Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Proyecto de Grado

Algoritmos evolutivos multiobjetivo para despacho de tareas en redes heterogéneas

Darío De León

2008

Supervisor: Sergio Nesmachnow

Contenido

Lista de Algoritmos

Capítulo 1. Introducción

La principal virtud que ofrece la utilización de la computación heterogénea es la posibilidad de contar con un entorno de maquinas (entorno HC) con distintas capacidades de computo, esto es maquinas que están mejor preparadas para ejecutar tareas con cierto tipo de paralelismo. Es importante lograr que las tareas se asignen a las maquinas de una manera adecuada, para medir esto se pueden utilizar distintas métricas.

El problema de despacho de tareas en redes heterogéneas consiste en determinar la asignación de las tareas a las maquinas presentes en el entorno HC, de modo de optimizar alguna métrica. En la mayoría de los casos estudiados en la literatura se proponen varias métricas a optimizar, por lo tanto nos encontramos ante un problema de optimización multiobjetivo. En el caso de los problemas de optimización multiobjetivo se busca encontrar un conjunto de soluciones que logren balancear los criterios de optimización que se esta intentando optimizar. Luego de obtener el conjunto de soluciones óptimas se puede elegir la solución más adecuada según la prioridad que se le establezcan a los objetivos. Es muy poco frecuente obtener una única solución que optimice a todas las funciones objetivo a la vez.

El problema de despacho de tareas en redes heterogéneas en un problema NP-difícil, por lo tanto no es posible resolverlo utilizando técnicas deterministas, como por ejemplo backtracking. En estos casos surge la necesidad de utilizar técnicas que obtengan buenas soluciones rápidamente, aunque las mismas no sean las soluciones óptimas globales. Estas técnicas son conocidas como heurísticas. Dentro de las que se encuentran una familia de heurísticas llamadas algoritmos evolutivos, que están basadas en la teoría de la evolución, que operan sobre un conjunto de soluciones, que es evolucionado a partir de la aplicación de distintos operadores evolutivos, hasta que se cumple una condición de parada.

El paralelismo es una forma de resolver problemas con necesidades de cómputo muy grandes, cuando no se cuenta con un procesador con el suficiente poder de cómputo para resolver dicho problema en un tiempo razonable. Esta técnica consiste en hacer que varios procesos se ejecuten en distintos procesadores de forma colaborativa, para resolver un problema común. El funcionamiento de los algoritmos evolutivos permite que sean fácilmente paralelizables. La aplicación del paralelismo a la computación evolutiva hace que se puedan resolver problemas de gran complejidad utilizando un conjunto de equipos con poder de cómputo relativamente pequeño.

El presente documento resume la investigación realizada acerca de la computación heterogénea, sus características y aplicaciones a la resolución de problemas, así como los aspectos relacionados a las distintas versiones del problema de despacho de tareas en redes heterogéneas propuestas en la literatura, en especial acerca de la versión de dicho problema abordada en este trabajo. Por otra parte se resume la investigación realizada acerca de los algoritmos evolutivos como técnica de resolución de problemas, siendo los algoritmos evolutivos paralelos y los algoritmos evolutivos multiobjetivo técnicas fundamentales de este trabajo. Como producto de este trabajo se detalla la implementación de algunos algoritmos evolutivos multiobjetivos paralelos que buscan resolver el problema de asignación de tareas a procesadores de manera optima.

El resto del documento se organiza del modo que se describe a continuación: el capítulo 2 introduce los aspectos relacionados con la computación heterogénea, que sirven como base para el resto del documento. En el capítulo 3 se explican los conceptos relacionados con la computación evolutiva, el paralelismo y su

aplicación a la resolución de problemas de optimización multiobjetivo, estos conceptos son la base de la implementación propuesta. En el capítulo 4 se presenta la propuesta de resolución del problema estudiado y su implementación, a su vez se describe las bibliotecas utilizadas como base de la implementación. En el capítulo 5 se presentan los resultados experimentales obtenidos, y por ultimo en el capitulo 6 se presentan las conclusiones acerca del trabajo.

Capítulo 2. Computación heterogénea

La computación heterogénea se trata de la utilización de un conjunto de computadores con distintas capacidades de cómputo para ejecutar un conjunto de tareas que presentan diferentes necesidades de cómputo. A lo largo del capitulo se presentarán diferentes aspectos relacionados con la computación heterogénea.

En primer lugar se presentan conceptos generales relacionados con la computación heterogénea, como son: las ventajas de su utilización, los tipos de computaciones heterogéneas existentes, los perfiles de tareas que se pueden ejecutar, las técnicas de predicción de tiempos de ejecución de las tareas, entre otros aspectos.

También se exponen algunos ejemplos de la utilización de la computación heterogénea para resolver problemas complejos. Por último se describe un modelo que se utiliza comúnmente para modelar los diferentes entornos HC y se presentan algunos trabajos realizados en el área. Estos trabajos incluyen: propuestas de heurísticas para la resolución del problema de asignación de tareas a maquinas en entornos HC, comparación del desempeño de distintas heurísticas, modelos de estimación de de tiempos de ejecución de tareas en maquinas, entre otros temas.

El contenido de este capitulo esta basado fundamentalmente en algunos artículos publicados entre los que se encuentran [1, 2, 3, 4, 5, 36].

2.1 Conceptos de computación heterogénea

Antes de hablar de computación heterogénea es necesario definir algunos conceptos. Una meta-tarea es una colección de tareas independientes con distintas necesidades computacionales, además estas tareas se pueden descomponer en subtareas donde las mismas también tengan distintas necesidades computacionales. Por otro lado llamamos entorno HC a un conjunto de maquinas de alto rendimiento, donde las mismas pueden tener distintas capacidades de cómputo. La computación heterogénea se trata de la utilización de un entorno HC para ejecutar una meta-tarea. Un entorno HC es particularmente bueno para ejecutar un conjunto de tareas en el que las mismas necesitan distintos tipos de paralelismo (por ejemplo: MIMD, SIMD, etc.), ya que se pueden asignar las distintas tareas a la maquina que mejor se adapte a su ejecución. Es necesario aclarar que un entorno HC consiste, además de un conjunto heterogéneo de maquinas, en un conjunto heterogéneo de distintos elementos relacionados con su funcionamiento, como son: redes de alta velocidad, interfaces, sistemas operativos, protocolos de comunicación, entornos de programación, etc.

Uno de los mayores retos de la computación de alto rendimiento es que las arquitecturas de máquina que son cada vez más avanzadas puedan obtener un mayor rendimiento. Sólo una pequeña fracción de este rendimiento se logra en muchos conjuntos de aplicaciones reales, esto se debe a que una meta-tarea típica puede tener diferentes tareas y/o subtareas con diferentes requisitos computacionales. Cuando dicha meta-tarea se ejecuta en una determinada máquina, la máquina pasa la mayor parte de su tiempo ejecutando tareas (o subtareas) que no son las más adecuadas para su arquitectura. Con los avances en las comunicaciones digitales de alta velocidad, se ha hecho posible la utilización de diferentes

colecciones de máquinas de alto rendimiento en coordinación para resolver tareas de meta-tareas computacionalmente intensivas.

La forma de realizar la programación en un entorno HC es distinta a la forma de realizar la programación en una maquina local. El programador local generalmente solo maneja un único sitio o un cluster y por lo general posee los recursos, en cambio el programador del entorno HC tiene que descubrir los recursos, planificar la ejecución de las tareas en el grid, y manejar la ejecución de las tareas.

Desde la perspectiva de usuario un entorno HC es un entorno colaborativo para resolver problemas donde una o mas tareas pueden ser enviadas, sin que a este le interese si los recursos están o a quien le pertenecen dichos recursos. Con el fin de construir un entorno HC, es fundamental contar con un sistema de software común al entorno (o sea un middleware grid).

Dado un conjunto de aplicaciones grid (meta-tarea) debemos determinar como planificarlas en los distintos recursos descentralizados con que cuenta el entorno HC. Estas aplicaciones como ya se mencionó anteriormente tienen un conjunto de tareas, y estas tareas son las que se tienen que mapear a las maquinas del entorno. Para esto hay varios aspectos a considerar como son, por ejemplo: las relaciones de dependencia que existen entre las distintas tareas y como pueden afectar a las decisiones de planificación, la heterogeneidad de los recursos y como esto afecta a la planificación, y los modelos de rendimiento que va a utilizar el planificador para determinar la calidad de la planificación.

Las tareas pueden depender unas de otras, lo que hace más difícil la planificación. Cabe destacar que el mayor esfuerzo en la investigación de la planificación de tareas se ha ocupado de la planificación de tareas independientes, o planificación de tareas débilmente dependientes. Estos enfoques usan métodos analíticos para hacer una planificación deterministica o usan un análisis de datos empíricos y heurísticas de búsqueda para encontrar una buena solución.

Si consideramos el problema de la planificación local vemos que los principales objetivos al momento de realizar la planificación de las tareas dentro del entorno HC son: i) superar la heterogeneidad de los recursos informáticos, ii) maximizar el rendimiento general del sistema, logrando alto rendimiento y alta tasa de utilización de recursos, iii) el apoyo a diversas aplicaciones de computación intensiva, como trabajos en lotes y aplicaciones paralelas (MPI, PVM, etc), y iv) suministrar funcionalidades para una ejecución robusta de tareas remotas, tales como E/S remotas confiables y una eficiente administración de trabajos que impliquen manejo de prioridad, migración y checkpointing [4].

El resultado que se obtiene es un sistema centralizado para la gestión de los recursos que ofrece un entorno fiable de computación distribuida local y maximiza la utilización de los recursos de un sitio local. Su planificación de recursos suele ser un proceso de emparejamiento oportunista que requiere una cierta manera de especificar y expresar los requisitos de las aplicaciones y los recursos.

La planificación en un entorno HC es intrínsecamente más compleja que la planificación de los recursos locales ya que debe manipular la gestión de recursos a gran escala. En este entorno dinámico de computación distribuida, la disponibilidad de recursos varía dramáticamente. Por lo tanto, la planificación se convierte en muy difícil. Se ha realizado una amplia labor de investigación sobre problemas de planificación en sistemas distribuidos[4].

La investigación en el campo de la computación heterogénea está motivada por el hecho de que las máquinas de alto rendimiento pueden variar en sus capacidades y, por lo tanto, pueden variar su aptitud para los diferentes tipos de tareas y subtareas. Ejemplos de este tipo de máquinas son arquitecturas distribuidas de las grandes

máquinas con memoria compartida (por ejemplo, un SIG 2800), multiprocesadores con memoria distribuida (por ejemplo, un IBM SP2), y pequeñas máquinas de memoria compartida (por ejemplo, un Sun Enterprise 3000 Server). Por otra parte, dos implementaciones de un determinado tipo de máquina pueden variar en la velocidad de la CPU, tamaño y estructura de la memoria caché, ancho de banda de E/S, etc.

La aplicabilidad y la fuerza de los sistemas de HC derivan de su capacidad para mapear las distintas necesidades de las aplicaciones a los recursos apropiados. En un sistema HC general, se realiza una planificación que consiste en asignar tareas a máquinas (matcheo), y calcular el orden de ejecución de las tareas asignadas a cada máquina (planificación). El proceso de hacer el matcheo y la programación de las tareas se conoce como el mapeo.

Las heurísticas que se utilizan para realizar el mapeo pueden agruparse en dos categorías: heurísticas de modo inmediato (online) y heurísticas de modo tratamiento por lotes (batch). En el modo inmediato una tarea es mapeada a una maquina tan pronto como llega al entorno. En el modo por lotes las tareas no son mapeadas a las maquinas cuando estas llegan, sino que se recogen en un conjunto que se examina por la estrategia de mapeo en momentos preestablecidos llamados eventos de mapeo.

El objetivo de un entorno HC es asignar las tareas y subtareas a maquinas y planificar su ejecución de manera de optimizar alguna medida de performance. Esta medida puede ser tan simple como el tiempo de ejecución de la meta-tarea, o puede ser más compleja y ser una función matemática de varios factores tales como la ponderación de las prioridades de las tareas, tiempo de ejecución de las tareas, los requisitos de seguridad y necesidades de calidad de servicio (QoS).

En [2] se presentan varios ejemplos de entornos HC, entre los que se encuentran: SmartNet, NetSolve, PVM and Hence, Globus Metacomputing Infrastructure Toolkit.

2.1.1 Ventajas de los sistemas heterogéneos

Cuando se habla de sistemas homogéneos se hace referencia a un conjunto de maquinas de alto rendimiento, que tienen capacidades de computo similares. Los sistemas homogéneos convencionales suelen utilizar un modo de paralelismo en una determinada máquina (como SIMD, MIMD, o vector de transformación) y, por tanto, no puede satisfacer adecuadamente las necesidades de meta-tareas que requieren más de un tipo de paralelismo. Como resultado, cualquier tipo de máquina a menudo gasta mucho de su tiempo de ejecución en código al que no está adaptada. Por otro lado, muchas aplicaciones necesitan procesar la información en más de una categoría simultáneamente, con diferentes tipos de paralelismo en cada nivel.

Por ejemplo, en el nivel más bajo de visión del computador, las operaciones de procesamiento de imágenes se aplican a la imagen cruda. Estos cálculos tienen un elevado porcentaje de paralelismo del tipo SIMD. En contraste, la computación de la compresión de imágenes en alto nivel expone características de paralelismo del tipo MIMD de grano grueso [1]. Para este tipo de aplicaciones, los usuarios de un sistema multiprocesador convencional deben conformarse con un rendimiento degradado en el hardware existente o adquirir maquinas más potentes. Cada tipo de sistema homogéneo adolece de limitaciones inherentes. Por ejemplo, se pueden emplear máquinas vectoriales que intercalan memoria con una unidad aritmética

lógica pipelined, lo que hace un alto rendimiento con millones de operaciones punto flotante por segundo (Mflops), pero si la distribución de datos de una meta-tarea y los cálculos resultantes no pueden explotar estas características el rendimiento se degrada severamente.

Considerando la posibilidad de una aplicación de código mixto que tenga incorporados diferentes tipos de paralelismo. Supongamos que el código, al ejecutarse en una máquina de serial gasta 100 unidades de tiempo. Cuando este código se ejecuta en una máquina vectorial, la parte del código vectorial se ejecuta con rapidez, mientras que otras partes del código todavía tienen tiempos de ejecución relativamente más altos. Del mismo modo, el mismo código, al ejecutarse en un conjunto heterogéneo de máquinas (de modo que cada porción del código se ejecuta en una maquina que tenga el tipo de paralelismo adecuado para la ejecución de dicha porción de código) es probable que se pueda lograr un tiempo de ejecución menor.

La heterogeneidad en los sistemas de computación no es un concepto totalmente nuevo. Varios tipos de procesadores para fines especiales han sido utilizados para proporcionar servicios específicos para mejorar el rendimiento del sistema. Uno de los más comunes es el manejo de la entrada/salida. Agregar unidades de punto flotante para complementar a los ordenadores centrales no es más que otro enfoque heterogéneo para mejorar el rendimiento del sistema. En computadoras de alto rendimiento, el concepto de heterogeneidad se manifiesta a nivel de instrucción en forma de varios tipos de unidades funcionales, tales como tuberías de aritmética vectorial y procesadores escalares rápidos. Sin embargo, los sistemas multiprocesador actuales en su mayoría siguen siendo homogéneos en lo que respecta al tipo de paralelismo apoyado por ellos. Tales sistemas han sido tradicionalmente clasificados en función del número de instrucciones y flujos de datos.

Un entorno HC debe contar con los siguientes componentes: i) un conjunto heterogéneo de máquinas, ii) una red inteligente de alta velocidad conectando todas las maquinas, y iii) un entorno de programación fácil de usar.

HC permite a un sistema dado adaptarse a una amplia gama de aplicaciones aumentándolas con funcionales específicas o con capacidades de rendimiento, sin necesidad de un rediseño completo. Dado que HC está compuesto por varios ordenadores autónomos, en general la tolerancia de fallas del sistema y la longevidad pueden mejorar.

Como ya mencionamos los sistemas homogéneos tienen limitaciones para obtener una ejecución adecuada para una amplia gama de aplicaciones. Esto se debe a que los sistemas homogéneos utilizan un modo de paralelismo en una determinada maquina (SIMD, MIMD, etc.), de esta forma no pueden satisfacer las necesidades de las aplicaciones que requieren más de un tipo de paralelismo. Si una tarea necesita varios tipos de paralelismo y se ejecuta en un sistema homogéneo, la parte del código que corresponde al tipo de paralelismo de esa maquina se ejecutara rápidamente, pero el resto del código tendrá tiempos de ejecución grandes. Para solucionar este problema HC utiliza los sistemas existentes en un entorno integrado. Esto que permite adaptarse a una ampliar gama de aplicaciones, dando la posibilidad de que las tareas se ejecuten en una maquina que cuente con el tipo de paralelismo que la tarea requiera.

2.1.2 Tipos de sistemas heterogéneos

Principalmente hay dos tipos de entornos HC: i) metacomputing, y ii) mixed-modecomputing [1].

En metacomputing existe una heterogeneidad de grano grueso, las instrucciones que requieren una clase particular de paralelismo se agrupan en módulos, cada modulo se ejecuta en una maquina paralela adecuada. Metacomputing se refiere a la heterogeneidad a nivel de módulo.

En mixed-modecomputing existe una heterogeneidad de grano fino, Los programas que tienen este tipo de heterogeneidad no son adecuados para si ejecución en un conjunto heterogéneo de maquinas porque la sobrecarga de la comunicación debido a los frecuentes intercambios de información entre maquinas puede convertirse en un cuello de botella. Aquí existe heterogeneidad a nivel de instrucciones. Sin embargo, estos programas se pueden ejecutar de manera eficiente en una sola máquina, como PASM (Particionable SIMD/MIMD), que incorpora modos heterogéneos de computación. mixed-modecomputing se refiere a la heterogeneidad a nivel de instrucciones. Las maquinas de modo mixto pueden lograr grandes speedups para una heterogeneidad de grano fino utilizando el modo mixto de procesamiento disponible en una sola máquina. Una maquina de modo mixto, puede usar su capacidad de conmutación de modo para apoyar paralelismo SIMD/MIMD y superar la limitación de la sincronización del hardware, mejorando así su rendimiento de una máquina que opera solo en modo SIMD o MIMD.

2.1.3 Perfiles de tareas

Los perfiles de las tareas especifican los tipos de cálculos que están presentes en cada tarea, esto se logra descomponiendo el código de la meta-tarea en bloques de código homogéneos basados en sus necesidades computacionales. El análisis de los perfiles de las tareas proporcionan la información necesaria para el mapeo de las tareas en las distintas maquinas del entorno HC.

El conjunto de tipos de código definidos se basa en las características de las arquitecturas de máquina disponibles y los requisitos del procesamiento de la meta-tarea que se están considerando para la ejecución en el entorno HC. Este conjunto de tipos de código será una función de los códigos de las tareas de la meta-tarea, y los tipos y tamaños de los conjuntos de datos del proceso.

Por otro lado la evaluación comparativa proporciona una medida de qué tan bien cada una de las máquinas disponibles en el entorno HC realiza cada uno de los tipos de código. En combinación, los perfiles de tareas y evaluación comparativa analítica proporcionan la información necesaria para el paso del matcheo y la programación. El desempeño de un determinado tipo de código en un determinado tipo de máquina es una función multi- variable. Las variables que pueden influir en esta función son: los requisitos de la meta-tarea (por ejemplo, la precisión de los datos), el tamaño del conjunto de datos a ser procesados, el algoritmo que debe aplicarse, los esfuerzos del programador y compilador para optimizar el programa, y el sistema operativo y arquitectura de la máquina que ejecutará el tipo de código específico.

La teoría de selección es una colección de formulaciones matemáticas que se han propuesto para optar por la maquina más adecuada para cada bloque de código. Muchas formulaciones definen a la evaluación comparativa como método para

medir la aceleración óptima de un determinado tipo de máquina que ejecuta el tipo de código mejor matcheado a un sistema de referencia. La relación entre la aceleración actual y la aceleración óptima define que tan bien matcheado es un bloque de código con cada tipo de máquina. En general, la aceleración real es inferior a la aceleración óptima [2].

El sistema paralelo de ventanas de evaluación (PAWS, *parallel assessment windows system*) y el sistema de gestión de supercomputación distribuida heterogénea (DHSMS, *distributed heterogeneous supercomputing management system*) representan ejemplos preliminares de los marcos para la aplicación de perfiles y evaluación comparativa. El prototipo PAWS consta de cuatro herramientas: la herramienta de caracterización de aplicación (meta-tarea), la herramienta de caracterización de la arquitectura, la herramienta de evaluación del rendimiento, y la herramienta de la pantalla gráfica interactiva. En primer lugar, la herramienta de caracterización de la aplicación transforma un programa escrito en un subconjunto específico de Ada en un lenguaje gráfico acíclico que ilustra la dependencia de datos del programa. La herramienta utiliza conjuntos de nodos y aristas para describir el comportamiento de las distintas funciones y procedimientos de un programa. Sin embargo, esta herramienta no realiza la descomposición de las tareas basada en las necesidades computacionales y capacidades de la máquina.

Para las máquinas de referencia, la herramienta de caracterización de la arquitectura divide la arquitectura de un tipo específico de máquina en cuatro categorías: cálculo, movimiento de datos y comunicaciones, Entrada/Salida, y control. Cada categoría puede ser dividida en subsistemas. La herramienta de evaluación del rendimiento utiliza la información de la herramienta de caracterización de la arquitectura para generar información para las operaciones en una determinada máquina. Dos conjuntos de parámetros de rendimiento de una aplicación, los perfiles de paralelismo y los perfiles de ejecución, son generados por la herramienta de evaluación del rendimiento. Los perfiles de paralelismo describen límites superiores teóricos de rendimiento de las aplicaciones. Los perfiles de ejecución representan las estimaciones de rendimiento de las aplicaciones después de que se han dividido y se asigna a una máquina en particular. La herramienta de la pantalla gráfica interactiva es la interfaz de usuario para acceder a todas las demás herramientas en PAWS.

DHSMS clasifica los resultados de los perfiles de tareas y de la evaluación comparativa dentro de un marco sistemático. En primer lugar, DHSMS genera un conjunto universal de códigos (USC) para los perfiles de tareas. El USC se puede considerar como un conjunto normalizado de programas evaluación comparativa. Al igual que la información del hardware organizacional mantenida por la herramienta de caracterización arquitectónica en PAWS, un USC se construye con una estructura jerárquica basada en las máquinas en el entorno HC. En el nivel más alto de esta estructura jerárquica, los modos de paralelismo son seleccionados para especificar las arquitecturas de máquinas. En el segundo nivel, características arquitectónicas más finas, como la organización del sistema de memoria, se pueden elegir. Esta estructura jerárquica es organizada de tal manera que la características arquitectónicas de cualquier nivel, son las opciones para una categoría determinada (por ejemplo, el tipo de interconexión de redes se utiliza). DHSMS asigna un tipo de código (es decir, las características computacionales) para cada ruta desde la raíz de la estructura jerárquica a un nodo hoja. Cada uno de dichos camino representa un conjunto específico de características arquitectónicas, definido por los nodos en el camino.

2.1.4 Técnicas de predicción

Los recursos mas comúnmente solicitados por una aplicación grid son recursos de computación, datos y red. Es fácil obtener la información estática de recursos de las maquinas (velocidad de la CPU, memoria, etc.). Sin embargo los recursos en tiempo de ejecución (carga del CPU, memoria disponible, etc.) son variables debido a que pueden existir muchas aplicaciones corriendo.

En el momento de tomar las decisiones para la planificación, se necesita alguna técnica de predicción de los parámetros de los recursos en tiempo de ejecución, esto es fundamental para maximizar el rendimiento de las aplicaciones. En la investigación de la predicción, el análisis se basa en la información histórica sobre la disponibilidad de los recursos y registros del rendimiento de los trabajos. Se pueden aplicar diversas técnicas estadísticas. Si el comportamiento de los recursos sigue o se supone que sigue cierta distribución, el proceso de análisis estocástico se puede utilizar para predecir las futuras disponibilidades de recursos en un punto fijo del tiempo o durante un cierto intervalo de tiempo. La técnica de regresión puede ser usada para la predicción en presencia de un modelo de desempeño.

Hay dos maneras de predecir el tiempo de ejecución esperado de una aplicación, dependiendo si existe un modelo de performance o no. Si no existe un modelo de performance, se puede usar el análisis de datos empíricos. Usando este método, en primer lugar se utilizan registros de tiempo de ejecución anteriores de aplicaciones similares a través de algunos procedimientos de búsqueda basados en criterios de categorización de parámetros, tales como el rango de tiempos de ejecución admisibles. Entonces se puede calcular el promedio de tiempo de ejecución de aplicaciones anteriores dentro de un margen de error tolerable y utilizar esta estimación como predicción del tiempo de ejecución de la aplicación actual. La tasa de error se calcula utilizando alguna medida estadística, como el error absoluto, o al menos la raíz cuadrada de error. Si existe un modelo de performance, se puede predecir valores para los recursos que varían en tiempo de ejecución.

En la investigación acerca de programación en un entorno HC, la mayoría de planificadores obtienen los parámetros de predicción de recursos de las predicciones del NWS (Network Weather Service) [4]. NWS es un agente desplegado en el entorno HC para supervisar periódicamente los recursos y el rendimiento. Normalmente, los monitores NWS realizan mediciones del rendimiento en intervalos de 10 segundos.

Un ejemplo de uso del valor predicho por NWS, es el método de planificación estocástica propuesto por Schopf [4] para la planificación de los datos de entrada a los procesadores en tiempo de ejecución. Asume que w_i sigue una distribución normal, y w_i define como:

$$w_i = m_i + sd_i * TF$$

donde m_i es la media de tiempo de finalización previsto, sd_i es la desviación estándar del tiempo de finalización previsto, y TF es un factor de ajuste estadístico que indica el número de sd_i se pueden añadir a la predicción. El TF más grande es, la predicción más conservadora que puede ser considerado porque el planificador permitiría más variación en la disponibilidad de recursos. La introducción del factor TF permite al planificador controlar la calidad del modelo de desempeño para una aplicación específica.

Intuitivamente, se planificará una tarea en una máquina con mayor potencia y menor variabilidad. Entre la potencia y la variabilidad, planificador optara por asignar una tarea a una máquina con baja potencia y baja variabilidad, en lugar de una máquina con mayor potencia pero mayor variabilidad. La idea detrás de la

elección es que una máquina con una alta variabilidad tiene mayor posibilidad de no contar con los recursos, lo que afecta gravemente a la ejecución de tareas.

La principal limitación de esta predicción es la suposición de que w_i tiene una distribución normal. El comportamiento de w_i aún se desconoce, lo que hace el uso de una distribución normal con media y varianza sea cuestionable.

Schopf y Yang sostienen que un valor estimado no es suficiente para predecir la información de los recursos en tiempo de ejecución. Así que presentan el intervalo de predicción y el intervalo de variación de predicción como dos medidas más de predicción. En lugar de utilizar una sola estimación, el intervalo de predicción utiliza un intervalo estimado que incluye valores de puntos de predicción. Ellos usan el grado de agregación para decidir cuántos valores continuos de puntos utilizan, donde el grado de agregación = tiempo de ejecución de la aplicación * frecuencia de los valores de los puntos. El tiempo de ejecución de una aplicación es el promedio de los tiempos de ejecución de aplicaciones anteriores similares. Asimismo, la variación en el intervalo se calcula de manera similar. Una máquina con un bajo intervalo de variación se considera "fiable", por lo que un planificador asigna menos trabajo de alta varianza recursos.

Otras técnicas estadísticas (empíricas) de análisis, como las técnicas de regresión, se pueden aplicar a la predicción de los tiempos de ejecución de una meta-tarea. Estas técnicas son puramente basadas en datos históricos. Vazhdukai utiliza una técnica de regresión para predecir la transferencia de datos en entornos HC.

Los métodos de regresión son herramientas matemáticas que se utilizan a menudo para predecir el comportamiento de una variable (por ejemplo, tiempo de ejecución esperado), la variable dependiente, a partir de múltiples variables independientes (por ejemplo, la entrada, el número de procesadores a utilizar, y el estado de recursos). El principio subyacente en el método de regresión es minimizar la suma de la desviación cuadrada de los valores predichos de las observaciones actuales. El proceso de regresión calcula los coeficientes de regresión entre las variables independientes en primer lugar, entonces usa los coeficientes derivados para calcular la variables dependientes basado en un modelo de regresión especifico, como el modelo de regresión de lineal, modelo de regresión cuasi-lineal, o modelo de regresión polinómica. La investigación de los coeficientes también ayuda a comprender las relaciones entre las mediciones de recursos independientes sobre el comportamiento de las aplicaciones.

2.1.5 Mapeo

Encontrar un mapeo de las tareas que optimice el rendimiento de alguna medida de performance es, en general un problema NP-completo. Por lo tanto es necesario disponer de alguna heurística para encontrar asignaciones que estén cerca de ser óptimas sin utilizar una búsqueda exhaustiva. Hay varios factores que influyen en el mapeo dentro de los que encontramos: 1) matcheo de requerimientos de las tareas con las capacidades de las maquinas, 2) overhead en la comunicación de código y datos entre las maquinas, 3) carga esperada de las maquinas y congestión esperada de la red, y 4) restricciones de precedencia entre tareas.

Hay varios tipos de heurísticas para el mapeo. En una heurística de mapeo estático las decisiones se toman antes de que se ejecute la meta tarea, estas heurísticas se usan cuando: 1) Las tareas que conforman la meta-tarea se conocen a priori, 2) las predicciones acerca de la disponibilidad de recursos heterogéneos de computación es probable que sean precisas, y 3) la estimación de tiempos de ejecución que se espera de cada tarea en cada maquina, se conoce con una precisión razonable.

En una heurística de mapeo dinámico las decisiones se toman durante la ejecución de la meta tarea. Se utilizan cuando no se puede prever: 1) los tiempos de llegada de tareas, 2) las maquinas disponibles en el sistema de computación heterogéneos y 3) los tiempos de ejecución esperados de las tareas en las maquinas. Una heurística de mapeo dinámica solo tiene información acerca de las tareas que ya han llegado para su ejecución.

En [4] se menciona la planificación de tareas independientes. Esta procede de la siguiente manera: dado un conjunto de tareas independientes y un conjunto de los recursos disponibles, la planificación de tareas independientes intenta minimizar el tiempo total de ejecución de la tarea buscando un mapeo óptimo de tareas a las máquinas. Las métrica utilizada para encontrar este mapeo es el tiempo de finalización (tiempo en que la máquina este disponible + tiempo esperado de computo). Como mencionamos anteriormente, encontrar el mejor mapeo es un problema de optimización combinatoria, que en este caso es NP-completo. Para resolver el problema de la planificación de tareas independientes se han utilizado distintas heurísticas como Minimum Completion Time (MCT), Oportunistic Load Balancing (OLB), min-min, max-min; y técnicas metaheurísticas como algoritmos genéticos (GA), Simulated Annealing (SA), tabu search, etc.

A continuación se describe distintos problemas que se presentan en el momento de buscar un mapeo de tareas a maquinas.

Particionamiento: Se debe detectar el paralelismo de las tareas, además se requiere una clasificación del código basada en el tipo de paralelismo. Podemos dividir el problema del particionamiento en dos subproblemas. En primer lugar tenemos el subproblema de la detección del paralelismo, que consiste en determinar el paralelismo presente en un determinado programa. En segundo lugar tenemos el subproblema de la agrupación, que es combinar varias operaciones en un módulo de programa y, por tanto, la partición de la aplicación en varios módulos. Estos dos subproblemas pueden ser manipulados por el usuario, el compilador, o la máquina en tiempo de ejecución. En HC, la detección de paralelismo no es el único objetivo; la clasificación del código de basada en el tipo de paralelismo es también necesaria. Esto se logra por los perfiles de tipos de código, que también plantea nuevas restricciones a la agrupación.

Selección de maquinas: Un problema que aparece en el diseño de entornos HC es como se puede encontrar el conjunto heterogéneo de máquinas más adecuado para la ejecución, sujeta a alguna una limitación como el costo o el tiempo de ejecución, de una determinada meta-tarea. Una propuesta para elegir una configuración óptima de máquinas para la ejecución de una meta-tarea en un conjunto heterogéneo de computadoras es la teoría de selección óptima (OST), esta teoría supone que el número de máquinas disponibles es ilimitado. También se supone que las máquinas que machean al conjunto de tipos de código están disponibles y que el código de la meta-tarea se descompone en módulos de igual tamaño. Otra de las propuestas es la teoría de selección óptima aumentada (A0ST) donde se incorpora el rendimiento de los segmentos de código en opciones de máquinas no óptimas, asumiendo que el número de máquinas disponibles para cada tipo de código es limitado. En este enfoque, el módulo de programa más adecuado para un tipo de máquina es asignado a otro tipo de máquina. En la formulación de OST y AOST, se ha supuesto que la ejecución de todos los módulos de programa de una determinada meta-tarea están totalmente ordenados en el tiempo. Por otra parte, el paralelismo puede estar presente dentro de un módulo, lo que resulta en una mayor descomposición de los módulos del programa. Por otra parte, el efecto de diferentes asignaciones en diferentes máquinas disponibles para un módulo de programa no se ha considerado en la formulación de estas teorías de selección. Por otro lado, la teoría de selección óptima heterogénea (HOST) extiende AOST de dos

maneras. Incorpora el efecto de diferentes técnicas de mapeo disponibles en diferentes máquinas para la ejecución de un módulo de programa. Asimismo, las dependencias entre los módulos del programa se especifican como un gráfo dirigido. En la formulación de HOST, un código de la meta-tarea se supone que constará de tareas que se ejecutan en serie. Cada tarea contiene una colección de módulos de programa. Cada módulo del programa se descompone en bloques paralelos de instrucciones, llamado los bloques de código. Para encontrar un conjunto óptimo de máquinas, tenemos que asignar el programa de módulos a las máquinas a fin de que $\sum T^i$ sea mínimo. Mientras que $\sum C^i < C_{max,}$ donde T^i es el tiempo para ejecutar el modulo de programa i, C^i es el costo de la máquina en la que se ejecuta el modulo de programa i, y C_{max} es una limitación en el costo de las máquinas. El costo C^i y el tiempo de ejecución T^i correspondiente a la asignación en consideración pueden obtenerse mediante el uso de perfiles de tipo de código y/o mediante el análisis de los algoritmos. Por ultimo, tenemos la propuesta de Iqbal que presentó un esquema de selección que busca una asignación de los módulos del programa a las máquinas en el entorno HC para que el tiempo de procesamiento total se minimice, mientras que el costo total de las máquinas empleadas en la solución no exceda un límite superior. Este sistema es aplicable a todas las teorías de selección anteriores. La precisión del sistema, sin embargo, depende del método utilizado para asignar los módulos de programa a las máquinas.

Planificación: En entornos homogéneos, un planificador asigna cada módulo de programa a un procesador para lograr el rendimiento deseado en términos de utilización del procesador. Los diseñadores suelen emplear tres niveles de planificación. La planificación de nivel alto, también llamada planificación de trabajos, selecciona un subconjunto de todos los trabajos que están compitiendo por los recursos disponibles. La planificación de nivel intermedio responde a las fluctuaciones a corto plazo en el sistema de carga mediante una suspensión temporal y la activación de los procesos para lograr un buen funcionamiento del sistema. La planificación de bajo nivel determina el próximo proceso listo para ser asignado a un procesador por un cierto tiempo. Diferentes políticas de planificación, como FIFO, round-robin, tarea más corta primero (shortest-job-first), y más corta en tiempo restante (shortest-remaining-time), pueden ser empleadas en cada nivel de planificación. Si bien estos tres niveles de planificación puede residir en cada una de las máquinas en un entorno de HC, un cuarto nivel que se necesita para llevar a cabo la planificación a nivel del sistema. Este planificador mantiene la carga de trabajo equilibrada en todo el sistema mediante el monitoreo del progreso de todos los módulos de programa. Además, el programador necesita conocer los diferentes tipos de módulos y tipos de máquinas disponibles en el entorno, ya que los módulos pueden tener que ser reasignados cuando la configuración del sistema o los cambios producen situaciones de sobrecarga.

2.1.6 Entornos de programación

Un entorno de programación paralela debe incluir una serie de herramientas que permitan a los programadores aprovechar las cualidades que ofrece un entorno HC. Un entorno de programación paralela incluye lenguajes paralelos, compiladores inteligentes, depuradores paralelos, editores dirigidos por la sintaxis, configuración de las herramientas de gestión, y otras ayudas a la programación. En computación homogénea, los compiladores inteligentes detectan el paralelismo en el código secuencial y lo traducen en código máquina paralelo. Paralelamente lenguajes de programación se han desarrollado para apoyar la programación paralela, como

MPL para máquinas MasPar, y Lisp, y C para la máquina de conexión. Además, varios entornos de programación paralela y modelos han sido diseñados, como el Code, Faust, Schedule, y Linda.

HC requiere un lenguaje de programación y herramientas que sean independientes de la máquina y portables. Este requisito crea la necesidad de diseñar compiladores cross-paralelo para todas las máquinas del entorno, y depuradores paralelos cruzados para depurar el código de máquina. Varios modelos de programación y entornos se han desarrollado en el pasado para la computación heterogénea.

La Paralell Virtual Machine (PVM) se compone de software que proporciona un entorno virtual de computación concurrente en redes de propósito general de maquinas heterogéneas. Se compone de un conjunto de primitivas de interfaz de usuario y el software que permiten la computación concurrente en redes flojamente unidas de maquinas de alto rendimiento. Puede ser aplicado sobre una base de hardware que consta de diferentes arquitecturas, incluyendo un sistemas con un solo CPU, máquinas de vectoriales, y multiprocesadores.

Los programas de aplicación ven al sistema PVM como un recurso general y flexible de computación paralela que apoya la memoria compartida, pasaje de mensajes, y modelos híbridos de computación. Una aplicación heterogénea puede descomponerse en varias tareas basado en los tipos de cálculos embebidos y luego ejecutarse mediante el uso de subrutinas PVM en las diferentes máquinas disponibles en la red. Las primitivas PVM se presentan en forma de bibliotecas vinculadas a los programas de aplicación escritos en lenguajes imperativos. Apoyan proceso de apertura y gestión, pasaje de mensajes, la sincronización, otras facilidades.

2.1.7 Evaluación de desempeño

Para evaluar el desempeño de aplicaciones se utilizan herramientas de performance para resumir el comportamiento en tiempo de ejecución de una aplicación, incluyendo el análisis de la utilización de los recursos y la causa de cualquier cuello de botella que hacen bajar el rendimiento. Dependiendo de su diseño, una herramienta de rendimiento puede describir el comportamiento de un programa en muchos niveles de detalle. Los dos niveles más comunes son la intraprocesos e interprocesos. Las herramientas de rendimiento Intraprocesos, tales como: the gprof facility on BSD Unix, the HP sampler/3000, and the Mesa Spy, proporcionan información sobre los distintos procesos.

Las herramientas de rendimiento para los sistemas de computación distribuida se concentran en las interacciones entre los procesos. El diseño de herramientas de evaluación de rendimiento para los sistemas de computación distribuida consiste en recoger, interpretar y evaluar la información sobre el performance de la ejecución de programas de aplicación, el sistema operativo, la red de comunicaciones, y otros módulos de hardware que trabajan en el entorno. La concurrencia inherente en un entorno de computación distribuida, la falta total de orden de eventos en diferentes máquinas, y la naturaleza no deterministica de los retrasos en la comunicación entre los procesos hace que el problema de la evaluación del rendimiento sea más complejo.

El impacto del tipo de código debe ser considerado en esta evaluación de desempeño. Además, se debe tener en cuenta que algunos parámetros de rendimiento tales como la utilización del procesador, speedup, y la eficacia son difíciles de calcular.

2.1.8 Algunas medidas de performance

En esta sección vamos a comentar algunas medidas de performance con las que se evalúa la planificación de tareas dentro de un entorno HC. Algunas de estas medidas consideran la planificación de tareas independientes, mientras que en otros casos se considera la planificación de tareas que tienen una relación de dependencia.

En [5] se propone una métrica basada en la calidad de servicio (QoS, *Quality of Service*). El término calidad de servicio (QoS) puede tener un significado diferente para los distintos tipos de recursos. Por ejemplo, QoS de la red puede significar el ancho de banda deseado por la aplicación; mientras que para la CPU puede significar la velocidad, o los flops, o la utilización de la CPU. En dicho artículo se distingue dos tipos de tareas: las tareas que tienen requerimientos de QoS y las tareas que no tienen dichos requerimientos, además se distinguen las maquinas con alto y bajo nivel de QoS. También se plantea que las tareas sin requerimientos de QoS se pueden ejecutar tanto en cualquiera de las maquinas del entorno, mientras que las tareas con requerimientos de QoS se deben ejecutar en las maquinas con alto nivel de QoS. Es posible que tareas sin requerimientos de QoS se ejecuten en maquinas con alto nivel de QoS, mientras tareas que tengan dichos requerimientos se ejecuten en maquinas con bajo nivel de QoS. En el artículo se plantea una modificación al algoritmo Min-min para contemplar este tipo de requerimientos en la planificación.

En [6] se proponen dos métricas para optimizar la asignación de tareas a maquinas. La primer métrica que se propone es el makespan, que es el tiempo en que finaliza la última tarea. La segunda métrica propuesta es el Consumo Total de Ciclos de Procesador (TPCC, *Total Processor Cycle Consumption*) que depende del makespan de la planificación y de la velocidad de los procesadores en cada intervalo de tiempo.

En [8] se presentan otras dos métricas para optimizar la asignación de tareas a maquinas. El modelo que se utiliza en este artículo es el siguiente: La red de computadoras se representa como un grafo conexo no dirigido G=(M,N) donde M representa a las maquinas de computo y de comunicación con las que se dispone, y N representa a los links de comunicación. Por otro lado m_j representa a una maquina y $n_{l,k}$ representa un link entre la maquina m_k y m_l. Dentro de las maquinas mj es una maquina de computo si se cumple que $1 \leq j \leq p$ y es una maquina de comunicación si $p+1 \leq j \leq p+q$. Por otro lado se define $R = M \cup N$ como el conjunto de recursos de la red, siendo $r_i \in R$ un recurso de la red. Un camino simple $p_{s,t}$ entre las maquinas m_s y m_t se define como un conjunto de recursos que forman un camino entre m_s y m_t donde no se pasa más de una vez por cada recurso.

Una meta-tarea a ejecutar es representada por un grafo acíclico dirigido T=(V,E) donde $V=\{v_1,v_2,..,v_n\}$ es el conjunto de tareas y E es el conjunto de aristas dirigidas que representan la comunicación de datos entre pares de tareas. Además $t_{i,j}^{E}$ representa el tiempo de ejecución esperado de la tarea v_i en la maquina m_j con $1 \leq i \leq n$ y $1 \leq j \leq p$, este tiempo se asume conocido. Por otro lado, $e_{k,l} \in E$ indica que existe comunicación de datos de v_k hacia v_l, por lo tanto v_l se debe ejecutar después de que finalice v_k. Para cada $e_{k,l}$ existe un valor $d_{k,l}$ que representa el tamaño de los datos que v_k le envía a v_l. Se asume que cada recurso $r_i \in R$ falla siguiendo una distribución de Poisson con media λ_{r_i}. También se define la función M: V→M donde M(i), con $1 \leq i \leq n$, define a que maquina fue asignada la tarea v_i. La función

S_j: V→{0,1,..n} donde $S_j(i)$, con $1 \leq i \leq n$ y $1 \leq j \leq p$, indica el orden en que se ejecuta la tarea v_i en la maquina m_j ($S_j(i) = 0$ indica que la tarea v_i no esta asignada a la maquina m_j)

La primer métrica que se considera es el largo de la planificación, donde el largo de la planificación es el máximo tiempo de finalización de una tarea y el tiempo de finalización de las tareas se calcula en función del tiempo de arribo de los datos de entrada, el tiempo en que la maquina que va a ejecutar esta disponible y el tiempo de ejecución de la tarea en la maquina. A continuación se explica esta métrica:

Sea *X* una asignación de tareas dada. Debido a las limitaciones de precedencia, la tarea v_i no puede empezar a ejecutarse en la máquina *M(i)* a menos que todos los datos de su tarea predecesor inmediata v_k hallan sido recibidos por la máquina *M(i)*.

$t_{i,k}^D$ denota el momento en que v_i tarea ha recibido los datos de la tarea v_k y

$$t_{i,k}^D = \begin{cases} t_K^F & si\ M(k) = M(i) \\ t_k^F + d_{k,i} * c_{M(k),M(i)} & en\ otro\ caso \end{cases}$$

donde t_k^F denota el tiempo de finalización de la tarea v_k y $c_{s,t}$ denota el tiempo esperado de transmisión del envío de un byte de datos desde máquina m_s a la máquina m_t. El momento en que todos los datos de la tarea v_i han sido recibidos por la máquina *M(i)* se refiere a el tiempo de llegada de los datos. Aquí se formaliza el tiempo de legada de los datos de una tarea

$$t_i^D = \max_{e_{k,i} \in E} \left\{ t_{i,k}^D \right\}$$

Con el fin de que la ejecución de una tarea comience en una máquina, todos los elementos de información para la tarea que deben haber sido recibidos, y la máquina debe estar disponible. Así, el tiempo de inicio de la tarea v_i, se denota por t_i^S, se define como $t_i^S = \max\left\{t_i^M, t_i^D\right\}$ donde t_i^M denota el tiempo en que la máquina *M(i)* estará disponible para ejecutar la tarea v_i ($t_i^M = 0$ si la tarea v_i es la primera tarea a ejecutar en la máquina *M(i)*) y t_i^M es igual al tiempo de finalización de la (k-1)ª tarea si es el Kª tarea. Por último, el tiempo de finalización de la tarea v_i (t_i^F) se define como

$$t_i^F = t_i^S + t_{i,M(i)}^E$$

por lo tanto, el largo de la planificación de la aplicación en virtud de asignación de tareas X viene dada por

$$J_1(X) = \max_{v_i \in V} \left\{ t_{i,k}^D \right\}$$

La segunda métrica es la probabilidad de falla, en el artículo se presenta un complejo modelo matemático para obtener la probabilidad de falla de la ejecución de una tarea en una maquina.

Sea $R_J(T, X)$, $1 \leq j \leq p$, la fiabilidad de la máquina de computo m_j, que es la probabilidad de que la máquina m_j sea funcional para la ejecución de las tareas que le han sido asignadas en virtud de asignación de tareas *X*.

Además, $R_j(T, X)$, $p+1 \leq j \leq p+q$, denota la fiabilidad de la máquina de comunicación m_j y $R_{K,l}(T,X)$, denota la fiabilidad del link $n_{l,k}$, donde la fiabilidad de un recurso de comunicación (máquina o link) es la probabilidad que el recurso de comunicación sea funcional para el desempeño de comunicación inter-tarea durante la ejecución de la meta-tarea en virtud de la asignación de tareas *X*. Hay que tener en cuenta que, desde que la falla de un recurso se rige por un proceso de Poisson, la fiabilidad del recurso r_i en el tiempo t es $e^{-\lambda_{r_i} t}$.

