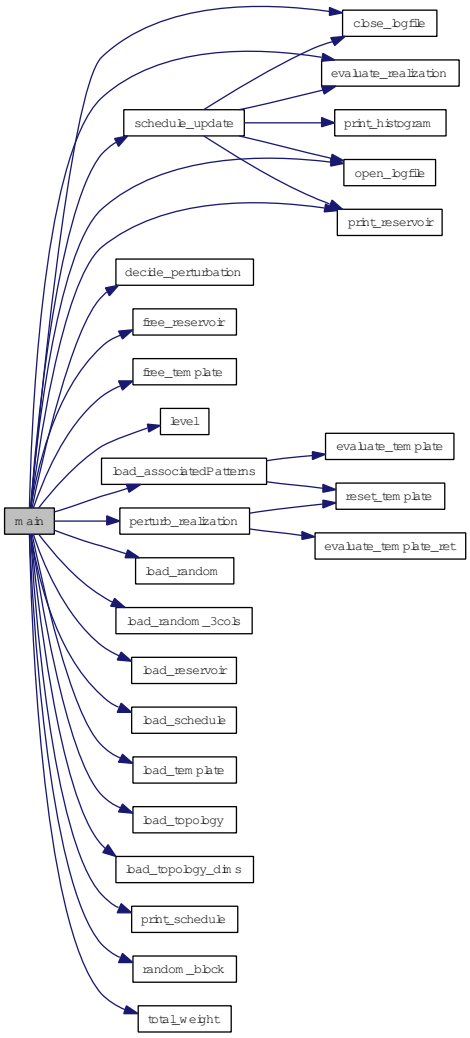
- Archivo con grilla asociada a imagen de entrenamiento.
- Archivo con grilla asociada a una realización (inicial).
- Archivo donde se guardará el histograma de frecuencias asociadas a un template, encontradas en la imagen de entrenamiento, en formato "columna-1,columna-2", con columna-1 el id del patrón y columna-2 la frecuencia (entero>=0).
- Archivo donde se guardará el histograma de frecuencias asociadas a un template, encontradas en la realización, en formato "columna-1,columna-2", con columna-1 el id del patrón y columna-2 la frecuencia (entero>=0).
- Archivo donde se especifica la topología del árbol binario que se utilizará para la paralelización.
- Archivo donde se guardara la grilla asociada a la realización final.
- Archivo donde se guardará el schedule utilizado en cada iteracion. Sólo se utiliza cuando hay varios procesos corriendo.
- Archivo con puntos aleatorios de la grilla. Se utiliza para tener un camino predeterminado de perturbaciones.
- Archivo con valores aleatorios entre 0 y 1. Se utiliza para tener un camino predeterminado de números aleatorios común a todos los procesos.
- Dimensión x de la grilla.
- Dimensión y de la grilla.
- Dimensión z de la grilla.
- Dimensión x del template.
- Dimensión y del template.
- Dimensión z del template.
- Tamaño máximo que puede alcanzar el map donde se almacenan las frecuencias. No se utiliza, pero se implementó por posibles usos futuros.

- Tamaño máximo del buffer que almacenara los id's de los patrones asociados a un template.
- Temperatura inicial.
- Factor de reducción lambda.
- Número máximo de perturbaciones.
- Número máximo de repeticiones.
- Número máximo de intentos, sin éxito, para reducir la función objetivo. Cuando se alcanza el máximo, se reduce la temperatura.
- Número de iteraciones que deben ocurrir para realizar un reporte del estado.

En esta función se implementa el Recocido Simulado a modo de prueba, para chequear la efectividad del speedup teórico $\log_2(P+1)$. Puede ser ejecutado con P procesos, donde P debe satisfacer que $\log_2(P+1)$ es un número natural. Esta condicion se debe a que sólo se consideraron árboles binarios para la Computación Especulativa.

Definición en la línea 222 del archivo mps-anneal.c.

Gráfico de llamadas para esta función



B.1.2.21. void open_logfile (ofstream & *file*, char * *logfilename*)

Abrir archivo log, donde quedará registrado el histograma de frecuencias en cada actualización.

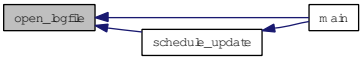
Parámetros:

file

logfilename

Definición en la línea 1505 del archivo mps-anneal.c.

Gráfico de llamadas a esta función



B.1.2.22. void perturb_realization (geometry * *tem*, geometry * *reservoir*, int * *coord*, map< string, int > & *hashrealization*, int *my_id*, int *iterout*)

Perturbar un estado, cambiando de color un punto de la grilla, tras lo cual se actualiza el map asociado a la grilla, donde se guarda el histograma de frecuencias.

