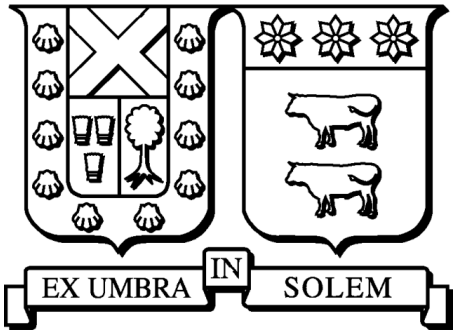
2017

UNA METAHEURISTICA PARA LA RESOLUCION DEL MACHINE REASSIGNMENT PROBLEM

CANALES ROJAS, DARÍO ANDRÉS

Universidad Técnica Federico Santa María
Departamento de Informática
Valparaíso - Chile



Una Metaheurística para la resolución del Machine Reassignment Problem

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA INGENIERÍA INFORMÁTICA

Darío Andrés Canales Rojas
dario.canales@alumnos.usm.cl
Profesor Guía: Dra. María Cristina Riff Rojas
Profesor Correferente: Dr. Carlos Castro
Profesor Correferente Externo: Dr. Bertrand Neveu

Octubre 2017

Agradecimientos

Agradezco a mis padres por apoyarme siempre en mis nuevos emprendimientos, también por enseñarme lo genial de aprender y entender cómo funciona el mundo, y en especial por motivarme a ser perseverante.

Agradezco a la Javi por emprender conmigo este viaje del conocer y superarnos, redefiniendo nuevas formas de entendernos y entender la realidad, ¡sigamos cumpliendo muchas metas más!.

Agradezco a la profesora María Cristina Riff por su gran paciencia, por apoyarme en los momentos difíciles y por empujarme a perfeccionar y buscar la simpleza.

Agradezco a la Eli y al Nico, por aquellas innumerables veces en que hablamos de nuevas ideas, códigos y los resultados. También por su preocupación y dedicación, que son un referente para mí, y sin duda alguna por la buena onda que irradian.

Sin todos ustedes no hubiese podido dar el extra, lo que se proyecta hacia el universo...

Resumen

El Problema de Reasignación de Máquinas (MRP) fue propuesto en el marco de la competencia ROADEF/EURO en conjunto con Google en el año 2012, y está definido por un conjunto de máquinas y procesos. Cada máquina está asociada con un conjunto de recursos, tales como CPU, RAM, Disco duro, y cada proceso tiene requerimientos de algunos de estos recursos. Inicialmente, cada proceso está asignado a una máquina específica, el objetivo del problema es reasignar los procesos, de tal forma, que se mejore su distribución y optimice el uso de las máquinas, los cuales están definidos por una función objetivo específica. Además, se debe cumplir con una serie de restricciones duras.

En este trabajo se describe el problema en detalle, y se propone un algoritmo para su resolución, correspondiente a un enfoque colaborativo basado en dos metaheurísticas simples y de fácil implementación: Hill Climbing y Simulated Annealing. Mediante experimentos se muestra que el enfoque propuesto permite obtener resultados competitivos en las instancias más complejas, superando incluso a los mejores algoritmos de la competencia.

Palabras Clave: *Problema de Reasignación de Máquinas, MRP, ROADEF, Hill Climbing, Simulated Annealing, Metaheurísticas de Búsqueda, Optimización Combinatoria.*

Abstract

The Machine Reassignment Problem (MRP) was proposed in the context of the ROADEF /EURO competition with Google in the year 2012, and it's defined by a set of machines and a set of processes. Each machine is associated with a set of resources, such as CPU, RAM, Hard disk, and each process has requirements of some of these resources. Initially, each process is assigned to one specific machine, the objective of the problem is to reassign the processes, in such a way, that improves distribution and optimize the use of the machines, which are defined by a specific objective function. In addition, a series of hard constraints have to be satisfied.

In this work a description of the problem is presented, along with an algorithm to solve it, consisting of a collaborative approach of two simple metaheuristics very easy to implement: Hill Climbing and Simulated Annealing. In computational experiments it is shown that with the proposed approach achieves competitive results on the more complex instances, surpassing even the best algorithms presented in the competence.