La conclusión con éxito de la ejecución de la meta-tarea requiere que cada máquina de computación sea funcional durante el tiempo que sus tareas asignadas se están ejecutando y que cada recurso de comunicación que se utilizará en las comunicaciones inter-tareas sea funcional durante el tiempo en que se está llevando a cabo la comunicación inter-tarea, esto depende de la fiabilidad de los recursos donde la meta-tarea es asignada. La probabilidad de que la meta-tarea T se pueda ejecutar con éxito en un sistema de HC en virtud de asignación de tareas *X* se denota por *R(T, X)*, que también representa la fiabilidad del sistema de HC cuando la meta-tarea *T* se asigna de *X*. Asumiendo que las fallas de los recursos son estadísticamente independientes, *R(T, X)* se define como

$$R(T,X) = \prod_{m_j \in R_K} R_j(T,X) . \prod_{n_{k,l}} R_{k,l}(T,X)$$

$$= \prod_{m_j \in R_K} e^{-\lambda_j t_j^A} . \prod_{n_{k,l} \in R_K} e^{-\lambda_{k,l} t_{k,l}^A}$$

$$= e^{(-COST(X))}$$

donde $COST(X) = \sum_{m_j \in R_K} \lambda_j t_j^A + \sum_{n_{k,l} \in R_K} \lambda_{k,l} t_{k,l}^A$, y λ_j y $\lambda_{k,l}$ son las tasas de fallas de la máquina m_j y link $n_{k,l}$, respectivamente. Además, t_j^A, $1 \leq j \leq p$, denota el momento en que la maquina de cómputo m_j completa la ejecución de tareas en X y se define como

$$t_j^A = \max_{S_j(i)>0} \{t_i^F\}$$

las variables t_j^A, $p+1 \leq j \leq p+q$, y $t_{k,l}^A$ denotan el momento en que la máquina de comunicación m_j y el enlace $n_{k,l}$ completan la comunicación de datos inter-tareas en X respectivamente, y son definidos como

$$t_j^A = \max_{e_{u,v} \in E} \{I_j(M(u), M(v)) t_{v,u}^D\}$$

$$t_{k,l}^A = \max_{e_{u,v} \in E} \{I_{k,l}(M(u), M(v)) t_{v,u}^D\}$$

donde

$$I_j(s,t) = \begin{cases} 1, & m_j \text{ esta en } p_{s,t} \\ 0, & \text{en otro caso} \end{cases}$$

$$I_{k,l}(s,t) = \begin{cases} 1, & n_{k,l} \text{ esta en } p_{s,t} \\ 0, & \text{en otro caso} \end{cases}$$

Por último, R_K es un conjunto de recursos que se compone de máquinas de cómputo y de comunicación, y de enlaces. Es decir, R_K es un subconjunto de los

recursos utilizados para la ejecución de la meta-tarea en virtud de la asignación X y decidida con respecto al conjunto $K = \{m_j \mid m_j = M(i) \wedge v_i \in V\}$, que es el conjunto de máquinas de cómputo para la que al menos una tarea de la meta-tarea se le asigna. Si la topología de la red de un sistema de HC puede ser modelada por un árbol (es decir, existe un único camino simple entre dos máquinas) el conjunto R_K se determina como sigue. (i) Incluir todas las máquinas de cómputo para que al menos una tarea se le asigna en R_K. (ii) Para todos los $e_{k,l} \epsilon$ E, se incluyen todas las máquinas de comunicación y los links que constituyen el único camino simple entre la máquina m_s y la maquina m_t en R_K . Si la topología de la red no es un árbol, el cómputo de R_K no es trivial.

Como resultado de ello, la probabilidad de falla de la aplicación virtud de la asignación de tareas X se define como: $J_2(X) = 1 - R(T, X)$

2.1.9 Algunas heurísticas propuestas

En la literatura se han sido propuestas varias heurísticas para resolver el problema de mapeo de tareas a maquinas dentro de un entorno HC. Por un lado podemos clasificar estas heurísticas en dos grandes grupos: las heurísticas estáticas y las dinámicas. Las heurísticas estáticas se corren antes de que se empiece a ejecutar la meta-tarea en el entorno HC, en ese momento se determina el mapeo de todas las tareas a las maquinas del entorno; por otro lado las heurísticas dinámicas se pueden ejecutar antes del inicio de la ejecución de las tareas en entorno, además pueden seguir en ejecución en paralelo a la ejecución de las tareas y modificar y/o completar el mapeo que se determinó al inicio de la ejecución. Otra posible clasificación es dividir las heurísticas que consideran que las tareas son independientes y las que consideran precedencias entre las tareas. En esta sección se presenta una serie de heurísticas que se encuentran en la literatura.

En [2] se proponen una serie de heurísticas entre las que se encuentran: heurísticas simples como son MCT, MET, KPB, y OLB; heurísticas más especificas como son Min-min, Max-min, Sufferage; y por último métodos más complejos y metaheurísticas como son Mapeo cluster-M, LMT (levelized min-time), AG para matcheo y planificación, programación generacional, planificación auto ajustable para HC. A continuación se detallan las mismas.

Tiempo mínimo de finalización (MCT): esta heurística asigna cada tarea a la máquina en que la tarea tiene menor tiempo de finalización. Esto provoca que algunas tareas se asignen a máquinas que no tienen el tiempo mínimo de ejecución para ellas. La Heurística MCT se utiliza como referencia para el modo inmediato, es decir, el desempeño de los demás heurísticas se compara con el desempeño de la heurística MCT. Cuando una tarea llega, todas las máquinas en el entorno HC son analizadas para determinar la máquina que le da el menor tiempo para completar dicha tarea. Por lo tanto, esta heurística es de O(m) para dar un mapeo.

Tiempo mínimo de ejecución (MET): esta heurística asigna cada tarea a la máquina que realiza la ejecución de la tarea en el menor tiempo de ejecución, esta heurística también es conocida como la mejor asignación limitada (LBA) y asignación directa del usuario (UDA). Esta heurística, en contraste con MCT, no se considera el tiempo en que la máquina esta disponible para ejecutar las tareas. Esta heurística puede causar un grave desequilibrio en la carga de las máquinas. Las ventajas de este método son que asigna cada tarea a la máquina que la ejecuta en la menor cantidad de tiempo, y la heurística es muy simple. Esta heurística tiene O(m)

tiempo para encontrar la máquina que tenga el mínimo tiempo de ejecución de una tarea.

Algoritmo de conmutación (SA): esta heurística está motivada por las siguientes observaciones. La heurística MET puede crear desequilibrio de carga entre las máquinas mediante la asignación de muchas más tareas que algunas máquinas que a otros, mientras que la heurística MCT trata de equilibrar la carga mediante la asignación de tareas a la maquina con menor tiempo de finalización. Si las tareas están llegando en una mezcla aleatoria, es posible utilizar MET a expensas de equilibrio de carga hasta un determinado umbral y luego usar el MCT para suavizar la carga a través de las máquinas. La heurística SA utiliza las heurísticas MCT y MET de forma cíclica en función de la distribución de carga a través de las máquinas. El propósito es contar con una heurística con las propiedades deseables de MCT y MET.

Mejor k-porciento (KPB): esta heurística considera sólo un subconjunto de las máquinas mientras que mapea una tarea. El subgrupo está formado por las m*(k/100) mejores máquinas basadas en los tiempos de ejecución para la tarea, en donde 100/m<=k<=100. La tarea se asigna a la máquina que proporciona el menor tiempo de terminación del subgrupo. Si k = 100, entonces la heurística KPB se reduce a la heurística MCT. Si k = 100/m, entonces la heurística KPB se reduce a la heurística MET. Un "buen" valor de k mapea una tarea a una máquina que esta dentro de un subconjunto formado por las maquinas computacionalmente superiores. El propósito no es tanto matchear la tarea actual a una maquina que sea computacionalmente adecuada para esa tarea, sino que es evitar poner la tarea actual en una máquina que podría ser más adecuada para alguna tarea que aun no ha llegado. Su "previsión" sobre la heterogeneidad de las tareas es lo que falta en el MCT, lo que podría asignar una tarea a una máquina mal matcheada por una mejora local marginal en tiempo de finalización, posiblemente privando a algunas tareas que llegan posteriormente de una mejor adecuación de las tareas de esa máquina, y finalmente conducen a un mayor makespan en comparación con la KPB. Cabe señalar que, si bien tanto la KPB y la SA combinan elementos de la MCT y la MET en su funcionamiento, sólo la KPB es la que en cada asignación de tareas intenta optimizar los objetivos de la MCT y la MET simultáneamente. Sin embargo, en los casos en que un determinado subconjunto de las máquinas no se encuentra entre los mejores k% para cualquiera de las tareas, la KPB causara mayores tiempos de inactividad en las máquinas en comparación con el MCT, y puede resultar en un desempeño mucho más pobre. Por lo tanto, el rendimiento relativo de la KPB y el MCT puede depender de las características del entorno HC de máquinas y de las tareas en ejecución. Para cada tarea, el tiempo que se gasta en clasificar las máquinas para determinar el subconjunto de las máquinas a examinar es de O(m log m). Una vez que el subconjunto de las máquinas está decidido, toma O((m * k) / 100), es decir O(m), para determinar la máquina de destino. En general, la heurística KPB es de O(mlogm).

La heurística balanceo de carga oportunista (OLB) asigna una tarea a la próxima máquina que pasa a estar lista, sin tener en cuenta el tiempo de ejecución de la tarea en esa máquina. En caso de que muchas máquinas estén disponibles al mismo tiempo, entonces una máquina es arbitrariamente elegida. La complejidad de la OLB heurística depende de la aplicación. El mapeo puede tener que examinar las m máquinas, para encontrar la próxima máquina que pasa a estar disponible. Por lo tanto, toma O(m) para encontrar la asignación.

Min-min: es una de las heurísticas implementadas en SmartNet. Sea r_j el tiempo esperado en que la maquina m_j se encuentre lista para ejecutar una tarea después de terminar la ejecución de todas las tareas que le han sido asignadas en ese momento. En primer lugar se calculan todos los valores de c_{ij} (tiempo esperado de

finalización de la tarea t_i en la maquina m_j) utilizando los valores de e_{ij} (tiempo de ejecución esperado de la tarea t_i en la maquina m_j) y r_j. Para cada tarea t_i, se define la máquina que da el tiempo de terminación previsto más temprano mediante la recorrida de la i-ésima fila de la matriz c (compuesta por los valores c_{ij}). Luego se determina la tarea t_k que tiene el mínimo tiempo de terminación previsto más temprano y luego es asignado a la máquina correspondiente. Luego se actualizan la matriz c y el vector r, y el proceso anterior se repite con las tareas que aún no han sido asignadas a una máquina. Min-min comienza la planificación de las tareas que cambian el tiempo en que la maquina estará lista en menor medida. Si las tareas t_i y t_k son contendientes para una máquina m_j, entonces Min-min asigna m_j a la tarea (por ejemplo t_i) que va a cambiar en menor medida el tiempo en que la maquina m_j vuelva a estar disponible. Esto aumenta la probabilidad de que la tarea t_k siga teniendo su tiempo de terminación previsto más temprano en la maquina m_j, y sea asignado a la misma. Porque en t = 0, la máquina que termina una tarea más temprana es también en la que se ejecuta más rápido, y la heurística min-min cambia el tiempo en que la máquina esta disponible en la menor medida en cada asignación. La expectativa es que se puede obtener un menor makespan si un mayor número de tareas se asigna a las máquinas que no sólo las completan antes, sino también la ejecución de ellas las realizan más rápido. La inicialización de la matriz c es de O(Sm). El bucle de hacer la heurística min-min se repite S veces y cada iteración es de O(Sm). Por lo tanto, la heurística es de toma $O(S^2m)$.

Max-min: es similar a la heurística min-min, y es otra de las heurísticas implementadas en SmartNet. Se distingue de la heurística Min-min en que, una vez que se encuentra la máquina que proporciona el tiempo de terminación previsto más temprano para cada tarea, se determina la tarea t_k que tiene el mayor tiempo de terminación previsto más temprano y luego asignado a la correspondiente máquina. La heurística max-min tiene la misma complejidad que la heurística min-min. Es probable que la heurística max-min de una mejor planificación que la heurística min-min en los casos en que hay muchos más tareas cortas que tareas largas. Por ejemplo, si sólo hay una larga tarea, Max-min va a ejecutar muchas tareas cortas simultáneamente con la larga tarea. En este caso el makespan resultante podría ser determinado solamente por el tiempo de ejecución de la tarea larga. Sin embargo, min-min en primer lugar ejecuta las tareas mas cortas (que pueden ser más o menos uniformemente distribuidas en las máquinas) y luego ejecuta la tarea larga, aumentando el makespan en comparación con el max-min.

Sufferage: se basa en la idea de que la mejor asignación puede ser generada por la asignación de una máquina para una tarea que "sufra" más en términos de tiempo de finalización esperado si esa máquina particular no le ha sido asignada. El valor de sufragio de una tarea t_i es la diferencia entre su segundo tiempo de terminación previsto más temprano (en la maquina m_i) y tiempo de terminación previsto más temprano (en la maquina m_x). La fase de inicialización es similar a la realizada por las heurísticas min-min y max-min. Inicialmente, todas las máquinas están marcadas sin asignar. En cada iteración, se selecciona una tarea t_k de la meta-tarea arbitrariamente. Luego se encuentra la máquina m_j que da el tiempo de terminación previsto más temprano para la tarea t_k y se asigna provisionalmente t_k a m_j si m_j está sin asignar, se marca m_j como asignada y se elimina t_k de meta-tarea. Sin embargo, si la máquina m_j ha sido previamente asignada a una tarea t_i, elige entre t_i y t_k la tarea que tiene el mayor valor del sufragio, se asigna a m_j la tarea elegida, y se elimina la tarea elegida de la meta-tarea. La tarea que no se elige no será considerada de nuevo para esta iteración. Cuando todas las iteraciones se han completado, se actualizan los tiempos en que las máquina estarán disponibles para aquellas maquinas a las que se le ha asignado una nueva tarea. Esta iteración se repite hasta que se asignen todas las tareas a las maquinas.

Mapeo Cluster-M: el proceso de matcheo y planificación del entorno HC puede ser pensado como un mapeo de un grafo que representa un conjunto de tareas (grafo de tareas) en un grafo que representa el conjunto de máquinas del entorno HC (grafo del sistema). En Cluster-M, el mapeo se realiza en dos etapas. En la primera etapa, el grafo de tareas y el grafo del sistema se agrupan. La agrupación del grafo de tareas combina las tareas con comunicación intensiva en un mismo cluster. Del mismo modo, el grafo del sistema combina las máquinas que están estrechamente unidas (es decir, que tengan pequeños tiempos de comunicación inter-máquina) en el mismo cluster. La agrupación del grafo de tareas no depende de la agrupación del grafo del sistema y viceversa. Por lo tanto, el grafo de tarea o el del sistema tiene que ser agrupados sólo una vez. En la segunda fase, el grafo de tareas agrupado es mapeado en el grafo del sistema agrupado. El agrupamiento reduce la complejidad del problema de mapeo y mejora la calidad del mapeo resultante.

LMT (levelized min-time): Esta heurística es un algoritmo estático de equiparación y planificación de tareas en un entorno HC. El algoritmo LMT utiliza una estrategia en dos fases. La primera etapa utiliza una técnica llamada level sorting para ordenar las tareas basándose en las restricciones de precedencia. El nivel puede definirse como sigue. El nivel 0 contiene tareas que no tengan arcos incidentes en el grafo que representa la dependencia entre tareas. Todos los predecesores con arcos a tareas de nivel k se encuentran en los niveles k-1 a 0. Para cada tarea en el nivel k existe al menos un arco incidente (la dependencia de datos) tal que la tarea fuente está en el nivel k-1. La técnica de level sorting agrupa las tareas que se pueden ejecutar en paralelo. La segunda fase del algoritmo LMT utiliza el algoritmo min time para asignar las tareas nivel a nivel. El algoritmo min time es un método greedy que intenta asignar cada tarea a la mejor máquina. Si el número de tareas es más grande que el número de máquinas, entonces las tareas más pequeñas se fusionan hasta que el número tareas sea igual al número de máquinas. A continuación, las tareas son ordenadas en orden descendente por su tiempo de procesamiento promedio. Cada tarea es asignada a la máquina para la que tiene el menor tiempo de finalización. Ordenar las tareas por el tiempo de procesamiento promedio aumenta la probabilidad de que tareas más grandes obtengan maquinas más rápidas.

Algoritmo genético para matcheo y planificación: En los algoritmos genéticos (GAs), las posibles soluciones se codifican como cromosomas, el conjunto de estas soluciones se denomina población. Esta población es operada iterativamente hasta que se cumple un criterio de parada aplicando los siguientes pasos. El primer paso es la selección, donde algunos cromosomas se eliminan y otros se duplican sobre la base de su valor de fitness (una medida de la calidad de la solución representada por un cromosoma). Esto es seguido por el cruzamiento, donde se eligen parejas de cromosomas y algunos de los componentes correspondientes de los cromosomas son intercambiados. Luego, los cromosomas son mutados al azar, con la limitación de que el cromosoma resultante debe seguir representando soluciones válidas para el problema físico. Para aplicar un algoritmo genético la problema de matcheo y planificación de tareas en un entorno HC se puede utilizar el siguiente enfoque: los cromosomas son codificados con dos partes: un string para el matcheo (mat) y un string para la planificación (ss). Si mat (i) = j, entonces la tarea t_i se ha asignado a la máquina m_j. Si ss (k) = i, entonces tarea t_i es el k-ésima tarea en el orden total. Cada cromosoma está asociado con un valor de fitness, que es el momento de terminación de la solución que representa este cromosoma (es decir, el tiempo de ejecución que se prevé para la aplicación si se usa el mapeo que este cromosoma especifica).

Reasignador Híbrido: es un algoritmo dinámico para el matcheo y la planificación de tareas en los sistemas de HC. Al inicio, el reasignador hibrido recibe como

entrada un mapeo estático. El reasignador híbrido se ejecuta en dos fases. En la primera fase del reasignador híbrido, realizada antes de la ejecución de la aplicación, las tareas son divididas en L niveles como en la heurística LMT. A cada tarea se le asigna un rango mediante el examen de las tareas de nivel L-1 a nivel 0. El rango de cada tarea en el (L-1)-ésimo nivel está establecido como su tiempo esperado de cálculo en la máquina a la que fue asignado por el mapeo inicial. El rango de una tarea s_i en el nivel k se determina calculando la longitud del camino crítico de s_i a la tarea que termina la ejecución. La segunda fase del reasignador híbrido se produce durante la ejecución de la aplicación. El reasignador híbrido realiza cambios el matcheo y la programación de las tareas del nivel k, mientras las tareas en el nivel (k-1) o anterior se están ejecutando. Las tareas en el nivel k se examinan en orden descendente según el rango estático, y cada tarea se asigna a la máquina con menor de tiempo de terminación para esa tarea. El reasignador híbrido comienza la programación de las tareas de nivel k cuando la primera tarea de nivel (k-1) comienza su ejecución, y la reasignación del nivel k debe terminar antes de que cualquier tarea de nivel k tenga los datos de entrada y las máquinas disponibles que necesita para ejecutarse. Cuando el nivel k está programado, es muy probable que la información de los tiempos de ejecución actuales pueda ser utilizada para muchas tareas de los niveles 0 y (k-2). Puede haber algunas tareas de los niveles 0 a (k-2) que aún se encuentran en funcionamiento o en espera de ejecución cuando se está considerando la posibilidad de reasignación para las tareas de nivel k. Para este tipo de tareas, se utilizan los tiempos de terminación esperados. Resultados de Simulaciones indican que el reasignador híbrido puede mejorar el rendimiento de un mapeo estático obtenido de forma inicial hasta en un 15% para algunos casos.

Programación Generacional: La heurística de programación generacional (GS) es una heurística dinámica para mapear tareas en un entorno HC. Se trata de una heurística cíclica con cuatro etapas. En primer lugar, el GS forma un problema de planificación parcial sacando todas las tareas con limitaciones de precedencia iniciales en el DAG (grafo acíclico dirigido, Directed Acyclic Graph) que representa a la aplicación. Es decir, el problema de planificación parcial inicial consiste en tareas que son independientes o no tienen aristas incidentes en el DAG. Las tareas en el problema de planificación parcial inicial son entonces asignadas a las máquinas utilizando un programador auxiliar. El programador auxiliar realiza la asignación de las tareas utilizando el algoritmo primero en llegar, primero en servir. Una tarea es asignada a la máquina que minimiza el tiempo de finalización de esa tarea particular. Cuando una tarea del problema de planificación parcial inicial concluye su ejecución, la heurística GS realiza una reasignación. Durante la reasignación, el GS revisa el problema de planificación parcial para agregar y quitar las tareas. La terminación de una tarea desencadena el evento reasignación, porque esta tarea puede hacer que se satisfagan las limitaciones de precedencia de algunas tareas. Estas tareas se añaden al problema de planificación parcial inicial. Las tareas que ya han comenzado su ejecución se eliminan del problema de planificación parcial inicial. Una vez revisado el problema de planificación parcial se obtienen, las tareas que son asignados a alguna de las máquinas del entorno HC utilizando el programador auxiliar. Este procedimiento se realiza cíclicamente, hasta la finalización de todas las tareas.

Planificación auto ajustable para HC: La heurística de programación auto ajustable para sistemas heterogéneos (Sash) es un algoritmo dinámico para mapear un conjunto de tareas independientes (meta-tarea) en un entorno HC. Un procesador se dedica a calcular la planificación, y esta planificación se superpone con la ejecución de las tareas. Al final de cada fase de planificación, el procesador que calcula la planificación carga las tareas, en esta fase, en las colas locales de los procesadores que están trabajando. El procesador dedicado a continuación, calcula

la planificación para el próximo subconjunto de tareas mientras que el subconjunto de tareas anteriormente planificado está siendo ejecutado por los procesadores que están trabajando. La duración de la fase de planificación se determina por un límite inferior estimado de la carga en los procesadores que están trabajando. El primer procesador de los que están trabajando que completa la ejecución de todas las tareas en su cola local le indica esto al procesador encargado de calcular la planificación, luego este le asigna más tareas a todos los procesadores basándose en una planificación parcial que este computa. La heurística SASH calcula la planificación utilizando una variación del algoritmo branch-and-bound. En esta variación, se utiliza un árbol para representar el espacio de posibles planificaciones. Un nodo en el árbol representa una programación parcial que consiste en un conjunto de tareas asignadas a un correspondiente conjunto de procesadores. La fase de planificación se compone de uno o más iteraciones del SASH. En una iteración, el nodo con el costo más bajo se amplía para aumentar la programación parcial con otra asignación tarea-procesador. La expansión de nodos termina cuando todas las tareas se han planificado o el tiempo para la programación de la fase i expiró.

2.2 Aplicaciones de la computación heterogénea

Hay varios proyectos en los que se implementan soluciones a diferentes problemas utilizando técnicas de computación heterogéneas. En [11] se presenta un listado de varios de estos proyectos, aquí se detallan los proyectos activos y los próximos proyectos a realizar. Los proyectos más relevantes son *Berkeley Open Infrastructure for Network Computing* (*BOINC*) y *World Community Grid*.

2.2.1 Berkeley Open Infrastructure for Network Computing (BOINC)

El *Berkeley Open Infrastructure for Network Computing* (BOINC) es un sistema de middleware no comercial para computación distribuida y computación voluntaria. Originalmente fue desarrollado para apoyar el proyecto SETI@home, luego se convirtió en una plataforma útil para aplicaciones distribuidas en áreas tan diversas como las matemáticas, la medicina, la biología molecular, la climatología, y la astrofísica. La intención de BOINC es hacer posible que los investigadores puedan aprovechar la enorme potencia de procesamiento de los ordenadores personales en todo el mundo.

La plataforma BOINC activa tiene casi 565.000 computadoras (hosts) en todo el mundo, en promedio tiene 1.287 PFLOPS de procesamiento (diciembre de 2008). BOINC es un proyecto financiado por la National Science Foundation a través de premios SCI/0221529, SCI/0438443y SCI/0721124 [12].

La plataforma BOINC es soportada por diversos sistemas operativos, incluyendo Microsoft Windows, Mac OS X y los diversos sistemas Unix incluyendo Linux y FreeBSD. BOINC es un software gratuito que se distribuye bajo la licencia GNU.

En [13] aparece una lista de proyectos activos que se pueden elegir utilizando la plataforma BOINC, entre los que se destacan:

Docking@Home. Este proyecto tiene objetivos dentro de la biociencia y la informática. El proyecto apunta a un mayor conocimiento de los detalles atómicos de las interacciones de las proteínas ligadas y, al hacerlo, busca información para ayudar al descubrimiento de nuevos productos farmacéuticos.

Superlink@technion. Este proyecto busca ayudar a los genetistas de todo el mundo a encontrar los genes que provocan algunos tipos de diabetes, la hipertensión, el cáncer, la esquizofrenia y muchas otras enfermedades.

Malariacontrol.net. Los modelos de simulaciones de la dinámica de transmisión y los efectos en la salud de la malaria son una herramienta importante para el control de la malaria. Se pueden utilizar para determinar estrategias óptimas para la entrega de mosquiteros, la quimioterapia, o las nuevas vacunas que están actualmente en fase de desarrollo y prueba. Tal modelado tiene necesidades de computo extremadamente intensivas, esto se debe a requiere simulaciones de grandes poblaciones humanas con una serie de parámetros relacionados con factores biológicos y sociales que influyen en la distribución de la enfermedad.

Rosetta@home. En este proyecto se busca determinar las formas tri-dimensionales de las proteínas en la investigación que últimamente puede conducir a encontrar curas para las principales enfermedades humanas. El esfuerzo esta concentrado en diseñar nuevas proteínas para combatir enfermedades como el VIH, la malaria, el cáncer y la enfermedad de Alzheimer.

SIMAP. El objetivo de este proyecto es calcular similitudes entre las proteínas. SIMAP proporciona una base de datos pública de los datos resultantes, que desempeña un papel clave en muchos proyectos de investigación en bioinformática.

GPUGrid.net. Este proyecto busca abrir nuevos escenarios computacionales para el primer código de dinámica molecular full-atom (CellMD) especialmente optimizado para correr en la GPU NVIDIA y Sony PlayStation 3. Este proyecto presenta la particularidad de que se pueden integrar consolas Sony PlayStation 3 a la grid en que se corre.

Proyecto Lattice. Este proyecto suministra poder de cómputo a los científicos de la Universidad de Maryland que estudian las relaciones evolutivas basada en los datos de las secuencias de ADN; secuencias de proteínas de bacterias, plásmidos y virus; y la diversidad biológica en las reservas naturales.

Climateprediction.net. Se busca investigar las aproximaciones que se tienen que realizar en el estado del arte de los modelos climáticos. Corriendo el modelo miles de veces se quiere ver como el modelo responde a ligeras modificaciones de esas aproximaciones. Se busca mejorar la comprensión de que tan sensible son los modelos a los a pequeños cambios y también a cosas como los cambios en el anhídrido carbónico y el ciclo de azufre. Se busca explorar cómo puede cambiar el clima en el próximo siglo bajo una gama amplia de escenarios diferentes.

Quantum Monte Carlo at Home. El objetivo de este proyecto es estudiar la estructura y reactividad de las moléculas mediante la química cuántica.

Spinhenge@home. Se busca estudiar las moléculas magnéticas y el magnetismo controlado a nanoescala. Estas moléculas magnéticas pueden ser utilizadas para desarrollar diminutos interruptores magnéticos, con aplicaciones en la medicina (como la quimioterapia local del tumor) y la biotecnología.

Einstein@home. Es un programa que analiza datos procedentes del LIGO (Laser Interferometer Gravitational wave Observatory) en los Estados Unidos y también del observatorio de ondas gravitacionales GEO 600 en Alemania , para aprender mas acerca de las estrellas de neutrones, también llamadas púlsares.

Leiden Classical. Los cálculos de la ciencia de superficies utilizan Dinámica clásica. En contraste con otros proyectos, Leiden Classical permite que voluntarios, estudiantes y otros científicos envíen sus cálculos personales al grid. Cada usuario tiene su propia cola de trabajos de dinámica clásica.

Milkyway@home. El objetivo de este proyecto es crear un modelo tri-dimensional muy exacto de la galaxia de la Vía Láctea que usa los datos recogido por el Sloan Digital Sky Survey.

uFluids@home. El proyecto simula en dos fases el comportamiento de los fluidos en problemas de microgravedad y microfluídica.

LHC@home. El Gran Colisionador de Hadrones (LHC) es un acelerador de partículas que se está construyendo en el CERN, la Organización Europea de Investigación Nuclear, el laboratorio de física de partículas más grande del mundo. Cuando se encienda será el instrumento más poderoso jamás construido para investigar las propiedades de las partículas. LHC@home simula las partículas que viajan alrededor del LHC para estudiar la estabilidad de sus órbitas.

SETI@home. SETI (Búsqueda de Inteligencia Extraterrestre) es un área científica, cuyo objetivo es detectar vida inteligente fuera de la Tierra. Un enfoque, conocido como radio SETI, usa radiotelescopios a la escucha de señales de radio de ancho de banda estrecha desde el espacio. Como no se conoce la existencia natural de estas señales sería una prueba de detección de la tecnología extraterrestre.

SZTAKI Desktop Grid. El objetivo de este proyecto es encontrar todos los sistemas de números binarios generalizados (en el que las bases son las matrices y los dígitos son vectores) hasta dimensión 11.

SHA-1 Collision Search Graz. El objetivo de este proyecto es buscar colisiones (debilidades) de la ampliamente utilizada función de hash SHA-1.

PrimeGrid. En este proyecto se esta generando una base de datos con una secuencia publica de números primos, y esto se hace buscando parejas de grandes números primos de la forma $k*2^n+1$ y $k*2^n-1$.

Rectilinear Crossing Number. Aquí se busca determinar la configuración de los n puntos de un grafo completo Kn de forma de minimizar el número de cruces entre las aristas del mismo. Se ha logrado determinar este numero para n <= 17. Recientes (no publicadas aún) consideraciones matemáticas han determinado el numero de cruces rectilíneos para n = 19 y n = 21. El principal objetivo de este proyecto es determinar el verdadero valor para n = 18.

ABC*@home*. El objetivo de este proyecto es buscar “trios-abc”: enteros positivos a, b, c tales que a + b = c; a <b <c; a, b, c no tienen divisores comunes y c> rad (abc), donde rad (n) es el producto de los distintos factores primos de n. La conjetura ABC dice que solo hay muchos a,b,c, finitos talque log(c)/log(rad(abc))>h para un real h>1. La conjetura ABC es actualmente uno de los mayores problemas abiertos en matemática. Si se demuestra que es verdadera, muchos otros problemas abiertos pueden ser contestados directamente desde ella.

NQueens@home. El problema de las ocho reinas consiste en tratar de colocar ocho reinas en un tablero de ajedrez de modo que ninguna reina ataque a alguna de las otras reinas. Desde hace mucho tiempo se sabe que hay 92 soluciones, de los cuales 12 son diferentes. Este proyecto estudia la generalización a N reinas en un tablero NxN, para N >= 19.

AQUA@home. AQUA (algoritmos cuánticos Adiabáticos) es un proyecto de investigación cuyo objetivo es predecir el rendimiento de las computadoras cuánticas en una variedad de problemas difíciles que surgen en campos que van desde la ciencia de los materiales hasta el aprendizaje automático. Se busca diseñar y analizar los algoritmos de computación cuántica, utilizando técnicas de Monte Carlo cuántico.

Chess960@home. Este proyecto estudia el ajedrez 960, una variante del ajedrez ortodoxo. En el ajedrez clásico la posición inicial del juego nunca cambia. En el

ajedrez en 960 la configuración inicial de las piezas de ajedrez se determina, justo antes del inicio de cada juego, al azar.

También existe la posibilidad de acceder a otros proyectos similares a BOINC, como son World Community Grid o Ibercivis desde la plataforma BOINC.

2.2.2 World Community Grid (WCG)

World Community Grid (WCG) es un esfuerzo para crear la red informática pública más grande del mundo para hacer frente a proyectos de investigación científica que beneficien a la humanidad. Fue lanzada el 16 de noviembre de 2004, es financiada y operada por IBM con software cliente disponible actualmente para los sistemas operativos Windows, Linux, Mac OS X y FreeBSD.

Usando el tiempo ocioso de las computadoras de todo el mundo, los proyectos de investigación de la World Community Grid han analizado aspectos del genoma humano, el VIH, el dengue, la distrofia muscular, y el cáncer. WCG tiene más de 428.000 cuentas de usuario registradas. El software de la WCG usa el tiempo ocioso de las computadoras que están conectadas a Internet para realizar cálculos. Los usuarios instalan el software cliente de la WCG en sus computadoras.

En la actualidad la WCG tiene varios proyectos activos, a continuación se mencionan algunos de ellos [14].

El Proyecto de Energía Limpia. Los científicos del grupo Aspuru-Guzik de la Universidad de Harvard está utilizando la World Community Grid para descubrir materiales para la tecnología de energía renovable. El objetivo principal del proyecto es calcular las propiedades electrónicas de decenas de miles de nuevos materiales y para determinar cuáles de ellas son los mejores candidatos para la próxima generación de células solares económicas.

Arroz nutritivo para el mundo (Nutritious Rice for the World). Lo que se busca en este compilar módulos de cálculo que ayudarán a determinar las mejores opciones para maximizar las producciones y la calidad del arroz.

Ayudar a Superar el Cáncer (Help Conquer Cancer). El Ontario Cancer Institute (OCI), el Princess Margaret Hospital y la University Health Network formaron un equipo con el World Community Grid para mejorar los resultados de cristalografía de la radiografía de la proteína para poder comprender mejor el cáncer y su tratamiento.

Descubriendo Fármacos Contra el Dengue. Científicos de la Universidad de Texas y de la Universidad de Chicago se juntaron con los investigadores del World Community Grid para combatir algunas de las enfermedades virales más endémicas en los países desarrollados y en países en vías de desarrollo. La meta inmediata del proyecto es descubrir fármacos nuevos para tratar la fiebre hemorrágica del dengue, hepatitis C, encefalitis del Nilo occidental (West Nile encephalitis) y fiebre amarilla; estas enfermedades resultan de infección con virus de la familia Flaviviridae.

Plegado del proteoma humano – Fase 2. Este proyecto continúa desde donde se interrumpió la primera fase. Los dos objetivos principales del proyecto son: 1) obtener estructuras con mayor resolución para proteínas humanas específicas y proteínas de agentes patógenos y 2) explorar más ampliamente los límites de la predicción de estructuras de.

FightAIDS@Home (lucha contra el SIDA desde casa). El objetivo de este proyecto es usar métodos de computación para identificar los medicamentos que posean la forma y las características químicas correctas para bloquear la protease del HIV (virus del SIDA). Este método se denomina "Structure-Based Drug Design" (producción de medicamentos basándose en su estructura), y de acuerdo con el National Institute of General Medical Sciences, ya ha tenido un importante efecto en la vida de personas que conviven con el SIDA.

2.3 Modelo ETC

Como se mencionó anteriormente, para evaluar el comportamiento de una heurística se necesita un modelo de los tiempos de ejecución de las tareas en las máquinas, también es necesario que los parámetros de este modelo se puedan cambiar para investigar el desempeño de la heurística en diferentes entornos HC y con diferentes tipos de tareas a asignar. Un modelo frecuentemente utilizado en las investigaciones relacionadas a la asignación de tareas a procesadores es el modelo ETC. Este modelo requiere una matriz de tiempos esperados de ejecución (ETC), donde la entrada (i,j) de esta matriz representa el tiempo esperado de ejecución de la tarea t_i en la maquina m_j. En una matriz ETC los elementos de una determinada fila representan el tiempo estimado de ejecución de una tarea dada en las distintas maquinas del entorno, mientras que los elementos de una determinada columna representan el tiempo de ejecución esperado de las diferentes tareas en una maquina dada.

El modelo presentado puede ser caracterizado por tres parámetros: heterogeneidad de maquinas, heterogeneidad de tareas y consistencia. La variación a lo largo de una fila refiere a la heterogeneidad de maquinas, esto es el grado en que los tiempos de ejecución de una tarea dada varia en las distintas maquinas. La heterogeneidad de maquinas del entorno es basada en la combinación de la heterogeneidad de maquinas para todas las tareas. Cuando tenemos un entorno de maquinas con características computacionales similares decimos que es un entorno con baja heterogeneidad de maquinas, mientras que si tenemos un entorno de maquinas con características computacionales variadas decimos que es un entorno con alta heterogeneidad de maquinas.

De manera similar, la variación a lo largo de una columna refiere a la heterogeneidad de tareas, esto es el grado en que los tiempos de ejecución de las distintas tareas varían en una maquina dada. La heterogeneidad de tareas del entorno es basada en la combinación de la heterogeneidad de tareas para todas las maquinas. Cuando tenemos conjunto de tareas con necesidades computacionales similares decimos que tenemos baja heterogeneidad de tareas, mientras que si tenemos conjunto de tareas con necesidades computacionales variadas decimos que tenemos alta heterogeneidad de maquinas.

Una matriz ETC se puede clasificar como: consistente, inconsistente o parcialmente consistente. Una matriz ETC es consistente si cuando una maquina m_x tiene menor tiempo de ejecución que la maquina m_y para una tarea t_x, entonces tiene también menor tiempo de ejecución para todas las tareas t_i. En una matriz ETC inconsistente las relaciones entre las necesidades computacionales las tareas y las capacidades de las máquinas no están estructuradas como en el caso consistente. Las matrices ETC inconsistentes se dan en la práctica cuando: (1) existe una gran variedad de arquitecturas de máquina en el entorno, y (2) existe una gran variedad de necesidades de cómputo de las distintas tareas. Una combinación de los dos casos anteriores, que pueden ser más realistas en muchos entornos, es

una matriz parcialmente-consistente, que es una matriz inconsistente que contiene una matriz inconsistente.

En la literatura hay existen dos métodos para generar matrices ETC, el método basado en rangos (Braun et al.) y el método basado en coeficiente de variación (Ali et al.) [10]. Para generar matrices ETC con el método basado en rangos se procede de la siguiente manera: en primer lugar se genera un vector columna B con valores de punto flotante, estos valores se obtienen a partir de una distribución uniforme en el intervalo $[1,\phi_b)$, cada elemento en la fila i se crea a partir de B(i) multiplicando por un numero que se obtiene a partir de una distribución uniforme en el intervalo $[1,\phi_r)$, cada fila necesita un multiplicador por cada maquina.

Para generar matrices ETC con el método basado en coeficiente de variación se define el coeficiente de variación, V, de los valores de tiempos de ejecución como una medida de heterogeneidad. El coeficiente de variación de un conjunto de valores es una mejor medida de la dispersión en los valores que la desviación estándar, ya que expresa la desviación estándar como porcentaje de la media de los valores. Sean σ y μ la desviación standard y la media, respectivamente, de un conjunto de valores de tiempos de ejecución, se define $V = \sigma/\mu$. El procedimiento de generación es el siguiente: Se define un vector de tareas q, donde q[i] es el tiempo de ejecución de la tarea ti en una maquina “promedio” del entorno HC. Para generar q se necesitan dos parámetros σ_{task} y V_{task}. El valor del parámetro μ_{task} es usado para fijar el promedio de los valores en q. V_{task} cuantifica la heterogeneidad de las tareas. Cada elemento de vector de tareas q es usado para producir una fila de la matriz ETC utilizando el coeficiente deseado de variación, V_{mach}, de valores en cada una de las filas, este valor es otro parámetro de entrada.

Para variar la consistencia de las matrices en ambos métodos se procede de la misma manera: para generar matrices inconsistentes no se realiza ninguna modificación a los valores obtenidos con algunos de los métodos definidos anteriormente. Las matrices ETC consistentes pueden ser generadas desde matrices ETC inconsistentes ordenando los tiempos de ejecución para cada tarea en todas las máquinas (es decir, ordenando los valores dentro de cada fila y hacerlo de forma independiente todas las filas).

2.4 Trabajos relacionados

2.4.1 Estimating the Execution Time Distribution for a Task Graph in a Heterogeneous Computing System

Yan Alexander Li y John K. Antonio propusieron en 1997 [17] un modelo matemático para la estimación del tiempo de ejecución de un grafo de tareas, considerando elementos que agregan incertidumbre y afectan el tiempo de ejecución. En este trabajo se utiliza un enfoque probabilístico para estimar la distribución del tiempo de ejecución global de las tareas, esta distribución depende de varios aspectos como son: la distribución de tiempos de comunicación inter-maquinas, la dependencia de datos entre subtareas, el mapeo de subtareas a maquinas, la programación de las subtareas en una maquina.

Se consideran tres variables aleatorias separadas para cada subtarea: El tiempo de inicio, el tiempo de ejecución y el tiempo de finalización. Se considera que la derivación de la distribución exacta del tiempo de ejecución de la tarea es posible, pero no es claramente sistemática para automatizar.

Las subtareas que tienen algún predecesor común pueden tener una correlación en las variables aleatorias relacionadas con el tiempo de comienzo de inicio y fin. Sin embargo, existen condiciones en que estas variables pueden ser consideras como si no tuvieran correlación. Para esto se considera la aproximación de independencia de Kleinrock, el mismo concluye que la fusión de varios flujos de paquetes en una línea de transmisión tiene un efecto similar a restaurar la independencia de los tiempos de interarribo y largos de paquetes. Entonces se asume que la entrada de datos de muchas otras subtareas tiene el efecto de restaurar la independencia entre los tiempos de entrada y salida de las subtareas que tienen un predecesor común.

Para estimar la distribución de tiempo inicial y final se particionan las subtareas en capas. Se considera que una subtarea Sj es un predecesor inmediato de la subtarea Sg si: Sg requiere datos de Sj, o si Sj y Sg son asignadas a la misma maquina y Sj se ejecuta inmediatamente antes que Sg. Las subtareas que no tienen predecesor inmediato se ponen en la capa 1, en la capa k+1 se ponen las subtareas que el numero de capa más grande de sus predecesores inmediatos es k. Primero se computan la distribución de tiempo inicial y final de la subtarea, para cada subtarea de la capa 1. También se calcula la distribución para los tiempos en que cada ítem de datos esta disponible en la maquina destino. Esto se repite para las sucesivas capas 2 hasta llegar a las subtareas terminales (son las subtareas que no son predecesoras de ninguna subtarea). A partir de las distribuciones de los tiempos finales de las subtareas terminales se obtiene la función de distribución del tiempo de ejecución de la tarea entera.

Se utiliza un grafo de tarea particular para, mediante simulación, determinar su distribución de tiempo de ejecución y compararla con la distribución obtenida mediante el enfoque propuesto. Los resultados muestran que el enfoque propuesto predice la función de distribución del tiempo de ejecución con gran exactitud, incluso en los peores casos las aproximaciones obtenidas son bastante buenas.

2.4.2 A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems

En el año 1998 Tracy D. Braun, junto a un grupo de investigadores en las áreas de ingeniería en computación e ingeniería eléctrica, realizaron un trabajo [15] donde comparan el rendimiento de once heurísticas a la hora de determinar un mapeo de tareas, que componen una meta-tarea, a maquinas donde se minimice el tiempo de ejecución de la meta-tarea. En este trabajo se asume que las heurísticas obtienen el mapeo de forma estática. Como ya mencionamos anteriormente el problema del mapeo de tareas a maquinas consiste en la asignación de las tareas a las maquinas y la determinación de orden en que se ejecutaran las tareas asignadas a una maquina. En este trabajo se define meta-tarea como una colección de tareas independientes, o sea, que no tienen dependencias de datos.

En el modelo de simulación que se utiliza en este trabajo los tiempos de ejecución de cada tarea en cada maquina se conocen a priori, por esto se utilizan matrices ETC para simular los distintos escenarios. Como se menciono en la sección anterior las matrices ETC son generadas con el método basado en rangos, se consideran las doce matrices posibles combinando los distintos tipos de heterogeneidad de maquinas, heterogeneidad de tareas y consistencias. Las instancias en las que se trabajó aquí tienen 512 tareas y 16 maquinas.

Se consideran las heurísticas OLB, MET, MCT, Min-min y Max-min (que fueron explicadas anteriormente), además de otras heurísticas que se detallan a continuación.

Duplex. Ejecuta el min-min y el max-min, y se queda con la mejor solución obtenida.

GA (Genetic algorithms). Se ejecutó un algoritmo evolutivo para el dominio de este problema particular. En el próximo capitulo se presentara un análisis en profundidad de esta heurística.

SA (simulated annealing): es una técnica iterativa que considera solamente una posible solución por vez. La solución inicial se genera aleatoriamente mediante una distribución aleatoria uniforme. Se utiliza la misma mutación que en GA y se evalúa el makespan. Si el nuevo makespan es mejor entonces se sustituye la solución anterior con la nueva, si es peor se elige un numero aleatorio z $\in$ [0,1), luego se compara con *y* (que es un valor determinado a partir del valor del nuevo makespan, del makespan anterior y de un parámetro llamado temperatura), si z > y acepta la nueva solución, en caso contrario se mantiene la solución vieja.

GSA (The Genetic Simulated Annealing). Es una combinación del SA y GA. Es básicamente un GA pero que usa el SA para el proceso de selección.

Tabu. Se parte de una solución inicial y se va ejecuta un procedimiento de saltos cortos. Para realizar estos saltos cortos se consideran los vecinos de la solución actual. Los vecinos se obtienen de la siguiente manera: se considera parar todos los posibles pares de tareas, todos los posibles pares de maquinas. Para un par de tareas (t_i, t_j) y un par de maquinas (m_i,m_j) se reasigna t_i a m_i y t_j a m_j dejando la asignación del resto de las tareas como estaban. Se evalúa la nueva solución y si es mejor que la solución actual, se sustituye la solución actual por la solución nueva. Cada vez que se reemplaza una solución por otra mejor se incrementa el valor de la variable successful_hops. El procedimiento de saltos cortos puede finalizar porque: 1) todos las posibles combinaciones de pares de tareas y pares de maquinas han sido evaluadas, o 2) el limite de reemplazos ($limit_{hops}$) ha sido alcanzado, o sea successful_hops = = $limit_{hops}$. Cuando el procedimiento de saltos cortos termina se guarda la mejor solución obtenida en la lista tabu. Luego se genera una nueva solución aleatoria, esto se llama salto largo. Esta nueva solución debe diferir en por lo menos la mitad de las asignaciones. Para cada salto largo exitoso se ejecuta el procedimiento de saltos cortos. El criterio de parada de la heurística es cuando la suma de saltos cortos exitosos y saltos largos exitosos alcanza el límite de saltos.

*A**. Es una técnica de búsqueda basada en árboles m-arios, que comienza en el nodo raíz de que representa a una solución nula. A medida que el árbol crece los nodos representan asignaciones parciales. La solución parcial de un nodo hijo tiene una tarea más que la solución parcial que representa el nodo padre, sea la tarea adicional t_a. Cada nodo padre genera m nodos hijos que representan las posibles asignaciones de la tarea t_a a alguna maquina. Después de esto el nodo padre pasa a ser un nodo inactivo. Se establece un número máximo de nodos activos, para controlar el tiempo de ejecución del algoritmo. Cada nodo n tiene una función de costo f(n) asociado a el. Esa función de costo es el límite inferior estimado del makespan de la mejor solución que incluye a la asignación parcial que representa n. A medida que se van generando nodos hijos se van podando los nodos con valores de f(n) más grande, esto evita que se realice una búsqueda exhaustiva. La heurística continúa su ejecución hasta que llega a por lo menos una asignación completa.

Este trabajo permite realizar interesantes conclusiones acerca de las soluciones obtenidas por las distintas heurísticas y acerca de los tiempos de ejecución de las mismas. El objetivo de este trabajo fue dejar una base para comparar el

rendimiento de otras heurísticas con las once mencionadas anteriormente. Para las situaciones estudiadas GA fue la heurística con mejor rendimiento. El rendimiento medio de la relativamente simple heurística Min-min fue siempre en el entorno del 12% respecto a la heurística GA.

2.4.3 The Relative Performance of Various Algorithms is Independent of Sizable Variances in Run-time Predictions

El trabajo realizado por Robert Armstrong, Debra Hensgen, y Taylor Kidd [16] en 1998 describe experimentos y simulaciones ejecutadas para comparar el rendimiento de algunos algoritmos de mapeo en diferentes entornos heterogéneos. Se asume que las tareas son independientes entre si. Se quiere determinar si es beneficioso usar algoritmos de mapeo inteligentes, incluso si las tareas no se realizan exactamente en el tiempo esperado. La métrica es el tiempo de finalización de la última tarea (makespan). En los sistemas en que cada tarea usa exclusivamente una maquina, las diferencias ocurren por: i) todas las características de cómputo no son conocidas o enumeradas por el diseñador del programa, o ii) El tiempo de acceso a memoria o a disco es estocástico y no determinista.