Parámetros:

tem Template utilizado

reservoir Grilla 3D

coord Coordenadas de un punto en la grilla 3D

hashrealization Map que contiene las estadísiticas, las cuales se actualizarán

my_id Id del proceso

iterout Número de iteración

La perturbación afecta solamente a los patrones asociados al punto perturbado. Para ello, se utilza la función evaluate_template() para obtener los patrones asociados, luego se revisa el map para ver si existen o no, y se reemplazan por los nuevos patrones obtenidos. Primero se resta 1 a la frecuencia de los patrones antes de la perturbación y luego se suma 1 a la frecuencia de los patrones después de la perturbación.

Definición en la línea 1629 del archivo mps-anneal.c.

Gráfico de llamadas para esta función

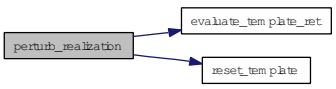
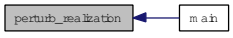


Gráfico de llamadas a esta función



B.1.2.23. void print_histogram (ofstream & *file*, map< string, int > & *hash*)

Imprimir el histograma de frecuencias almacenado en un map en un archivo.

Parámetros:

file

hash

Definición en la línea 1443 del archivo mps-anneal.c.

Gráfico de llamadas a esta función

B.1.2.24. void print_reservoir (geometry * *reservoir*, char * *filename*)

Imprimir una grilla en un archivo.

Parámetros:

reservoir Grilla 3D

filename Nombre del archivo donde se guardará la grilla

El formato del archivo de salida es compatible con el programa pixelplt, perteneciente a GSLIB.

Definición en la línea 1230 del archivo mps-anneal.c.

Gráfico de llamadas a esta función

B.1.2.25. void print_schedule (double *latestobj*, int *attempt*, double *T*, int *tred*, char * *filename*)

Imprime el schedule actual.

Parámetros:

latestobj Revisar schedule_update().

attempt Revisar schedule_update().

T Revisar schedule_update().

tred Revisar schedule_update().

filename Archivo donde se guardará el schedule.

Definición en la línea 886 del archivo mps-anneal.c.

Gráfico de llamadas a esta función

B.1.2.26. void print_to_log (ofstream & *file*, char * *line*)

Imprimir una línea en el archivo log.

Parámetros:

file

line

Definición en la línea 1515 del archivo mps-anneal.c.

B.1.2.27. int * random_block (geometry * *reservoir*, int *proc*, int *iter*, int * *rand1*, int * *rand2*, int * *rand3*)

Calcular un bloque o punto aleatorio en la grilla utilizando el camino aleatorio especificado por rand1, rand2 y rand3.

Parámetros:

reservoir

proc

iter

rand1

rand2

rand3

Definición en la línea 1555 del archivo mps-anneal.c.

Gráfico de llamadas a esta función

B.1.2.28. void reset_template (geometry * *tem*)

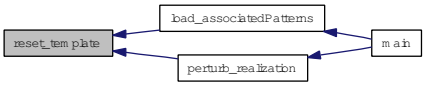
Resetear posiciones del template.

Parámetros:

tem

Definición en la línea 1115 del archivo mps-anneal.c.

Gráfico de llamadas a esta función



B.1.2.29. void schedule_update (int *iter*, int *irepo*, double *startobj*, double * *latestobj*, int * *attempt*, int *maxatt*, double * *T*, double *lambda*, int * *tred*, int *ntemp*, int *maxrepetitions*, int * *repcounter*, map< string, int > & *hash*, map< string, int > & *hashrealization*, geometry * *realization*, char * *outfile*, ofstream & *file*, char * *logfilename*, float *prom*)

Realiza la actualización del schedule, actualizando la temperatura, el número de intentos, el número de repeticiones y el factor de reducción lambda.

Parámetros:

iter Número de iteracion

irepo Revisar main().

startobj Valor Inicial de la Función Objetivo

latestobj Ultimo valor de la Función Objetivo.

attempt Número de veces que se ha reportado un movimiento hacia un estado con Función Objetivo mayor.

maxatt Revisar main().

T Temperatura.

lambda Factor de reducción.

tred Número de veces que se ha alcanzado el valor maxrepetitions. Si se alcanza ntemp veces, la simulación se detiene.

ntemp Revisar main().

maxrepetitions Número máximo de repeticiones en el valor de la función objetivo. Cuando se alcanza el máximo se reduce la temperatura.

repcounter Número de repeticiones.

hash Map con histograma de Imagen de Entrenamiento.

hashrealization Map con histograma de Realización.

realization Grilla con datos de la realización.

outfile Archivo donde se guarda la grilla asociada a la realización, si se ha detectado una mejora en la función objetivo. Sólo se utiliza para generar una animación de la evolución de la simulación.

file Stream donde se guarda el histograma de la realización.

logfilename Archivo donde se guarda el histograma de la realización.

prom Suma de las frecuencias inversas encontradas en la imagen de entrenamiento. Se utiliza cuando la función objetivo necesita calcular pesos.