Las heurísticas que se consideran en este trabajo son: OLB (opportunistic load balancing), LBA (limited best assignment), además se utilizaron dos algoritmos greedy.

Se estudian los efectos del no determinismo en la performance de los algoritmos, porque el LBA y los algoritmos greedy usan el tiempo esperado de ejecución para realizar sus mapeos. Se intenta determinar una distribución para los tiempos de ejecución de los programas. Primero se realizaron experimentos con programas reales para determinar distribuciones que pudieran caracterizar los tiempos de ejecución. Para esto se ejecutan programas en repetidas ocasiones para poder determinar algunas distribuciones realistas para realizar las simulaciones.

Luego de realizar estas ejecuciones se determina que las distribuciones más realistas son las distribuciones gaussiana y exponencial, por estos se generan tiempos de ejecuciones a partir de estas distribuciones, para probar el desempeño de las heurísticas mencionadas. Luego de realizar las simulaciones se observa que los algoritmos greedy obtienen mejores resultados que OLB y LBA tanto con tiempos de ejecuciones generados a partir de la distribución gaussiana como los generados a partir de la distribución exponencial.

2.4.4 A Taxonomy for describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems

En 1998 Tracy D. Braun y un grupo de investigadores [18] propusieron una taxonomia para poder comparar distintas heurísticas de mapeo de tareas a maquinas. Se considera que la comparación de heurísticas es complicada debido a que las mismas asumen distintas hipótesis acerca de la plataforma de destino, además tampoco existía un conjunto estándar de aplicaciones de referencia o plataforma para entornos HC.

La taxonomia propuesta utiliza tres componentes para describir las heurísticas: (1) el modelo de la aplicación y la caracterización de las comunicaciones, (2) las

características de la plataforma del modelo, y (3) las características de la estrategia de mapeo.

La caracterización del modelo de aplicación define los modelos utilizados para las aplicaciones que se ejecutan en el sistema HC y de las comunicaciones inter-maquina de la red. Las aplicaciones no son clasificadas por funcionalidad, sino por los rasgos que definen las características computacionales de la aplicación que pueden influir en las decisiones del mapeo. Además la taxonomia es capaz de incluir rasgos de la aplicación que sirven para simplificar las hipótesis de implementación. La definición de las características de las aplicaciones se basan en distintos elementos como son: tamaño de la aplicación, tipo de aplicación, patrones de comunicación, disponibilidad de datos, deadlines, modelo de tiempos de ejecución, heterogeneidad de la meta-tarea, múltiples versiones de las aplicaciones, prioridades de tareas, requerimientos de QoS de las tareas, heterogeneidad de las subtareas, distribución temporal (aplicaciones estáticas o dinámicas).

La caracterización del modelo de plataforma define las características del hardware disponible dentro del entorno HC. Un marco para la clasificación de los modelos utilizados consta de los siguientes elementos: parámetros analíticos (indican de que forma cada maquina disponible realizan cada tipo de cálculo), tiempo de comunicación, posibilidad de envío/recepción concurrente, interconexión de la red, arquitecturas de las maquinas, heterogeneidad de las maquinas, numero de conexiones, numero de maquinas, superposición computo/comunicación, sistema de control (entorno dedicado o compartido), y compatibilidad de tareas.

La caracterización de la estrategia de mapeo define las características utilizadas para describir la estrategia de mapeo. Dentro de los elementos a considerar se encuentran: modelo de aplicación soportado, tiempos de comunicación, ubicación del control (estrategia de mapeo centralizada o distribuida), transferencia de datos, dependencias entre las tareas consideradas, duplicación de tareas, mapeo dinámico o estático, ubicación de la ejecución (se refiere a si el mapeo lo ejecuta una maquina del entorno HC o si lo hace una maquina externa), tiempos de ejecución (se conocen a priori o no), consideración de la tolerancia a fallas, incorporación de feedback de las aplicaciones, función objetivo, modo de plataforma soportado, preventivo (si se pueden interrumpir aplicaciones ya iniciadas para asignar los recursos a aplicaciones más importantes), redistribución.

Con las tres categorías definidas en la taxonomia las técnicas de mapeo heterogéneas pueden ser clasificadas de forma más precisa. Esta clasificación permite a las investigaciones acerca de entornos HC describir con mayor facilidad las heurísticas de mapeo. Con esta taxonomia se podrá trabajar para generar casos de prueba estándar para los sistemas HC.

2.4.5 Efficient Batch Job Scheduling in Grids using Cellular Memetic Algorithms

Fatos Xhafa, Enrique Alba, y Bernabé Dorronsoro propusieron [19], en el año 2007, un algoritmo memetico celular para determinar un mapeo de tareas a maquinas. El modelo de simulación que se utiliza en este trabajo es el de matrices ETC (tiempo esperado de computo) propuesto en el trabajo de Braun et al. [15]. Se consideramos el mapeo de tareas como un problema de optimización bi-objetivo, en el que tanto el makespan como el flowtime son minimizados al mismo tiempo. Con el fin de hacer llevar a cabo la optimización simultánea de los objetivos su utiliza una suma ponderada del makespan y el flowtime.

En un cMA (algoritmo memetico celular) los individuos de la población están distribuidos espacialmente formando vecindades y los operadores evolutivos se aplican individuos vecinos. Para poder resolver el problema de mapeo de tareas a maquinas, se agrega conocimiento del problema a la estructura general del cMA. A continuación se describen algunos elementos que se utilizaron.

La topología de la población es una grid toroidal de dos dimensiones de pop_heigth x pop_width_size. Los patrones de vecindad utilizados son los siguientes: L5 (5 personas), L9 (9 personas), C9 (9 personas) y C13 (13 personas). Otro aspecto del algoritmo que se implemento en este trabajo es la actualización de las celdas, para esto se implementaron varios mecanismos asincrónicos como son: Fixed Line Sweep (FLS), Fixed Random Sweep (FRS) y New Random Sweep (NRS). La representación de las soluciones que se presenta en este problema es un vector de largo igual a la cantidad de tareas, donde la posición j indica la maquina donde fue asignada la tarea j. Para la inicialización de la población se obtiene un individuo a partir de la heurística Longest Job to Faster Resource – Shortest Job to Faster Resource (LJFR-SJFR) y el resto se obtiene a partir de este individuo mediante grandes perturbaciones. El operador de selección es un torneo N-ario. El operador de cruzamiento es el cruzamiento de un punto. El operador de mutación que se implemento esta basado en el rebalanceo de carga de la maquina más cargada. La estrategia de reemplazo es elitista, ya que un hijo reemplaza a su padre si tiene mejor valor de fitness. El último operador a considerar es el método de búsqueda local, que es una característica propia de los algoritmos memeticos, en este caso se utilizan tres métodos: Local Move (LM), Steepest Local Move (SLM) and Local Minimum Completion Time Swap (LMCTS).

A partir de este trabajo se puede afirmar que cMAs son una buena opción para la programación de tareas en grandes redes computacionales, ya que son capaces de ofrecer alta calidad en el mapeo en un tiempo muy corto.

2.5 Conclusiones

En este capitulo se presentan las ventajas de la utilización de entornos HC para la ejecución de una meta-tarea compuesta por tareas con distintas necesidades computacionales. Aquí se presentan dos ejemplos (BOINC y WCG) que muestran como se esta utilizando en la actualidad esta tecnología para la resolución de problemas muy complejos que requieren gran capacidad de computo.

Se puede observar que se ha trabajado en diversos aspectos relacionados con la computación heterogénea. En los trabajos relacionados que se presentan en este capitulo se abarcan temáticas que van desde la estimación de los tiempos de ejecución de tareas en maquinas hasta la comparación de distintos algoritmos que se utilizan para resolver la planificación de tareas en maquinas de un entorno HC. Un aspecto importante que se observa es que se ha trabajado fundamentalmente en el enfoque monoobjetivo del problema, ya sea considerando solamente una métrica (por ej. makespan) o la combinación lineal de varias métricas. Por lo tanto el enfoque multiobjetivo del problema presenta una nueva posibilidad de estudio del mismo, ofreciendo la oportunidad de obtener resultados de buena calidad.

Capítulo 3. Computación evolutiva

A partir de la década de 1970 se empezó a trabajar en base a la idea de emular los mecanismos de la evolución biológica para utilizarlo en la resolución de problemas. A partir de esta idea han surgido una gran cantidad de investigaciones, aplicaciones y propuestas de algoritmos en el área.

Dentro de la computación evolutiva han surgido una gran variedad de modelos computacionales que se conocen genéricamente como algoritmos evolutivos. Todos utilizan operadores de selección y reproducción basados en la evolución natural como base de su funcionamiento. Los operadores mencionados se basan en una función de fitness, que da una medida de que tan adaptada esta una solución al problema que se esta estudiando.

En este capitulo se comienza explicando los conceptos básicos de los algoritmos evolutivos. Luego se describe con mayor detalle una clase de algoritmo evolutivo: los algoritmos genéticos. También se mencionan diferentes modelos de evolución propuestos en la literatura. Por ultimo se desarrollan dos conceptos muy relevantes para este trabajo: los algoritmos evolutivos multiobjetivos y los algoritmos evolutivos paralelos.

3.1 Técnicas de computación evolutiva

Las técnicas de computación evolutiva son un conjunto de heurísticas que han sido utilizadas para la resolución de una amplia gama de problemas en las áreas de optimización combinatoria, diseño de artefactos, búsqueda de información, control de dispositivos y aprendizaje automático, etc. Estas técnicas basan su operativa en los mecanismos de la evolución natural, identificados por Charles Darwin en su célebre obra *El origen de las especies por medio de la selección natural: la selección natural, la reproducción y la diversidad genética de individuos* (Darwin, 1859).

Estas heurísticas trabajan sobre un conjunto de soluciones a un problema determinado (población), la metodología utilizada por estas técnicas se basa en el uso de mecanismos de selección de las mejores soluciones potenciales (padres) para luego construir nuevas soluciones candidatas (hijos) mediante la recombinación de características de las soluciones seleccionadas.

A continuación se presenta un mecanismo general de funcionamiento de algoritmo evolutivo que trabaja sobre una población P.

```
Inicializar(P(0))
generación=0
mientras (no CriterioParada) hacer
      Evaluar(P(generación))
      Padres = Seleccionar(P(generación))
      Hijos = Aplicar Operadores Evolutivos(Padres)
      NuevaPoblacion = Reemplazar(Hijos, P(generación))
      generación++
      P(generación) = NuevaPoblacion
Fin
retornar Mejor Solución Encontrada
```

Algoritmo 1 – Algoritmo Evolutivo Genérico

Un algoritmo evolutivo trabaja sobre individuos que representan soluciones al problema, estos individuos están codificados de acuerdo a un mecanismo establecido. Los individuos son evaluados de acuerdo a una función de fitness que determina que tan adecuada es cada solución para el problema que se intenta resolver.

El algoritmo evolutivo comienza con la generación de la población inicial, los individuos de esta población pueden ser generados de manera aleatoria o aplicando algún criterio de acuerdo a las características del problema a resolver. El algoritmo evolutivo podría inclusive tomar como población inicial a los individuos que se generen mediante alguna otra heurística de resolución del problema, que permita calcular buenas soluciones iniciales para el problema.

La evolución de la población se lleva a cabo en el ciclo donde se generan nuevos individuos a partir de la población actual, El procedimiento para generar los nuevos individuos aplica los operadores evolutivos definidos. En el ciclo hay cuatro etapas principales:

Evaluación: en esta etapa se le asigna un valor de fitness a cada individuo de la población. Este valor evalúa que tan bien resuelve cada individuo el problema en cuestión, y es muy importante en el desarrollo del algoritmo evolutivo.

Selección: en esta etapa se eligen los individuos (padres), de acuerdo a sus valores de fitness, para que mediante la aplicación de los operadores evolutivos generen los individuos de la nueva generación.

Aplicación de los operadores evolutivos: en etapa se genera el conjunto de descendientes (hijos) a partir de los individuos seleccionados en la etapa anterior (padres).

Reemplazo: en esta etapa se sustituyen los individuos de la generación anterior (padres) por los descendientes creados en la etapa anterior (hijos).

Se pueden aplicar distintas políticas de selección y reemplazo para modificar las características del algoritmo evolutivo. Por ejemplo se pueden privilegiar los individuos con mayor valor de fitness de cada generación (estrategias de elitismo), aumentar la presión selectiva sobre individuos mejor adaptados, generar un número reducido de descendientes en cada generación (modelos de estado estacionario), etc.

Los operadores evolutivos establecen como se explora el espacio de soluciones del problema. Han surgido una gran cantidad de operadores evolutivos, sus características son las que determinan las distintas variantes de algoritmos

evolutivos. Los operadores evolutivos más difundidos son los operadores de cruzamiento y de mutación.

La condición de parada de la fase iterativa es, generalmente, alcanzar un numero determinado de generaciones. Otras alternativas consideran como varían los valores de fitness o el valor estimado de error respecto al valor optimo del problema o una aproximación.

3.2 Algoritmos genéticos

Los algoritmos genéticos son una de las técnicas de computación evolutivas más utilizadas en la actualidad, ya que es una técnica muy flexible y permite resolver una amplia variedad de problemas. Como el resto de las técnicas evolutivas, los algoritmos genéticos basan su funcionamiento en la evolución natural de los seres vivos, estos algoritmos trabajan sobre una población de soluciones que va evolucionando mediante interacciones y transformaciones únicas. La selección utilizada en el proceso evolutivo determina los individuos que estarán presentes en la siguiente generación, esta selección prioriza los individuos más aptos. La aptitud de los individuos se evalúa de acuerdo al problema a resolver, para esto se define una función adecuada al problema, que es llamada función de fitness. En determinadas condiciones este mecanismo lleva a la población a converger hacia una solución cercana al óptimo del problema, esto luego de ejecutar un determinado número de generaciones.

La formulación clásica de los algoritmos genéticos se basa en el esquema genérico de un algoritmo evolutivo que se presenta en Algoritmo 1 – Algoritmo Evolutivo Genérico. Partiendo de ese esquema define los siguientes operadores evolutivos: el operador de cruzamiento que se encarga de combinar a los individuos, y el operador de mutación que se encarga de realizar una variación aleatoria a los individuos con el fin de proporcionar diversidad.

A continuación se presenta un esquema de un algoritmo genético.

```
Inicializar(P(0))
generación=0
mientras (no CriterioParada) hacer
      Evaluar(P(generación))
      Padres = Seleccionar(P(generación))
      Hijos = Aplicar Recombinación(Padres)
      Hijos = Aplicar Mutación(Padres)
      NuevaPoblacion = Reemplazar(Hijos, P(generación))
      generación++
      P(generación) = NuevaPoblacion
Fin
retornar Mejor Solución Encontrada
```

Algoritmo 2 – Esquema de un Algoritmo Genético

Una característica que distingue a los algoritmos genéticos del resto de las técnicas evolutivas es que utilizan el operador de cruzamiento como operador principal, mientras que la mutación se utiliza como operador secundario para agregar diversidad la exploración del espacio de soluciones. Incluso en algunas variantes de algoritmos genéticos no se incluye el operador de mutación y usan otros operadores para introducir diversidad.

En general se trabaja con codificaciones binarias para problemas de búsqueda en espacios de cardinalidad numerable, también se han propuesto codificaciones reales, e incluso se han propuesto codificaciones no tradicionales, dependientes de los problemas a resolver.

Un mecanismo de selección utilizado es el llamado selección proporcional, donde la cantidad de copias de individuos a considerar en el cruzamiento es proporcional a sus valores de fitness. También existen otros mecanismos de selección utilizados para modificar el proceso de exploración del espacio de soluciones.

3.2.1 Representación de soluciones

Los algoritmos genéticos no trabajan directamente sobre las soluciones del problema, sino que trabajan sobre una representación abstracta de dichas soluciones, estas representaciones generalmente son llamadas cromosomas. Un cromosoma es un vector de genes, el valor asignado a un gen se denomina alelo.

Se llama genotipo a las características de un individuo. El genotipo sometido al medio ambiente se llama fenotipo. En un algoritmo genético el genotipo esta construido por cromosomas, generalmente se utiliza un cromosoma por individuo. El fenotipo representa un punto del espacio de soluciones del problema.

Por lo mencionado anteriormente se debe definir una función de codificación sobre los puntos del espacio de soluciones, que establece la correspondencia entre los puntos del espacio de soluciones y los genotipos. La función inversa de la codificación es la función de decodificación que devuelve el fenotipo asociado a un cromosoma.

La codificación de los individuos es importante para el proceso de búsqueda de los algoritmos genéticos. Por lo general utilizan codificaciones binarias de largo fijo. Los individuos se codifican generalmente con vectores de valores binarios conocidos como string de bits.

Se han utilizado otros esquemas de codificación con menor frecuencia. Por ejemplo se han utilizado codificaciones basadas en números reales cuando se resuelven problemas en espacios de soluciones no numerables, por ejemplo en el caso de determinación de parámetros de control o entrenamiento de redes neuronales. Otro ejemplo de codificación son las basadas en permutaciones de enteros, se utilizan en problemas donde se buscan hallar ordenamientos óptimos. Otras codificaciones dependientes de los problemas son utilizadas con frecuencia, con el fin de incorporar conocimiento específico en la resolución de problemas complejos.

3.2.2 Función de fitness

Todos los cromosomas tienen un valor de fitness asociado, este valor evalúa la aptitud del individuo para resolver el problema que se esta evaluando. La función de fitness tiene el mismo tipo que la función objetivo del problema, por esto el cálculo de la función de fitness se realiza sobre el fenotipo del cromosoma.

La función de fitness guía el mecanismo de exploración, ya que representa al entorno que evalúa la aptitud del individuo solución para la resolución del problema.

La función de fitness debe tener en cuenta el criterio de optimización del problema, esto es si se trata de la minimización o maximización de un objetivo, y las restricciones que presenta el problema. La función de fitness debe asegurar que las soluciones no factibles que aparezcan no se perpetúen en el proceso evolutivo.

Si el entorno presenta problemas para la evaluación de la función de fitness, se tienen que implementar mecanismos para solucionar este problema como pueden ser evaluaciones parciales o funciones de fitness multivaluadas que asignen diferentes valores de fitness a un individuo para simplificar la labor del operador de selección.

Si las funciones de fitness varían dinámicamente durante la evolución del algoritmo genético se deben implementar mecanismo complejos, como las representaciones múltiples, para introducir memoria en la operativa del algoritmo genético.

Generalmente la función de fitness requiere un esfuerzo computacional mayor que el necesario para ejecutar el resto de los operadores evolutivos. Incluso puede darse el caso de que solamente se puedan obtener valores aproximados en tiempos razonables.

En el caso de problemas con objetivos múltiples, la función de fitness deberá contemplar a todos los objetivos. Otros aspecto a considerar es cuando se presentan problemas de dominancia de soluciones muy adaptadas en generaciones tempranas de la evolución, deben considerarse mecanismos de escalado del fitness para evitar el estancamiento de la evolución.

3.2.3 Operadores evolutivos

Los principales operadores que presentan los algoritmos evolutivos son el operador de selección, el operador de cruzamiento y el operador de mutación.

La selección es quien se encarga de que las buenas características de los individuos se mantengan, ya que las buenas características están presentes en los individuos más adaptados. En la literatura se pueden encontrar varios ejemplos de operadores de selección, aquí se detallan algunos. La selección proporcional o por ruleta elige individuos aleatoriamente, pero la probabilidad que tiene un individuo de ser elegido aumenta según su valor de fitness. Otros mecanismos de selección aplican criterios elitistas, conservando un número determinado de los mejores individuos a través de las generaciones. En la selección por torneo se eligen aleatoriamente un numero de individuos, que compiten entre ellos para determinar se seleccionaran para reproducirse, se seleccionan en base al valor de fitness de los mismos. La selección basada en el rango introduce el mayor elitismo posible, manteniendo un alto porcentaje de los mejores individuos de la población.

La codificación binaria de largo fijo permite definir operadores evolutivos simples. Por ejemplo el operador de cruzamiento de un punto, este operador obtiene dos hijos a partir de dos padres seleccionando un punto de corte al azar, luego corta a los cromosomas de los padres e intercambia los trozos de cromosomas, generando un hijo con la primer mitad del primer padre y la segunda mitad del segundo padre, y el hijo con la segunda mitad del primer padre y la primer mitad del segundo padre.

El operador de mutación es el encargado de introducir diversidad en la ejecución del algoritmo evolutivo. El operador más tradicional es el que modifica

aleatoriamente uno de los valores binarios del cromosoma. En un esquema de codificación binaria se invierte el valor de un alelo.

Los operadores de cruzamiento y de mutación son operadores probabilísticos, porque se aplican o no teniendo en cuenta una probabilidad de aplicación del operador. Por lo general se establece una probabilidad de cruzamiento alta, entre 0,5 y 0,9, y una probabilidad de mutación muy baja, por lo general es de 0,001 por cada bit en la representación.

En la literatura se han propuesto operadores evolutivos más complejos, con el objetivo de modificar la forma en que se explora el espacio de soluciones. Por ejemplo se proponen operadores de cruzamiento multipunto donde se utilizan dos o más puntos de corte para realizar el intercambio de trozos de cromosomas. Otro operador de cruzamiento es el cruzamiento uniforme, en este operador se determina para cada posición en el cromosoma se decide si se intercambia material genético de acuerdo a una probabilidad establecida.

También existen otros operadores evolutivos para los otros tipos de codificaciones como son las codificaciones reales o las permutaciones de enteros.

3.3 Modelos de evolución

Cuando se habla de modelos de evolución se hace referencia a la forma que utiliza un algoritmo evolutivo para realizar la renovación de la población con la que esta trabajando.

Tradicionalmente en los algoritmos evolutivos el paso evolutivo involucra la generación de una nueva población que sustituya a la población actual mediante la aplicación de los distintos operadores evolutivos explicados previamente.

Se dice que el paso atómico de la evolución, en este caso, es una generación. A partir de esta apreciación es que se denomina modelo generacional al modelo de ejecución de la evolución de los algoritmos genéticos tradicionales.

Una alternativa al modelo de evolución anteriormente mencionada, es el modelo de evolución de algoritmos evolutivos de estados estacionarios. En este modelo el paso atómico de la evolución es la generación de un único individuo.

En la programación genética se propuso un modelo de evolución en el que cada individuo generaba un hijo y luego se seleccionaban entre los padres y los hijos a los mejores individuos para formar la población. En las estrategias de evolución se puede optar por un modelo de evolución de poblaciones solapadas o un modelo de evolución de poblaciones no solapadas. En un modelo de poblaciones solapadas los padres y sus hijos compiten por ser parte de la población de la próxima generación, mientras que en modelo de poblaciones no solapadas los hijos sustituyen a los padres.

3.3.1 Modelo de estado estacionario

En este modelo de evolución en cada paso se seleccionan dos padres para aplicarles el operador de cruzamiento y generar un, o eventualmente dos, hijo/s. A los hijos se les pueden aplicar un operador de mutación de acuerdo a una probabilidad de mutación. Estos individuos que se generan se agregan a la población con la que se esta trabajando, para esto se debe determinar los individuos que se deben sacar de

la población. En este modelo de evolución la estrategia que se utiliza para reemplazar individuos de la población por los individuos nuevos que se generaron, tiene una mayor importancia que en un algoritmo evolutivo generacional.

Se pueden definir varias estrategias de reemplazo, por ejemplo, estas estrategias pueden ser aleatorias, proporcionales al fitness o elitistas. Normalmente se definen estrategias donde hay una competencia entre el descendiente generado y el individuo a reemplazar.

Este mecanismo de reproducción paso a paso fue inicialmente propuesto para evitar inconvenientes en la resolución de problemas de entrenamiento de redes neuronales. Luego se extendió para varios problemas y áreas de aplicación.

Este modelo mantiene el equilibrio entre los mecanismos de exploración y de explotación del algoritmo evolutivo. El algoritmo evolutivo se encarga de mantener las características de las poblaciones, garantizando una buena exploración del espacio de soluciones, y a la vez se realiza una explotación local que presiona hacia los individuos mejor adaptados. En este modelo de evolución son muy utilizadas las técnicas de selección elitistas, tanto el elitismo propiamente dicho como la selección por ranking.

Cabe aclarar que si siempre se reemplaza al peor individuo, esto puede generar convergencia prematura. El uso de estrategias de reemplazo aleatorias en general funciona correctamente, pero en contrapartida se obtiene una convergencia más lenta. Otra opción es utilizar un reemplazo probabilístico que sea inversamente proporcional al fitness, esto disminuye los efectos adversos mencionados anteriormente.

La mayor diferencia entre el modelo de estado estacionario y el modelo generacional es la cantidad de selecciones que realizan ambos, ya que el modelo generacional requiere realizar muchas más selecciones que el modelo de estado estacionario. Por esto la perdida de información o deriva genética será mayor en el caso del modelo generacional. También se observa que el algoritmo de estado estacionario tendrá tiempos de ejecución menores que el algoritmo generacional, pero a largo plazo el algoritmo generacional puede encontrar mejores soluciones que el algoritmo de estado estacionario, ya que el primero tiene un mejor patrón de exploración del espacio de búsqueda.

3.3.2 Otros modelos de evolución

En la literatura podemos encontrar otros modelos de evolución propuestos, para algoritmos evolutivos no tradicionales y para algunas variantes de los algoritmos genéticos. Dentro de estas propuestas podemos encontrar el modelo de GAP generacional, messy GA (algoritmo genéticos desordenados), entre otros.

El modelo de GAP generacional fue propuesto por De Jong, este es un modelo intermedio en el que se define un porcentaje de la población que se utiliza para participar en la generación de la nueva población, a este porcentaje se le denomina gap. Esta propuesta se puede ver como una generalización de los modelos de evolución antes presentados, ya que incluye tanto al enfoque del modelo generacional (cuando el gap es el 100%) como al modelo de estado estacionario (cuando el gap determina que solo se consideren dos individuos de la población). Otro aspecto que se puede definir en este modelo es el porcentaje de población que se solapa entre generaciones.

Por otro lado los messy GA (algoritmos genéticos desordenados) tienen una fase primordial donde crean la población utilizando una estrategia de enumeración parcial, distribuyendo bloques de construcción de soluciones y una fase de yuxtaposición donde se mezclan las soluciones encontradas en la fase primordial.

3.4 Algoritmos evolutivos multiobjetivos

En esta sección se describen los aspectos relacionados con los problemas de optimización multiobjetivo (MOP). En primer lugar se da una definición formal de dichos problemas y algunos conceptos relacionados a los mismos. Luego se presentan los algoritmos evolutivos para optimización multiobjetivo como un mecanismo eficiente para la resolución de los MOP, también se presentan diferentes propuestas surgidas a lo largo de la investigación en esta área. Por ultimo se explican las distintas métricas que se utilizan para evaluar a los resultados obtenidos por los algoritmos evolutivos.

3.4.1 Problemas de optimización multiobjetivo

En los problemas de optimización multiobjetivo se busca una o más soluciones que satisfagan de forma simultánea todas las restricciones, y optimicen una determinada función que relaciona el espacio de decisión al que pertenecen las soluciones con el espacio objetivo. Para un problema con $K \geq 2$ objetivos, se busca minimizar o maximizar la función objetivo f: minimizar/maximizar $(f_k(s)), \forall k \in [1,k]$ [32].

Las variables de decisión de un MOP son representadas por un vector que tiene los diferentes elementos que se consideran en el problema. Cada variable de decisión representa las cualidades numéricas para el problema en cuestión.

La función objetivo *f* en un MOP transforma las variables de decisión desde en espacio de búsqueda hacia un espacio objetivo de *K* dimensiones $Z \in R^K$, siendo $z = f(s), f(s) = \{f_1(s), f_2(s),.., f_K(s)\}, z \in Z$ [32].

Se establece una relación de dominancia de Pareto de la siguiente manera: sean dos soluciones *s* y *s'*, consideramos que estamos en el contexto de un problema de minimización, se dice que *s* domina a *s'* ($s \prec s'$) sii $\forall k \in [1,K] \exists k' \in [1,K]: f_k(s) \leq f_k(s') \wedge f_{k'}(s) \neq f_{k'}(s')$, se dice que *s* y *s'* son indiferentes o incomparables ($s \sim s'$) si *s* no domina *s'* y *s'* no domina a *s*.

Una solución s se dice que es un optimo de Pareto sii no existe otra solución s' talque $s' \prec s$. El conjunto de todas las soluciones que son óptimos de Pareto definen el frente de Pareto.

Las soluciones que pertenecen al frente de Pareto no pueden ser mejoradas en ningún objetivo sin degradar al menos otro objetivo, por esto se consideran como óptimos globales. También existen óptimos locales que son soluciones no dominadas en ciertas áreas locales [32].

Es importante observar que a la hora de diseñar un algoritmo Multiobjetivo se deben considerar varios aspectos: hay que minimizar la distancia entre las soluciones no dominadas encontradas con el frente de Pareto, las soluciones

encontradas deben abarcar toda la región del frente de Pareto, la distribución de las soluciones en el frente de Pareto debe ser lo más uniforme posible.

3.4.2 AE para optimización multiobjetivo (MOEA)

Los algoritmos evolutivos se encuentran dentro de las mejores técnicas metaheurísticas para explorar el frente Pareto-optimo. En el año 1967 Rosenberg proponía el uso de los algoritmos evolutivos para resolver problemas de optimización multiobjetivo (MOP). A pesar de esto la primer implementación de un algoritmo evolutivo aplicado a resolver un MOP fue realizada en 1984 por Schaffer, cuando implemento el algoritmo VEGA. Más adelante se detalla dicha propuesta [32].

La utilización de este enfoque ofrece varias ventajas: la posibilidad de trabajar en paralelo sobre un conjunto de soluciones brinda la potencialidad de tratar problemas con múltiples objetivos de forma de determinar un conjunto de soluciones aproximadas al frente de Pareto en cada ejecución, mientras que muchas técnicas de cálculo numérico obtienen solo una solución aproximada al frente de Pareto por ejecución. Además, los algoritmos evolutivos son menos sensibles a la forma o continuidad del frente de Pareto que dichas técnicas de cálculo numérico. También vale destacar que los MOEAs permiten trabajar con problemas cuyo espacio de soluciones es de gran dimensión.

Los principales propósitos que se pueden destacar de los MOEAs son: aproximarse al frente de Pareto, muestrear adecuadamente el frente del Pareto buscando distintas soluciones que representen diferentes compromisos entre las funciones a optimizar. Otro aspecto importante es obtener buenos valores de eficiencia computacional.

A continuación se muestra un esquema genérico de un MOEA, donde se observa la aplicación de dos operadores evolutivos que no se utilizan en los algoritmos evolutivos simples, estos operadores son el operador de diversidad y el operador de asignación de fitness. El operador de diversidad se utiliza para evitar la convergencia a un sector del frente de Pareto. El operador de asignación de fitness busca dar más chance de perpetuarse a los individuos con mejores características, considerando tanto las funciones objetivos como la métrica que se utiliza para evaluar la diversidad. No necesariamente todas las propuestas de MOEAs siguen este esquema.

```
generación=0
Inicializar(P(generación))
mientras (no CriterioParada) hacer
      Evaluar(P(generación))
      Operador de Diversidad(P(generación))
      Asignar Fitness(P(generación))
      Padres = Seleccionar(P(generación1))
      Hijos = Operadores de Reproducción(Padres)
      NuevaPop = Reemplazar(Hijos, P(generación))
      generación++
      P(generación) = NuevaPop
Fin
retornar Mejor Frente Encontrado
```

Algoritmo 3 – Esquema de un MOEA

Es habitual clasificar los MOEAs de acuerdo con el mecanismo que utilizan en la asignación del fitness, se distinguen entre non-Pareto based MOEAs y Pareto based MOEAs. Los primeros proponen mecanismos de asignación de fitness relativamente sencillos, que no reflejan la independencia entre los objetivos del problema y en general no garantizan una correcta resolución del MOP. Los segundos en cambio utilizan explícitamente la dominancia de Pareto en la asignación del fitness, lo que garantiza la independencia entre los objetivos del problema.

Otra clasificación posible para los MOEAs es una clasificación cronológica, dividiendo las propuestas de MOEAs en dos generaciones. Ambas clasificaciones se detallan en las próximas secciones, además se presentan propuestas de MOEAs que corresponden a cada una de las generaciones mencionadas anteriormente.

- **Non Pareto based MOEAs**

Como se mencionó anteriormente estas propuestas utilizan mecanismos de asignación de fitness relativamente sencillos, que no están basados en la dominancia de Pareto. En la practica se ha comprobado que son adecuados para problemas con no más de tres funciones objetivo.

Se han propuesto varios mecanismos de asignación de fitness, a continuación se detallan algunos de ellos.

Uno de estos mecanismos es la agregación o combinación lineal de los objetivos. Aquí el fitness se calcula como una suma ponderada de las funciones objetivo, para esto se le asigna a cada objetivo un peso a priori. Este mecanismo no refleja el carácter multiobjetivo del problema, ya que se procede como si se tratará de un problema de optimización monoobjetivo.

Otro mecanismo es el ordenamiento lexicografico, en este mecanismo se establece una jerarquía entre las funciones objetivo según su importancia. Se resuelve considerando las funciones objetivos en el orden establecido en la jerarquía, es necesario agregar restricciones ficticias para las funciones objetivo resultas previamente.

Por su parte el mecanismo de ε-constraint optimiza una función objetivo y considera al resto de las funciones objetivo como restricciones acotadas por ciertos niveles permisibles a las que se denomina ε_i

Otro mecanismo propuesto es el target vector optimization donde se busca alcanzar un vector ideal mediante la combinación de estrategias de optimización. Este mecanismo es utilizado por el algoritmo VEGA, que será explicado más adelante.

- **Pareto based MOEAs**

Cuando se habla de Pareto based MOEAs, se hace referencia a algoritmos que utilizan explícitamente la dominancia de Pareto para realizar la asignación de fitness. Generalmente utilizan mecanismos para preservar la diversidad. Dentro de los mecanismos utilizados para preservar la diversidad se encuentran: fitness sharing, restricción al cruzamiento, crowding, entre otros.

El método de fitness sharing le asigna un mayor valor de fitness a las soluciones que están en nichos menos poblados, esto permite que se mantenga la diversidad de la población, ya que entre dos soluciones con similar fitness tendrá más chance de ser elegida aquella que se encuentre en el nicho menos poblado.

En cambio la restricción al cruzamiento intenta preservar la diversidad evitando que se crucen dos soluciones muy similares, cuya distancia sea menor que un valor σ_{MATING}.

3.4.3 MOEAs de primera generación

Las primeras propuestas de MOEAs utilizaban agregación de funciones o estrategias basadas en orden lexicografico. En 1985 se propone un algoritmo llamado Vector Evaluated Genetic Algorithms (VEGA), que basaba su selección en optimizar un objetivo por vez. Luego en 1989 Golberg [20] propone utilizar estrategias de asignación de fitness basadas en el ranking de Pareto. También propuso utilizar la ordenación por dominancia de Pareto y el fitness sharing para evitar el problema de la convergencia prematura. La primer generación de MOEAs comienza en el año 1994 y se extiende hasta el año 1999. Los MOEAs de esta generación se caracterizaban también por el empleo de nichos para proveer diversidad, y en general no utilizaban estrategias de elitismo. A continuación se describen algunos MOEAs de la primer generación, y además se presenta el algoritmo VEGA que es considerado un antecedente de los mismos.

- **VEGA - Vector Evaluated Genetic Algorithms**

En 1985 Schaffer propuso una extensión del AG simple para contemplar las funciones de fitness vectoriales, a dicha extensión la que llamó Vector Evaluated Genetic Algorithm (VEGA). El operador de selección fue modificado para que, en cada generación, un número de sub-poblaciones sea generado por la realización de selección proporcional de acuerdo a cada función objetivo por turno. Así, en un problema con q funciones objetivo, se generaban q subpoblaciones de tamaño N/q, suponiendo que el tamaño de la población = N. Luego se mezclan los individuos de las distintas sub-poblaciones y se aplican los operadores evolutivos de la forma habitual de los algoritmos genéticos simples para generar la nueva población de tamaño N.

En el año 1989 Richardson et al. observaron que la selección propuesta en este algoritmo era equivalente a realizar una combinación lineal de las componentes del vector de la función de fitness para obtener un único valor de fitness, aunque los coeficientes de ponderación de esta combinación lineal dependen de la población actual.

Esto significa que, en el caso general, que la población tenderá a dividirse en especies diferentes, cada una de ellas particularmente fuerte en uno de los objetivos. Schaffer anticipó esta propiedad de VEGA y la llamó especialización. La especialización es indeseable en caso de que se busquen soluciones de compromiso entre los distintos objetivos a optimizar.

Para evitar la combinación de objetivos de cualquier manera se requiere un enfoque diferente para la selección. En 1989 Goldberg propuso modificar el operador de selección de VEGA por uno basado en el ranking de los individuos, el ranking del individuo i depende de la cantidad de individuos que dominan al individuo i [22].

- **NSGA - Non-dominated Sorting Genetic Algorithm**

Srinivas y Deb trabajaron más directamente en la idea de Goldberg. La idea detrás del NSGA es usar un método de selección basado en el ranking para dar mayor chance de ser seleccionado a los individuos no dominados de la población actual y usar una función de sharing para mantener la diversidad de la población.

NSGA difiere de un algoritmo genético simple solamente en la forma de utilizar el operador de selección. Los operadores de cruzamiento y mutación siguen funcionando de la manera en que lo hacen en un algoritmo genético simple.

Antes de aplicar el operador de selección se realizan dos procedimientos de forma serial. Primero, Se calcula el rango de los individuos de la población en base a un nivel de no-dominancia de ellos y luego se utiliza la función de sharing para asignar el fitness a cada individuo.

Para calcular el rango de los individuos se procede de la siguiente manera: Todas las soluciones no dominadas constituyen el primer frente no-dominado de la población. Con el fin de encontrar las soluciones pertenecientes al segundo nivel de no-dominación, no se consideran las soluciones del primer frente y se buscan las soluciones no dominadas entre las restantes. El procedimiento se repite hasta que todas las soluciones de la población se clasifiquen con un nivel de no dominancia. Es importante señalar que este procedimiento tiene una complejidad computacional de O (N^2).

```
generación=0
Inicializar(P(generación))
Evaluar(P(generación))
AsignarRanking(P(generación))
Calcular Cardinal Nicho(P(generación))
AsignarSharedFitness(P(generación))
mientras (no CriterioParada) hacer
      generación++
      Padres = Seleccionar(P(generación-1))
      P(generación)= Cruzamiento(Padres)
      Mutar(P(generación))
      Evaluar(P(generación))
      AsignarRanking(P(generación))
      Calcular Cardinal Nicho(P(generación))
      AsignarFitnessLineal(P(generación))
      AsignarSharedFitness(P(generación))
      SeleccionarSobrevivientes(P(generación))
Fin
retornar Mejor Frente Encontrado
```

Algoritmo 4 – Esquema del NSGA

En NSGA, el fitness es asignado a cada individuo de acuerdo a su nivel no-dominancia. A un individuo con alto nivel se le asigna un fitness bajo. Esto se hace con el fin de mantener una presión de selección para elegir las soluciones de los niveles inferiores de la no-dominación.

La asignación del fitness se realiza en dos etapas. Primero, a todas las soluciones que están en un mismo nivel de no-dominancia se les asigna un mismo valor de dummy fitness. Luego, basado en el crowding de las soluciones en el frente, se hace hincapié en las soluciones más solitarias usando la función de sharing[25].

- **NPGA - Niched Pareto Genetic Algorithm**

Horn, Nafploitis, and Goldberg propusieron en 1994 [26] un método que utiliza torneos basados en la dominancia de Pareto en lugar de un método de selección basado en el rango de los individuos. En este método, al inicio de cada proceso de selección se elige al azar un conjunto de determinada cantidad (t_{dom}) de los individuos de la población.

Dos individuos de la población son escogidos al azar para elegir un ganador de acuerdo con el siguiente procedimiento. Ambos individuos se comparan con los miembros del conjunto inicialmente determinado para establecer la dominación respecto a las funciones objetivo. Si uno de ellos es no dominado y el otro es dominado, el individuo no dominado es seleccionado.

Por otro lado, si ambos son no-dominados o dominados, se determina el cardinal de nicho (niche count) se calcula para cada individuo de la población entera. El niche count de un individuo se calcula simplemente contando el número de individuos de la población que están a una determinada distancia (σ_{share}). Luego se selecciona el individuo con menor niche count.

La elección del parámetro t_{dom} es importante para el desempeño del algoritmo, si es muy pequeño pueden seleccionarse individuos no dominados, si es muy grande puede obtenerse una convergencia prematura en el algoritmo [25].

```
generación=0
Inicializar(P(generación))
Evaluar(P(generación))
Calcular Cardinal Nicho(P(generación))
mientras (no CriterioParada) hacer
      generación++
      mientras no padres no esta completo
            elegir Dos Individuos y conjunto
            si ind1 domina a todo x ∈ conjunto
                  incluir ind1 en padres
            sino y si ind2 domina a todo x ∈ conjunto
                  incluir ind2 en padres
            sino
                  incluir ind con menor niche count
            fin si
      fin mientras
      P(generación)= Cruzamiento(Padres)
      Mutar(P(generación))
      Evaluar(P(generación))
      Calcular Cardinal Nicho(P(generación))
Fin
retornar Mejor Frente Encontrado
```

Algoritmo 5 – Esquema de NPGA

- **MOGA - MultiObjective Genetic Algorithm**

El primer algoritmo evolutivo multiobjetivo basado en frentes de Paretos que fue publicado fue el MOGA (multiobjective genetic algorithm), este fue propuesto por Fonseca y Fleming en el año 1993 [24] .

Aquí se implementa una variante de la jerarquización de Pareto propuesta por Goldberg. Aquí la población entera es marcada y a todos los individuos no dominados se le asigna ranking =1. El resto de los individuos se clasifican controlando la no dominancia de ellos respecto al resto de la población de la siguiente manera. Para un individuo solución, en primer lugar se busca el número de soluciones que este domina estrictamente en la población. Posteriormente, el rango que se le asigna a ese individuo es dicho número más uno. Al final de este proceso de ranking, pueden existir varios individuos con un mismo rango. El procedimiento de selección utiliza este rango para seleccionar individuos para generar la población de la generación siguiente. Este mecanismo puede producir una gran presión en la selección, lo que puede causar que se produzca convergencia prematura. MOGA utiliza un método de formación de nichos para distribuir la

población sobre la región del óptimo de Pareto. En lugar de realizar sharing en los valores de los parámetros, se utiliza en los valores de las funciones objetivos. A pesar de mantener la diversidad en los valores de la función objetivo, no se mantiene la diversidad en el conjunto de parámetros, que es una cuestión importante para una toma de decisiones. Además, MOGA puede no ser capaz de encontrar soluciones múltiples en problemas donde diferentes puntos pertenecientes al frente de Pareto corresponden a un mismo valor en alguna de las funciones objetivo [25].

```
generación=0
Inicializar(P(generación))
Evaluar(P(generación))
Calcular Cardinal Nicho(P(generación))
AsignarFitnessLineal(P(generación))
AsignarSharedFitness(P(generación))
mientras (no CriterioParada) hacer
      generación++
      Padres = Seleccionar(P(generación-1))
      P(generación)= Cruzamiento(Padres)
      Mutar(P(generación))
      Evaluar(P(generación))
      Calcular Cardinal Nicho(P(generación))
      AsignarFitnessLineal(P(generación))
      AsignarSharedFitness(P(generación))
      SeleccionarSobrevivientes(P(generación))
Fin
retornar Mejor Frente Encontrado
```

Algoritmo 6 – Esquema del MOGA

3.4.4 MOEAs de segunda generación

La segunda generación de MOEAs comienza en el año 1999 y se extiende hasta la actualidad. La característica fundamental de los algoritmos de esta generación consiste en la incorporación del elitismo en su ejecución. El elitismo fue incluido mediante la utilización de métodos de selección (μ+λ), o mediante el uso de una población externa en la que se almacenan todas las soluciones no dominadas que el algoritmo va encontrando.

Varios MOEAs han implementado exitosamente la utilización de una población externa. En este enfoque hay que considerar varios aspectos: la interacción entre la población y la población externa, los criterios para realizar las inserciones de elementos en la población externa, y las acciones a tomar cuando se completa la población externa.

A continuación se describen algunas propuestas de MOEAs de segunda generación.

- **SPEA - Strength Pareto Evolutionary Algorithm**

En 1999 Zitzler y Thiele propusieron [27] un Algoritmo evolutivo multi-objetivo elitista con el concepto de no-dominancia. Se propone mantener una población externa en cada generación donde se almacena el conjunto de las soluciones no-dominadas descubiertas desde la población inicial hasta la población actual.

La población externa participa en las operaciones genéticas. El fitness de cada individuo de la población actual o de la población externa es asignado en base al número de soluciones dominadas (Strength del individuo). Para esto se genera una población combinando la población externa con la población actual, luego a todos

los individuos no dominados de la población combinada se les asigna un fitness basado en el número de soluciones que domina.

Para mantener la diversidad se le asigna mayor fitness a las soluciones no dominadas que dominan más soluciones en la población combinada. Por otro lado, se le asigna un mayor fitness a las soluciones dominadas por más soluciones en la población combinada. El peor fitness de una solución no dominada no puede ser menor que el mejor fitness de una solución dominada.

Esta asignación del fitness asegura que la búsqueda se orientara hacia las soluciones no dominadas y simultáneamente a la diversidad ya que se mantienen soluciones dominadas y no-dominadas.

Para mantener la población externa se utilizan técnicas de clustering, de manera de quitar individuos similares[25].

```
generación=0
Inicializar(P(generación))
Inicializar(PobExterna)
Evaluar(P(generación))
Calcular Strength(P(generación))
mientras (no CriterioParada) hacer
      copiarNoDominados(P(generación), PobExterna)
      SacarNoDominados(PobExterna)
      Clustering(PobExterna)
      Padres = Seleccionar(P(generación)U PobExterna )
      P(generación+1) = Cruzamiento(Padres)
      Mutar(P(generación + 1))
      SeleccionarSobrevivientes(P(generación+1))
      Evaluar(P(generación+1))
      Calcular Strength(P(generación+1))
      generación++
Fin
retornar Mejor Frente Encontrado
```

Algoritmo 7 – Esquema del SPEA

- **SPEA-2**

En el año 2001 Eckart Zitzler, Marco Laumanns, y Lothar Thiele propusieron [28] una versión mejorada del SPEA la que llamaron SPEA-2. Este nuevo algoritmo incorpora al original una estrategia de asignación de fitness más fina, una técnica de estimación de densidad y un mejor método de truncamiento de archivo.

Además el tamaño del archivo (población externa) es fijo, cuando la cantidad de individuos no dominados en menor que el tamaño del archivo, el archivo se completa con individuos dominados.

Por otro lado, el método de clustering, que se invoca cuando el archivo de los individuos no-dominados supera el límite, ha sido sustituido por un método alternativo de truncamiento con características similares, pero no pierde puntos fronterizos.

```
generación=0
Inicializar(P(generación))
Inicializar(PobExterna)
Evaluar(P(generación))
Calcular Strength(P(generación))
mientras (no CriterioParada) hacer
      copiarNoDominados(P(generación), PobExterna)
      SacarNoDominados(PobExterna)
      Truncamiento(PobExterna)
      Si no esta lleno PobExterna
            LlenarConDominados(PobExterna)
      Fin si
      Padres = Seleccionar(P(generación)U PobExterna )
      P(generación+1) = Cruzamiento(Padres)
      Mutar(P(generación + 1))
      SeleccionarSobrevivientes(P(generación+1))
      Evaluar(P(generación+1))
      Calcular Strength(P(generación+1))
      generación++
Fin
retornar Mejor Frente Encontrado
```

Algoritmo 8 – Esquema del SPEA2

- **PAES - Pareto Archived Evolution Strategy**

Este algoritmo es una estrategia de evolución[29]. El algoritmo comienza con la inicialización de un único cromosoma, que es la solución actual. Luego se genera una copia de esta solución y se le aplica el operador de mutación. Esta copia mutada es evaluada y pasa a ser una solución candidata.

La aceptación es simple, si una solución domina a la otra es aceptada, pero en el caso de que ninguna solución sea dominante, la solución candidata se compara con una población de referencia (archivo) de soluciones no-dominadas previamente generada. En caso de empate la comparación favorece a las soluciones que están en el espacio de soluciones donde hay menos soluciones.

El archivo sirve para almacenar todas las soluciones no dominadas que a medida que se encuentran, y para ayudar en la selección exacta entre la solución actual y la candidata. Este último es lo que proporciona la presión de selección, impulsando el proceso para encontrar mejores soluciones.

Para mantener la diversidad en las diferentes regiones del espacio de soluciones utiliza crowding, para esto usa una grilla d-dimensional, donde d es el número de funciones objetivos del problema. Cuando cada solución se genera se encuentra su ubicación en la grilla utilizando una subdivisión recursiva y un árbol de codificación. Se mantiene un mapa de la grilla, indicando la ubicación de cada grilla el número de soluciones en el archivo que actualmente residen allí. Esto también se utiliza para la selección entre la solución actual y la solución candidata cuando la candidata no domina ni es dominada por ninguna solución del archivo. En este caso la solución con menor grid count es elegida. [29]

```
mientras (no CriterioParada) hacer
      inicializar(padre)
      evaluar(padre)
      agregarAlArchivo(padre)
      hijo = mutar(padre)
      si hijo domina padre
            agregarAlArchivo(hijo)
            hijo reemplaza padre
      sino si padre domina hijo
            descartar hijo
      sino
            esposible = comparar(hijo, padre, archivo)
            si esposible
                  agregarAlArchivo(hijo)
            fin si
      fin si
Fin
retornar Mejor Frente Encontrado
```

Algoritmo 9 – Esquema del PAES

- **NSGA-II**

En el 2000 Deb et al. propusieron una serie de mejoras al algoritmo NSGA [30], así surge el NSGA-II. Dentro de estas mejorar se encuentran: No utiliza población secundaria, incorpora elitismo mediante un esquema de selección ($\mu+\lambda$), el chequeo de dominancia fue mejorado para aumentar la performance computacional, en lugar de la técnica de sharing del NSGA utiliza un mecanismo de crowding que no requiere parámetros.

El operador de crowding guía al proceso de selección en las distintas etapas del algoritmo hacia a un frente de Pareto distribuido. Entre dos soluciones con diferentes grados de no-dominación seleccionara la solución con menor rango. En caso contrario, si las dos soluciones pertenecen al mismo frente, seleccionara la solución que se encuentre en la región con menor número de puntos.

```
generación=0
Inicializar(P(generación))
Evaluar(P(generación))
AsignarRankingBasadoPareto(P(generación))
Padres = Seleccionar(P(generación))
Hijos = cruzamiento(Padres)
Mutar(Hijos)
mientras (no CriterioParada) hacer
      AsignarRankingBasadoPareto(Padres U Hijos)
      Frentes = GenerarConjuntosFrentes(Padres U Hijos)
      i=1
      Mientras size(NuevaPop)+ size(frentes[i])< tamañoP
            Crowding(frentes[i])
            NuevaPop = NuevaPop U frentes[i]
            I++
      Fin
      Inds = tamañoP - (NuevaPop)+ size(frentes[i])
      Crowding(frentes[i])
      SortingPorDistanciadeCrowding(frentes[i])
      NuevaPop = NuevaPop U ElegirPorCrowding(frentes[i], Inds)
      generación++
      P(generación) = NuevaPop
      Padres = Seleccionar(P(generación))
      Hijos = cruzamiento(Padres)
Mutar(Hijos)
Fin
retornar Mejor Frente Encontrado
```

Algoritmo 10 – Esquema del NSGA-II

- **NPGA-2**

La propuesta del NPGA-2 fue realizada por Ericsson et al. en el año 2001, como una versión mejorada del algoritmo NPGA.

En NPGA-II se usan selección de torneo y ranking basado en frentes de Pareto. Las cardinalidades de los nichos son calculadas en NPGA-II usando individuos en la próxima generación parcialmente generada. Esto se llama fitness sharing continuamente actualizado[31].

```
generación=0
Inicializar(P(generación))
Evaluar(P(generación))
Calcular Cardinal Nicho(P(generación))
mientras (no CriterioParada) hacer
      generación++
      mientras no padres no esta completo
            elegir Dos Individuos basado en ranking y conjunto
            si ind1 domina a todo x ∈ conjunto
                  incluir ind1 en padres
            sino y si ind2 domina a todo x ∈ conjunto
                  incluir ind2 en padres
            sino
                  incluir ind con menor niche count
            fin si
      fin mientras
      P(generación)= Cruzamiento(Padres)
      Mutar(P(generación))
      Evaluar(P(generación))
      Calcular Cardinal Nicho(P(generación))
Fin
retornar Mejor Frente Encontrado
```

Algoritmo 11 – Esquema del NPGA-II

- **MICRO GA**

En esta propuesta se parte de la base de que las poblaciones de gran tamaño implican un gran número de evaluaciones de la función de fitness, por eso se propone como alternativa el uso de micro-algoritmos genéticos, que evolucionan poblaciones muy pequeñas lo que hace que la búsqueda en el espacio de soluciones se realice en de manera muy eficiente.

Evidentemente, las pequeñas poblaciones no son capaces de mantener la diversidad durante muchas generaciones, por eso la población puede ser reiniciada cuando se pierda la diversidad, manteniendo sólo los individuos que mejor se adapten (por lo general, sólo se mantiene el mejor, es decir, el elitismo es de un individuo).

El reiniciar la población varias veces durante la ejecución del algoritmo genético tiene el beneficio añadido de evitar la convergencia prematura, debido a la presencia de un ajuste individual, lo que plantea el riesgo de impedir una mayor exploración de la búsqueda de espacio y de esa manera hacer que el programa de convergencia un optimo local. También, ya que no son la evolución de grandes poblaciones, se puede lograr la convergencia más rápida y se requiere menos memoria para almacenar la población.

3.4.5 Métricas de evaluación de MOEAs

La comparación de los resultados de dos o más algoritmos multiobjetivo no es una tarea sencilla, ya que hay varios aspectos a considerar en dicha comparación. Para poder realizar dicha comparación han surgido una serie de métricas que hacen hincapié en diferentes aspectos del desempeño de los algoritmos. Los aspectos que se suelen considerar son los siguientes: la calidad de los resultados (en base a la cercanía al frente de Pareto), la diversidad de las soluciones encontradas, la cantidad de elementos del conjunto de óptimos de Pareto, la eficiencia computacional del algoritmo.

A continuación se detallan algunas de las métricas propuestas en la literatura.

- **Número de Puntos no Dominados**

Esta métrica evalúa la cantidad efectiva de puntos no dominados que encuentra el algoritmo. Aquí no se toma en cuenta el frente de Pareto real del problema, es una medida que permite ver la diversidad de las soluciones encontradas por el algoritmo. Podemos llamar FP_{know} al conjunto de las soluciones no dominadas por el algoritmo y n a la cardinalidad de dicho conjunto.

- **Tasa de Error**

Esta métrica indica el porcentaje de soluciones no dominadas que el algoritmo encontró que no pertenecen al frente de Pareto real del problema. Se requiere conocer el frente de Pareto real del problema, lo que no es simple posible.

Para calcular la tasa de error *ER* se utiliza la siguiente formula:

$$ER = \frac{\sum_{i=1}^{n} e_i}{n}, \text{ siendo } e_i = \begin{cases} 1 \text{ si la solución} \in FP_{real} \\ 0 \text{ en otro caso} \end{cases}$$

- **Distancia Generacional**

Esta métrica busca estimar que tan lejos del frente de Pareto Real del problema están las soluciones no dominadas encontradas por el algoritmo, utilizando la distancia euclidiana en el espacio de los objetivos. Esta métrica se determina de la siguiente manera:

$$GD = \frac{\left(\sum_{i=1}^{n} d_i^{\,p} \right)^{1/p}}{n}$$

donde p es la dimensión del espacio y esta dada por la cantidad de funciones objetivo, y $d_i^{\,p}$ es la distancia entre la i-ésima solución no dominada obtenida y la solución más próxima en el frente de Pareto real.

- **Spacing**

Esta métrica evalúa la distribución de los puntos no dominados hallados como solución del problema mediante el MOEA. Se calcula de la siguiente manera:

$$SP = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} \left(\bar{d} - d_i \right)^2}$$

El termino d_i mide la distancia en el espacio de las funciones objetivo entre la solución i-ésima y su vecino más próximo (a modo de ejemplo la j-ésima solución) en el conjunto de soluciones no dominadas obtenidas por el algoritmo. Este término se calcula de la siguiente manera:

$$d_i = \min(\left| f_1^i(x) - f_1^j(x) \right| + ... + \left| f_m^i(x) - f_m^j(x) \right|),\ i = 1,..,n$$

El termino $\overline{d}$ corresponde al promedio de los d_i.

Si se obtiene SP=0, esto indica que todos los puntos no-dominados obtenidos por el MOEA están equiespaciados.

- **Spread**

Esta métrica es similar a la anterior, con la salvedad de que incorpora la distancia a los extremos teóricos del frente de Pareto real del problema. Esto se hace con la finalidad de no obtener valores aceptables para la medida de distribución en los casos en que no se muestrea correctamente el frente de Pareto.

Esta métrica se calcula de la siguiente manera:

$$Spread = \frac{\sum_{k=1}^{M} d_k^e + \sum_{k=1}^{n} \left| \overline{d} - d_i \right|}{\sum_{k=1}^{M} d_k^e + n.\overline{d}}$$

El termino d_k^e mide la distancia entre el punto "extremo" del frente de Pareto, tomando en cuenta la k-ésima función y la solución no dominada más cercana. Mientras tanto, el termino d_i mide la en el espacio de las funciones objetivo entre la solución i-ésima y su vecino más próximo en el conjunto de soluciones no dominadas obtenidas por el algoritmo. Por ultimo el valor $\overline{d}$ corresponde al promedio de d_i.

3.5 Algoritmos evolutivos paralelos

A medida que se fueron abordando modelos y problemas cada vez más complejos, que a su vez necesitaban trabajar con volúmenes de datos cada vez más grandes, los requisitos de poder de cómputo fueron creciendo. Las aplicaciones que se desarrollaban para resolver estos problemas necesitaban un poder de computo que un computador tradicional no era capaz de proporcionar trabajando de manera aislada.

Como solución al problema planteado surge el procesamiento paralelo, donde varios procesadores trabajan cooperativamente para resolver un problema en común. Se puede decir que es una forma eficaz de procesar la información que favorece a la explotación de los sucesos que pueden ocurrir de manera concurrente.

Cabe destacar que los avances producidos en el procesamiento paralelo se deben a los avances en el poder de procesamiento de los microprocesadores y al desarrollo de las redes y la comunicación de datos.

Por otro lado surge el procesamiento distribuido, que se trata de procesos concurrentes ejecutándose en forma distribuida en varios computadores que no tienen porque estar localmente interconectados. Esta forma de ejecución mejora el desempeño, robustez y la tolerancia a fallos.

En 1996 Flynn [21] propone una taxonomía para describir las diferentes arquitecturas paralelas, en función del manejo de instrucciones y de datos que presentan las distintas arquitecturas de computadores. Se proponen cuatro categorías en base a las combinaciones de estos elementos. A continuación se describen dichas categorías.

La primer categoría que se define es la SISD, que significa una sola instrucción para un solo dato (Single Instruction Single Data). Esta categoría corresponde a la maquina de Von Neumann. En este caso un procesador es capaz de realizar acciones secuencialmente, dichas acciones son controladas por un programa que se encuentra almacenado en la memoria que esta conectada al procesador. Este tipo de arquitectura da soporte al procesamiento secuencial clásico.

La segunda categoría es SIMD, que significa una sola instrucción para múltiples datos (Single Instruction Multiple Data). En estas arquitecturas un único programa controla a múltiples procesadores. Son especialmente útiles en casos de aplicaciones uniformes, por ejemplo: procesamiento de imágenes, diferencias finitas, etc.

La tercer categoría es MISD, que significa múltiples instrucciones para un solo dato (Multiple Instruction Single Data). En esta categoría están las arquitecturas paralelas que involucran a más de un proceso interactuando con un solo dato. Por lo general se refiere a arquitecturas implementadas sobre un multiprocesador ya que se necesita tener memoria compartida.

La cuarta categoría es MIMD, que significa múltiples instrucciones para múltiples datos (Multiple Instruction Multiple Data). En esta categoría se encuentran las arquitecturas paralelas en que varios procesadores trabajan sobre un conjunto de datos. Aquí podemos hablar de arquitecturas que trabajan con memoria compartida, como con memoria distribuida. En el primer caso la escalabilidad de la arquitectura es muy limitada, mientras que en el segundo caso la arquitectura tiene una escalabilidad muy grande.

Un caso particular de la última categoría son los clusters. Un cluster es una arquitectura de procesamiento paralelo distribuido, compuesto por un conjunto de computadores que trabajan cooperativamente como un único recurso de cómputo.

Para poder medir la mejora del rendimiento que se obtiene al utilizar técnicas de paralelismo se define el speedup, que es el cociente entre el tiempo medio de ejecución de una versión serial sobre el tiempo de ejecución del algoritmo paralelo. Si un algoritmo paralelo que utiliza m procesos presente un speedup = m decimos que tiene speedup lineal, si es menor se dice que el speedup es sublineal y si es mayor se dice que tiene speedup superlineal.

La evolución natural se puede interpretar como un proceso paralelo, por eso se han propuesto varios modelos de paralelismo aplicados a algoritmos evolutivos. Los modelos más comunes propuestos en la literatura son: el modelo maestro-esclavo, el modelo de subpoblaciones distribuidas y el modelo celular. En las siguientes secciones se describen dichos modelos de algoritmos evolutivos paralelos.

3.5.1 Modelo maestro-esclavo

En este modelo un proceso maestro coordina la ejecución de k procesos esclavos. El proceso maestro es quien se encarga de la ejecución de los operadores evolutivos como son: la selección, el cruzamiento, la mutación, etc. Los procesos

esclavos se encargan de realizar la evaluación de las funciones de fitness de los individuos[20].

En este modelo el proceso maestro envía los individuos, o la información que se necesite de estos, a los procesos esclavos; luego los esclavos calcularán el fitness del individuo que les fue enviado y se lo enviaran al proceso maestro. El proceso maestro aplicará los operadores evolutivos con los resultados que le envían los procesos esclavos.

El sistema es sencillo y relativamente fácil de implementar, sin embargo, adolece de dos grandes inconvenientes. En primer lugar, una buena cantidad de tiempo se desperdicia si hay mucha varianza en el tiempo de evaluación de la función. En segundo lugar, el algoritmo no es muy fiable, ya que depende del estado del proceso maestro, si el proceso maestro cae, el sistema falla[20].

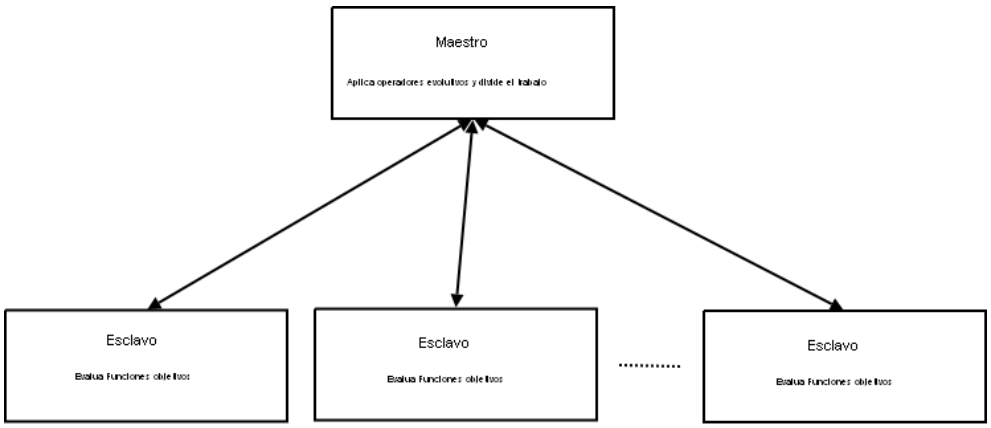

Figura 3-1: Diagrama del modelo maestro-esclavo

El funcionamiento de este modelo es similar al funcionamiento de un algoritmo genético serial, ya que se realiza una evolución panmítica como en el caso serial. También cabe destacar que se pueden implementar versiones sincrónicas y asincrónicas de este modelo de algoritmo genético.

3.5.2 Modelo de subpoblaciones distribuidas

En este modelo se divide la población inicial en k subconjuntos, luego se ejecutara un algoritmo genético sobre cada subconjunto. Cada una de estas subpoblaciones opera como un algoritmo genético secuencial. Cada uno de los procesos (islas) ejecuta sus operadores de selección, cruzamiento, mutación de manera independiente a los demás procesos, además estos operadores y su configuración pueden diferir entre las distintas islas.

Estos procesos interactúan mediante el envió de individuos (migración) de una isla a otra, para realizar esta comunicación se define de antemano una topología de comunicación entre los procesos, por lo general un anillo unidireccional.

Aquí vemos que la interacción entre los individuos es distinta a la que se presenta en el caso serial, ya que en el caso serial un individuo se puede recombinar con cualquier otro individuo de la población, mientras que en el caso de este modelo los individuos se pueden cruzar con los individuos de su subpoblación o con los de otra subpoblación en caso de que haya sido migrado.

Cuando se realiza una migración se busca que los individuos que son enviados de una isla a otra tengan buena adaptación, con el fin de que el material genético introducido genere individuos con buenas características.

Existen dos tipos de migraciones posibles: migraciones sincrónicas y asincrónicas. Cuando la migración es sincrónica todas las islas detienen la ejecución del algoritmo genético hasta que se complete el traslado de los individuos. Cuando la migración es asincrónica, las islas siguen ejecutando su algoritmo genético mientras realizan la transferencia de los individuos.

Debido a que la comunicación en este modelo se da con menor frecuencia que en el modelo anterior, las necesidades de ancho de banda son menores respecto al modelo anterior. La fiabilidad de este modelo es alta debido a la autonomía que tienen los distintos procesos[20].

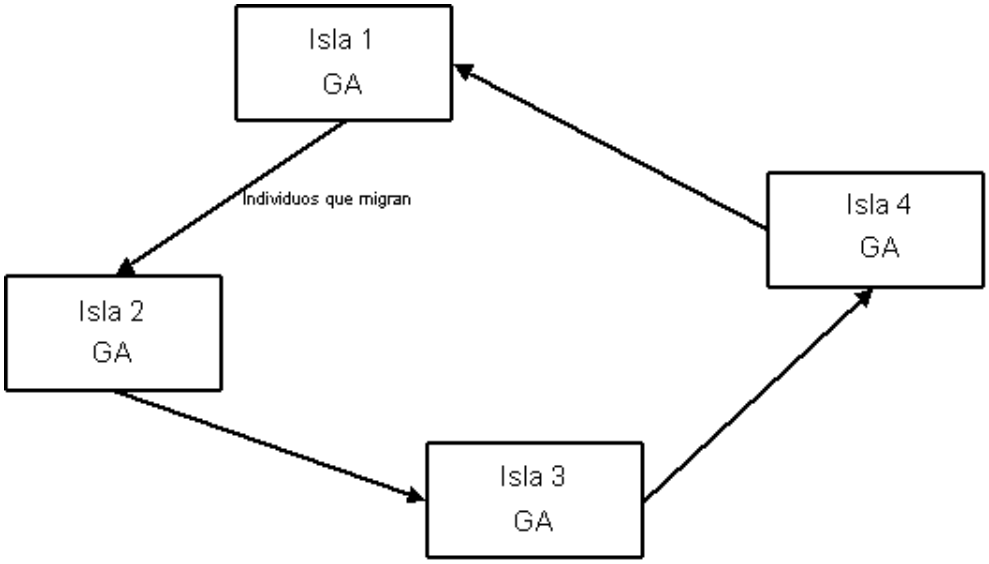

Figura 3-2: Diagrama del modelo de subpoblaciones distribuidas

3.5.3 Modelo celular

En este modelo se proponer dividir la población en subpoblación, como en el modelo anterior. Pero a diferencia del modelo anterior no existe la migración entre individuos de distintas subpoblaciones. En este modelo se definen vecindarios, que son conjuntos de subpoblaciones cercanas, y los individuos de una subpoblación se pueden recombinar con los individuos de su subpoblación y con los individuos de las subpoblaciones que pertenezcan al mismo vecindario que dicha subpoblación. Este mecanismo es conocido como la difusión de individuos, mediante este mecanismo se que las buenas características de los individuos se difunda a toda la población.

En la literatura se menciona a este modelo como el modelo de polinización de plantas, ya que su funcionamiento es similar al mecanismo de reproducción por polinización de algunas plantas[20].

El operador de selección que se utiliza en este modelo debe contemplar que se pueden elegir individuos de la propia subpoblación como individuos de las subpoblaciones vecinas. Esto plantea una modificación de este operador respecto a los modelos anteriormente presentados. En la Figura 3-3 se muestra un diagrama

de este modelo donde se puede observar a las distintas subpoblaciones y una vecindad definida con algunas de ellas.

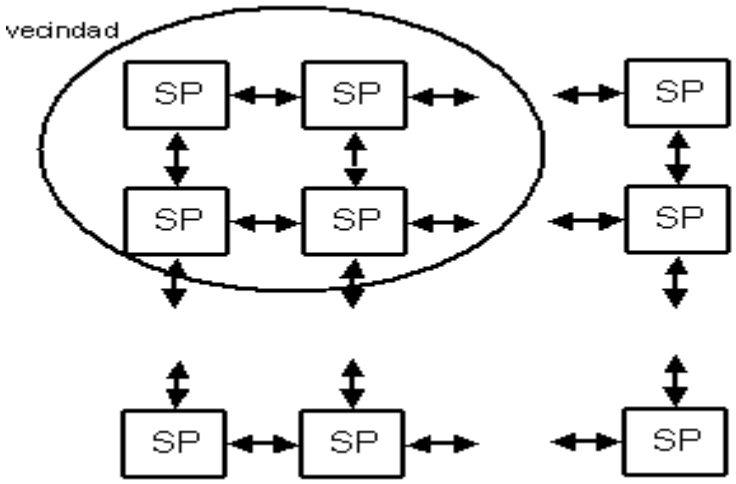

Figura 3-3: Diagrama del modelo celular

Otro aspecto importante de este modelo es que las estructuras que se utilizan para definir las vecindades son las que permiten configurar las características del modelo de difusión, lo que determina las características del propio algoritmo.

A medida que aumentan las interconexiones entre las subpoblaciones se observa que el mecanismo de búsqueda de este algoritmo es asemeja más al mecanismo de búsqueda de un algoritmo evolutivo serial.

En este modelo se presenta un nivel alto de interacciones entre las distintas subpoblaciones, esto implica que se tengan altos requerimientos de comunicaciones. Por lo tanto es modelo adecuado para implementar sobre arquitecturas de computadores paralelos modelo SIMD.

Capítulo 4. MOEAs para planificación en entornos HC

En este capitulo se presenta el problema de planificación de tareas en entornos HC, así como los algoritmos implementados. En primer lugar, se presenta el problema específico que se va a estudiar. También se describen los operadores propuestos para resolver este problema, así como las bibliotecas utilizadas como base de la implementación de los algoritmos propuestos. Por ultimo, se explican los detalles específicos de la implementación.

4.1 Descripción del problema

Como se mencionó anteriormente el problema de la planificación de tareas a maquinas consiste en la asignación de las tareas a las maquinas y la determinación de orden en que se ejecutaran las tareas asignadas a una maquina. En realidad la planificación de tareas en un entorno HC se trata de una familia de problemas que engloban las necesidades más importantes de de las meta-tareas para el alojamiento eficiente de sus tareas en un entorno HC. Por lo tanto se pueden formular varias versiones del problema de acuerdo a las necesidades de la meta-tarea a considerar. También se pueden formular distintas versiones del problema de acuerdo a las propiedades del entorno HC que se considera.

En este trabajo se define una meta-tarea como una colección de tareas independientes, o sea, que no tienen dependencias de datos. Se asume que se ejecuta una tarea por vez en cada maquina y que el orden en que las tareas de una maquina se ejecutan es de forma ascendente según el tiempo de ejecución de las tareas en dicha maquina.

Se va a trabajar con instancias del problema donde la cantidad de tareas y la cantidad de maquinas no varían durante la ejecución de la meta-tarea, además son parámetros conocidos de antemano.

Otra consideración es que las maquinas no tienen periodos de inactividad, por lo tanto cada maquina esta siempre disponible para ejecutar alguna de las tareas que le fueron asignadas, a menos que ya se encuentre ejecutando otra de las tareas que le fueron asignadas. Otro punto importante es que no se considera el tiempo que la maquina necesita para prepararse para ejecutar una tarea luego de ejecutar la tarea que la precedía, por lo tanto se considera que una vez que se termina de ejecutar una tarea t_i en la maquina m_j inmediatamente se inicia la ejecución de la siguiente tarea t_k en dicha maquina. En cuanto al conjunto de maquinas que componen el entorno HC, hay que aclarar que las maquinas del entorno no se sustituyen por maquinas con distintas capacidades de computo a lo largo del tiempo en que se ejecuta la meta-tarea, ni las mismas cambian sus capacidades de computo. De lo anterior se desprende que el tiempo de ejecución de una tareas t_i en una maquina m_j del entorno HC siempre va a ser el mismo, independientemente del momento en que se inicie la ejecución de dicha tarea.

Las tareas que se envíen a ejecutar al entorno HC siempre van a ser ejecutadas, es decir no se cancela la ejecución de una tarea. Por otro lado todas las tareas que se van a ejecutar arriban al entorno al inicio de la ejecución de las tareas (en $t = 0$). Otro parámetro del problema es la matriz ETC, tiempo esperado de computo, en esta matriz se definen los tiempos esperados de ejecución de las tareas en las

distintas maquinas del entorno. El tiempo esperado de ejecución de la tarea t_i en la maquina m_j viene dado por el valor ETC[i][j] y como se mencionó anteriormente no varia en función del momento en que la maquina m_j inicia la ejecución de la tarea t_i. A continuación se explica como se modela la heterogeneidad de maquinas y de tareas mediante las distintas matrices ETC.

En este trabajo se utilizaran las matrices ETC utilizadas en el trabajo de Braun et al. [15]. Los parámetros que se utilizan en este trabajo son los siguientes. Para tener una alta heterogeneidad de tareas se utilizó ϕ_b = 3000 y para tener una baja heterogeneidad de tareas se utilizó ϕ_b = 100. Para variar la heterogeneidad de las maquinas se cambian los valores de ϕ_r, para tener una alta heterogeneidad de maquinas se utilizó ϕ_r = 1000 y para tener una baja heterogeneidad de maquinas se estableció ϕ_r = 10. Además se usaron matrices ETC con diferentes consistencias. En total son doce matrices ETC que se logran realizando todas las combinaciones de heterogeneidad de maquinas, heterogeneidad de tareas y consistencia.

Otro aspecto importante es el criterio de optimización que se va a utilizar en este trabajo. Los objetivos que se buscan minimizar son el flowtime y el makespan. El makespan indica el tiempo en que finaliza la última tarea procesada. El makespan es una medida de la productividad de la programación.

El flowtime de una tarea es la deferencia entre el tiempo de finalización y el tiempo de arribo, el flowtime de la planificación es la suma de los flowtimes de las tareas. El flowtime es una medida de la calidad de servicio (QoS, quality of service) que reciben las tareas en la programación.

Sea c_j el tiempo de finalización de la tarea t_j y sea r_j el tiempo de llegada de la tarea t_j, podemos definir los objetivos de la siguiente manera:

$$makespan = \max_{t_j \in Tareas} \{c_j\}$$

$$flowtime = \sum_{t_j \in Tareas} (c_j - r_j)$$

En este trabajo se busca diseñar e implementar varios algoritmos evolutivos para resolver el problema de despacho de tareas en redes heterogéneas. Se opto por la utilización de algoritmos evolutivos multiobjetivos debido a que se busca minimizar dos funciones objetivos y es de esta forma se puede obtener un conjunto de soluciones optimas donde varia la importancia que se le otorga a cada una de las funciones objetivo. Como base de la implementación se eligieron los algoritmos evolutivos multiobjetivos NSGA-II y SPEA-2, esto se debe a que son dos MOEAs que han demostrado obtener resultados de buena calidad en diversos problemas en el área. A partir de la estructura de estos algoritmos se implementaron algunos operadores evolutivos que incorporan algunos mecanismos relacionados con el problema específico, a fin de obtener mejores soluciones que si se utilizarán operadores totalmente aleatorios.

Otro aspecto a tener en cuenta a la hora del diseño del algoritmo es la eficiencia computacional, ya que se espera obtener soluciones de buena calidad en tiempos razonables. Se estableció como tiempo de ejecución aceptable del algoritmo a lo sumo 10 minutos, mientras se estableció un tiempo de ejecución deseable del algoritmo a lo sumo 3 minutos. Para aumentar la eficiencia computacional de los algoritmos implementados se utilizaron técnicas de programación paralela. Debido a las características del problema y del entorno de ejecución que se disponía para el algoritmo se optó por el modelo de subpoblaciones distribuidas.

Se utilizaron varios frameworks como soporte a la implementación de los algoritmos. Para la implementación del NSGA-II se utilizó el MOE framework [33], que en un framework que surgió a partir de un proyecto de grado de la facultad de ingeniaría. El MOE Framework provee una implementación del algoritmo NSGA-II y la posibilidad de realizar su ejecución de forma paralela utilizando el modelo de subpoblaciones distribuidas de manera transparente para el usuario del framework. A su vez es sencillo incorporar nuevos operadores evolutivos al algoritmo ya que el framework esta diseñado utilizando técnicas de programación orientada a objetos.

Para la implementación del SPEA-2 se utilizaron los frameworks JMetal y Sparrow. JMetal [35] provee la implementación de varios algoritmos evolutivos multiobjetivos, entre los que encontramos: SPEA-2, NSGA-II, PAES, PESA-II, así como algoritmos evolutivos simples, estrategias de evolución y otros tipos de metaheurísticas. En este framework también es sencillo incorporar nuevos operadores evolutivos al algoritmo que se este utilizando, ya que el framework esta diseñado utilizando técnicas de programación orientada a objetos. En este framework solamente esta implementada la ejecución serial de las distintas metaheurísticas, para poder implementar la ejecución paralela del SPEA-2 implementado en JMetal se utilizó el framework Sparrow. El framework Sparrow [34] permite implementar la ejecución paralela de tareas de una forma transparente para el usuario, extendiendo determinadas clases que provee el framework.

Mas adelante en este capitulo se explicará detalladamente el funcionamiento de cada uno de estos frameworks.

4.2 Trabajo previo. Despacho de tareas en redes heterogéneas: enfoque multiobjetivo

En el marco del curso de Algoritmos Evolutivos se realizó el proyecto*: Despacho de tareas en Redes Heterogéneas: Enfoque Multiobjetivo*, este trabajo es un primer acercamiento al problema que se estudia en este trabajo.

En este trabajo también se buscaba una planificación que minimizara dos funciones objetivo: el makespan y el flowtime. Aquí también se eligió un algoritmo evolutivo multiobjetivo como método de resolución para abordar el problema. En este caso se eligió el algoritmo evolutivo multiojetivo NSGA-II original de Deb et al. (2000). Esta implementación del NSGA-II, a diferencia de la que se utilizó en este proyecto, solo se puede ejecutar de manera serial. En este trabajo se implementaron dos propuestas, las que variaban entre si en la forma de inicializar la población.

El modelo utilizado para representar los tiempos de ejecución de las tareas en las máquinas es el mismo que el utilizado en este trabajo, el modelo ETC (expected time to compute). Los supuestos acerca del conjunto de tareas y del entorno HC donde estas se van a ejecutar son los mismos que se consideran en este trabajo.

La representación de las soluciones elegida fue una array de largo n_jobs, donde n_jobs es la cantidad de tareas, y cada posición toma un valor entero que representa a que procesador fue asignado la tarea.

El operador de selección elegido fue la selección por torneo, el operador de cruzamiento elegido fue el cruzamiento de un punto. La inicialización de la población de una propuesta se realiza generando una solución con la heurística min-min y el resto se genera a partir de esta solución. En la otra propuesta se genera una solución con la heurística min-min y otra solución con la heurística max-min, el resto de las soluciones se generan a partir de estas dos soluciones.

También se propuso una mutación basada en la ejecución de la heurística min-min a un conjunto pequeño de tareas.

Hay que mencionar que en este trabajo también se presentaron otras propuestas que fueron descartadas porque no los resultados que obtenían eran muy pobres.

Una de estas propuestas realizaba una inicialización de la población totalmente aleatoria y utilizaba un operador de mutación totalmente aleatorio, los resultados de esta propuesta eran muy malos, eran entre tres o cuatro veces más grandes que los resultados que fueron utilizados como referencia [19].

Otra propuesta realizaba una inicialización de la población totalmente aleatoria y utilizaba un operador basado en un operador propuesto por Xhafa en un trabajo de referencia en el área [19]. Los resultados de este algoritmo eran bastante malos, aunque mejores que los obtenidos con el algoritmo anterior. El operador de mutación propuesto fue descartado como operador genético, porque no hacia cambiar a los individuos durante la ejecución del algoritmo.

Otras dos propuestas analizadas proponían una inicialización de la población basada en la heurística min-min y otra inicialización de la población basada en la heurística min-min y la heurística max-min. El problema que presentaron estas propuestas era la poca cantidad de individuos no dominados que generaban, en el mejor de los casos llegaba a obtener tres individuos no dominados, y siempre uno de los individuos no dominados era el individuo generado con la heurística min-min, incluso para algunos escenarios el único individuo no dominado que devolvía era el generado con min-min. Por esto se decidió descartar estas propuestas, ya que los resultados obtenidos no eran satisfactorios, ya que no tenía sentido ejecutar un algoritmo evolutivo para que devolviera el individuo que había sido generado con el min-min.

La realización de este trabajo permitió obtener ciertas conclusiones acerca del problema estudiado que sirvieron como base del trabajo que se presenta aquí. En ese trabajo se llego a la conclusión de que el algoritmo implementado se comportaba razonablemente bien, esto viendo como evolucionaban los valores de fitness de los individuos a los largo de la ejecución del algoritmo. Otro aspecto que se pudo observar es que mejora mucho la calidad de los resultados si se utiliza una heurística como base para inicializar la población. También se comprobó que el operador de mutación propuesto es valido como operador evolutivo, ya que permite explorar nuevos individuos. En cuanto a los resultados obtenidos se puede decir que los valores obtenidos de makespan son bastante buenos y que los valores obtenidos de flowtime tendrían que mejorar. Como líneas de trabajo a futuro planteadas en ese trabajo se destacan: modificar la evaluación del flowtime y analizar otros operadores genéticos, con el fin de mejorar la calidad de los resultados obtenidos.

Todos los aspectos mencionados de este trabajo fueron considerados a la hora de diseñar e implementar los algoritmos realizados en este trabajo.

4.3 Diseño del algoritmo

En esta sección se explican los aspectos relacionados con el diseño de los distintos elementos que componen a los algoritmos evolutivos que fueron implementados. Se mencionarán aspectos relacionados con los distintos operadores evolutivos, así como aspectos relacionados con los distintos parámetros que se eligieron para la ejecución de ambos algoritmos. También se mencionan otros aspectos de diseño como, por ejemplo, la codificación elegida de las soluciones.

4.3.1 Codificación

Para representar la asignación de las tareas a las distintas maquinas que componen el entorno se va a utilizar un vector de enteros, cuyo largo es igual a la cantidad de tareas que se quiere enviar al entorno HC para que las mismas sean ejecutadas. El valor de la i-esima posición del vector corresponde a la maquina a la que fue asignada la i-esima tarea. Esta representación es posible gracias al hecho de que no es necesario que la solución refleje explícitamente el orden en que se ejecutan las tareas en cada una de las maquinas del entorno.

Como ya se mencionó anteriormente el orden en que se ejecutan las tareas esta implícito en la misma asignación, esto se debe a la forma en que se evalúan las funciones objetivo. Se observa claramente que el orden en que se ejecuten las tareas dentro de una maquina no va a variar el tiempo en que la maquina finalice la ejecución de las mismas, por lo tanto el valor del makespan siempre será el mismo. Por otro lado el valor del flowtime de una asignación varía según el orden de ejecución de las tareas, pero se cumple que la forma de minimizar el valor del flowtime de una asignación particular es que las tareas asignadas a una maquina se ejecuten en orden ascendente según el tiempo que requieren para ser ejecutadas en la maquina a la que fueron asignadas.

Esta codificación es una representación sencilla, que permite utilizar operadores evolutivos relativamente sencillos en el algoritmo evolutivo. En la siguiente figura se muestra un ejemplo de la codificación utilizada.

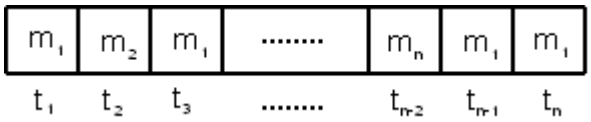

Figura 4-1: Codificación de las soluciones.

4.3.2 Inicialización de la población

Tomando en cuenta los resultados obtenido en el proyecto final realizado en el curso de algoritmos evolutivos, donde se pudo concluir que la utilización de una heurística en la inicialización de la población permite obtener resultados de mayor calidad que si se inicializa la población de forma totalmente aleatoria. La heurística que se utiliza en la inicialización de la población es el min-min, debido a que es una heurística muy simple, presenta un tiempo de ejecución breve, y se obtienen resultados de calidad aceptable incluso mejores que otras heurísticas mucho más complejas [15].

En primer lugar se genera un individuo que corresponde a la solución obtenida por la heurística min-min, este individuo se utilizará como base para generar otros individuos de la población. Cada uno de los individuos restante de la población se genera de la siguiente manera: se elige de manera equiprobable si el mismo se generará de manera aleatoria o basándose en el primer individuo.

Si el individuo se genera de manera aleatoria a cada uno de sus genes se le asigna un valor aleatorio entre 0 y maquinas-1, o sea, que cada tarea es asignada a una maquina al azar. Con este método se busca generar diversidad entre los individuos de la población, con el objetivo de explorar la mayor parte del espacio de soluciones.

Si el individuo se genera de basándose en el primer individuo para cada uno de sus genes se elige de manera equiprobable si se copia el valor del gen correspondiente de dicho individuo o si se asigna aleatoriamente. Con este método se busca captar algunas características buenas del individuo generado con la heurística min-min y a su vez generar nuevas características, a partir de las asignaciones aleatorias, tratando de generar individuos que nos permitan explorar zonas del espacio de soluciones donde existan buenas soluciones.

El tamaño de población elegido es de 120 individuos, que se dividen en 8 islas de 15 individuos cada una. Estos parámetros fueron elegidos considerando la necesidad de tener soluciones con un cierto grado de diversidad en cada una de las islas y también buscando que el algoritmo evolutivo presente tiempos de ejecuciones breves en cada una de las islas.

4.3.3 Selección

El operador de selección utilizado es la selección por torneo. En este operador de selección se eligen dos individuos de la población de manera aleatoria, luego se comparan entre si para decidir cual individuo es seleccionado. Primero se determina la dominancia entre dichos individuos, si un individuo domina al otro individuo, respecto a las dos funciones objetivos, se selecciona el individuo no-dominado.

En el caso de que ninguno de los dos individuos domine al otro se compara el valor de crowding que tienen dichos individuos, luego de realizada dicha comparación se selecciona el individuo que tiene mayor crowding.

Con esto se busca que la presión de la selección se dirija hacia los individuos que son dominados por menos individuos, también se busca en segundo lugar darle prioridad a los individuos con mayor valor de crowding para poder explorar zonas lo más diversas posibles dentro del espacio de soluciones.

4.3.4 Cruzamiento

El hecho de utilizar una representación sencilla nos permite utilizar un operador de cruzamiento sencillo. Los operadores de cruzamiento que fueron utilizados fueron el cruzamiento de un punto y el cruzamiento uniforme. En el trabajo previo se había utilizado el cruzamiento de un punto y se había planteado como línea de trabajo a futuro analizar otros operadores, entre los que se mencionaba el cruzamiento uniforme.

Luego de analizar los resultados obtenidos utilizando ambos operadores de cruzamiento, se llego a la conclusión de que se obtenían resultados de mejor calidad si se optaba por el cruzamiento uniforme.

Cabe recordar que el objetivo de este operador es explotar buenas secciones del espacio de soluciones, para poder lograrlo se busca combinar las buenas características con la intención de generar nuevos individuos que también tengan buenas características.

Debido a la importancia de este operador se le asigno una probabilidad de aplicación grande.

4.3.5 Migración

Cuando se trabaja con algoritmos paralelos siguiendo el modelo de subpoblaciones distribuidas es necesario utilizar el operador de migración. Este operador es el encargado de enviar un individuo que presente buenas características a otra isla, con el objetivo de generar nuevos individuos que puedan adaptarse mejor al problema en estudio.

Cada 500 generaciones se aplica la migración de un individuo por población. Se establece una topología de anillo entre las islas, por lo tanto de una isla se envía un individuo a la siguiente isla y recibe uno de la isla anterior. Para elegir el individuo que se va a migrar se utiliza un mecanismo de selección proporcional al fitness, rueda de ruleta, para darle mayor chance de ser elegidos a los individuos con mayor fitness.

Luego de que un individuo i_m es enviado a otra isla, se le aplica una política de migración para ver si el mismo se agrega a esa isla o se descarta. Esta política busca un individuo i talque i_m domina a i, si encuentra dicho individuo el individuo i_m se agrega a la población de la isla sustituyendo al individuo i, en caso contrario se descarta al individuo i_m. Vale aclarar que el individuo que se selecciona para ser migrado no se saca de la población de la isla desde la que fue migrado.

4.3.6 Mutación

El objetivo del operador de mutación de un algoritmo evolutivo es agregar diversidad a la población y permitir al algoritmo explorar nuevas secciones del espacio de búsqueda. Al momento de diseñar este operador se tomaron en cuenta las conclusiones del trabajo previo mencionado anteriormente, donde se puede observar el pobre desempeño logrado al utilizar un operador de mutación totalmente aleatorio y que se obtienen soluciones aceptables incorporando inteligencia a la ejecución de este operador.

Se definieron varias estrategias para realizar en la mutación. El operador de mutación se encarga de elegir una de estas estrategias de manera probabilística y aplicar la estrategia elegida.

La primer estrategia se trata de una estrategia de mutación aleatoria. En primer lugar se elige la cantidad de genes (tareas) que se van a mutar, entre un mínimo y máximo establecido (en nuestro caso 1 y 5 respectivamente). Una vez se determinada la cantidad de genes a mutar se eligen los mismos, y para cada a mutar se determina aleatoriamente su nuevo valor.

Se diseñaron cuatro estrategias de mutación que buscan balancear la carga de las tareas en las maquinas para generar soluciones de mejor calidad. En la primera de las estrategias de balance se busca determinar cual es la maquina más cargada y cual es la maquina menos cargada, una vez que se determina esto se elige al azar una tarea de la maquina más cargada y se asigna a la maquina menos cargada. La segunda estrategia de balance también busca determinar cual es la maquina más cargada y cual es la maquina menos cargada, pero en este caso se determina cual es la tarea que requiere mayor tiempo de ejecución en la maquina más cargada y se asigna a la maquina menos cargada. En la tercer estrategia de balance se busca determinar cual es la maquina menos cargada y cual es la tarea, que esta asignada a otra maquina, que es la más adecuada para asignar a la maquina menos cargada, la tarea más adecuada es la que hace incrementar en menor medida el makespan de la

maquina menos cargada una vez que es asignada a esta. En la cuarta estrategia de balance se busca determinar cual es la maquina menos cargada y cual es la tarea, que esta asignada a otra maquina, que es la más adecuada para asignar a dicha maquina, la tarea más adecuada es la que hace incrementar en menor medida el makespan de la maquina menos cargada una vez que es asignada a esta.

También se definieron dos estrategias de mutación basada en movimientos de tareas. La primer estrategia de movimiento consiste en elegir una tarea de manera aleatoria y moverla a la mejor maquina, esto es mover la tarea elegida a la maquina que incremente su makespan en menor medida. Además con probabilidad 0,5 se puede realizar un movimiento complementario, que es mover una tarea desde la maquina a la que se asigno la tarea elegida en el primer paso hacia la maquina a la que dicha tarea estaba asignada originalmente, en este caso se elige la tarea más adecuada para realizar este movimiento.

La segunda estrategia de movimiento consiste en elegir una tarea t_1 de manera aleatoria e realizar un intercambio con la tarea t_2 más adecuada, esto es asignar t_1 a la maquina donde esta t_2 y viceversa. La tarea t_2 se elige de forma que de minimizar el makespan de la maquina más cargada luego de realizar el intercambio.

Por ultimo se definió una estrategia basada en la heurística min-min. En primer lugar se determinan de manera aleatoria las tareas que se van a integrar el conjunto U (tareas sin asignar). Luego se ejecuta la heurística min-min para asignar las tareas que integran el conjunto U. Cabe destacar el no determinismo de esta estrategia, ya que las tareas que no integran el conjunto U no tienen porque estar asignadas siguiendo los criterios de la heurística min-min, lo que hace que la forma en que se asignan las tareas del conjunto U va a ser diferente a la asignación resultante de ejecutar la heurística min-min sobre todas las tareas.

En conclusión se definieron ocho estrategias de mutación, cuando se ejecuta el operador de mutación este elige probabilisticamente una de las estrategias mencionadas y la ejecuta sobre el individuo. Se le asigno a todas estas estrategias la misma probabilidad, salvo a la estrategia aleatoria que se le asigno una probabilidad menor. Con estas estrategias se busca explorar distintas secciones del espacio de búsqueda, buscando captar distintas características que nos aportan las estrategias mencionadas.

4.4 Frameworks utilizados

La implementación de los algoritmos mencionados anteriormente se realizó sobre la base de algunas bibliotecas que contaban con las implementaciones básicas de los MOEAs que se decidió utilizar.

Para implementar el algoritmo evolutivo multiobjetivo NSGA-II se utilizo el MOE framework, que es una biblioteca de MOEAs paralelos implementada en el lenguaje C++ utilizando mpich como mecanismo de comunicación entre los procesos. Esta biblioteca surge en un proyecto de grado de la Facultad de Ingeniería de la UdelaR, y por ahora solo cuenta con la implementación del NSGA-II.

Para implementar el algoritmo evolutivo multiobjetivo SPEA2 se utilizo jMetal, que es una biblioteca de MOEAs implementada en el lenguaje java. Esta biblioteca no ofrece soporte a la ejecución de MOEAs paralelos. Para poder realizar la ejecución paralela se utilizó la biblioteca Sparrow, dicha biblioteca permite al usuario definir tareas y enviarlas a procesar en paralelo de una forma bastante transparente.

A continuación se presenta una explicación detallada de estas tres bibliotecas.

4.4.1 MOE Framework

Como se mencionó anteriormente el MOE framework es una biblioteca de MOEAs paralelos implementada en C++ que utiliza mpich para realizar las comunicaciones entre los procesos que se están ejecutando en paralelo.

Esta biblioteca esta compuesta por tres paquetes principales: engine, algorithms y problems. El paquete engine contiene las clases base, estas clases permiten el modelado de los problemas, soluciones y algoritmos, además de cubrir las funcionalidades mínimas del sistema. El paquete algorithms contiene las clases que implementan los algoritmos para resolver problemas. Por ultimo, el paquete problems contiene las clases que definen los problemas: codificación de las soluciones, operadores para cada codificación, criterios de optimización y restricciones. En la Figura 4-2 se muestra las dependencias entre estos paquetes.