Esta función realiza la actualización del schedule. Cuando se utilizan varios procesos, solamente el proceso 0 tiene acceso a esta función. Los demás procesos utilizan load_schedule(). El valor irepo representa las iteraciones donde se realizará un reporte, es decir, una llamada a esta función. Si se desean comparar los tiempos de ejecución con N y M procesos, la variable irepo debe ser divisible por $log_2(N+1)$ y $log_2(M+1)$ y lo más cercana posible al valor original ingresado por el usuario. Por ejemplo, para 1,3 y 7 procesos, con irepo=10000 ingresado por el usuario, se escoje irepo=9996. De esa manera se tiene un irepo igual para distintos números de procesos, y es posible comparar tiempos de ejecución.

Definición en la línea 830 del archivo mps-anneal.c.

Gráfico de llamadas para esta función

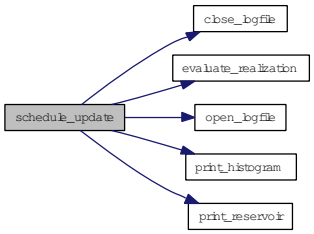


Gráfico de llamadas a esta función

B.1.2.30. float total_weight (map< string, int > & *hash*, float *c*)

Peso total de los inversos de las frecuencias. Se utiliza cuando se setea la función objetivo con Pesos.

Parámetros:

hash

c En la variable c se almacena el siguiente resultado $\sum_{i \in \mathcal{P}} \frac{1}{f_i^{TI}}$

Definición en la línea 1537 del archivo mps-anneal.c.

Gráfico de llamadas a esta función

Bibliografía

[Bur85] F. W. Burton. Speculative computation, parallelism, and functional programming. *IEEE Trans. Computers*, 34(12):1190–1193, 1985. 19

[db4] Db4o. http://www.db4o.com/. 12

[Deu92] C.V. Deutsch. *ANNEALING TECHNIQUES APPLIED TO RESERVOIR MODELING AND THE INTEGRATION OF GEOLOGICAL AND ENGINEERING (WELL TEST) DATA*. PhD thesis, stanford university, 1992. 7

[Deu02] C. V. Deutsch. *Geostatistical Reservoir Modeling*. Oxford University Press, 2002. 2, 5, 7

[DJ92] C. V. Deutsch and A. G. Journel. *GSLIB Geostatistical Software Library and User's Guide*. Oxford, England, 1992. 5, 27

[Goo97] P. Goovaerts. *Geostatistics for Natural Resources Evaluation*. Oxford University Press New York, 1997. 2, 5, 7

[Hay88] B. Hayek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13:311–328, 1988. 17

[HMS+98] J. M. D. Hill, B. McColl, D. C. Stefanescu, M. W. Goudreau, K. Lang, S. B. Rao, T. Suel, T. Tsantilas, and R. H. Bisseling. BSPlib: The BSP programming library. *Parallel Computing*, 24(14):1947–1980, 1998. 26

[KGV83] S. Kirkpatrick, C. D. Gelatt, and Jr. M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598), 1983. 6, 7, 13

[Kri51] DG Krige. A statistical approach to some mine valuations and allied problems at the Witwatersrand. *Master's thesis, University of Witwatersrand*, 1951. 1

[Mar05] S. Martínez. *Apuntes del curso Procesos de Markov*. 2005. 15

[Mat62] G. Matheron. Traite de Geostatistique Appliquee, Tome I. *Memoires du Bureau de Recherches Geologiques et Minieres*, 14, 1962. 1

[Mat63] G. Matheron. Traite de geostatistique appliquee, tome II. *Editions Techniques, Paris*, 1963. 1

[OD04] J. Ortiz and C. V. Deutsch. Indicator simulation accounting for multiple-point statistics. *Mathematical Geology*, 36(5), 2004. 7

[ope] Open mpi. http://www.open-mpi.org. 26

[Ort07] J. Ortiz. *Apuntes del curso Evaluación de Yacimientos*. 2007. 2

[Pac96] P. S. Pacheco. *Parallel programming with MPI*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996. 20, 21, 26

[pos] Postgresql. http://www.postgresql.org/. 11

[PVM] Pvm. http://www.csm.ornl.gov/pvm/pvm_home.html. 26

[RRD90] P. Roussel-Ragot and G. Dreyfus. A problem independent parallel implementation of simulated annealing: models and experiments. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 9(8):827–835, 1990. 19

[Sic52] H. S. Sichel. New methods in the statistical evaluation of mine sampling data. *Transactions of the Institution for Mining and Metallurgy*, March 1951-1952. 1

[Str02] S. Strebelle. Conditional simulation of complex geological structures using multiple-point statistics. *Mathematical Geology*, 34(1), 2002. 7

[vLA87] P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. Reidel Publishing Company, 1987. 15

[WCF91] E. E. Witte, R. D. Chamberlain, and M. A. Franklin. Parallel simulated annealing using speculative computation. *IEEE Transactions on Parallel and Distributed Systems*, 2(4), 1991. 21, 23

Índice alfabético