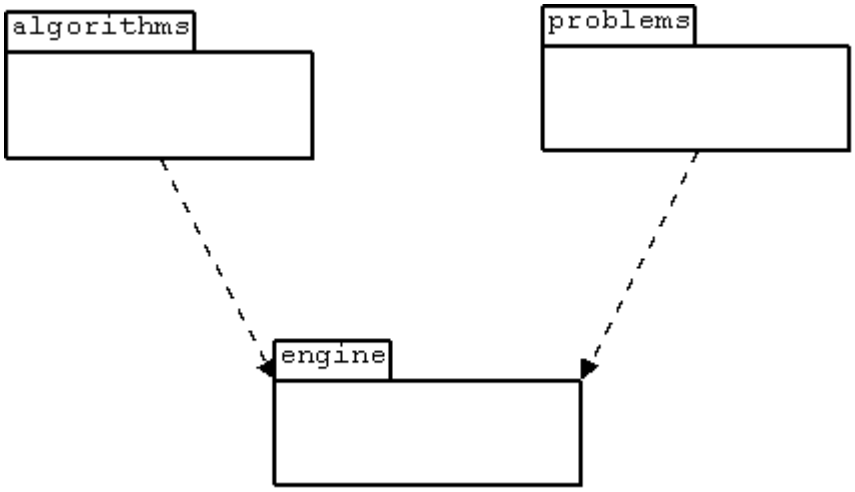

Figura 4-2 : Paquetes que componen el MOE Framework

En el paquete engine están las clases que implementan el esqueleto de un algoritmo evolutivo multiobjetivo, estas clases proveen los mecanismos básicos necesarios para la ejecución del MOEA, de forma que no se necesario implementarlos en las clases que extiendan a estas para resolver un problema particular.

La clase principal es MOEA, en esta clase se implementan los mecanismos que son comunes a todos los moeas. En esta clase se implementa la administración de los operadores evolutivos y los operadores de selección, también se controlan las condiciones de término y de migración, así como se mantiene el estado actual de la evolución con el fin de actualizar las estadísticas, históricos y mantener la población. Los distintos MOEAs que se implementen, que estarán incluidos en el paquete algorithm, deberán extender esta clase e implementar los métodos abstractos que se definen en ella, como por ejemplo: stopCondition(), generation(), migrationCondition().

Otra clase importante en este paquete es NetStream, que es la encargada de proveer las primitivas de comunicación necesarias para que los diferentes procesos puedan interactuar entre si.

Cada uno de los operadores evolutivos, operadores de selección, y operadores de migración están representados por una clase distinta. En el paquete engine se encuentran las clases Crossover, Selection, Mutation, y Migration. Todas estas

clases deberán ser extendidas para implementar los operadores que se deseen utilizar para resolver el problema que se esté estudiando, por ejemplo: si se quiere utilizar el operador de cruzamiento uniforme se implementara una clase uniformCrossover que extienda a la clase Crossover.

La clase MOEA utiliza a la clase Population para mantener a la población con la que esta trabajando, la clase Population tiene una colección de objetos Solution. Solution es la clase base que deberán extender las clases que representen a una codificación particular de un individuo. Esta clase tiene una colección de objetos Objetive, estos objetos tienen el código necesario para evaluar la función objetivo que representan así como la dirección de optimización. También tiene una colección de objetos Restriction, estos objetos tienen el código necesario para determinar si una solución es factible o no para el problema.

En la Figura 4-3 se muestran las relaciones y dependencias entre las clases mencionadas.

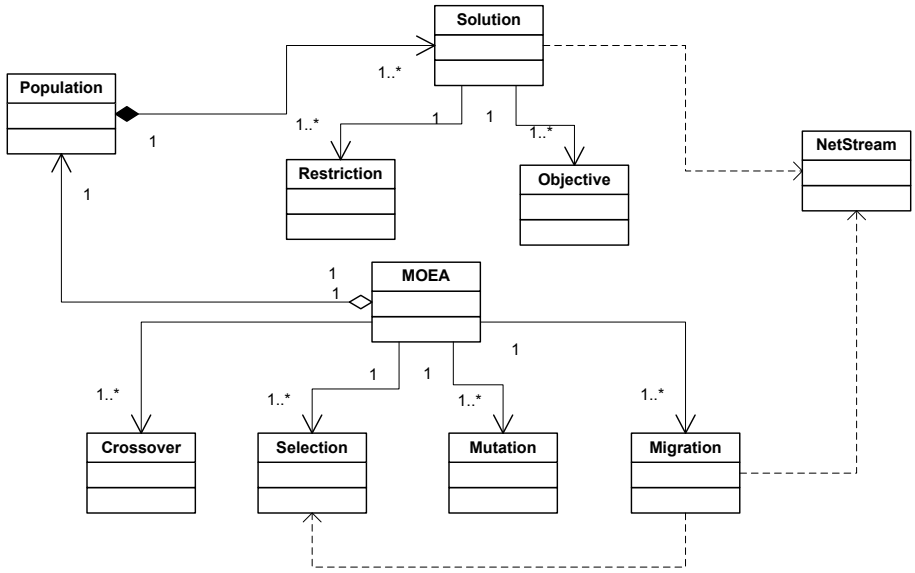

Figura 4-3: Clases básicas del paquete engine

Además de las clases mencionadas anteriormente podemos destacar otro grupo de clases que son las encargadas de implementar las estadísticas y el histórico de la evolución. En la Figura 4-4 se puede observar la aplicación del patrón de diseño observer para implementar tales tareas.

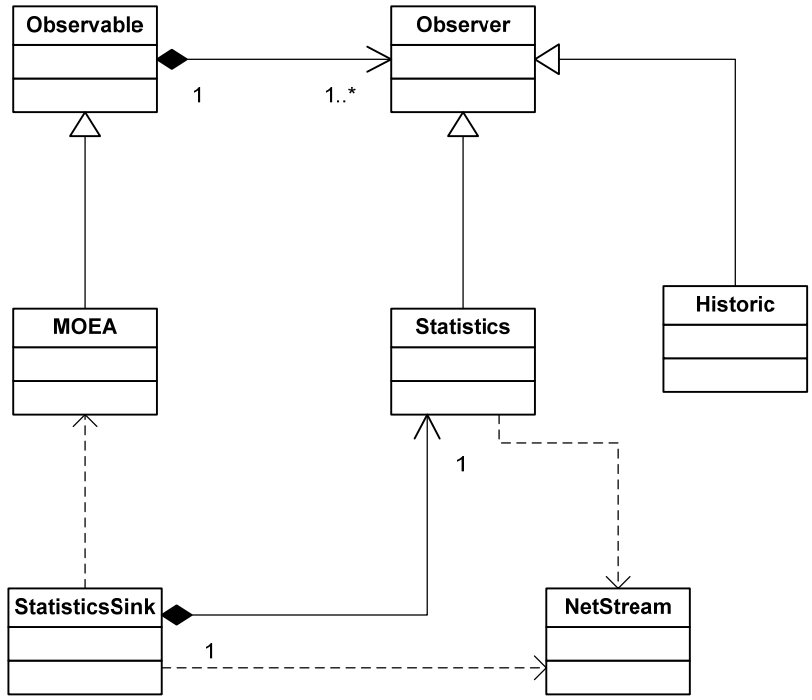

Figura 4-4: Clases del paquete engine encargadas de las estadísticas

Además de las clases mencionadas anteriormente, en el paquete engine se encuentran otras clases auxiliares que pueden resultar útiles para la implementación de las clases necesarias para realizar un MOEA particular. Estas clases son: Random, Properties, y ArrayList.

En el paquete algorithms se encuentran las clases que implementan a un MOEA particular. En la actualidad solamente hay un MOEA implementado, por esto solamente se encuentra en dicho paquete la clase NSGAII que extiende a la clase MOEA, e implementa todos los métodos abstractos definidos en la clase base. En particular la clase NSGAII implementa el método generation() que es el encargado de aplicar todos los mecanismos que corresponden a una generación de la evolución.

En el paquete problems se encuentran las clases que implementan las codificaciones de los individuos, operadores de mutación, cruzamiento, migración y mutación, además de las clases que modelan a los problemas. Las clases que están disponibles en la biblioteca son las que se necesitan para resolver las instancias de los problemas ZDT. Podemos encontrar las clases IntArraySolution y RealArraySolution que extienden a la clase Solution, las clases PerturbMutation y RealPerturbMutation que extienden a la clase Mutation, las clases SinglePointCrossover, DualPointCrossover y SBX que extienden a la clase Crossover, las clases RouletteWheelSelection y TournamentSelection que extienden a la clase Selection, y la clase RingMigration que extiende a la clase Migration.

Por otra parte vale la pena mencionar al paquete Launcher, en el que se encuentra la clase Launcher que es la encargada de orquestar la ejecución del MOEA, en esta clase se crea la instancia del MOEA a ejecutar, en este caso NSGAII, y se determinan los distintos operadores a utilizar. Esta clase es la clase main que se utiliza en la ejecución del algoritmo.

Por ultimo en la Figura 4-5 se muestra el despliegue de un MOEA implementado utilizando el MOE Framework, considerando un ejemplo donde se ejecuta un MOEA paralelo dividido en tres subpoblaciones distribuidas. En este ejemplo se

observa que en las islas donde se ejecutan instancias del MOEA también se ejecutan instancias de la clase Statistics o de una clase que extienda a esta, estas instancias corresponden a las estadísticas dinámicas, estas son las estadísticas que se van actualizando en tiempo de ejecución. En el nodo que corresponde al Proceso de control se ejecuta una instancia de la clase StatisticsSink o de una clase que extienda a esta, que corresponde a las estadísticas estáticas, estas corresponden al análisis que se realiza al finalizar la ejecución del MOEA paralelo [33].

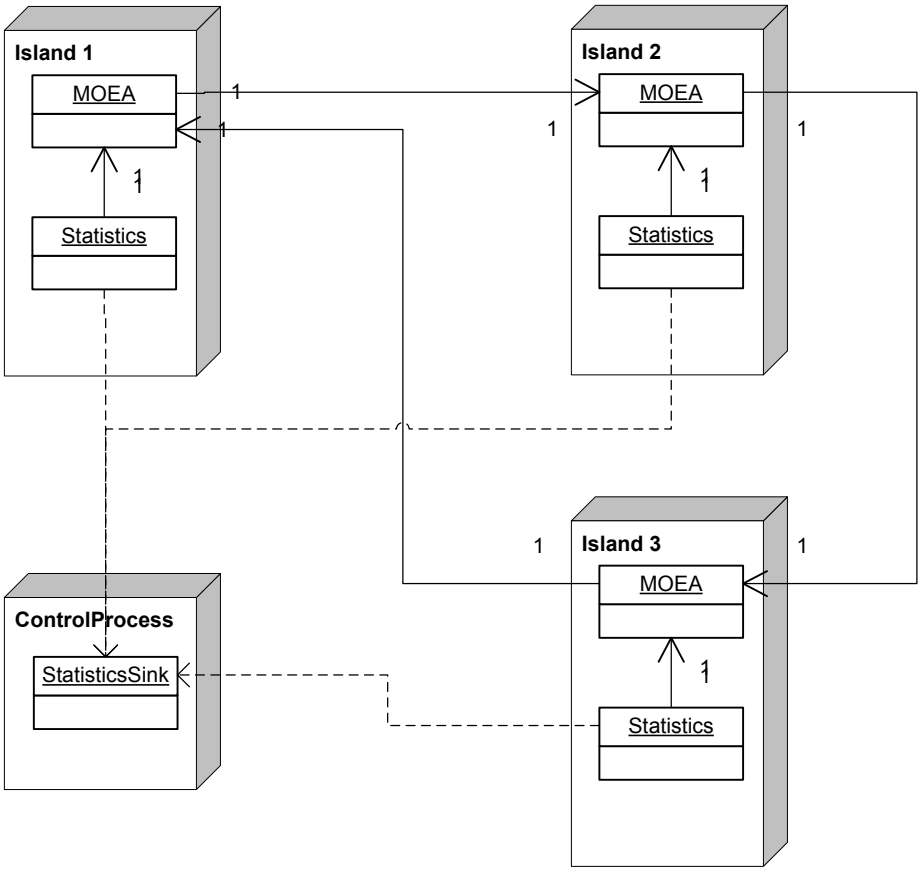

Figura 4-5: Diagrama de despliegue MOE Framework

4.4.2 JMetal

JMetal es una biblioteca de algoritmos evolutivos, entre los que se encuentran implementaciones de MOEAs, algoritmos genéticos simples, estrategias evolutivas, etc. A diferencia del MOE Framework, esta biblioteca no provee de mecanismos de ejecución paralela para los distintos algoritmos. Esta biblioteca esta implementada utilizando el lenguaje Java.

Esta biblioteca esta compuesta por cuatro paquetes principales: base, metaheuristics, problems y util. El paquete base contiene las clases base, estas clases permiten el modelado de los algoritmos, problemas, y los distintos operadores, además de cubrir las funcionalidades mínimas del sistema. El paquete metaheuristics contiene las clases que implementan las distintas metaheurísticas para resolver problemas. El paquete problems contiene una gran cantidad de clases

que definen los problemas. Por ultimo, el paquete util provee un conjunto de clase auxiliares que se utilizan en las distintas metaheurísticas. En la Figura 4-6 se muestra las dependencias entre estos paquetes.

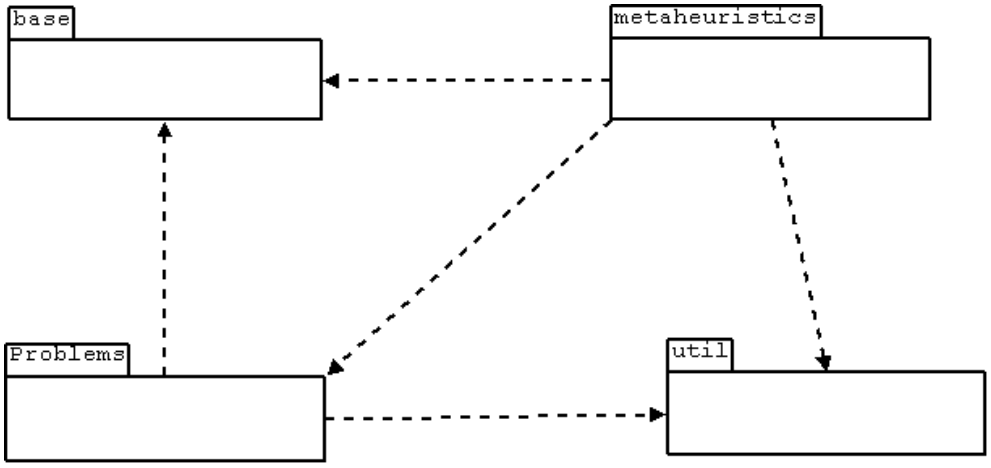

Figura 4-6: Diagrama de paquetes de jMetal

La clase principal de JMetal es Algorithm, todas las metaheurísticas extienden a esta clase. Esta clase provee los métodos addParameter() y getParameter() que permiten agregar y acceder parámetros específicos de las distintas metaheurísticas. De manera similar provee los métodos addOperator() y getOperator (), para agregar y acceder a los operadores necesario para la ejecución de la metaheurística. Por ultimo, el método execute() inicia la ejecución del algoritmo.

La clase SolutionSet representa un conjunto de objetos Solution, estos objetos están compuestos por objetos DesicionVariables, que a su vez están compuestos por un array de objetos Variable. Variable es una interfase que deben implementar las diferentes representaciones que se utilicen en los distintos algoritmos. Estas clases representan a distintos elementos de los algoritmos evolutivos: población, individuos, cromosomas y genes respectivamente.

La clase Problem contiene dos métodos: evaluate() y evaluateConstraints(). Ambos métodos reciben un objeto Solution; el primer método evalúa las funciones objetivos, mientras que el segundo evalúa si la solución viola alguna de las restricciones definidas. Todas las clases que representen a un problema específico deben definir ambos métodos.

La clase Operator representa a un operador genérico utilizado en un algoritmo (cruzamiento, selección, mutación, etc.). La clase Algorithm tienen un conjunto, parameters_, que se utiliza para almacenar los distintos parámetros que necesitan los operadores. En la Figura 4-7 se muestra la dependencia entre las diferentes clases básica de jMetal [35].

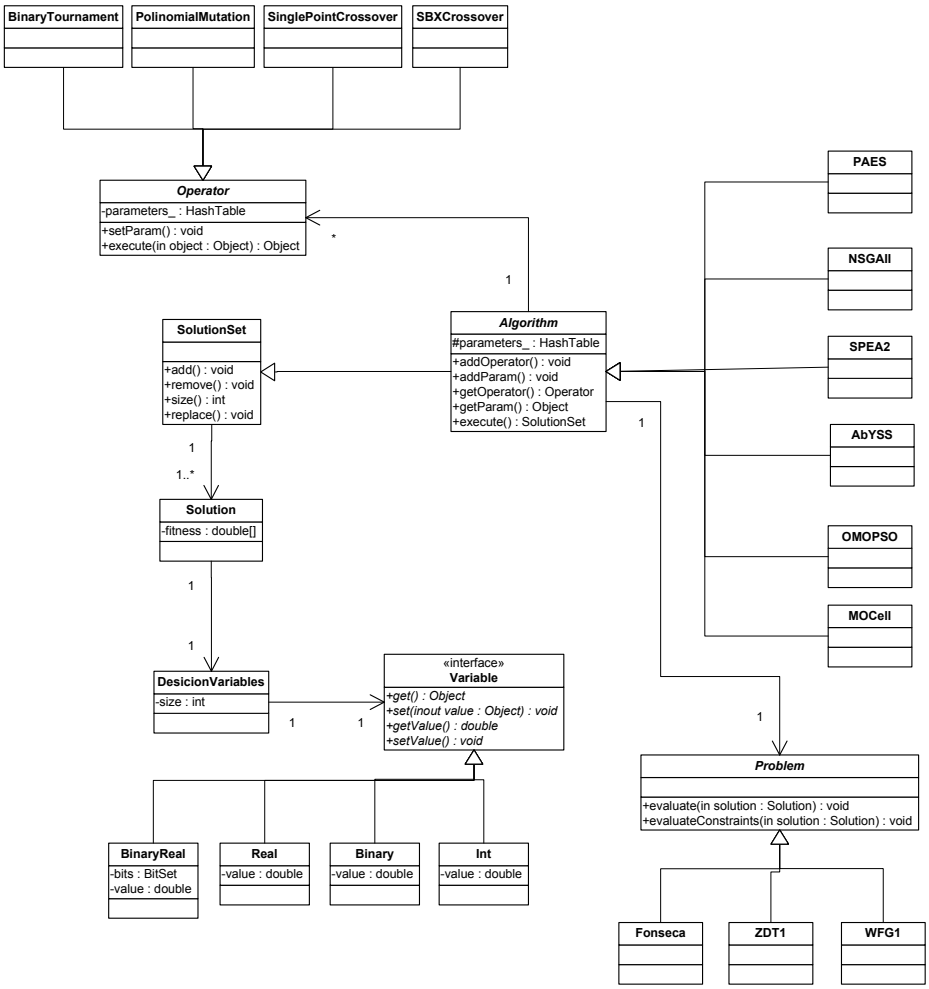

Figura 4-7: Diagrama de clases de jMetal

4.4.3 Sparrow

Sparrow es una biblioteca que brinda al usuario mecanismos para poder ejecutar tareas en forma paralela, de manera transparente para el mismo. Esta biblioteca esta implementada en java

Esta biblioteca esta compuesta por cuatro paquetes principales: driver, worker, aplications, y util. El paquete driver contiene a la clase MasterWoker, esta clase es la que tiene el funcionamiento básico necesario para la ejecución paralela de tareas. El paquete workers contiene las clases worker y workerServer, además de algunas clases auxiliares que sirven de soporte a estas clases. El paquete applications contiene a la clase monitor, que se encarga de mostrar cuantos workers están trabajando y el status que tienen. Por ultimo, el paquete util contiene clases auxiliares que son utilizadas por las clases de los demás paquetes. En la Figura 4-8 se muestran las dependencias entre los paquetes que componen la biblioteca sparrow.

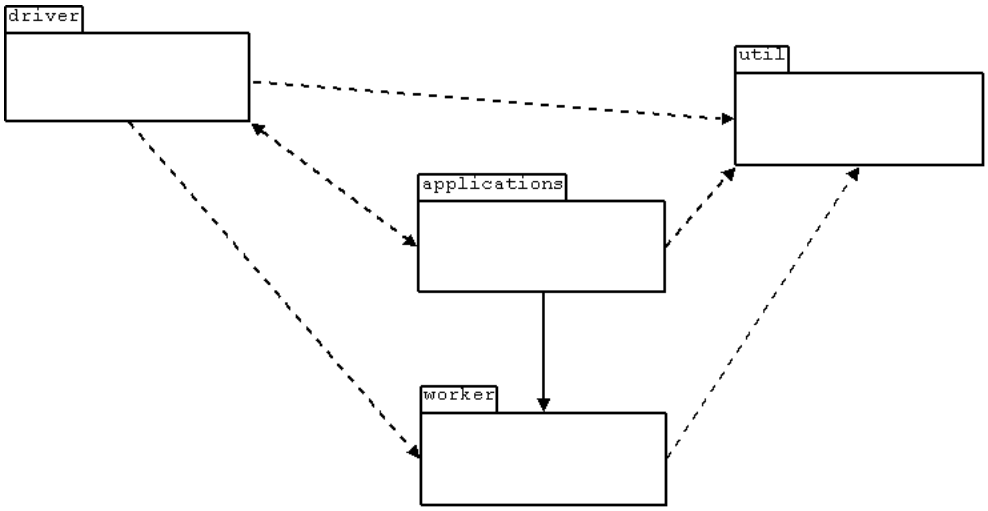

Figura 4-8 : Diagrama de Paquetes de Sparrow

Cuando se quiere implementar un algoritmo utilizando Sparrow es necesario implementar una clase que extienda a la clase MasterWorker, en particular es necesario implementar cuatro métodos de dicha clase: setInitData(), CreateInitialTasks(), newCompletedTask(), y printResults().

En el método setInitData() se inicializan los datos necesarios para la ejecución del algoritmo. Este método debe devolver una instancia de la clase InitData o una subclase de esta, dicha instancia tendrá la definición de la tarea que ejecutará cada uno de los workers.

En el método CreateInitialTasks() se envían las primeras tareas a ejecutarse en paralelo. Para realizar este envío se utiliza la clase Task, o una subclase de esta, donde se definen los parámetros específicos para la ejecución de una tarea particular.

El método newCompletedTask() es ejecutado por el MasterWorker cuando finaliza la ejecución de una tarea. La utilidad de este método es procesar los resultados de la ejecución de una tarea, que vienen en el parámetro completedTask que es una instancia de la clase CompletedTask. En este método se pueden enviar otras tareas a ejecutar o indicar que se termino la ejecución de las tareas.

El MasterWorker invoca al método printResults(), en este método se definen los resultados que se muestran luego de finalizada la ejecución del algoritmo y en que forma se hace.

Para realizar el desplegado de una aplicación utilizando sparrow se procede de la siguiente manera. Primero, en cada uno de los hosts que se va a utilizar para correr un worker, se debe ejecutar el worker pasándole como argumento el puerto TCP en el que va a utilizar. Luego en el host que va a correr el WorkerServer se debe ejecutar el WorkerServer pasándole los siguientes argumentos: un archivo que indique la dirección IP y el puerto TCP de cada Woker, y el puerto TCP en el que va a utilizar. En el MasterWoker se debe tener una referencia al WorkerServer, para esto se le debe indicar la dirección IP y el puerto TCP que utiliza. En la Figura 4-9 se muestra el despliegue de una aplicación implementada sobre Sparrow.

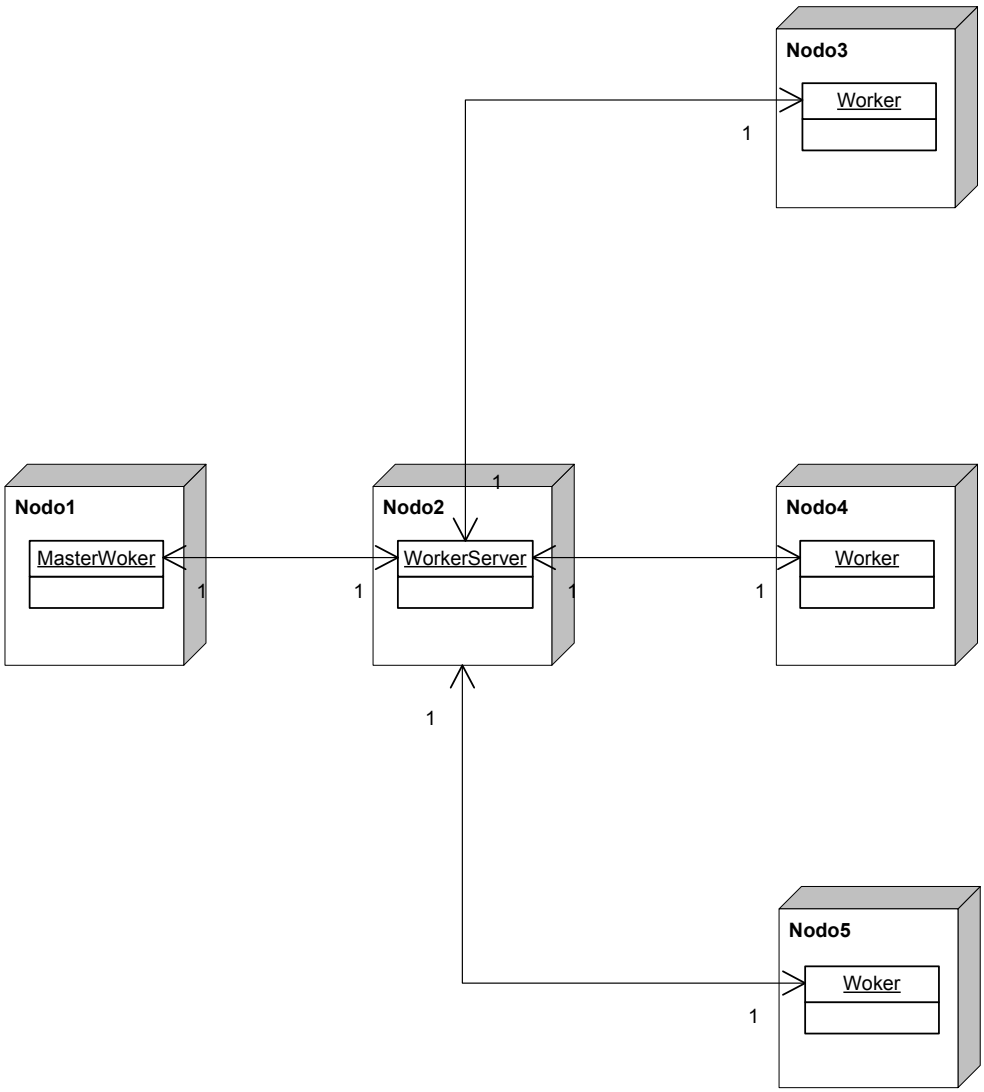

Figura 4-9: Diagrama de Despliegue de Sparrow

4.5 Detalles de la implementación

En esta sección se explica como se lleva a cabo la implementación de los dos MOEA mencionados anteriormente. Se van a mencionar decisiones de diseño que se tomaron a la hora de implementar las clases necesarias para llevar a cabo el algoritmo propuesto.

4.5.1 NSGAII

El algoritmo NSGA-II fue implementado utilizando la biblioteca MOE Framework. Para poder adaptar este MOEA a las necesidades del problema que se intenta resolver fue necesario implementar varias clases, que se explican a continuación.

En primer lugar se implementó la clase ETC, esta clase representa a la matriz ETC que tiene los tiempos de ejecución de las tareas en las distintas maquinas que componen el entorno. Esta clase se implementó siguiendo el patrón de diseño singleton porque es una clase a la cual se debe acceder desde otras clases que no pueden tener una referencia directa a la clase ETC, por ejemplo el operador de mutación necesita obtener datos desde esta clase y no es posible pasarle la clase ETC por parámetro. Esta clase tiene los métodos: getInstance() que nos devuelve una instancia de la clase ETC, load() que se utiliza para inicializar la matriz ETC, getMachineCount() que devuelva la cantidad de maquinas que componen el entorno, getTasksCount()que devuelva la cantidad de tareas que componen el entorno, y getValue() que devuelve el tiempo de ejecución de una tarea en una maquina.

Los operadores de selección (selección por torneo) y de migración (ring migration) elegidos forman parte de la biblioteca, por lo tanto no fue necesario implementarlos. En cambio fue necesario implementar los operadores de cruzamiento y mutación propuestos. El operador de cruzamiento elegido fue el cruzamiento uniforme, como no estaba implementado en la biblioteca fue implementado siguiendo la descripción de este operador.

El operador de mutación es específico para este problema, ya que incorpora inteligencia relacionada al problema. El método mutate, que implementa el método abstracto de la clase base utilizado en la ejecución del algoritmo, se encarga de elegir la estrategia de mutación entre las ocho estrategias de mutación explicadas anteriormente. Luego de determinar que estrategia de mutación utilizar invoca al método correspondiente a dicha estrategia.

Por ultimo se implementó la clase DTRH, donde se definen las funciones objetivo que se buscan optimizar, así como la dirección de optimización (en este caso se busca minimizar las dos funciones objetivo). Además, fue necesario modificar la clase Launcher, que es la encargada de orquestar la ejecución del algoritmo, a fin de incorporar los nuevos operadores implementados.

4.5.2 SPEA-2

Para implementar el algoritmo SPEA-2 fue necesario utilizar dos bibliotecas, jMetal y Sparrow. La primer biblioteca se utilizó para implementar los mecanismos relacionados con el algoritmo genético en si, mientras que la segunda se utilizó para implementar los mecanismos necesarios para la ejecución paralela del algoritmo evolutivo implementado.

Aquí también se implementó una clase ETC, siguiendo el patrón de diseño singleton, con el objetivo de que la misma sea visible desde todas las clases. Esta clase provee los mismos métodos que en el NSGA-II.

Fue necesario implementar el operador de cruzamiento, y de mutación. En casos las clases implementadas extienden a la clase Operator y redefinen el método execute(), procediendo de manera similar que en el SPEA-2.

Por otro lado, fue necesario extender la clase Problem. Con este objetivo se implemento la clase dtr, donde se define el método evaluate() que es el encargado de evaluar el valor de las funciones objetivo que le corresponde a cada solución. No se define el método evaluateConstraints() porque todas las soluciones son factibles.

Como se mencionó anteriormente la biblioteca jMetal no permite la implementación de MOEAs paralelos, para poder realizar dicha implementación se utilizó como complemento a la biblioteca Sparrow. Dicha biblioteca esta diseñada para implementar MOEAs paralelos siguiendo el modelo maestro-esclavo. En este trabajo se utilizó esta biblioteca para implementar la ejecución paralela del SPEA2 siguiendo el modelo de subpoblaciones distribuidas. A continuación se explica como se logró esto.

En primer lugar se implemento la clase spea2InitData que extiende a la clase InitData, esta clase se utiliza para realizar la inicialización de los workers. En nuestro caso particular, se le agrega a la clase InitData un atributo de tipo Algorithm, que va a ser el algoritmo evolutivo que va a ejecutar cada worker (en nuestro caso una instancia de SPEA2).

La clase spea2Task extiende a la clase Task, esta clase tiene dos atributos de tipo SolutionSet: archive_ y solutionSet_. Estos atributos representan al archivo y la población que utiliza el SPEA2 respectivamente. Debido a que los workers ejecutan una tarea pero no se les puede enviar datos mientras se están ejecutando, se ejecutan las generaciones que ocurren entre dos migraciones, por esto se necesitan obtener el archivo y la población resultante de cada isla, para poder enviarla al worker que ejecute la siguiente etapa de la evolución. Esto se explicará en detalle más adelante.

La clase spea2CompletedTask extiende a la clase CompletedTask. Esta clase es utilizada por la clase spea2Task para devolver el archivo y la población obtenida al ejecutar la etapa actual de la evolución. Esto es necesario debido a que la clase que extiende a la clase Task debe implementar el método execute(), dicho método retorna una instancia de la clase CompletedTask.

La clase spea2Driver, que extiende a la clase MasterWorker, es la encargada de ejecutar el algoritmo, para esto la misma provee un método main(). En dicho método se inicializan los parámetros necesarios para la ejecución del algoritmo, así como se genera la población inicial del algoritmo. Dicha población se divide en subpoblaciones para ser enviadas a las islas donde se va a ejecutar el algoritmo. Al final del método main() se invoca al método execute() de la superclase MasterWorker, que es quien se encarga de la comunicación con el WorkerServer para la ejecución paralela del algoritmo evolutivo.

Además del método main() se deben implementar varios métodos necesarios para la ejecución del algoritmo. Estos métodos son: setInitData(), CreateInitialTasks(), newCompletedTask(), printResults().

En el método setInitData() se setean los parámetros necesarios de los algoritmos genéticos que se ejecutaran en las distintas islas. Estos parámetros son: tamaño de la población, tamaño del archivo, operadores evolutivos y cantidad de generaciones en que se ejecutara el algoritmo evolutivo. La cantidad de generaciones en que se va a ejecutar un algoritmo evolutivo en una isla corresponde al periodo de migración, debido a la imposibilidad de enviarle datos a un worker durante la ejecución es necesario realizar la migración en el spea2Driver.

El método CreateInitialTasks() es el encargado de generar las primeras tareas que se le van a enviar a los workers, para esto se crea una instancia de la clase spea2Task para cada isla y se le asignan la subpoblación inicial que le corresponde.

El método newCompletedTask() se ejecuta cuando un worker envía los resultados de la ejecución de la tarea que le fue asignada. En el caso de existan otras islas ejecutándose se almacenan el archivo y la población que se obtiene de la ejecución de la isla y se espera el resto de los resultados. En el caso de que no queden más islas ejecutándose y aun quedan generaciones por ejecutar, se determinan los

migrantes de cada isla y se aplica una politica de migración para determinar si estos son aceptados en la isla a la que se los migró. En caso de que haya finalizado la ejecución paralela, se unen todas las poblaciones y los archivos obtenidos en las islas y se ejecuta la integración panmítica.

El método printResults() es el encargado de imprimir los resultados al finalizar la ejecución del algoritmo, en este caso se imprime en un archivo el frente de Pareto obtenido y en otro archivo se imprime el tiempo de ejecución del algoritmo.

Capítulo 5. Evaluación experimental

Este capítulo describe los resultados obtenidos en los experimentos realizados para validar el comportamiento de los algoritmos descritos en el capítulo anterior. Se describen los casos de prueba utilizados, la plataforma computacional, y la configuración de los parámetros de ejecución utilizadas en los experimentos realizados. Se analizan varios aspectos relacionados a los resultados obtenidos, como son: calidad de dichos resultados en comparación a los resultados de referencia del trabajo, evolución de los frentes de Pareto a lo largo de la ejecución de los algoritmos, y la forma de los frentes de Pareto obtenidos. También se analiza el desempeño computacional, así como la ventaja del uso del paralelismo.

5.1 Casos de prueba

Se utilizaron las instancias de prueba provistas por Braun et. al [15]. Estas instancias consisten en matrices ETC de 512 tareas y 16 maquinas. La elección de dichas instancias de prueba se debe a que han sido utilizadas ampliamente en la evaluación de algoritmos que resuelven el problema de asignación de tareas a procesadores, lo que permite comparar las soluciones obtenidas con las mejores soluciones encontradas hasta el momento y determinar la calidad de las mismas.

Cada instancia de prueba esta definida en un archivo de nombre *u_x_yyzz.0* donde: u indica que se utilizó la distribución uniforme para la generación de la matriz, x indica la consistencia de la matriz (c para las matrices consistentes, i para las matrices inconsistentes, s para las matrices parcialmente consistentes), yy indica la heterogeneidad de las tareas (lo en caso de baja heterogeneidad, hi en caso de alta heterogeneidad), zz indica la heterogeneidad de las maquinas (lo en caso de baja heterogeneidad, hi en caso de alta heterogeneidad).

5.2 Plataforma computacional

Para la ejecución de los algoritmos se utilizó una red de computadores con las siguientes características:

Sistema Operativo: Linux Fedora Core 10.

Procesador: Intel Pentium D 2.80Ghz (dual core)

Memoria: 1GB

Red: Ethernet 10/100 Mb switcheada

MPICH: versión 1.2.7

Java: versión 1.6.0_0

En el caso del algoritmo NSGA-II implementado usando el MOE Framework se necesitaban nueve computadores de dicha red para la ejecución del algoritmo. Por otra parte, se utilizaban once computadores de la red para la ejecución del algoritmo SPEA2 implementado usando JMetal y Sparrow.

5.3 Parámetros de Ejecución

No se realizaron experimentos para analizar la configuración óptima de los valores de los parámetros para los MOEAs implementados. Los valores de los parámetros se eligieron de forma de obtener resultados de buena calidad en tiempos de ejecución breves. A lo largo de la etapa de implementación se realizaron experimentos no formales con el objetivo de evaluar como impactaban los distintos valores de los parámetros en el desempeño de los MOEAs, tanto desde el punto de vista de la calidad de los resultados como del tiempo de ejecución. Se realizaron experimentos considerando cuatro u ocho islas, poblaciones de 200 individuos o de 120 individuos, además se analizaron los desempeños de los operadores de cruzamiento simple y uniforme obteniéndose mejores resultados con el último.

Se definió como criterio de parada de esfuerzo prefijado, los algoritmos se detienen al alcanzar un número fijo de generaciones (10000). Se eligió un valor alto para el límite de generaciones con el objetivo de obtener resultados de buena calidad.

Se buscó un tamaño de población relativamente pequeño (120 individuos, 15 individuos por cada subpoblación) para lograr que la ejecución de los algoritmos se desarrolle en plazos breves. Se descarto usar subpoblaciones con menor cantidad de individuos, ya que esto podría generar problemas de diversidad en las mismas y afectar la calidad de los resultados obtenidos.

Los operadores evolutivos (selección, cruzamiento, mutación) fueron explicados en el capitulo anterior. Se definió que se realizaran las migraciones de individuos entre islas cada 500 generaciones buscando que el tiempo necesario para las comunicaciones entre las distintas islas no afecte el desempeño del algoritmo.

En la tabla 5-1 se detallan los valores de los parámetros utilizados.

Parámetro	Valor
Tamaño de la población	120
Cantidad de subpoblaciones	8
Cantidad de generaciones	10000
Operador de cruzamiento	Uniforme
Probabilidad de cruzamiento	0.9
Operador de mutación	Especifico para el problema
Probabilidad de mutación	0.025
Migración	Sincrónica cada 500 generaciones migrando 1 individuo por vez
Topología de las islas	Anillo unidireccional
Interacciones finales	20

Tabla 5-1: Parámetros de ejecución

5.4 Análisis de resultados experimentales

Para cada instancia se realizaron veinte ejecuciones, obteniéndose luego las soluciones no dominadas del conjunto de todas las soluciones obtenidas. En la tabla 5-2 se presentan los mejores valores de makespan y flowtime obtenidos por los algoritmos, para cada una de las instancias de prueba. Estos valores se pueden comparar con los mejores valores reportados en la literatura, ya que en ambos casos se trata de las mejores soluciones obtenidas por los algoritmos. En dicha tabla se puede observar que los valores de makespan y/o flowtime que mejoran los valores de referencia aparecen en negrita.

En primer lugar se observa que los dos MOEAs implementados obtienen soluciones de buena calidad. Tomando en cuenta las mejores soluciones obtenidas para los distintos escenarios de prueba, se observa que los MOEAs llegan a valores muy buenos de flowtime, donde en la mayoría de los casos se superan los valores de referencia. Se observa claramente que la implementación del algoritmo SPEA2 obtiene mejores valores de flowtime que los valores de referencia en diez de doce instancias, mientras que la implementación del NSGA-II obtiene mejores valores de flowtime que los valores de referencia en once de doce instancias.

	Mejores Resultados		Mejores soluciones obtenidas			
	MA+TS	cMA	SPEA2		NSGA-II	
Instancia	Makespan	Flowtime	Makespan	Flowtime	Makespan	Flowtime
u_c_hihi.0	7530020,2	1037049914,2	7664708,50	**1035559104,00**	7669870,00	**1034100000,00**
u_c_hilo.0	153917,2	27487998,9	154800,41	27511050,00	154650,00	**27470600,00**
u_c_lohi.0	245288,9	34454029,4	250619,17	**34406000,00**	250148,00	**34319000,00**
u_c_lolo.0	5173,7	913976,2	5238,95	915994,44	5230,90	914269,00
u_i_hihi.0	3058474,9	361613627,3	**2972172,00**	**350388640,00**	**2964590,00**	**350056000,00**
u_i_hilo.0	75108,5	12572126,6	**74043,52**	**12434667,00**	**73825,10**	**12433400,00**
u_i_lohi.0	105808,6	12707611,5	**103379,61**	**12228552,00**	**103336,00**	**12230400,00**
u_i_lolo.0	2596,6	439073,7	**2567,31**	**434308,97**	**2561,48**	**434148,00**
u_s_hihi.0	4321015,4	513769399,1	**4313499,00**	**505288224,00**	**4273710,00**	**504447000,00**
u_s_hilo.0	97177,3	16300484,9	**97138,22**	**16194711,00**	**96855,90**	**16181900,00**
u_s_lohi.0	127633,0	15179363,5	**125533,72**	**14863620,00**	**125121,00**	**14848700,00**
u_s_lolo.0	3484,1	594666,0	3497,65	**591651,06**	3490,17	**591381,00**

Tabla 5-2: mejores soluciones obtenidas por los algoritmos

Al comparar los mejores valores de makespan obtenidos por los MOEAs con los valores de makespan de referencia también se observa que en varios casos se mejoraron dichos valores. Se observa que la implementación del algoritmo SPEA2 obtiene mejores valores de makespan que los valores de referencia en siete de doce instancias, mientras que la implementación del NSGA-II también obtiene mejores valores de flowtime que los valores de referencia en siete de doce instancias. Se observa que estas siete instancias coinciden para ambos MOEAs. Se observa que en ninguna de las instancias consistentes se logra obtener mejores valores que los de referencia.

Por otra parte, en la tabla 5-3 se encuentra una comparación entre los valores promedio de makespan y flowtime obtenidos por los algoritmos para cada instancia y los mejores valores presentados en la literatura. También se presenta la desviación estándar (σ) de los promedios de makespan y flowtime obtenidos por los algoritmos para cada una de las instancias de prueba, estos valores nos dan la pauta de que tan robusto es el algoritmo. Esto nos da la pauta de los valores que se pueden llegar a obtener al ejecutarse los algoritmos, teniendo en cuenta que la ejecución de los mismos no es determinista puede ocurrir que en una determinada ejecución no se alcancen los mejores valores reportados. Como en el caso anterior los valores de promedio de makespan y/o flowtime que mejoran los valores de referencia aparecen en negrita.

La importancia de observar los valores promedio de cada una de las soluciones que obtienen los MOEAs se debe a que la ejecución de dichos algoritmos es no determinista lo que implica que no siempre se puedan obtener los mejores resultados reportados. Se ve que en algunas instancias los valores promedio obtenidos de las métricas superan a los mejores valores de referencia, este aspecto da la pauta de que se pueden esperar obtener soluciones de buena calidad en cualquier ejecución de dichas instancias. Se observa que los valores promedios de

flowtime que se obtuvieron superan a mejores valores de flowtime de referencia en la mayoría de los casos (nueve de doce instancias para SPEA2 y diez de doce para NSGA-II), mientras que en el caso del makespan la proporción es bastante menor (tres de doce instancias para SPEA2 y cinco de doce para NSGA-II).

	Mejores Resultados		Promedio de soluciones obtenidas			
	MA+TS	cMA	SPEA2		NSGA-II	
Instancia	Makespan	Flowtime	Makespan	Flowtime	Makespan	Flowtime
u_c_hihi.0	7530020,2	1037049914,2	7874284,53	**1036777936,00**	7752213,50	**1035712000,00**
σ			205953,81	1309050,36	80848,47	1030261,08
u_c_hilo.0	153917,2	27487998,9	155563,50	27542485,67	156045,38	27495533,33
σ			770,72	20569,80	1585,18	23054,94
u_c_lohi.0	245288,9	34454029,4	253019,59	34482939,20	253690,00	**34391467,86**
σ			1983,33	62603,65	3056,94	56541,54
u_c_lolo.0	5173,7	913976,2	5299,45	917000,84	5283,53	915253,25
σ			56,34	656,12	64,22	893,58
u_i_hihi.0	3058474,9	361613627,3	3059688,93	**352031771,59**	**3026429,17**	**352493305,56**
σ			83837,86	1292053,99	69979,50	1513725,98
u_i_hilo.0	75108,5	12572126,6	**74890,12**	**12449207,48**	**74494,91**	**12457086,67**
σ			1118,37	12800,39	1297,52	16682,53
u_i_lohi.0	105808,6	12707611,5	**105527,46**	**12277812,29**	**105120,84**	**12285568,75**
σ			2584,49	42166,60	2152,35	42553,79
u_i_lolo.0	2596,6	439073,7	**2578,78**	**434646,72**	**2594,23**	**434471,05**
σ			10,05	433,16	38,60	321,71
u_s_hihi.0	4321015,4	513769399,1	4403745,06	**507305741,58**	4378180,97	**507235290,32**
σ			93682,65	1776812,32	139688,25	2005151,29
u_s_hilo.0	97177,3	16300484,9	97571,16	**16222015,76**	97626,52	**16209521,43**
σ			936,76	15757,58	848,80	21187,37
u_s_lohi.0	127633,0	15179363,5	127818,64	**14911901,10**	**127129,34**	**14917727,59**
σ			1952,09	40134,64	2058,25	56236,91
u_s_lolo.0	3484,1	594666,0	3518,18	**592671,33**	3523,58	**592425,77**
σ			19,75	740,74	41,29	1005,80

Tabla 5-3: promedios de las soluciones obtenidas por los algoritmos

5.5 Evolución de los frentes de Pareto

A continuación se muestra como evolucionan los frentes obtenidos por los algoritmos implementados para alguna de las instancias con las que se probó el funcionamiento de dichos algoritmos. Se muestra una instancia de cada tipo de consistencia y con distintos tipos de heterogeneidad de tareas y maquinas para que se pueda observar como trabajaron los algoritmos en los distintos escenarios. Este aspecto es muy importante de los algoritmos evolutivos, ya que permite observar si el algoritmo evolutivo se estanca debido a problemas de convergencia prematura.

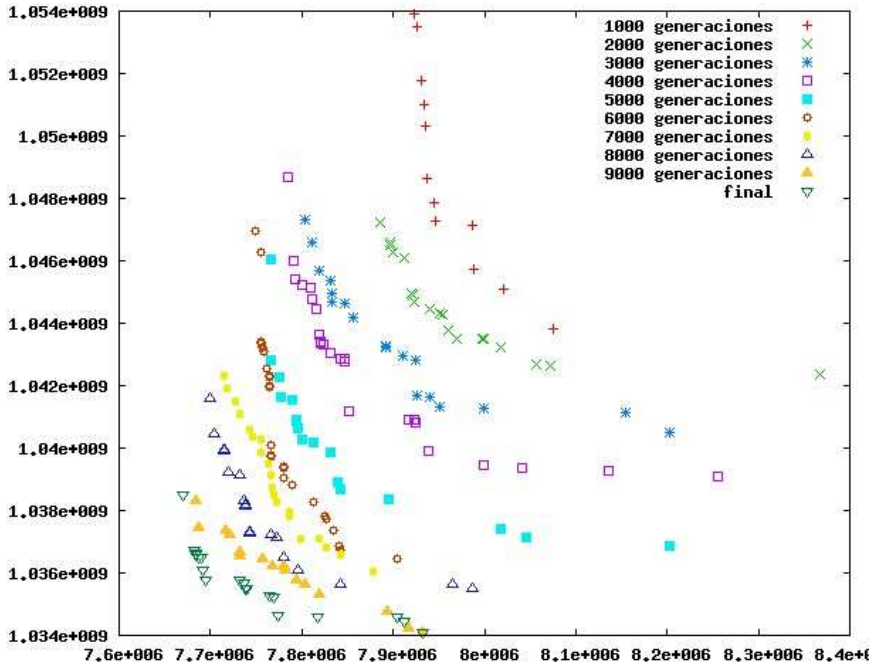

Figura 5-1: Evolución del frente de Pareto en el NSGA-II en la instancia consistente con alta heterogeneidad de tareas y alta heterogeneidad de maquinas.

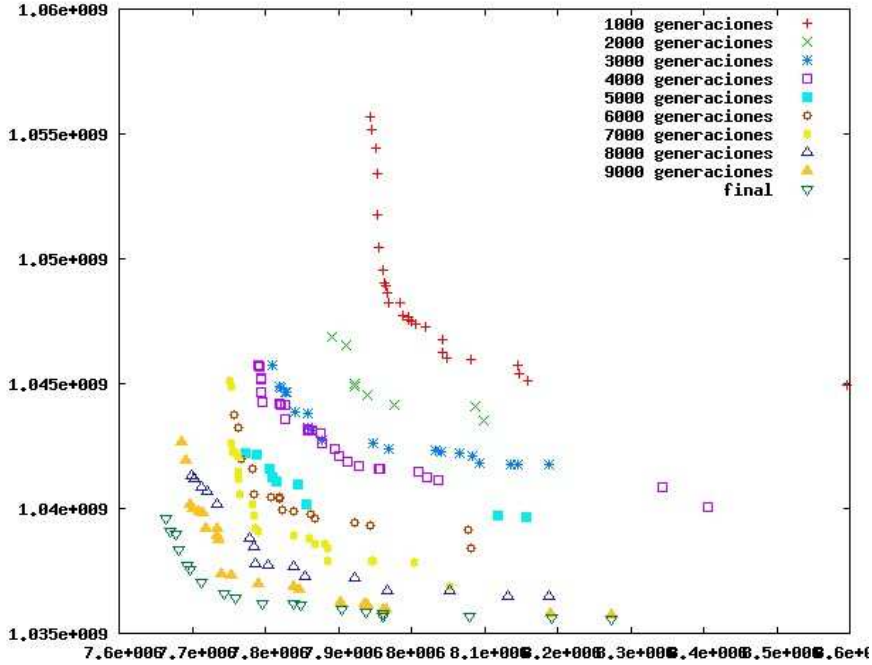

Figura 5-2: Evolución del frente de Pareto en el SPEA2 en la instancia consistente con alta heterogeneidad de tareas y alta heterogeneidad de maquinas.

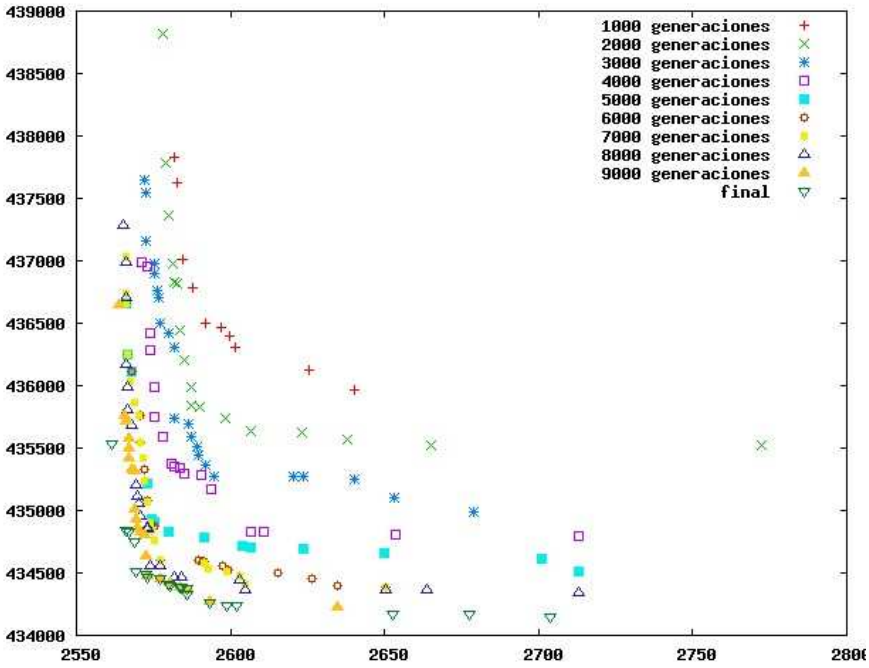

Figura 5-3: Evolución del frente de Pareto en el NSGA-II en la instancia inconsistente con baja heterogeneidad de tareas y baja heterogeneidad de maquinas.

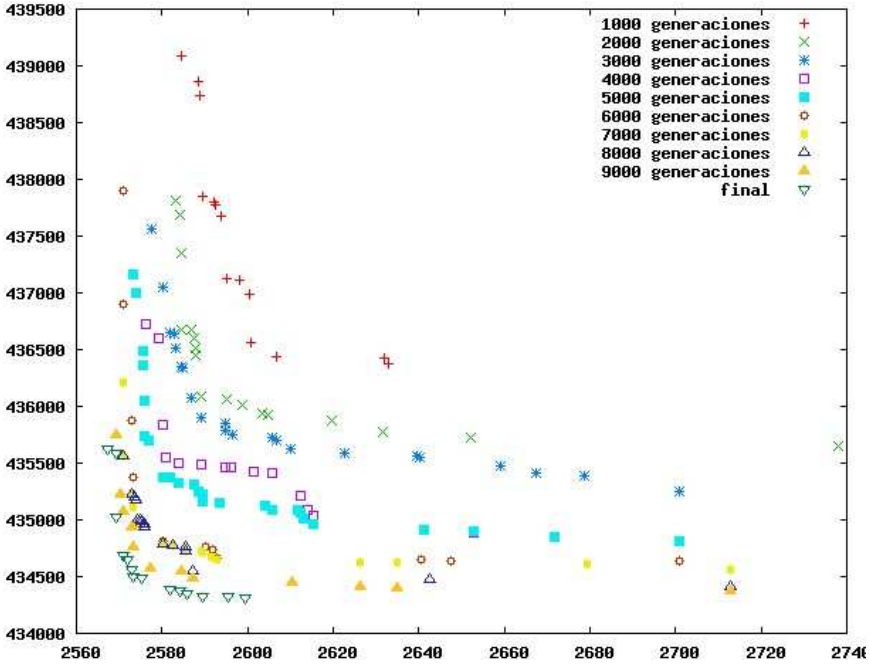

Figura 5-4 : Evolución del frente de Pareto en el SPEA2 en la instancia inconsistente con baja heterogeneidad de tareas y baja heterogeneidad de maquinas.

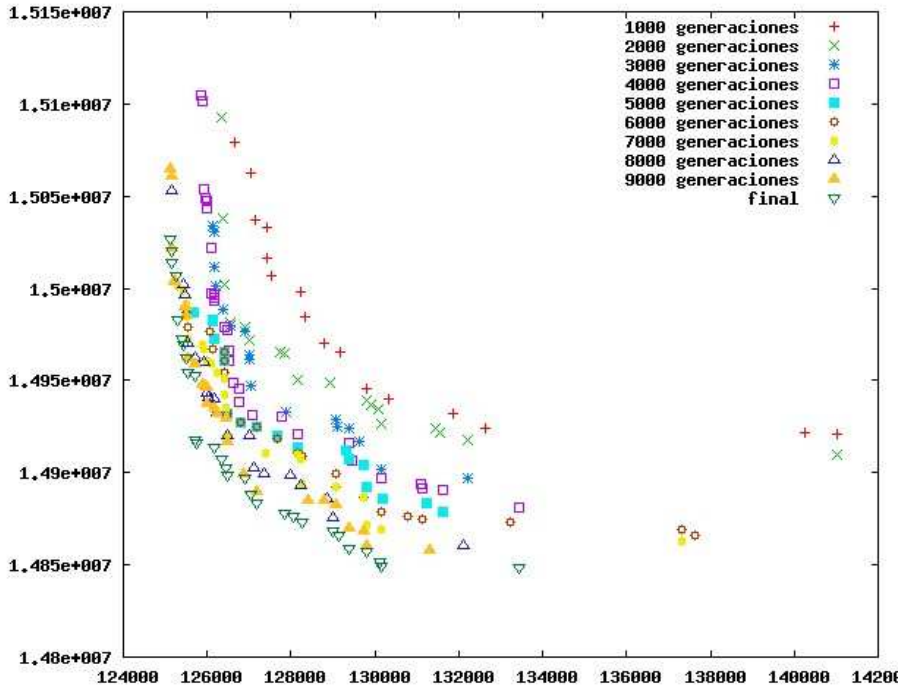

Figura 5-5: Evolución del frente de Pareto en el NSGA-II en la instancia parcialmente-consistente con baja heterogeneidad de tareas y alta heterogeneidad de maquinas.

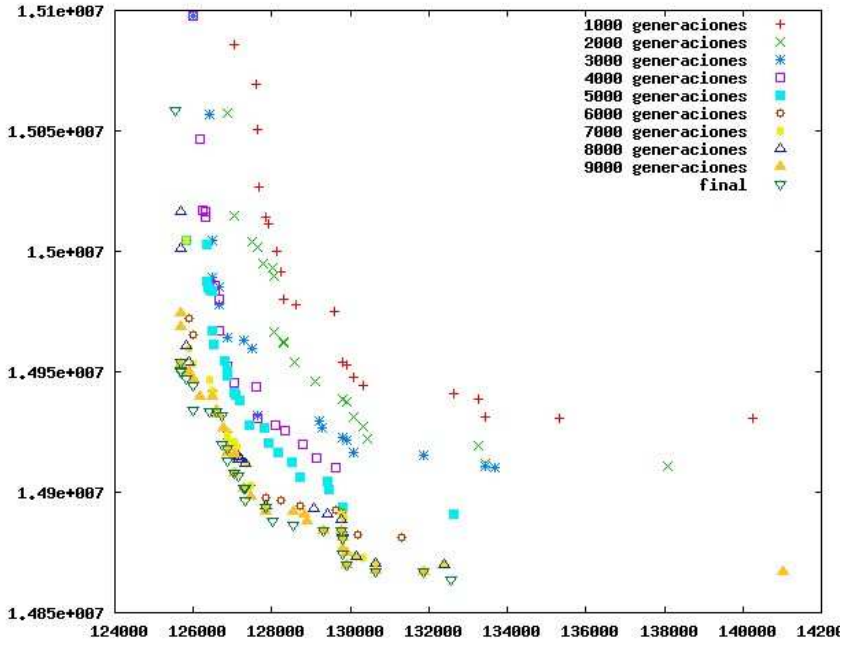

Figura 5-6: Evolución del frente de Pareto en el SPEA2 en la instancia parcialmente-consistente con baja heterogeneidad de tareas y alta heterogeneidad de maquinas.

Observando la evolución de los frentes de Pareto a lo largo de la ejecución de los MOEAs se puede comprobar la validez de los operadores evolutivos implementados, ya que los mismos van mejorando a medida que avanza la ejecución. Esto da la pauta que el algoritmos explora el espacio de soluciones de manera correcta y no converge prematuramente a algún optimo local.

5.6 Frentes de Pareto obtenidos

En esta sección se muestran los frentes de Pareto obtenidos al finalizar la ejecución de los algoritmos implementados para alguna de las instancias con las que se probó el funcionamiento de dichos algoritmos. Es importante observar la forma de los frentes de Pareto que se obtienen, ya que es importante obtener soluciones los más diversas posibles.

Se muestran los frentes obtenidos para las mismas instancias que fueron consideradas en la sección anterior.

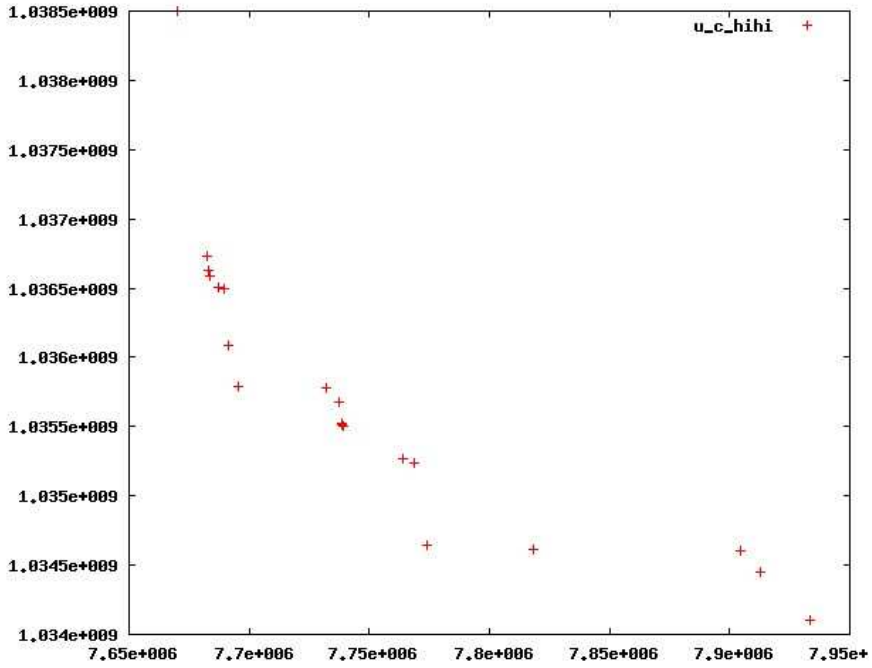

Figura 5-7: Frente de Pareto obtenido en el NSGA-II para la instancia consistente con alta heterogeneidad de tareas y alta heterogeneidad de maquinas.

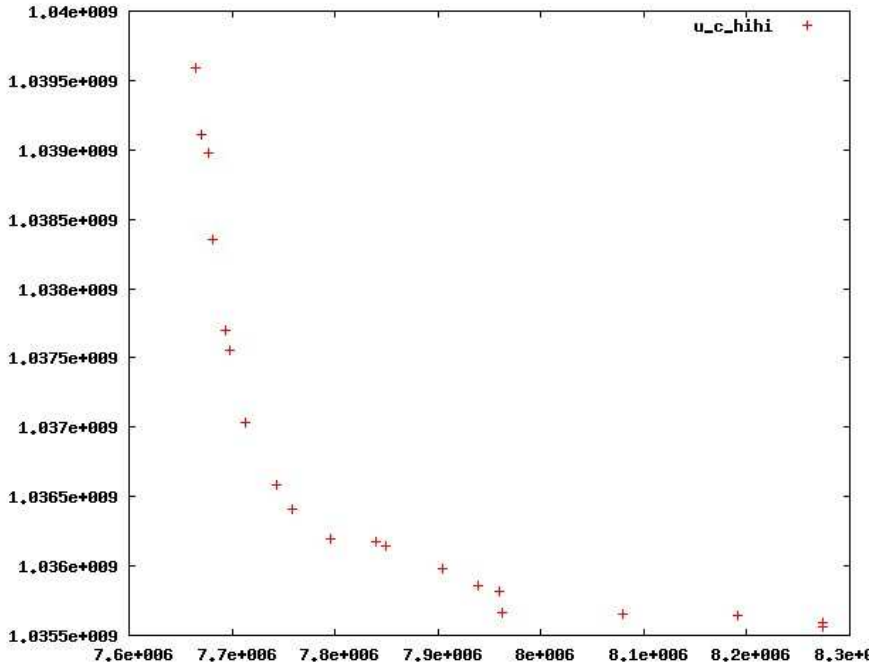

Figura 5-8: Frente de Pareto obtenido en el SPEA2 para la instancia consistente con alta heterogeneidad de tareas y alta heterogeneidad de maquinas.

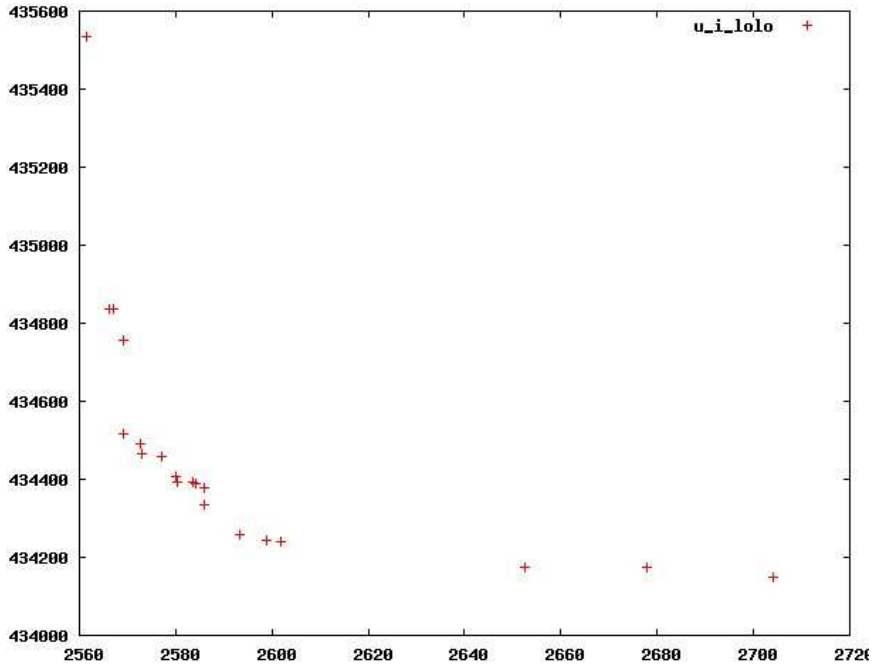

Figura 5-9: Frente de Pareto obtenido en el NSGA-II para la instancia inconsistente con baja heterogeneidad de tareas y baja heterogeneidad de maquinas.

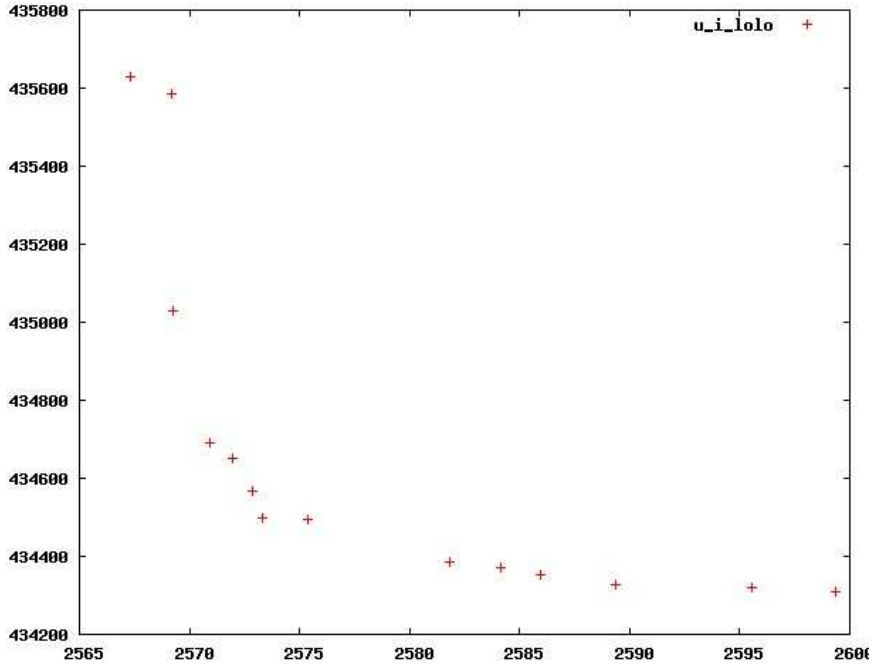

Figura 5-10: Frente de Pareto obtenido en el SPEA2 para la instancia inconsistente con baja heterogeneidad de tareas y baja heterogeneidad de maquinas.

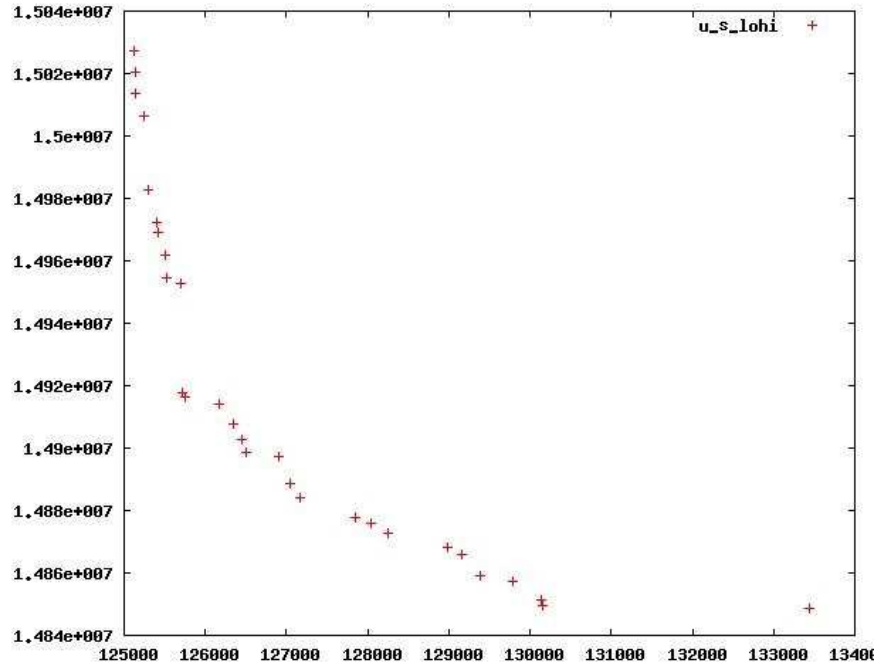

Figura 5-11: Frente de Pareto obtenido en el NSGA-II para la instancia parcialmente-consistente con baja heterogeneidad de tareas y alta heterogeneidad de maquinas.

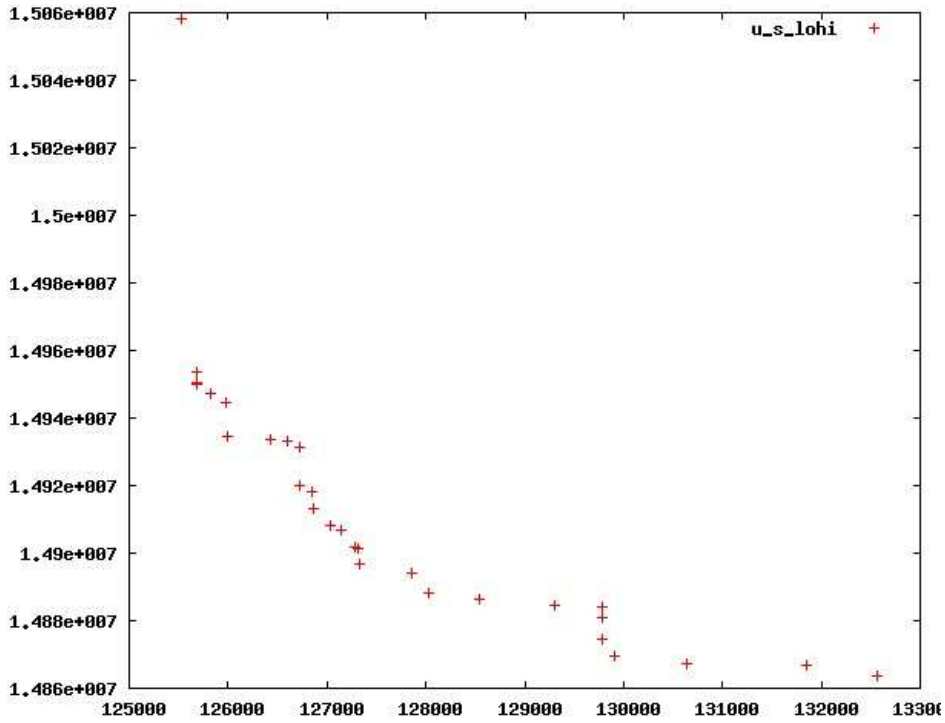

Figura 5-12: Frente de Pareto obtenido en el SPEA2 para la instancia parcialmente-consistente con baja heterogeneidad de tareas y alta heterogeneidad de maquinas.

Observando la forma de los frentes de Parte obtenidos, se puede comprobar que se obtienen puntos dispersos en el espacio de soluciones. En este caso no es posible determinar si se logra muestrear toda la zona correspondiente al frente de Pareto, ya que para eso sería necesario conocer el frente de Pareto real del problema que en este caso no se conoce.

5.7 Análisis de métricas de calidad

Debido a que no se conocen los frentes de Pareto reales para los casos de pruebas utilizados no es posible evaluar los resultados obtenidos por los MOEAs implementados con todas las métricas presentadas en la subsección 3.4.5. Las métricas utilizadas son: numero de puntos no dominados y spacing. Como se menciono anteriormente, el número de puntos no dominados permite ver la diversidad de las soluciones encontradas por el algoritmo. El spacing, en cambio, evalúa la distribución de los puntos no dominados.

En la tabla 5-4 se muestran los valores obtenidas de las métricas utilizadas para determinar la calidad de las soluciones, donde ND se refiere al numero de puntos no dominados. Estas son las únicas métricas que pueden ser aplicadas a los frentes de Pareto obtenidos, ya que no se conoce el frente de Pareto real de estos problemas.

	SPEA2		NSGA-II	
Instancia	ND	Spacing	ND	Spacing
u_c_hihi.0	20	29,912151	20	31,622776
u_c_hilo.0	18	29,065342	21	29,033602
u_c_lohi.0	20	28,938740	28	30,033316
u_c_lolo.0	30	6,901066	16	11,428040
u_i_hihi.0	29	31,622776	36	31,622776
u_i_hilo.0	27	21,419180	15	30,541304
u_i_lohi.0	35	29,847479	32	29,315855
u_i_lolo.0	15	6,849953	20	7,743452
u_s_hihi.0	33	31,622776	31	31,622776
u_s_hilo.0	17	27,206064	28	27,894762
u_s_lohi.0	29	28,565592	29	31,622776
u_s_lolo.0	27	7,763334	31	8,540646

Tabla 5-4: métricas de calidad de las soluciones

Se observa que los puntos no dominados varían entre un 12% y un 30% del total de los individuos de la población. Se puede concluir que los algoritmos obtienen frentes de Pareto con una buena diversidad de sus soluciones.

Por otro lado se observa que los valores de spacing obtenidos por los algoritmos son mayores a cero, por lo que podemos concluir que los puntos obtenidos no están equiespaciados a lo largo del frente de Pareto, se observa menores valores de spacing en los casos de baja heterogeneidad de tareas y baja heterogeneidad de maquinas. Observando la forma de los frentes de Pareto que se presentan en la sección anterior, se comprueba que hay una mayor concentración de puntos no dominados en la parte central de los frentes de Pareto obtenidos, mientras que aparecen más dispersos en los extremos de los frentes.

5.8 Análisis de la eficiencia computacional

Otro objetivo planteado en el trabajo es la obtención de buenos resultados en tiempos de ejecución breves, para cumplir con dicho objetivo se optó por la implementación de MOEAs paralelos. Para poder determinar la mejora en el desempeño que presentan dichos algoritmos se realizó la ejecución de algoritmos

seriales con la misma configuración de parámetros que los algoritmos paralelos. En la Tabla 5-5 se presenta un breve análisis de la eficiencia computacional, mostrando los tiempos promedio de ejecución (en segundos) de las ejecuciones independientes realizadas. También se presentan los valores obtenidos de las métricas de performance utilizadas para evaluar la mejora del desempeño.

	NSGA-II				SPEA2			
Instancia	Paralelo	Serial	Speedup	Eficiencia	Paralelo	Serial	Speedup	Eficiencia
u_c_hihi.0	136,54	981,82	7,19	0,90	144,90	1820,83	12,57	1,57
u_c_hilo.0	80,68	634,69	7,87	0,98	149,00	1384,58	9,29	1,16
u_c_lohi.0	129,83	979,99	7,55	0,94	160,90	1169,83	7,27	0,91
u_c_lolo.0	79,10	619,24	7,83	0,98	162,95	1368,24	8,40	1,05
u_i_hihi.0	67,47	529,94	7,85	0,98	139,45	994,07	7,13	0,89
u_i_hilo.0	67,16	529,78	7,89	0,99	172,95	1211,07	7,00	0,88
u_i_lohi.0	66,74	532,09	7,97	1,00	137,60	1079,62	7,85	0,98
u_i_lolo.0	66,35	528,88	7,97	1,00	153,35	1372,25	8,95	1,12
u_s_hihi.0	83,28	638,39	7,67	0,96	149,75	914,21	6,10	0,76
u_s_hilo.0	70,85	566,30	7,99	1,00	190,85	1305,82	6,84	0,86
u_s_lohi.0	82,63	1241,26	15,02	1,88	155,40	1005,85	6,47	0,81
u_s_lolo.0	73,07	1079,65	14,77	1,85	167,40	1292,68	7,72	0,97

Tabla 5-5: Tiempos de ejecución promedio (segundos).

Analizando los tiempos de ejecución para cada algoritmo, es posible observar que los algoritmos paralelos proveen una mejora sustancial en el desempeño. Ambos algoritmos implementados presentan muy buenos valores de speedup y de eficiencia. Incluso se observan algunos casos en que los algoritmos obtienen valores de speedup superlineal, o sea que el rendimiento del algoritmo paralelo respecto al algoritmo serial aumenta en un factor mayor a la cantidad de procesadores que se utilizan en paralelo.

Los valores de eficiencia obtenidos son también muy buenos, ya que los registros más bajos obtenidos son valores muy cercanos al valor 1.0, que es el valor que identifica situaciones ideales de mejora de performance.

Otro aspecto a analizar es la comparación de los tiempos de ejecución que presentan los algoritmos implementados. En todos los casos se observa que el algoritmo NSGA-II obtiene mejores tiempos de ejecución que el algoritmo SPEA2. Hay que tomar en cuenta que los algoritmos fueron implementados utilizando plataformas distintas, y era previsible que el algoritmo NSGA-II implementado utilizando C++ presentará tiempos de ejecución menores que el algoritmo SPEA2 implementado utilizando Java.

En la figura 5-13 se muestra el speedup promedio obtenido por los algoritmos a lo largo de los experimentos, observándose que el NSGA-II presenta speedup superlineal mientras que el SPEA2 presenta un speedup casi lineal. Observando esta grafica se confirma que los algoritmos presentan buenos valores de speedup.

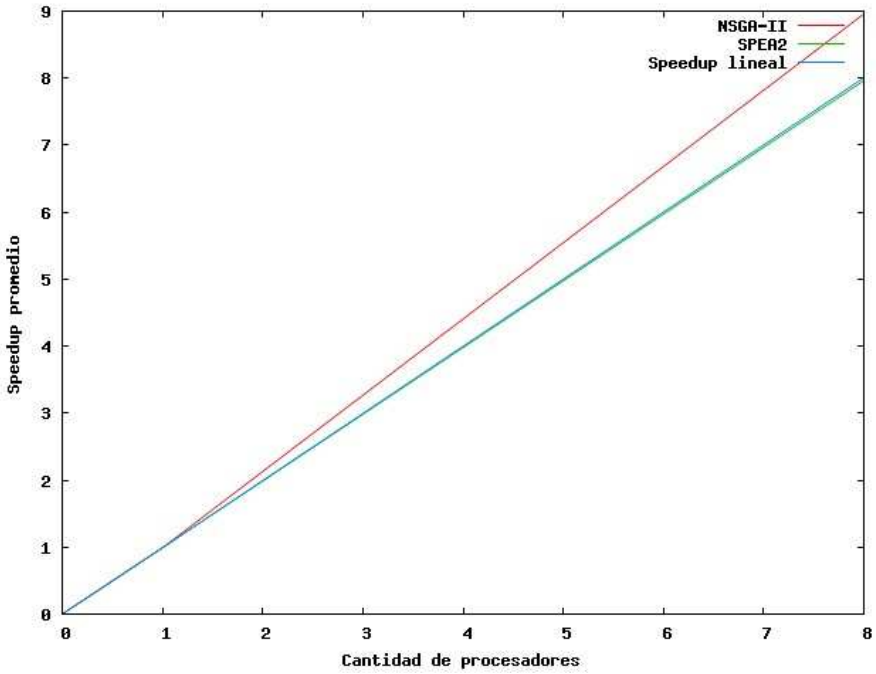

Figura 5-13: Speedup promedio de los algoritmos

Capítulo 6. Conclusiones y trabajo futuro

En el marco del proyecto se llevaron a cabo varias tareas de investigación, diseño, implementación y análisis experimental. En primer lugar se estudiaron en profundidad los fundamentos de las técnicas de computación heterogénea desde un punto de vista general. En esta etapa se obtuvieron conocimientos sobre la utilización de entornos HC para resolver problemas donde es necesario ejecutar un conjunto de tareas con distintas necesidades computacionales. Se analizaron diferentes propuestas planteadas para resolver la planificación de tareas en entornos HC, observando fundamentalmente los supuestos en que se basan los mismos y las métricas que utilizan para evaluar la calidad de las planificaciones que obtenían. Por otra parte se hizo un relevamiento de dos ejemplos (BOINC, WCG) del uso actual de la computación heterogénea para la resolución de problemas. Al finalizar el relevamiento bibliográfico se llego a la conclusión de que en la literatura no se ha utilizado el enfoque multiobjetivo para resolver el problema de planificación de tareas en entornos HC utilizando las instancias de Braun et al., y por lo tanto era un enfoque que, debido a las características del problema, planteaba la posibilidad de obtener resultados de buena calidad usando el mismo.

En el marco del trabajo se realizó un relevamiento de los métodos de aplicación de las técnicas de procesamiento paralelo y distribuido a los algoritmos evolutivos. En primer lugar se determinó que los MOEAs a utilizar como bases de la implementación de los algoritmos serian el NSGA-II y el SPEA2, esto debido a que son los MOEAs que han presentado mejores resultados respecto a otras propuestas de MOEAs en la investigación en el área. Una vez determinados los MOEAs a utilizar se determinaron las bibliotecas a utilizar. Para la implementación del Algoritmo NSGA-II se utilizó el MOE Framework que es una biblioteca de MOEAs paralelos surgida en un proyecto de grado de la facultad en el año 2007, se optó por esta biblioteca porque los resultados obtenidos en las pruebas iniciales de la biblioteca eran buenos, presenta mecanismos de incorporación de operadores evolutivos sencillos, posee un manejo transparente del paralelismo, y además se consideró una buena oportunidad para probar dicha biblioteca en la resolución de un problema más complejo que los problemas que se habían resuelto con la misma.

En el caso del algoritmo SPEA2, no se pudo utilizar el MOE Framework ya que esta metaheurística no esta implementada en dicha biblioteca. Por esto se optó por utilizar la biblioteca JMetal, que es una biblioteca desarrollada en Java que incluye la implementación de una gran cantidad de metaheurísticas entre las que se encuentra la implementación de SPEA2. Esta biblioteca también presenta mecanismos de incorporación de operadores evolutivos relativamente sencillos, hecho que facilitó el trabajo. La dificultad que presentaba esta biblioteca era que no contaba con la posibilidad de paralelizar la ejecución de sus algoritmos, por estos se consultó a quienes desarrollaron dicha biblioteca. Dichas personas enviaron una biblioteca complementaria, llamada Sparrow, que permite la ejecución de algoritmos evolutivos de forma paralela utilizando el modelo maestro-esclavo y un código de ejemplo de una implementación de NSGA-II distribuido siguiendo este modelo. Combinando ambas bibliotecas y adaptando la ejecución paralela para seguir el modelo de subpoblaciones distribuidas en lugar del modelo maestro-esclavo, se logro implementar el algoritmo SPEA2 paralelo.

Para la implementación de los algoritmos mencionados se analizaron los distintos operadores evolutivos a utilizar, con el objetivo de obtener resultados de buena calidad en tiempos de ejecución breves. Se utilizaron mecanismos que incorporen inteligencia a los operadores y no mecanismos puramente aleatorios, porque de esta forma se obtienen mejores resultados.

Para la evaluación experimental de los algoritmos se utilizaron las instancias de prueba de Braun et al, debido a que han sido ampliamente utilizados en el estudio de este problema y da la posibilidad de comparar los resultados obtenidos con los resultados publicados para el caso monoobjetivo.

Sobre dichas instancias de prueba, se realizó un estudio de los resultados obtenidos por los MOEAs Paralelos. Además se ejecutaron las versiones seriales de dichos MOEAs a fin de realizar un análisis de la eficiencia computacional de los algoritmos implementados.

El análisis comparativo de los resultados obtenidos con los resultados publicados para el caso monoobjetivo, permitió determinar que los resultados obtenidos por ambos algoritmos son de buena calidad y de la validez de la utilización del enfoque multiobjetivo para la resolución del problema de planificación de tareas en entorno HC. Por otra parte, se pudo comprobar las que la utilización de modelos paralelos presenta grandes ventajas, tanto desde el punto de vista de la calidad de resultados obtenidos como desde el punto de vista de la mejora de eficiencia computacional.

Por otra parte, seria deseable analizar la utilización de otros operadores evolutivos buscando mejorar los resultados obtenidos por los algoritmos presentados en el marco de este proyecto. En este aspecto es clave revisar la implementación del operador de mutación, ya que es el encargado de incorporar inteligencia específica del problema a la ejecución de los algoritmos.

Sería interesante implementar otros algoritmos evolutivos para resolver el problema de planificación de tareas en entornos HC, con el objetivo de comparar su desempeño respecto a los algoritmos implementados. La utilización de otras metaheurísticas como simulated annealing, tabu search, e incluso algoritmos híbridos surgidos de la combinación de alguna de las metaheurísticas mencionadas anteriormente, por ejemplo: algoritmo genético + simulated annealing, o un algoritmo genético que utilice tabu search para realizar una búsqueda local. También se podría analizar la utilización de otras técnicas evolutivas como por ejemplo: estrategias de evolución, programación genética, etc.

Otro aspecto interesante para profundizar el estudio presentado en este proyecto es la utilización de otros modelos multiobjetivos, o sea utilizar otros criterios de optimización. Aparecen como modelos posibles la optimización del makespan y utilización de las maquinas de una planificación, y la optimización del makespan y confiabilidad de una planificación. Para llevar a cabo este punto es necesario revisar los operadores evolutivos utilizados y ver su aplicabilidad a estos modelos, e incluso evaluar modificaciones a dichos operadores a fin de mejorar el desempeño del algoritmo.

En otra línea de trabajo se podrían realizar pruebas con instancias más realistas de entornos HC, instancias con cientos de procesadores o incluso con instancias que representen a un grid (miles de procesadores), con el objetivo de evaluar el desempeño de los algoritmos implementados para este tipo de entornos.

Referencias

[1]. A. Khokhar, V. Prasanna, M. Shaaban, y C. Wang. Heterogeneous Computing: Challenges and Opportunities. In *Computer*, vol. 26, no. 6, pp. 18-27, June 1993.

[2]. M. Maheswaran, T. Braun, y H. Siegel. Heterogeneous Distributed Computing. In *the Encyclopedia of Electrical and Electronics Engineering, J. G. Webster*, editor, John Wiley & Sons, New York, NY, Vol. 8, pages 679-690, 1999.

[3]. S. Ali, T. Braun, H. Siegel, y A. Maciejewski. Heterogeneous computing. In *Encyclopedia of Distributed Computing*, J. Urbana and P. Dasgupta, eds., Kluwer Academic Publishers, Norwell, MA, 2002.

[4]. Y. Liu. Grid Scheduling. Technical Report, Department of Computer Science, University of Iowa.

[5]. X. He, X. Sun, G. Von Laszewski. A QoS Guided Scheduling Algorithm for Grid Computing. In *Int. Workshop on Grid and Cooperative Computing (GCC02)*, pages 442-450, 2002.

[6]. N. Fujimoto, K. Hagihara. A Comparison among Grid Scheduling Algorithms for Independent Coarse-Grained Tasks. In *Proc. of the 2004 Symp. on Applications and the Internet-Workshops*, pages 674-680, 2004.

[7]. M. Maheswaran, S. Ali, H. Siegel, D. Hensgen, and R. Freund. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. In *Journal of Parallel and Distributed Computing*, volumen 59, pages 107-131, 1999.

[8]. A. Dogan y F. Ozguner. Biojective Scheduling Algorithms for Execution Time-Reliability Trade-off in Heterogeneous Computing Systems. In *The Computer Journal Vol. 48 No. 3*, pages 300- 314, 2005.

[9]. S. Song, Y. Kwok, y K. Hwang. Security-Driven Heuristics and A Fast Genetic Algorithm for Trusted Grid Job Scheduling. *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers - Volume 01*, Page: 65.1, 2005.

[10]. S. Ali, H. Siegel, M. Maheswaran, D. Hensgen, y S. Ali. Representing Task and Machine Heterogeneities for Heterogeneous Computing Systems. In *Journal of Science and Engineering, Special 50 th Anniversary Issue*, volume 3, pages 195-207, 2000.

[11]. List of distributed computing projects. Web Site: http://en.wikipedia.org/wiki/List_of_distributed_computing_projects. Consultado en Noviembre 2008.

[12]. BOINC Web Site: http://boinc.berkeley.edu/. Consultado en Febrero 2009.

[13]. Chossing BOINC Project. Web Site: http://boinc.berkeley.edu/projects.php. Consultado en Febrero 2009.

[14]. World Community Grid - Research. Web Site: http://www.worldcommunitygrid.org/projects_showcase/viewResearch.do. Consultado en Febrero 2009.

[15]. T. Braun, H. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. Reuther, J. Robertson, M. Thies, B. Yao, D. Hensgen, and R. Freund. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. In *Journal of Parallel and Distributed Computing 61*, pages 810-837, 2001.

[16]. R. Armstrong, D. Hensgen, and T. Kidd. The Relative Performance of Various Algorithms is Independent of Sizable Variances in Run-time Predictions. In 7th IEEE Heterogeneous Computing Workshop (HCW ’98). pages 79-87, 1998.

[17]. Y. Li, J. Antonio. Estimating the Execution Time Distribution for a Task Graph in a Heterogeneous Computing System. In *Proceedings of the 6th Heterogeneous Computing Workshop (HCW '97)*, page: 172, 1997.

[18]. T. Braun, H. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. Reuther, J. Robertson, M. Thies, B. Yao. A Taxonomy for describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems. In *IEEE Workshop on Advances in Parallel and Distributed Systems*, West Lafayatte, IN, October 1998, pages 330-335, (included in the *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, 1998).

[19]. F. Xhafa, E. Alba, and B. Dorronsoro. Efficient Batch Job Scheduling in Grids using Cellular Memetic Algorithms. In *Journal of Mathematical Modelling and Algorithms*, pages 217-236, Springer Netherlands, febrero de 2008.

[20]. D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[21]. M. Flynn and K. Rudd. Parallel architectures. *ACM Computing Surveys*, 28(1):67–70, 1996.

[22]. C. Fonseca and P. Fleming. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In *Genetic Algorithms: Proceedings of the Fifth International Conference*, pages 416–423. Morgan Kaufmann, 1993.

[23]. K. Deb. Non-linear Goal Programming Using Multi-Objective Genetic Algorithms. Technical Report No. CI-60/98. Department of Computer Science/XI. University of Dortmund, Germany. October 1998.

[24]. R. Purshouse, P. Fleming. The MultiObjective Genetic Algorithm Applied To Benchmark Problems - An Análisis. Research Report No. 796. Department of Automatic Control and Systems Engineering University of Sheffield, Sheffield, S1 3JD, UK. 2001.

[25]. K. Deb. Multi-Objective Evolutionary Algorithms: Introducing Bias Among Pareto-Optimal Solutions. KanGAL Report No. 99002, Kanpur Genetic

Algorithms Laboratory (KanGAL), Department of Mechanical Engineering, Indian Institute of Technology Kanpur, Kanpur, PIN 208 016, India. 1999.

[26]. J. Horn and N. Nafpliotis. Multiobjective Optimization Using The Niched Pareto Genetic Algorithm. Published, in part, in the *Proceeding of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, Volume 1, 1994 (ICEC '94)*.

[27]. E. Zitzeler and L. Thiele. An evolution Algorithm for Multiobjective Optimization: The Strength Pareto Approach. Technical Report 43, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 1998. url: http://e-collection.ethbib.ethz.ch/view/eth:24834

[28]. E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Department of Electrical Engineering, Swiss Federal Institute of Technology (ETH) Zurich, ETH Zentrum, Gloriastrasse 35, CH-8092 Zurich, Switzerland, mayo 2001.

[29]. J. Knowles, and D. Corne. The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Pareto Multiobjective Optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on Volume 1, Issue , 1999 Page(s): - 105 Vol. 1*.

[30]. K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 849–858, Paris, France, 2000. Springer.

[31]. M. Kilic. Multiobjective Genetic Algorithm Approaches To Project Scheduling Under Risk. MSc Thesis. Sabanci Univesity. Agosto 2003.

[32]. R. Baños Navarro. Meta-heurísticas Hibridas para Optimización Mono-objetivo y Multi-objetivo. Paralelización y Aplicaciones. PhD Thesis, Departamento de Arquitectura de Computadores y Electronica, Universidad de Almería, España, 2006.

[33]. A. Rodríguez. Paralelismo Aplicado a Algoritmos Evolutivos Para Optimización Multiobjetivo. Documentación técnica, 2007. Proyecto de grado. Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay.

[34]. J. Durillo, A. Nebro, F. Luna and E Alba. A Study of Master-Slave Approaches to Parallelize NSGA-II. Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, E.T.S.I. Informática.

[35]. J. Durillo, A. Nebro, F. Luna, B. Dorronsoro, E. Alba. jMetal: a Java Framework for Developing Multi-objective Optimization Metaheuristics. Technical Report ITI-2006-10, Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, E.T.S.I. Informática, Campus de Teatinos, diciembre 2006. url: http://neo.lcc.uma.es/staff/paco/pdfs/TECHREP%20ITI-2006-10.pdf

[36]. S. Nesmachnow. Una Versión Paralela del Algoritmo Evolutivo para Optimización Multiobjetivo NSGA-II y su Aplicación al Diseño de Redes de

Comunicaciones Confiables. Technical Report TR0403, Centro de Cálculo, Instituto de Computación, Universidad de la República, Uruguay, 2004. url: http://www.fing.edu.uy/inco/pedeciba/bibliote/reptec/TR0403.pdf

Anexo

Aquí se muestran la evolución de los frentes de Pareto para todas las instancias de prueba en ambos algoritmos. También se muestran todos los frentes de Pareto obtenidos al finalizar la ejecución de los algoritmos para todas las instancias de prueba.

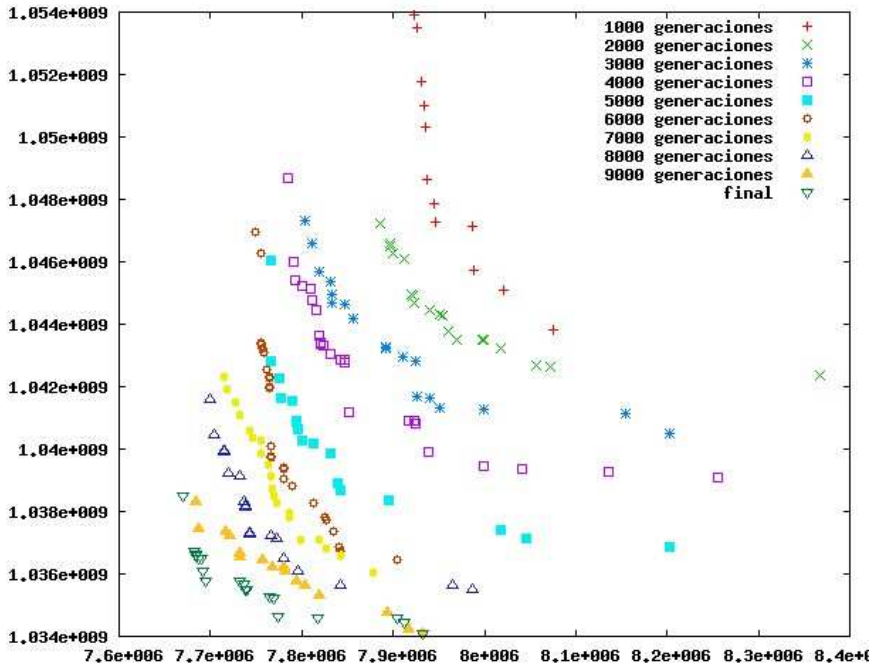

Figura A - 1: Evolución del frente de Pareto en el NSGA-II en la instancia consistente con alta heterogeneidad de tareas y alta heterogeneidad de maquinas.

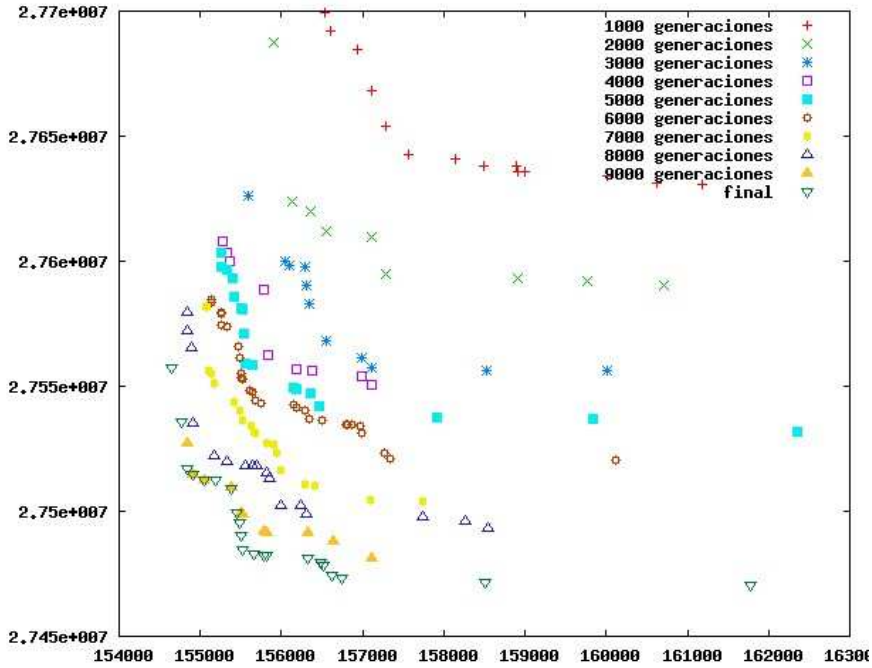

Figura A - 2 : Evolución del frente de Pareto en el NSGA-II en la instancia consistente con alta heterogeneidad de tareas y baja heterogeneidad de maquinas.

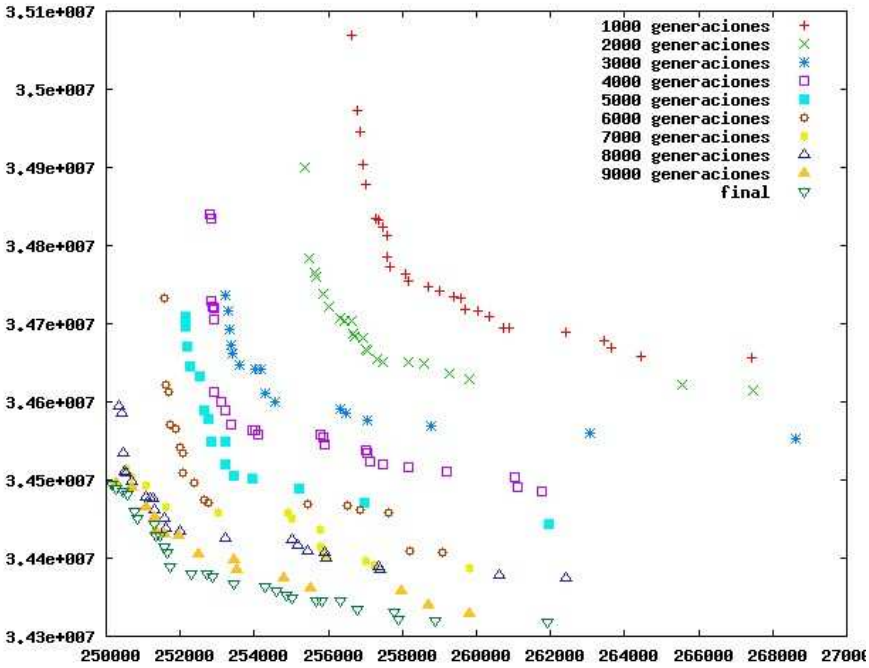

Figura A - 3: Evolución del frente de Pareto en el NSGA-II en la instancia consistente con baja heterogeneidad de tareas y alta heterogeneidad de maquinas.

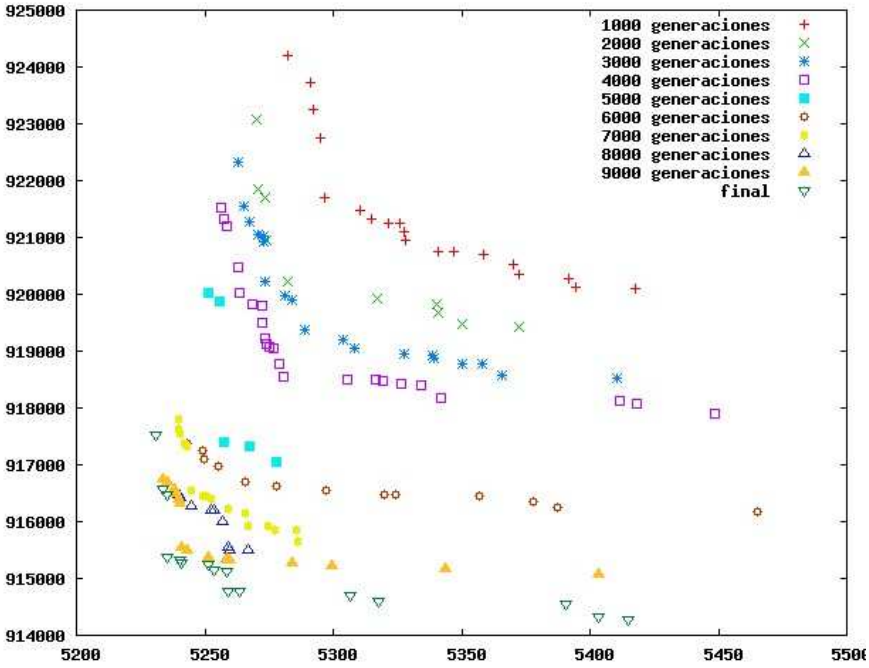

Figura A - 4: Evolución del frente de Pareto en el NSGA-II en la instancia consistente con baja heterogeneidad de tareas y baja heterogeneidad de maquinas.

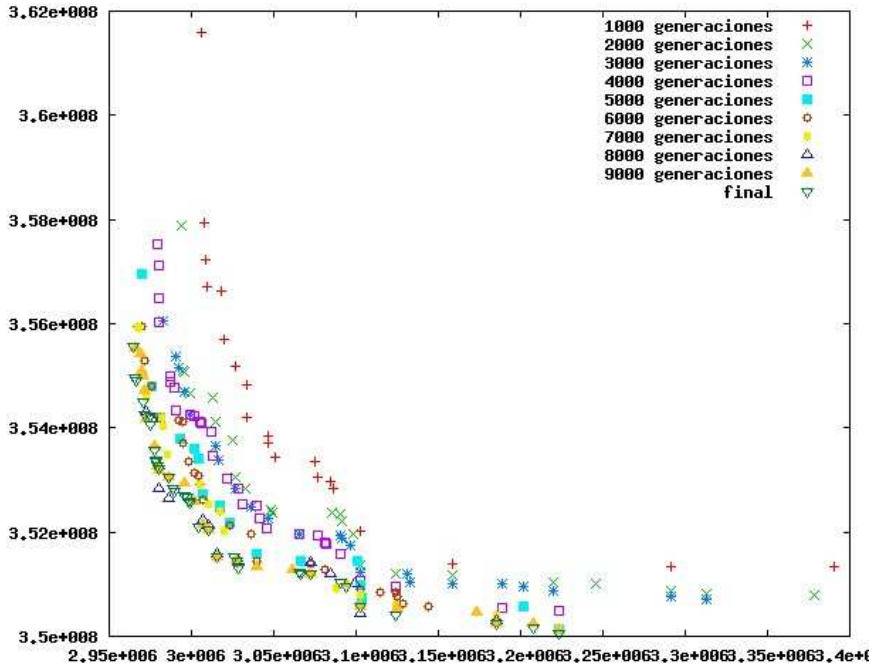

Figura A - 5: Evolución del frente de Pareto en el NSGA-II en la instancia inconsistente con alta heterogeneidad de tareas y alta heterogeneidad de maquinas.

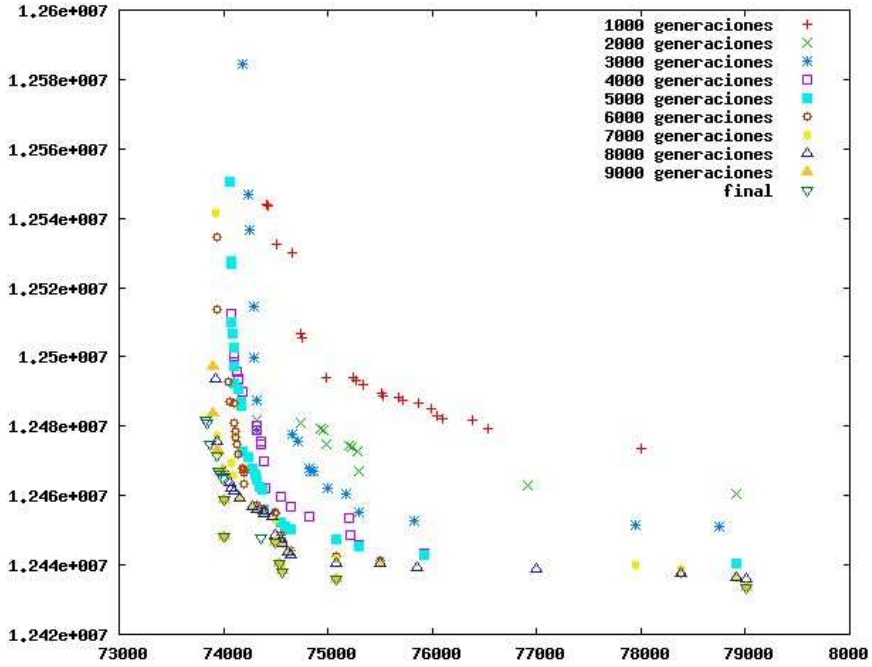

Figura A - 6: Evolución del frente de Pareto en el NSGA-II en la instancia inconsistente con alta heterogeneidad de tareas y baja heterogeneidad de maquinas.

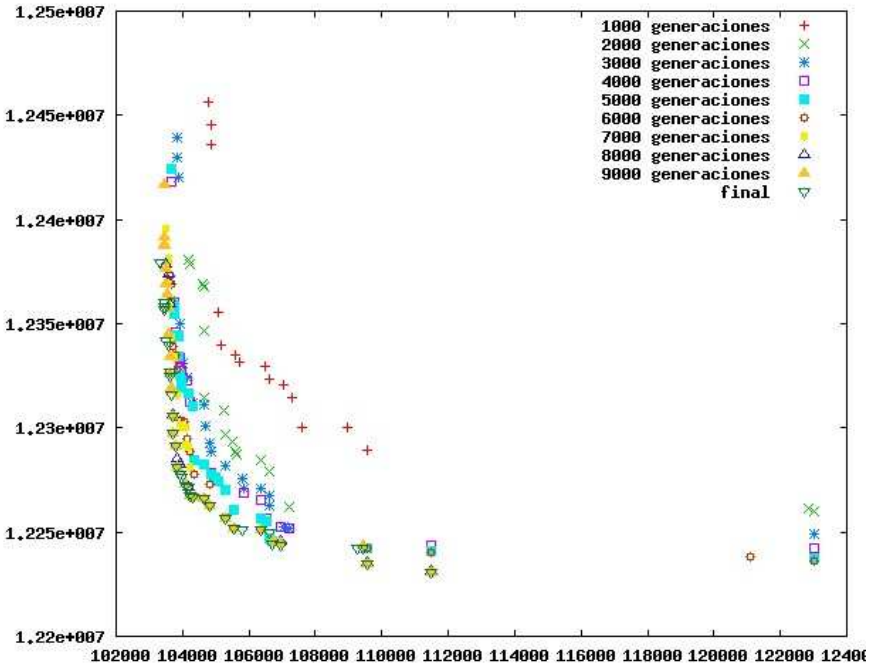

Figura A - 7: Evolución del frente de Pareto en el NSGA-II en la instancia inconsistente con baja heterogeneidad de tareas y alta heterogeneidad de maquinas.

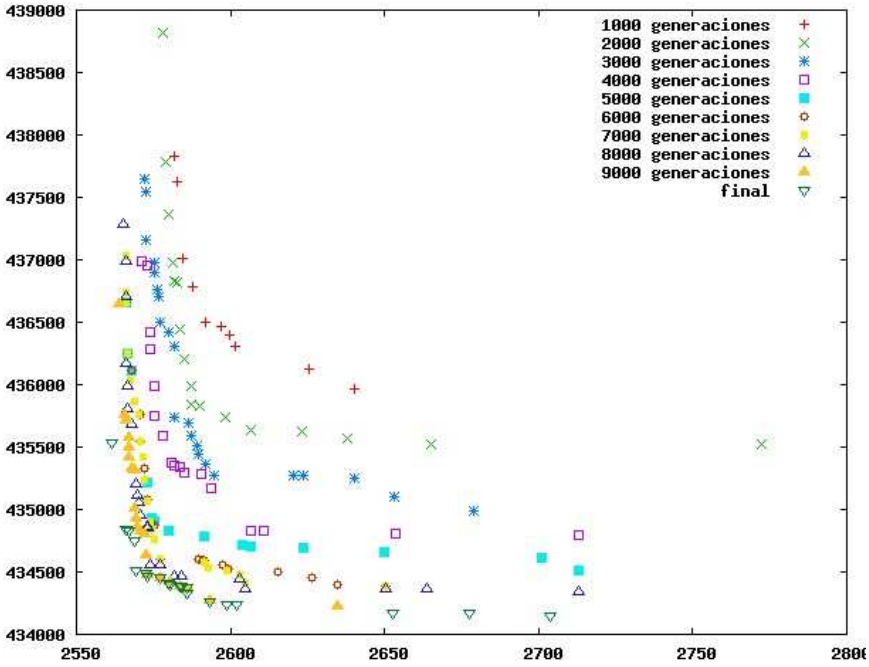

Figura A - 8: Evolución del frente de Pareto en el NSGA-II en la instancia inconsistente con baja heterogeneidad de tareas y baja heterogeneidad de maquinas.

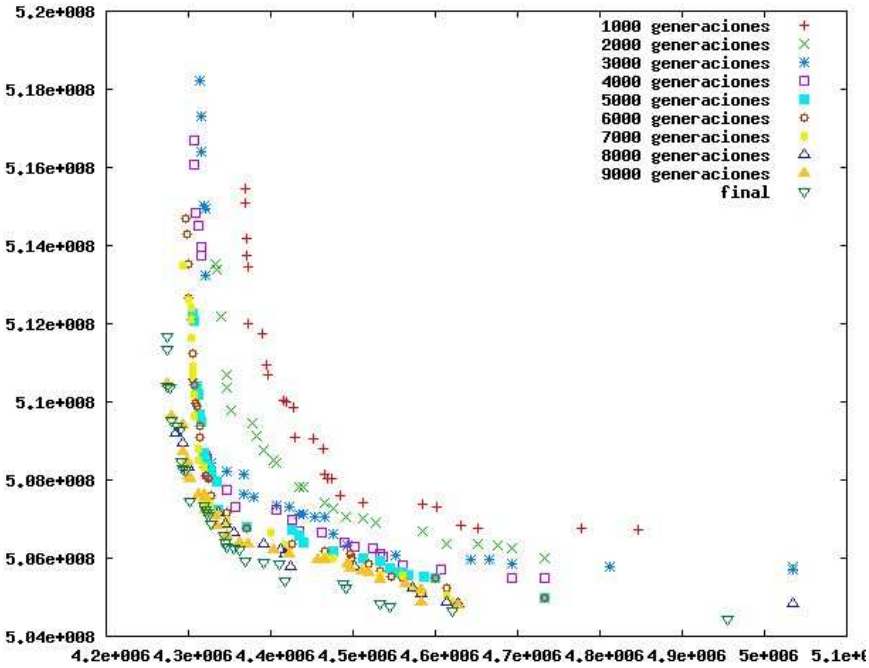

Figura A - 9: Evolución del frente de Pareto en el NSGA-II en la instancia parcialemente-consistente con alta heterogeneidad de tareas y alta heterogeneidad de maquinas.

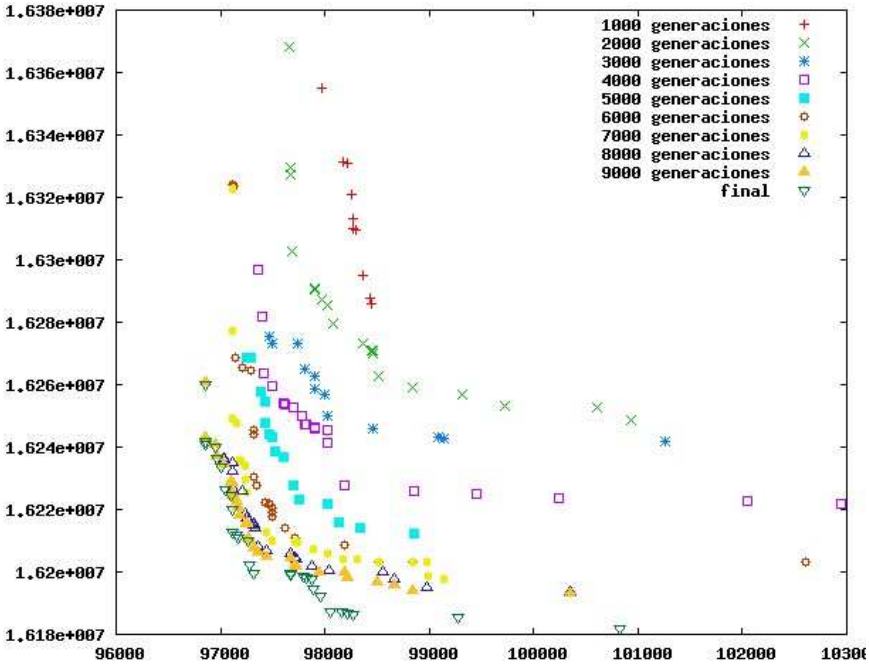

Figura A - 10: Evolución del frente de Pareto en el NSGA-II en la instancia parcialemente-consistente con alta heterogeneidad de tareas y baja heterogeneidad de maquinas.

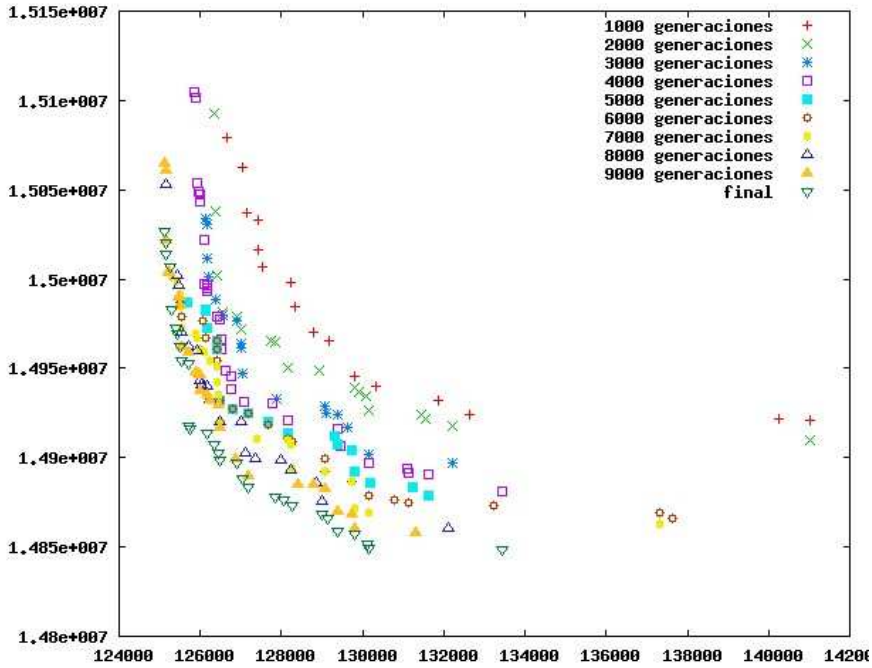

Figura A - 11: Evolución del frente de Pareto en el NSGA-II en la instancia parcialmente-consistente con baja heterogeneidad de tareas y alta heterogeneidad de maquinas.

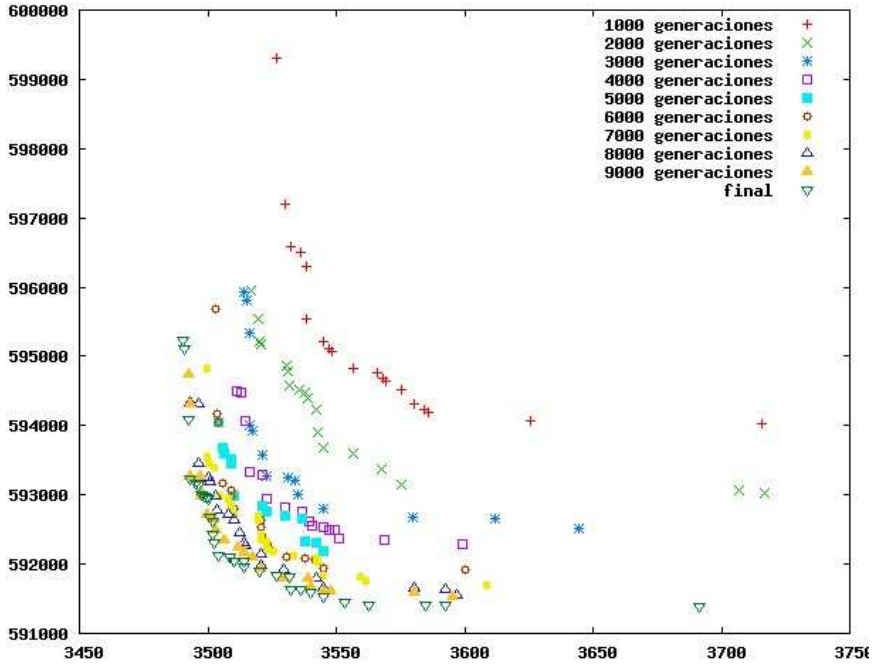

Figura A - 12: Evolución del frente de Pareto en el NSGA-II en la instancia parcialmente-consistente con baja heterogeneidad de tareas y baja heterogeneidad de maquinas.

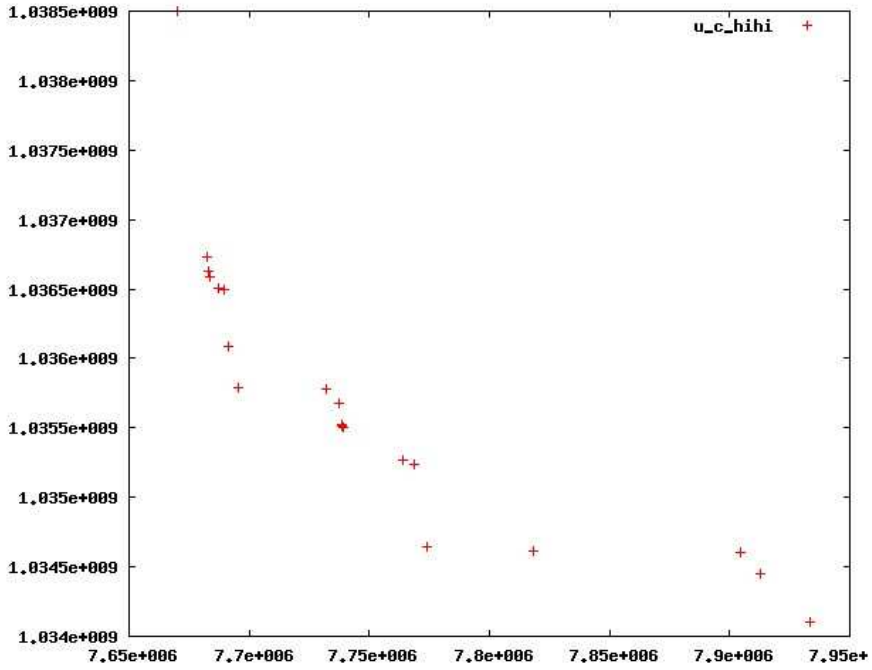

Figura A - 13: Frente de Pareto obtenido en el NSGA-II para la instancia consistente con alta heterogeneidad de tareas y alta heterogeneidad de maquinas.

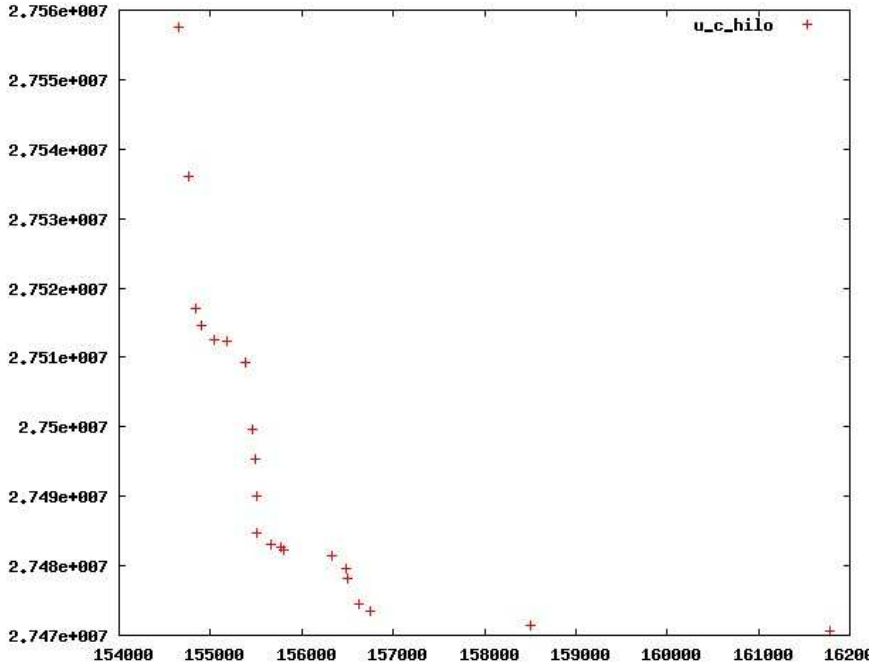

Figura A - 14: Frente de Pareto obtenido en el NSGA-II para la instancia consistente con alta heterogeneidad de tareas y baja heterogeneidad de maquinas.

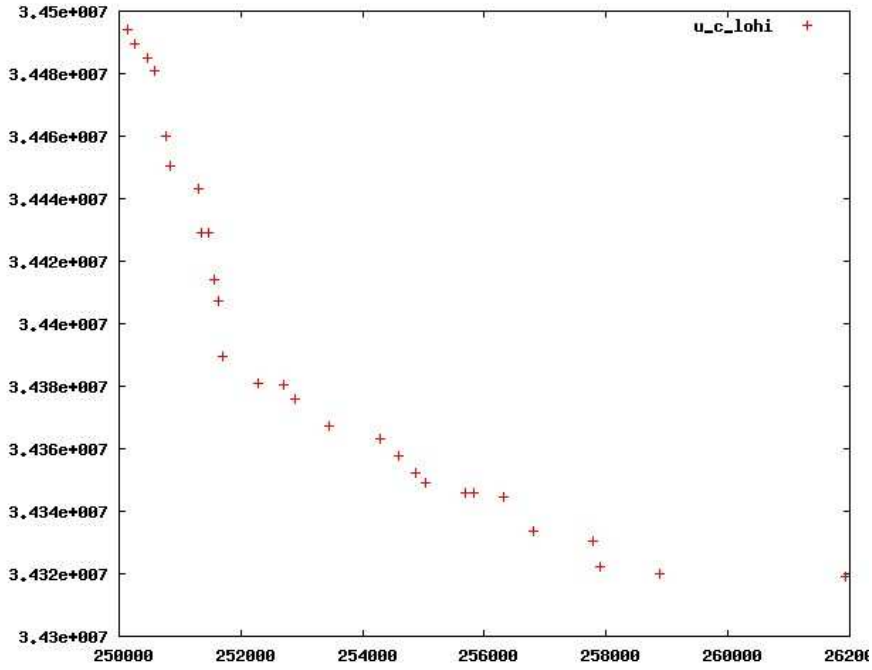

Figura A - 15: Frente de Pareto obtenido en el NSGA-II para la instancia consistente con baja heterogeneidad de tareas y alta heterogeneidad de maquinas.

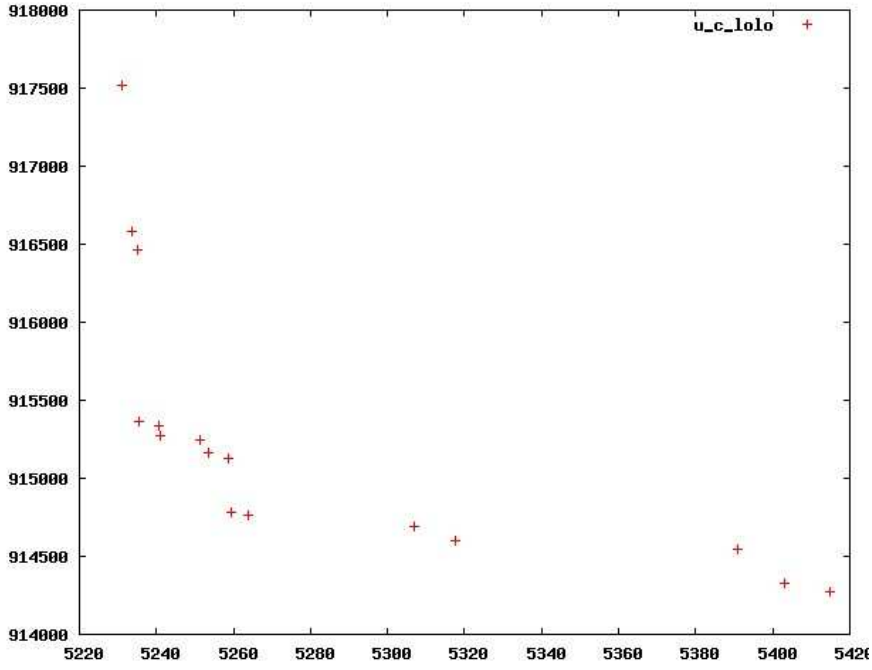

Figura A - 16: Frente de Pareto obtenido en el NSGA-II para la instancia consistente con baja heterogeneidad de tareas y baja heterogeneidad de maquinas.

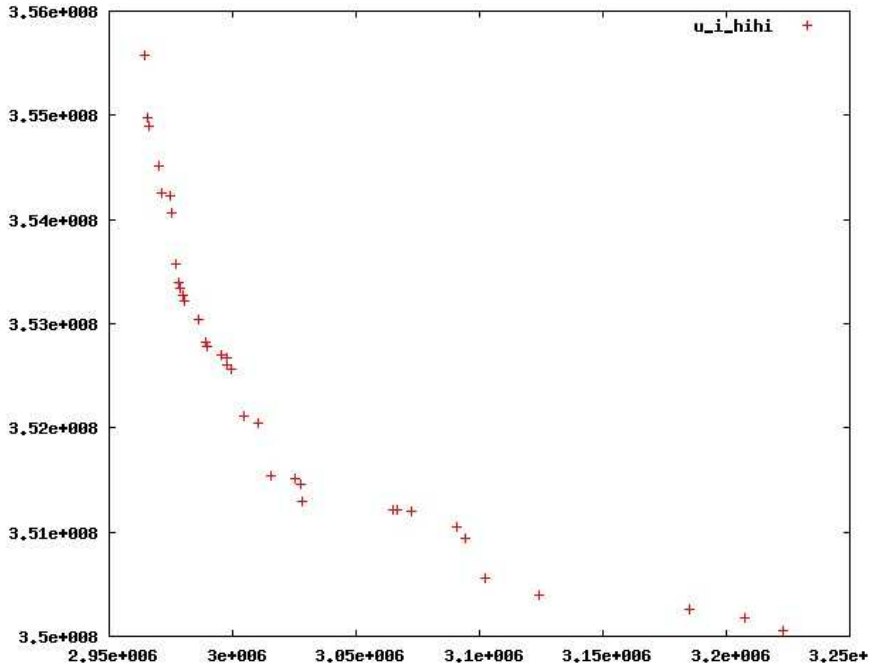

Figura A - 17: Frente de Pareto obtenido en el NSGA-II para la instancia inconsistente con alta heterogeneidad de tareas y alta heterogeneidad de maquinas.

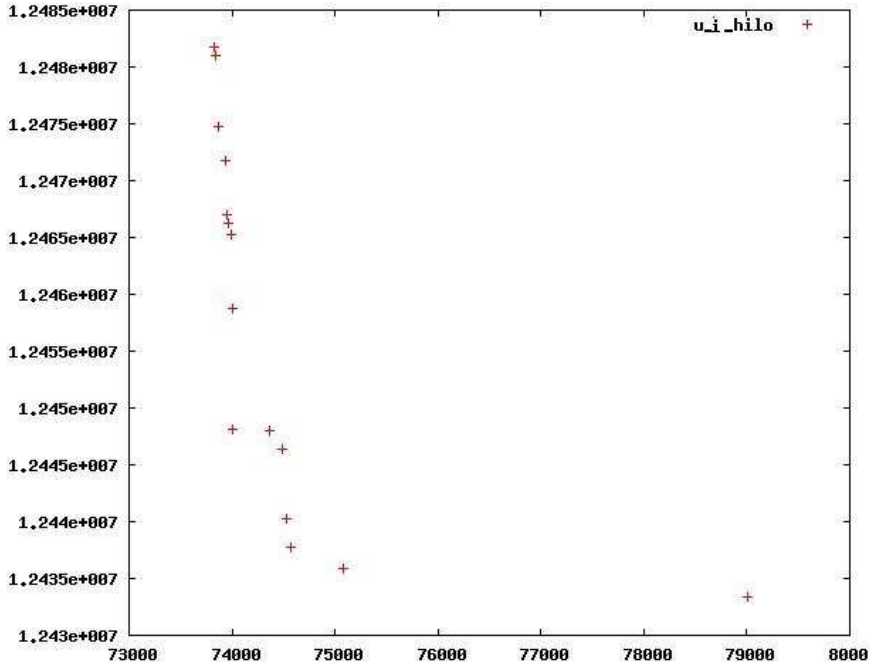

Figura A - 18: Frente de Pareto obtenido en el NSGA-II para la instancia inconsistente con alta heterogeneidad de tareas y baja heterogeneidad de maquinas.

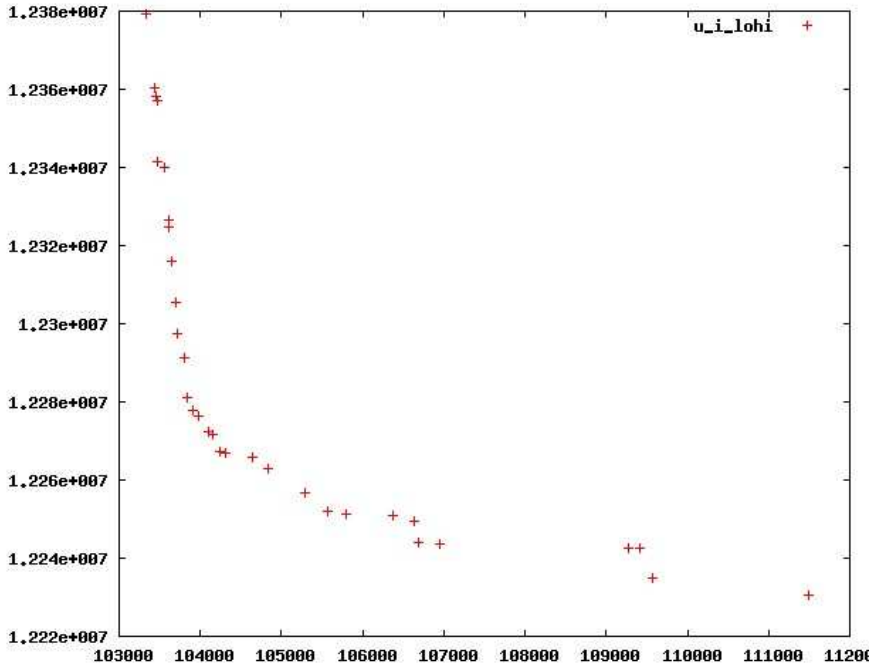

Figura A - 19: Frente de Pareto obtenido en el NSGA-II para la instancia inconsistente con baja heterogeneidad de tareas y alta heterogeneidad de maquinas.

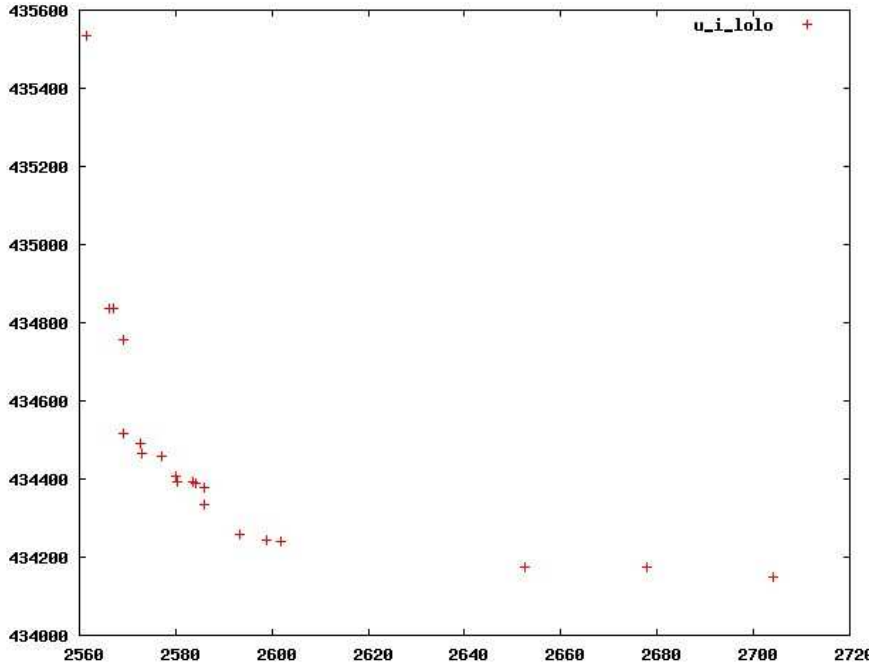

Figura A - 20: Frente de Pareto obtenido en el NSGA-II para la instancia consistente con baja heterogeneidad de tareas y baja heterogeneidad de maquinas.

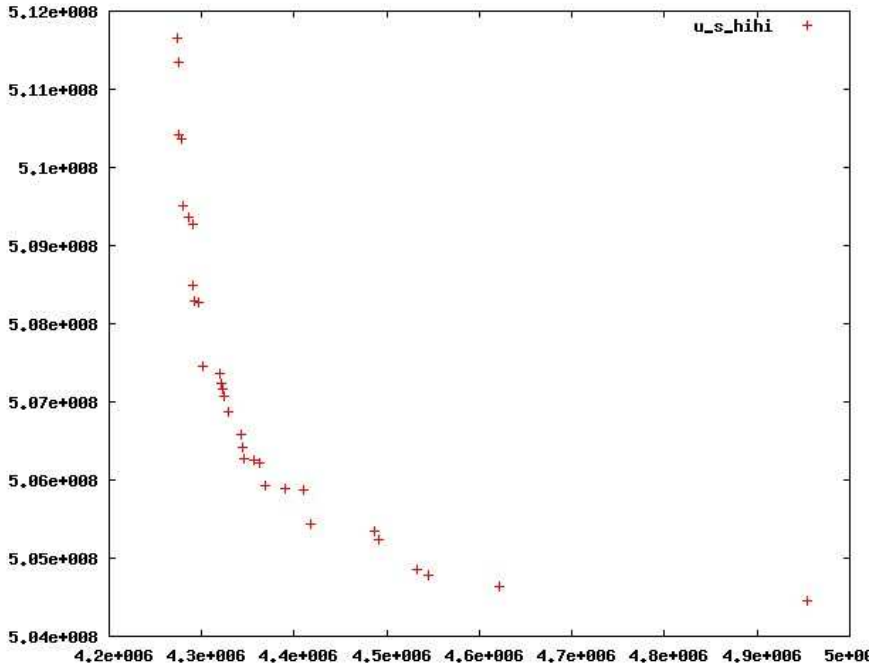

Figura A - 21 : Frente de Pareto obtenido en el NSGA-II para la instancia parcialmente-consistente con alta heterogeneidad de tareas y alta heterogeneidad de maquinas.

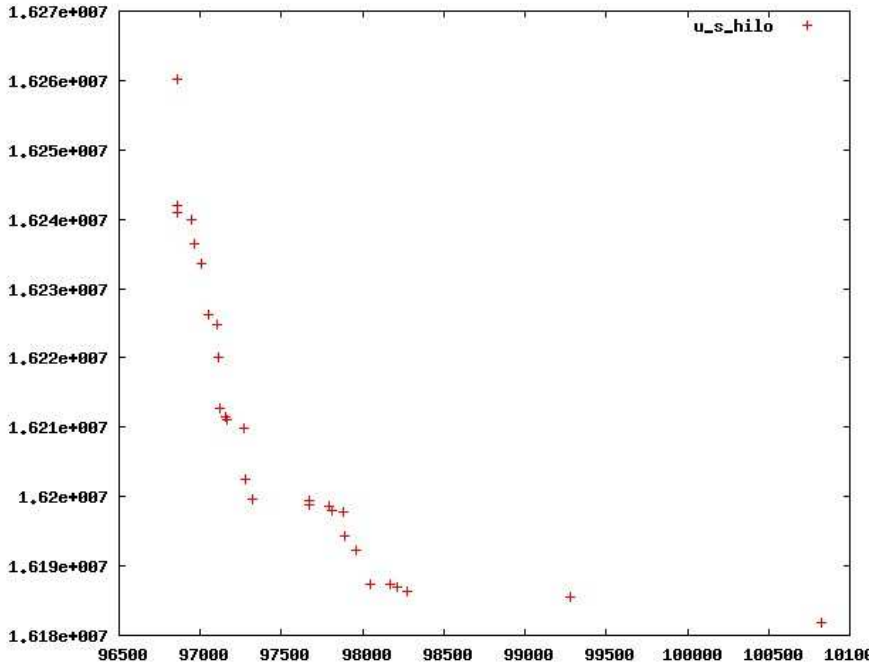

Figura A - 22: Frente de Pareto obtenido en el NSGA-II para la instancia parcialmente-consistente con alta heterogeneidad de tareas y baja heterogeneidad de maquinas

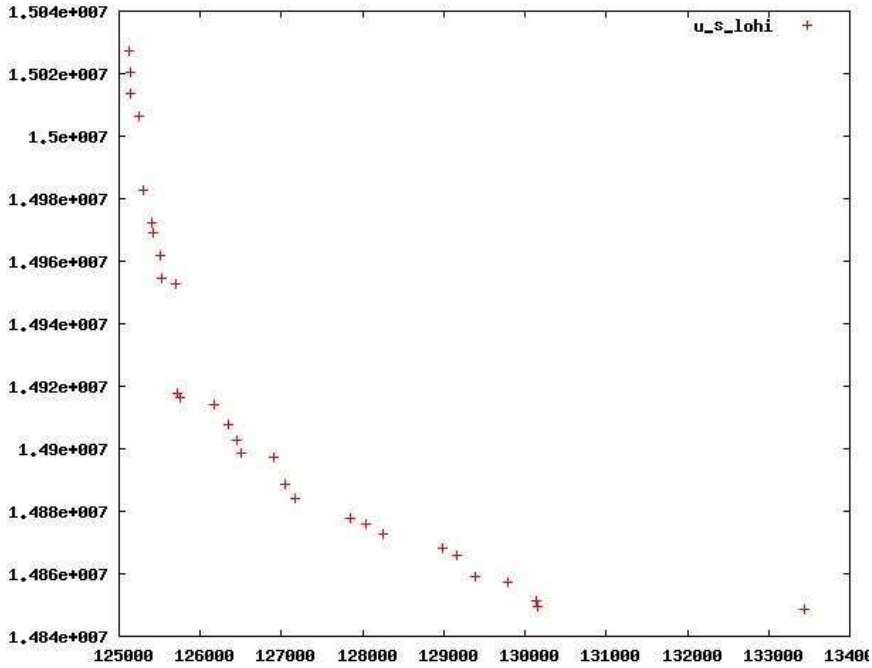

Figura A - 23: Frente de Pareto obtenido en el NSGA-II para la instancia parcialmente-consistente con baja heterogeneidad de tareas y alta heterogeneidad de maquinas.

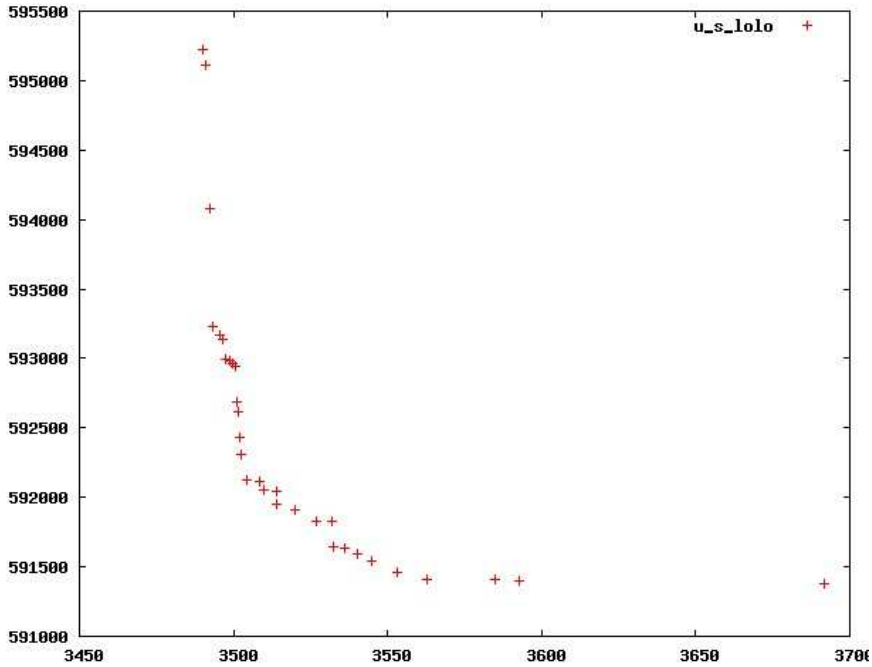

Figura A - 24: Frente de Pareto obtenido en el NSGA-II para la instancia parcialmente-consistente con baja heterogeneidad de tareas y baja heterogeneidad de maquinas.

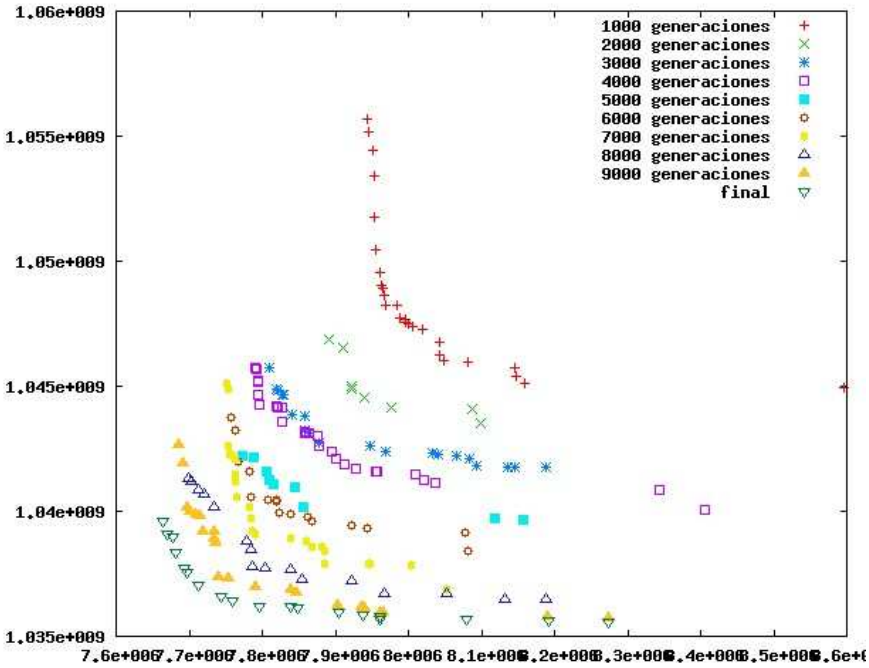

Figura A - 25: Evolución del frente de Pareto en el SPEA2 en la instancia consistente con alta heterogeneidad de tareas y alta heterogeneidad de maquinas.

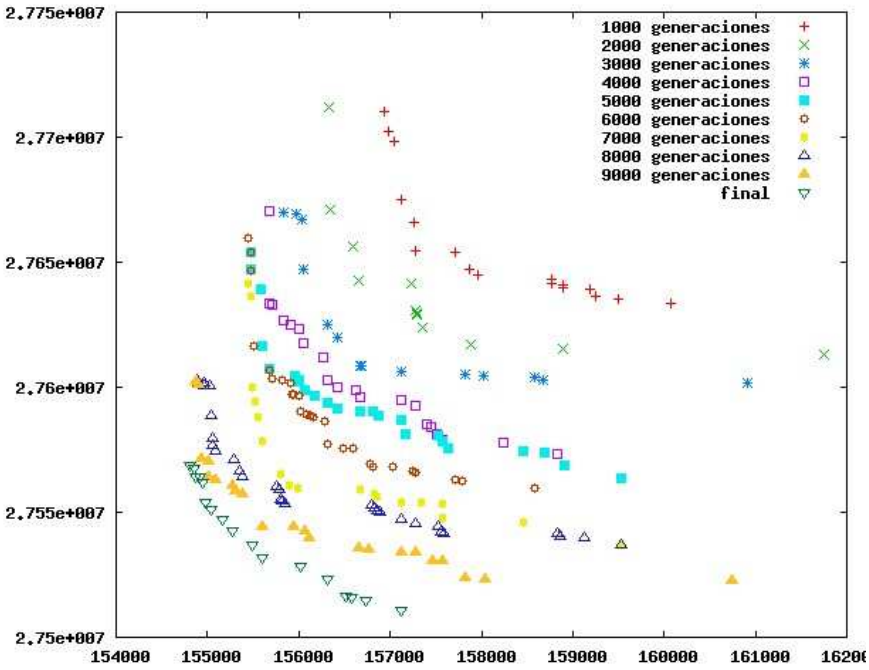

Figura A - 26: Evolución del frente de Pareto en el SPEA2 en la instancia consistente con alta heterogeneidad de tareas y baja heterogeneidad de maquinas

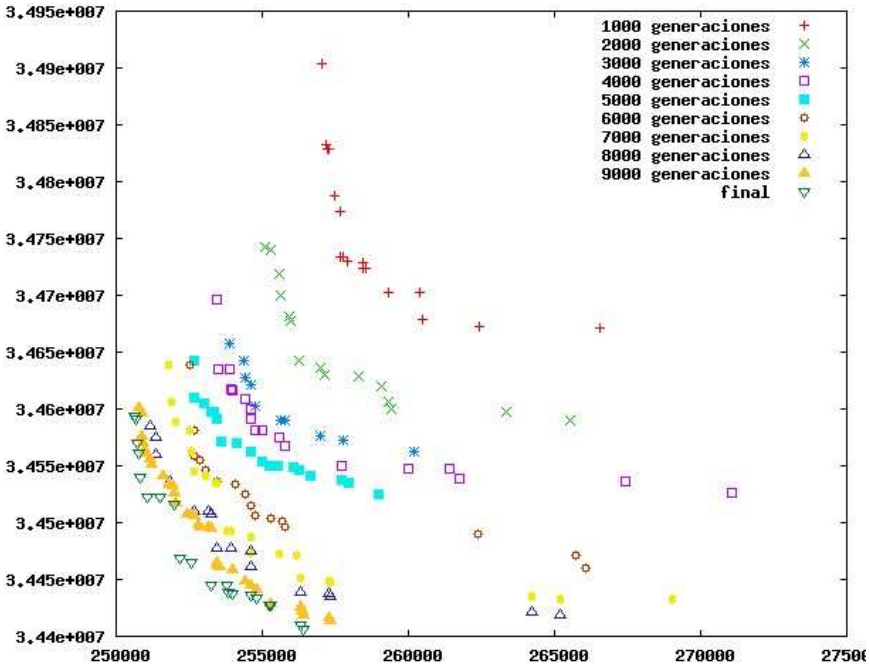

Figura A - 27: Evolución del frente de Pareto en el SPEA2 en la instancia consistente con baja heterogeneidad de tareas y alta heterogeneidad de maquinas

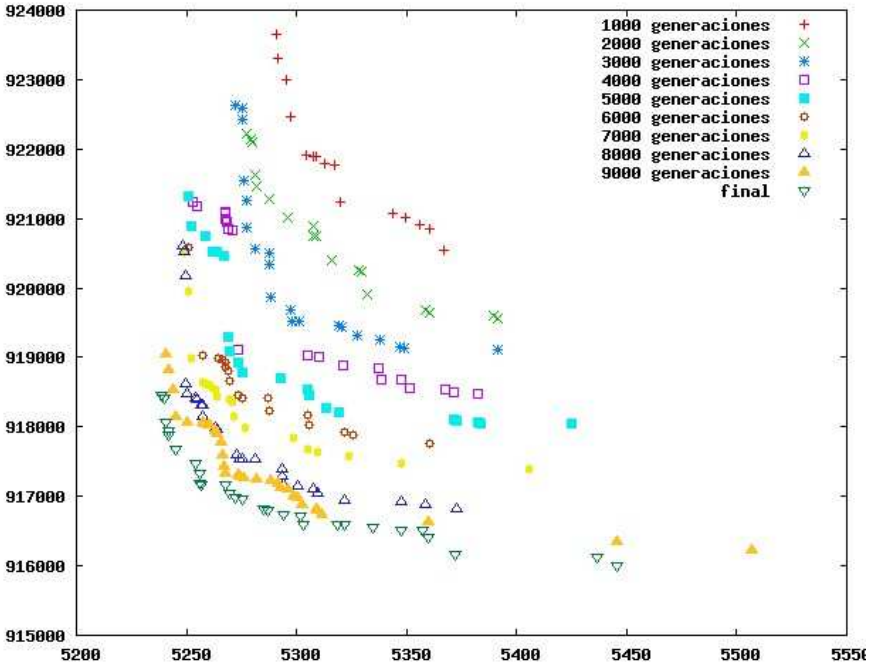

Figura A - 28: Evolución del frente de Pareto en el SPEA2 en la instancia consistente con baja heterogeneidad de tareas y baja heterogeneidad de maquinas

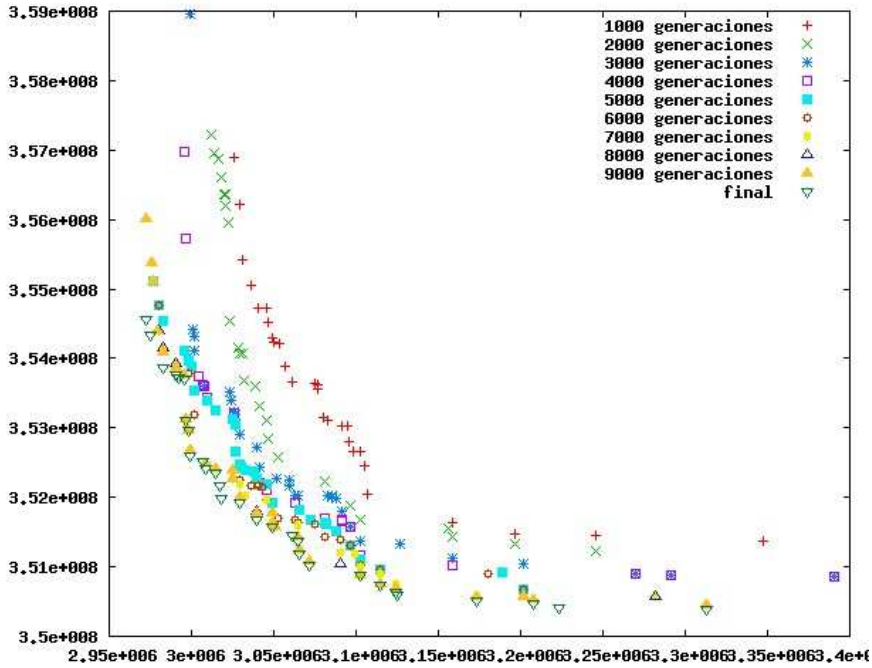

Figura A - 29: Evolución del frente de Pareto en el SPEA2 en la instancia inconsistente con alta heterogeneidad de tareas y alta heterogeneidad de maquinas

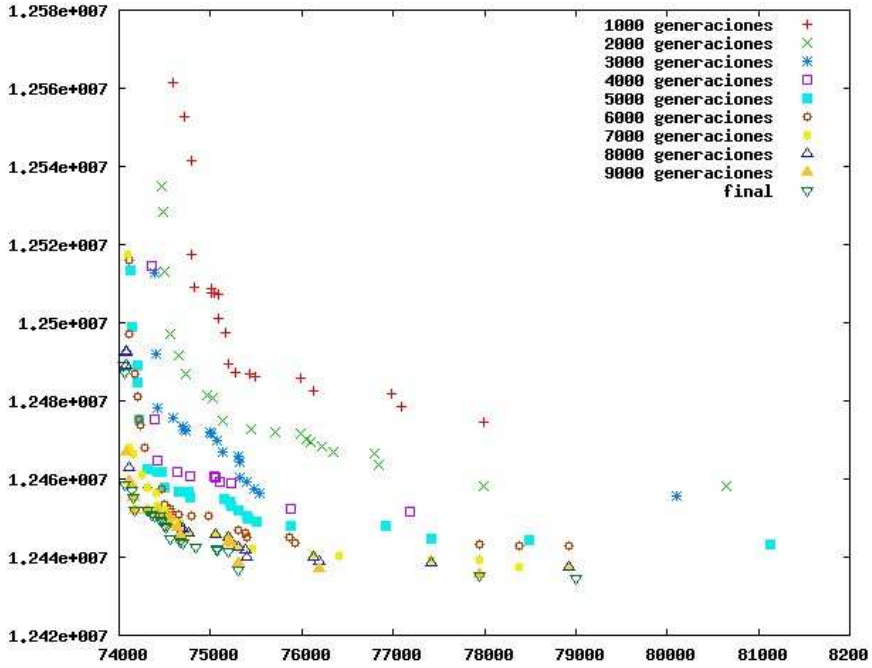

Figura A - 30: Evolución del frente de Pareto en el SPEA2 en la instancia inconsistente con alta heterogeneidad de tareas y baja heterogeneidad de maquinas

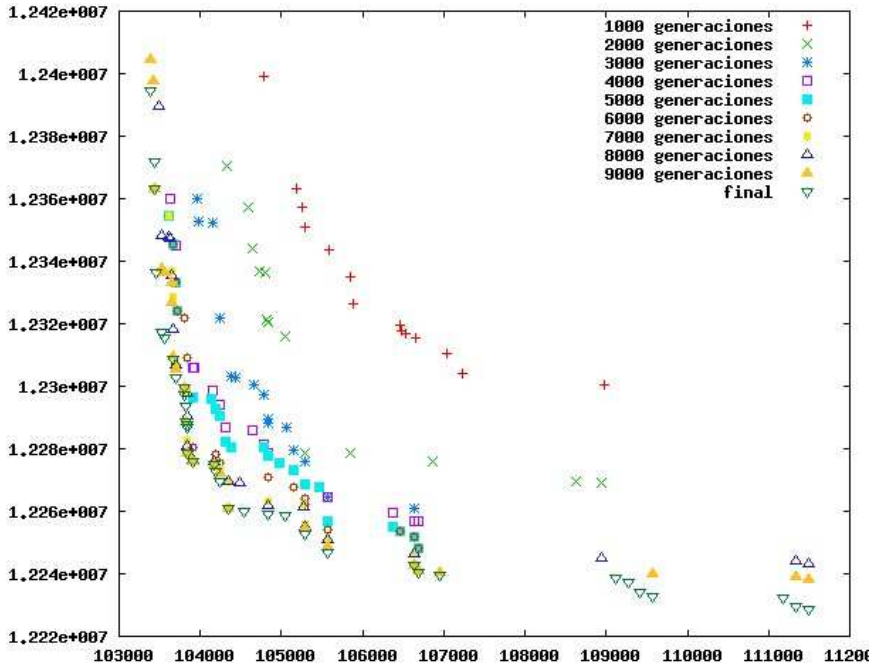

Figura A - 31: Evolución del frente de Pareto en el SPEA2 en la instancia inconsistente con baja heterogeneidad de tareas y alta heterogeneidad de maquinas

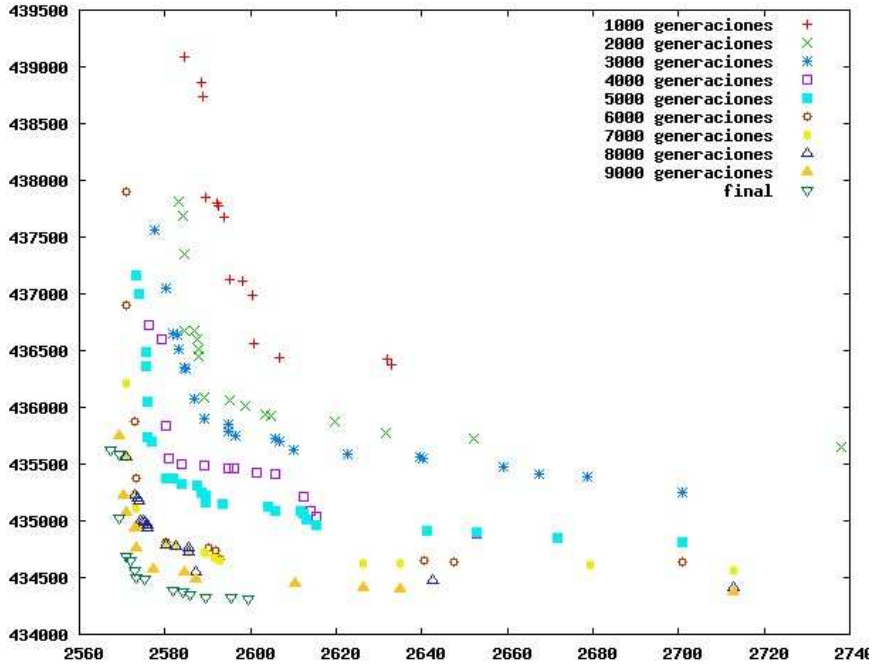

Figura A - 32: Evolución del frente de Pareto en el SPEA2 en la instancia inconsistente con baja heterogeneidad de tareas y baja heterogeneidad de maquinas

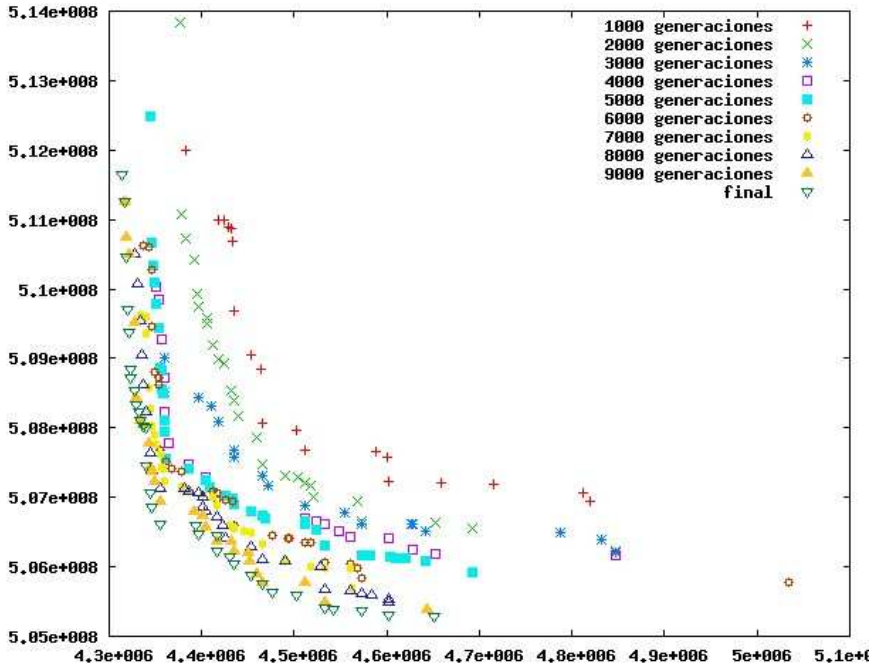

Figura A - 33: Evolución del frente de Pareto en el SPEA2 en la instancia parcialmente-consistente con alta heterogeneidad de tareas y alta heterogeneidad de maquinas

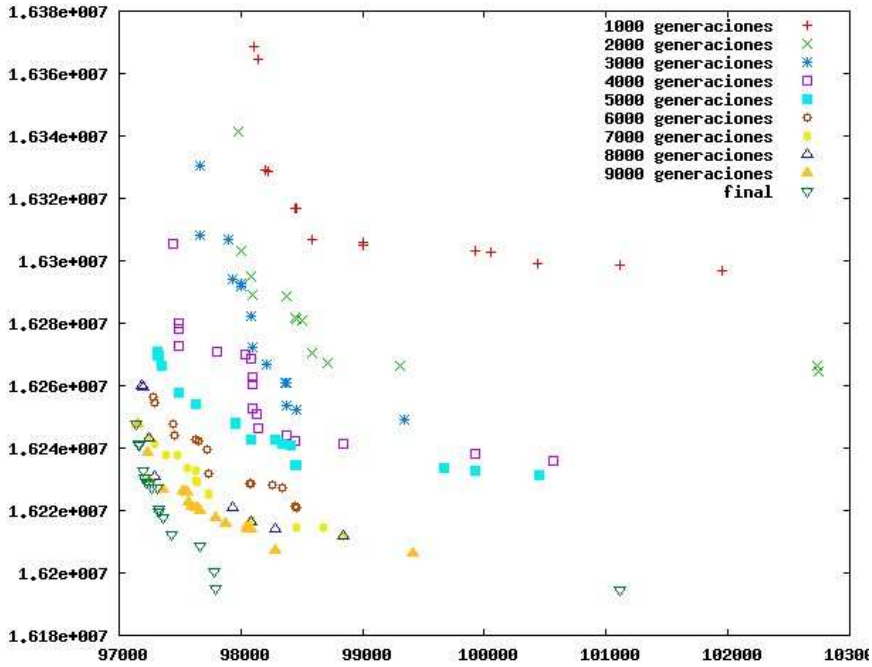

Figura A - 34: Evolución del frente de Pareto en el SPEA2 en la instancia parcialmente-consistente con alta heterogeneidad de tareas y baja heterogeneidad de maquinas

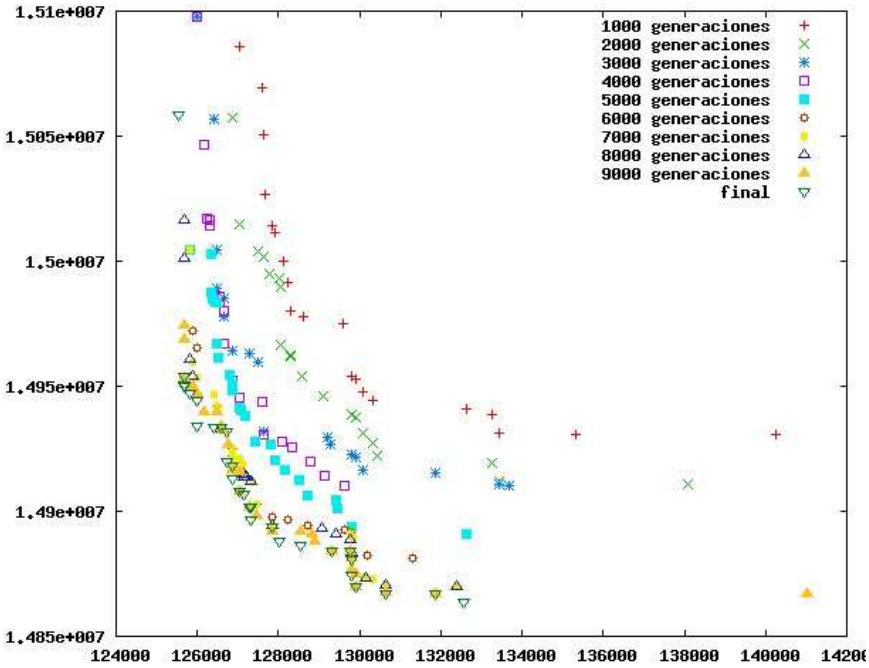

Figura A - 35: Evolución del frente de Pareto en el SPEA2 en la instancia consistente con baja heterogeneidad de tareas y alta heterogeneidad de maquinas

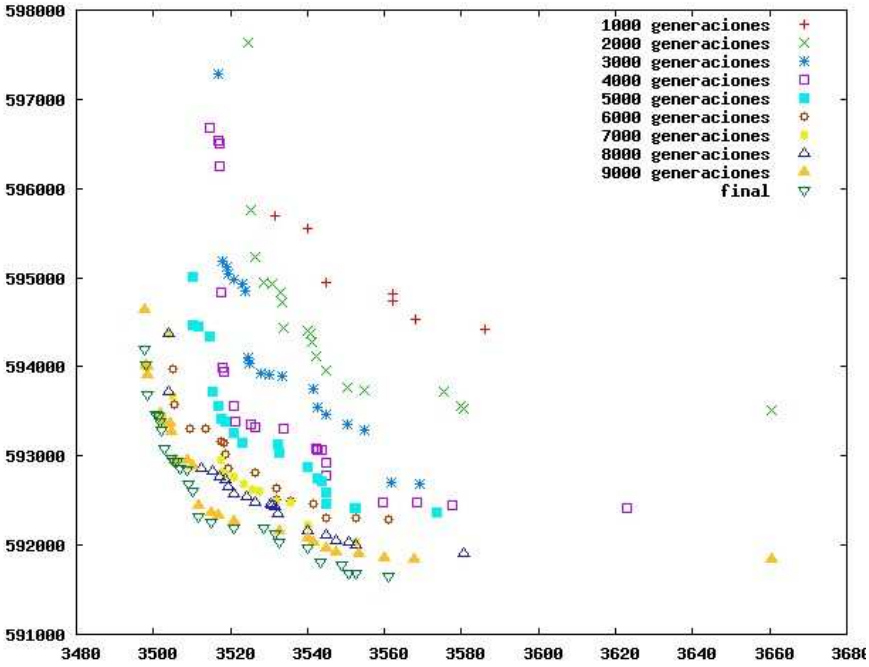

Figura A - 36: Evolución del frente de Pareto en el SPEA2 en la instancia parcialmente-consistente con baja heterogeneidad de tareas y baja heterogeneidad de maquinas

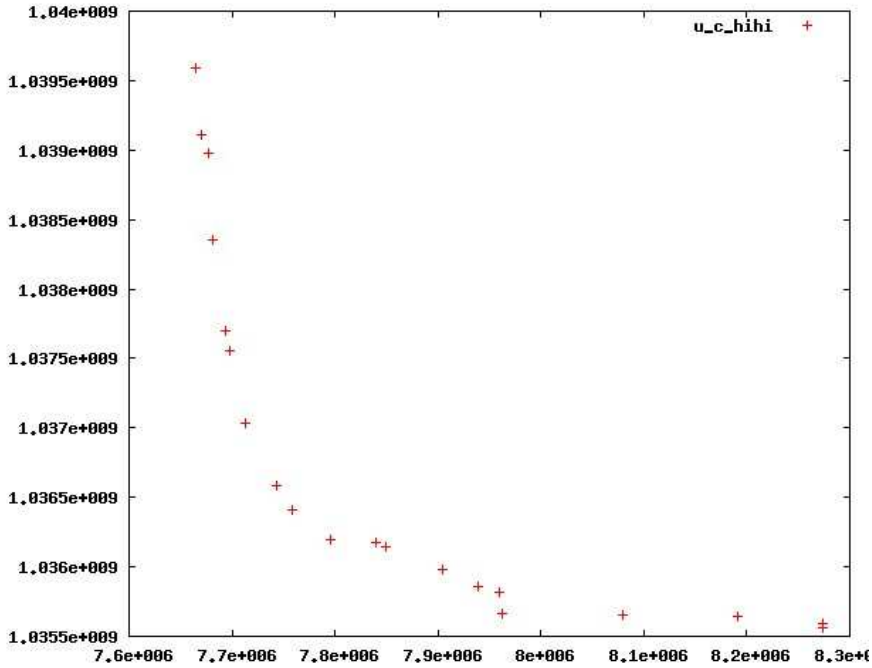

Figura A - 37: Frente de Pareto obtenido en el SPEA2 para la instancia consistente con alta heterogeneidad de tareas y alta heterogeneidad de maquinas.

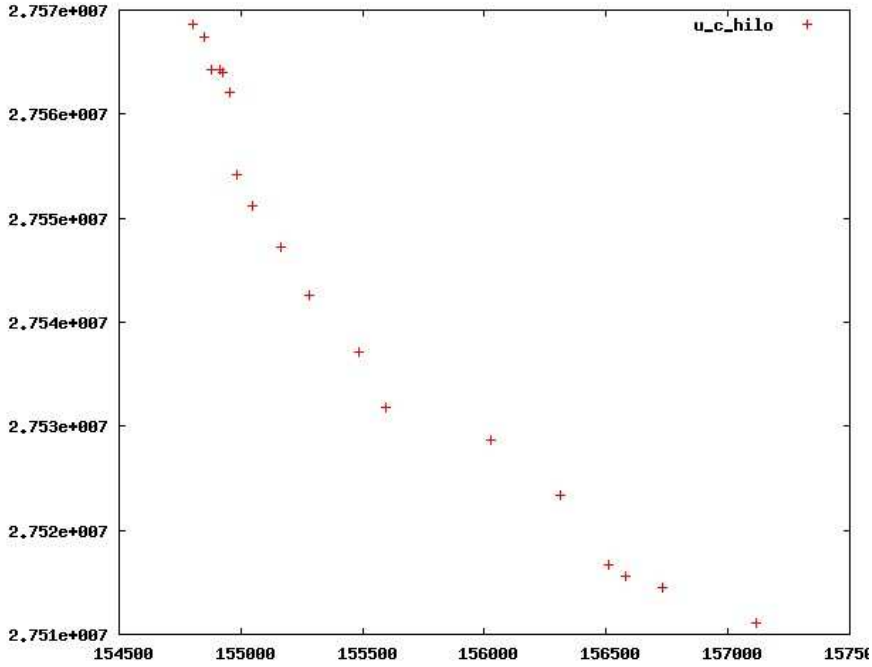

Figura A - 38: Frente de Pareto obtenido en el SPEA2 para la instancia consistente con alta heterogeneidad de tareas y baja heterogeneidad de maquinas.

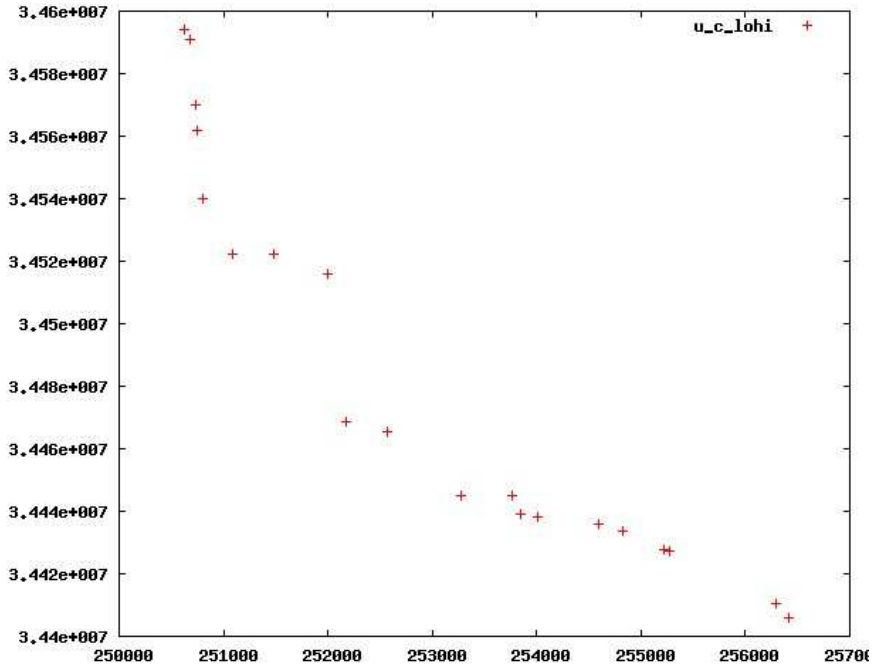

Figura A - 39: Frente de Pareto obtenido en el SPEA2 para la instancia consistente con baja heterogeneidad de tareas y alta heterogeneidad de maquinas.

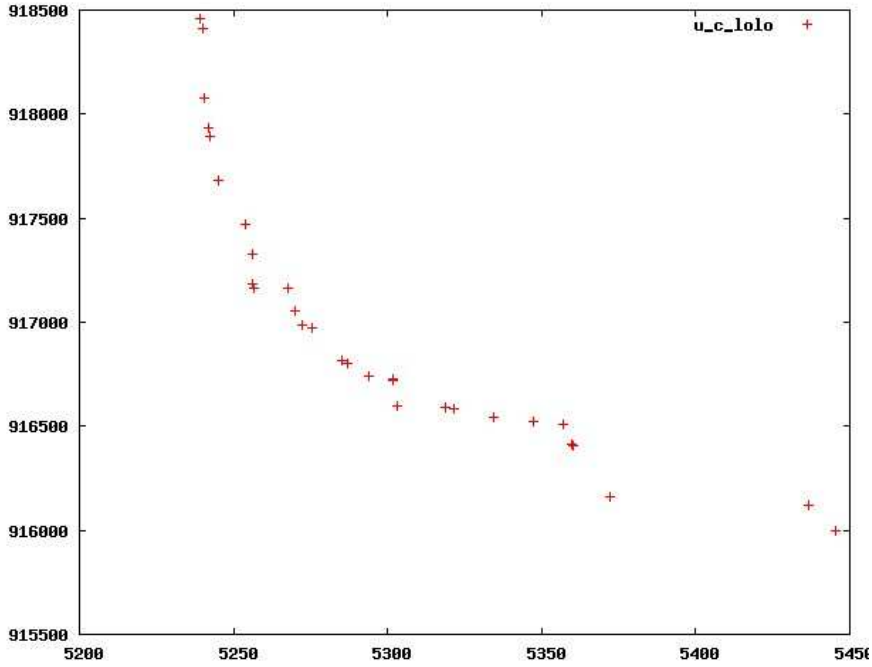

Figura A - 40: Frente de Pareto obtenido en el SPEA2 para la instancia consistente con baja heterogeneidad de tareas y baja heterogeneidad de maquinas.

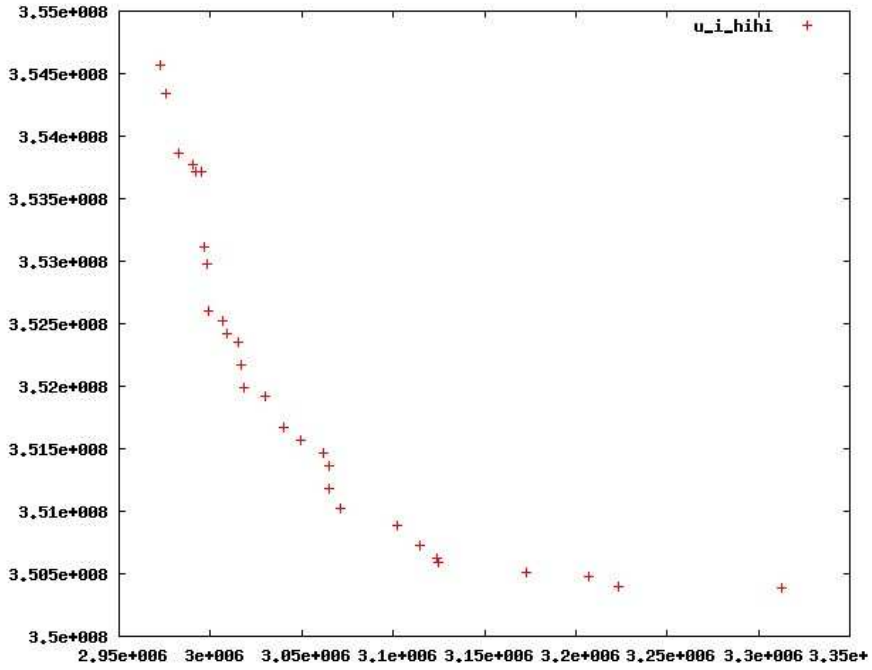

Figura A - 41: Frente de Pareto obtenido en el SPEA2 para la instancia inconsistente con alta heterogeneidad de tareas y alta heterogeneidad de maquinas.

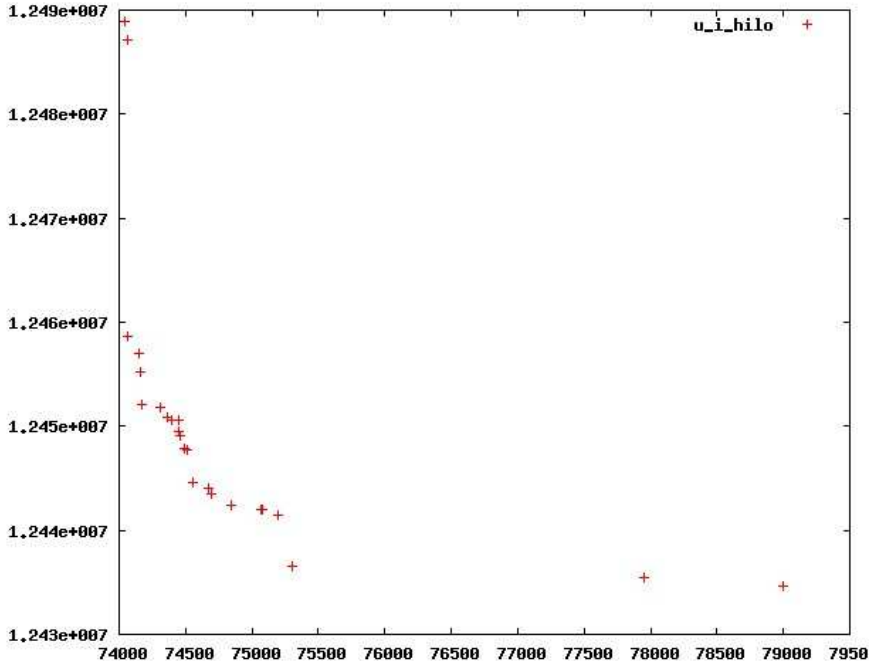

Figura A - 42: Frente de Pareto obtenido en el SPEA2 para la instancia inconsistente con alta heterogeneidad de tareas y baja heterogeneidad de maquinas.

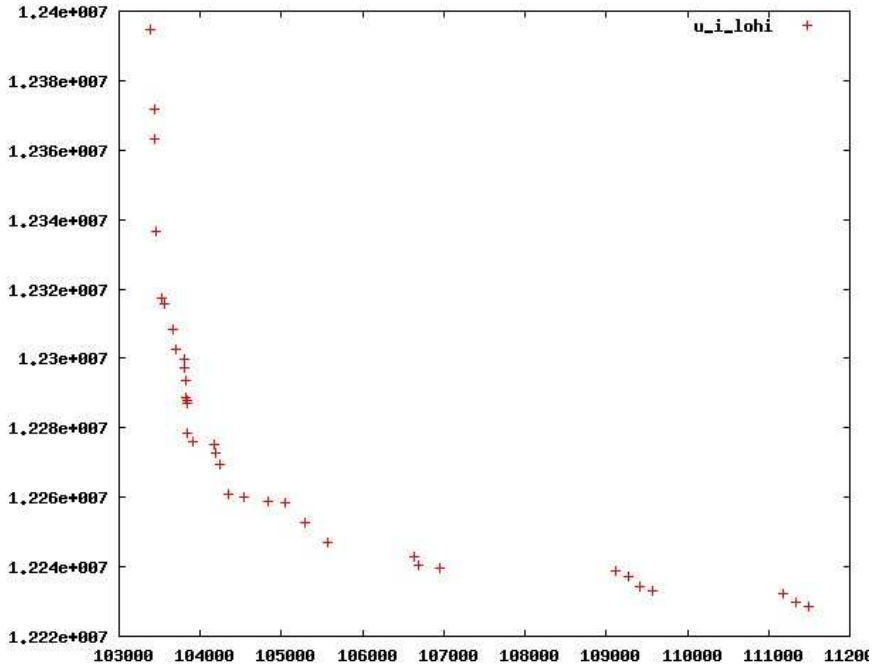

Figura A - 43: Frente de Pareto obtenido en el SPEA2 para la instancia inconsistente con baja heterogeneidad de tareas y alta heterogeneidad de maquinas.

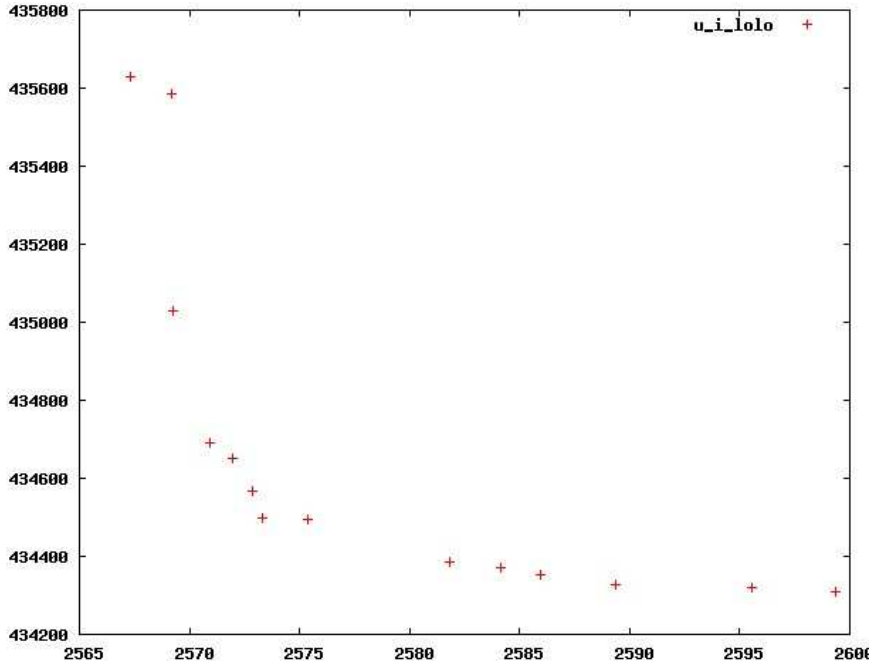

Figura A - 44: Frente de Pareto obtenido en el SPEA2 para la instancia inconsistente con baja heterogeneidad de tareas y baja heterogeneidad de maquinas.

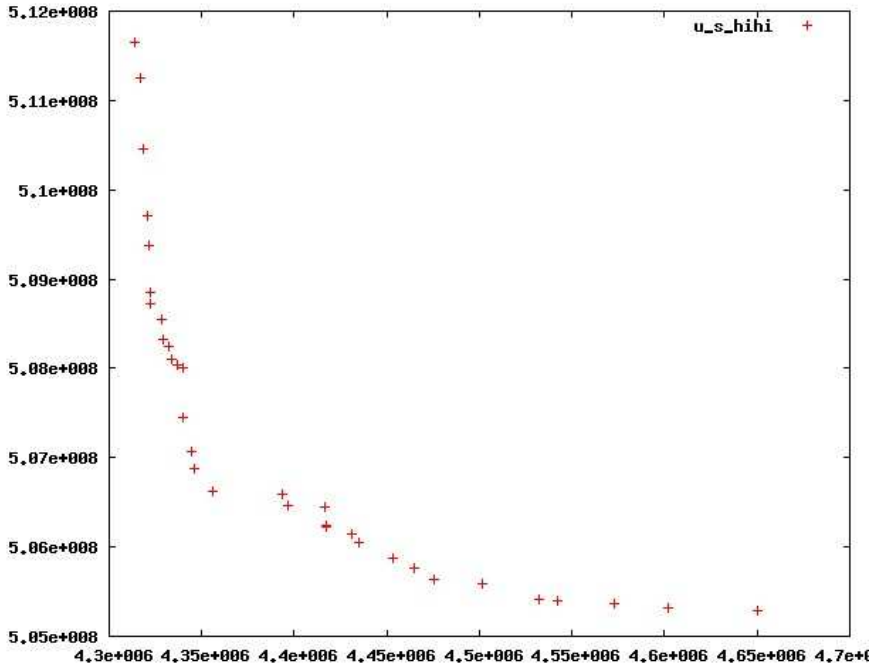

Figura A - 45: Frente de Pareto obtenido en el SPEA2 para la instancia parcialmente-consistente con alta heterogeneidad de tareas y alta heterogeneidad de maquinas.

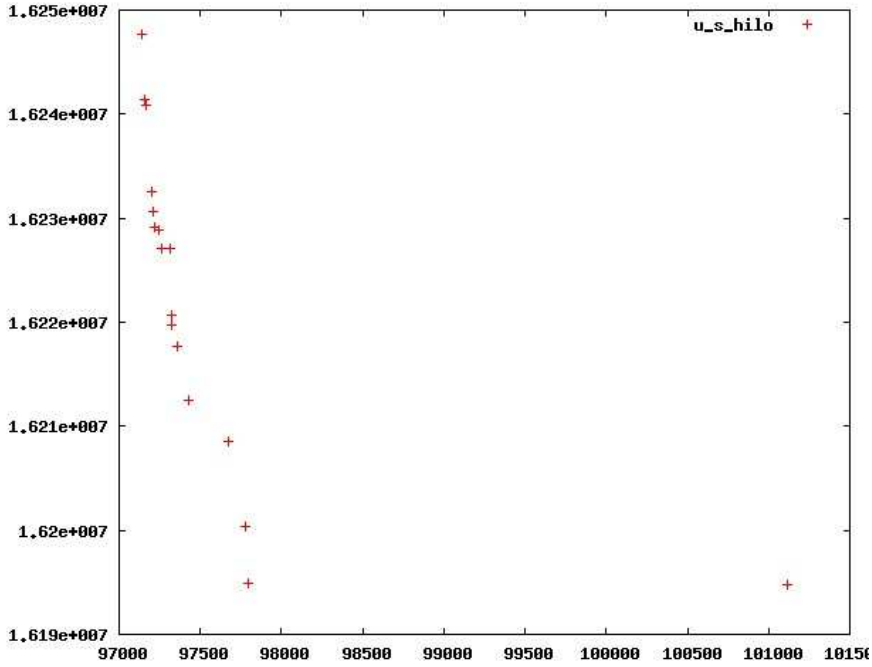

Figura A - 46: Frente de Pareto obtenido en el SPEA2 para la instancia parcialmente-consistente con alta heterogeneidad de tareas y baja heterogeneidad de maquinas.

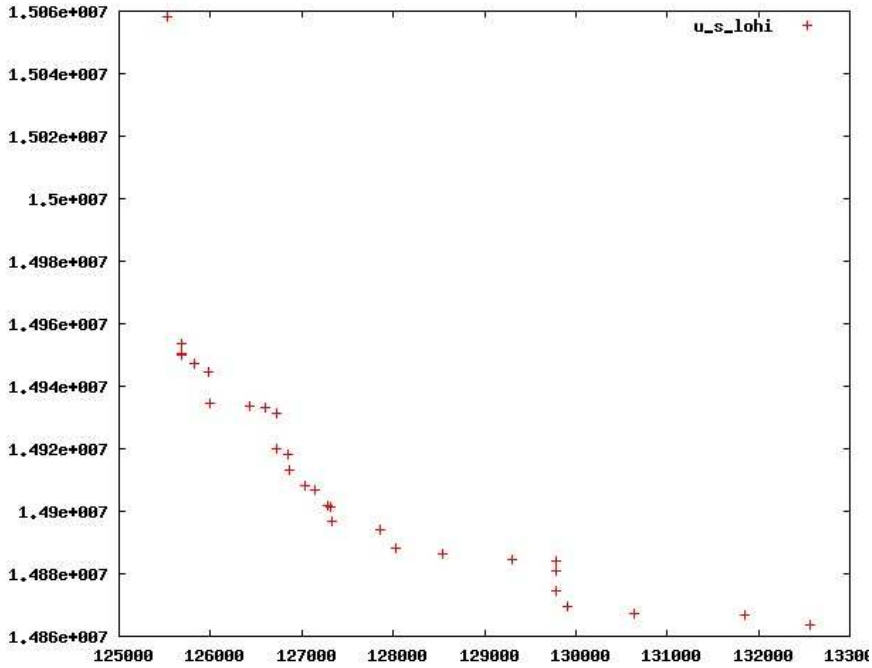

Figura A - 47: Frente de Pareto obtenido en el SPEA2 para la instancia parcialmente-consistente con baja heterogeneidad de tareas y alta heterogeneidad de maquinas.

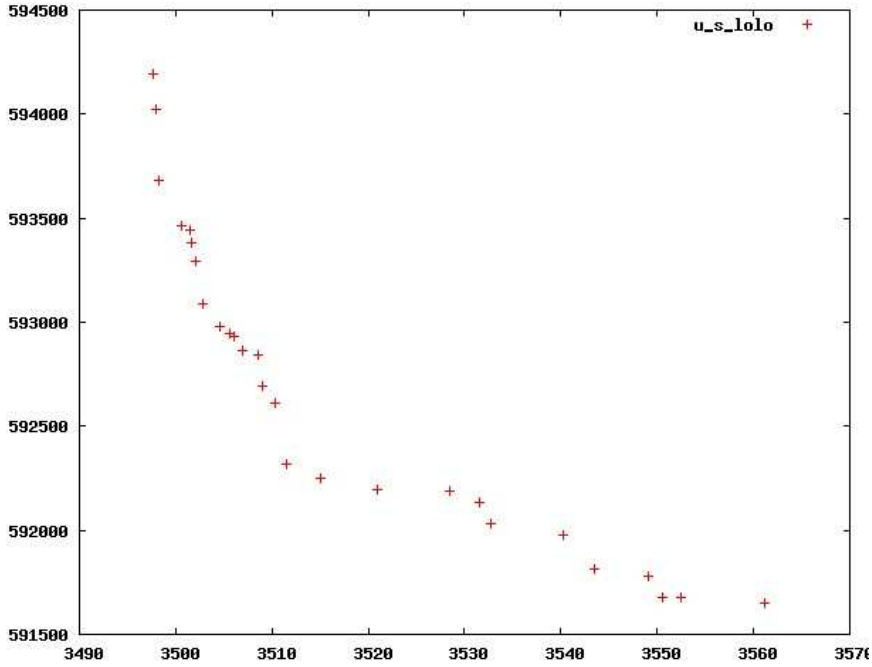

Figura A - 48: Frente de Pareto obtenido en el SPEA2 para la instancia parcialmente-consistente con baja heterogeneidad de tareas y baja heterogeneidad de maquinas.