Keywords: *Machine Reassignment Problem, MRP, ROADEF, Scheduling, Hill Climbing, Simulated Annealing, Local Search Metaheuristics, Combinatorial Optimization.*

Índice general

1 Introducción

En los últimos años, el emergente desarrollo de las tecnologías que ofrecen servicios en la nube y almacenamiento de datos han generado un gran interés en la optimización de los recursos ocupados, lo cual afecta directamente la calidad del servicio y económicamente a las empresas. Para presentar una mayor calidad, un aspecto clave a tomar en cuenta en servicios como Google Apps, Facebook, Microsoft, es la maximización de los beneficios a la par de la minimización de los costos operacionales. Por otro lado, la creciente preocupación por disminuir los gases de efecto invernadero como forma de combatir el cambio climático que está sufriendo el planeta [26, 27], se presenta como una fuerte motivación para la disminución del consumo eléctrico, tomando en cuenta que sólo en Estados Unidos el 30 por ciento de los gases invernadero en 2014 fueron emitidos debido a la generación de electricidad [1].

Tomando en cuenta a un servicio computacional como un conjunto de procesos, hay varias causas por las cuales su desempeño se puede ver afectado. Por este motivo, se han usado varias estrategias para reducir el impacto ante eventuales caídas, entre las cuales se destaca el que las máquinas que ejecutan los procesos sean distintas, que estas a su vez se encuentren en distintas localizaciones, la no sobrecarga de las máquinas a las cuales están asignados y, en caso de existir relaciones de dependencia entre servicios, que los procesos de un servicio se encuentren asignados a máquinas donde están los procesos del servicio del que depende. Por otro lado, es preciso tomar en consideración que un uso sobrecargado de los recursos de una máquina puede incurrir en costos adicionales asociados al deterioro de ésta o un mayor consumo eléctrico.

Esta tesis se centra en la resolución del problema Machine Reassignment Problem (MRP), el cual fue propuesto en el marco de la competencia ROADEF/EURO en conjunto con Google en el año 2012. Se define por un conjunto de máquinas y procesos. Cada máquina está asociada con un conjunto de recursos, tales como CPU, RAM y Disco duro, y cada proceso tiene requerimientos de algunos de estos recursos. Inicialmente, cada proceso está asignado a una máquina específica, el objetivo del problema es reasignar los procesos, de tal forma que se mejore su distribución y optimice el uso de las máquinas, los cuales están definidos por una función objetivo específica. Además se debe cumplir con una serie de restricciones duras que hacen que este problema sea NP-Hard [28].

Este trabajo está organizado de la siguiente manera. En el Capítulo 2 se estudiará el problema en profundidad, indicando cada una de los componentes y restricciones que se

deben considerar. Posteriormente en el Capítulo 3 se revisarán algunas de las técnicas utilizadas para atacar el problema, tendencias actuales, dando paso a la presentación formal de algunos modelos matemáticos que sirven para abordar el problema. En el Capítulo 4 se propone un algoritmo para resolver el problema Machine Reassignment Problem (MRP), mediante una combinación de las metaheurísticas Hill Climbing y Simulated Annealing. En el capítulo 5 se presentan los resultados experimentales, y se compara con las mejores versiones de las técnicas propuestas en la competencia. Finalmente en el Capítulo 6 se presentan las conclusiones generales y trabajo futuro.

2 Definición del problema

El problema de reasignación de máquinas (MRP), consiste en la reasignación de un conjunto de procesos a un conjunto de máquinas que serán las encargadas de procesarlos. Se cuenta con una asignación inicial, en la que todos los procesos están distribuidos en diferentes máquinas. A partir de esta asignación inicial se desea encontrar modificaciones con el objetivo de mejorar su distribución. Esta mejora se puede medir en términos de una función de costo que exprese cuan buena es la distribución de procesos de una asignación. Cada proceso tiene una demanda de recursos, tales como CPU, RAM y disco duro, que son necesarios para satisfacer su ejecución; por otro lado, cada máquina tiene una cierta cantidad de estos recursos, que van quedando utilizados a medida que algunos procesos van siendo asignados a éstas. Se requiere que cada reasignación cumpla con una serie de restricciones, que incluyen por ejemplo que la suma de los recursos requeridos para satisfacer la demanda total de los procesos asignados a una máquina no sobrepase su capacidad, o que ciertos conjuntos de procesos deben distribuirse de una manera específica cuando son asignados a las máquinas, entre otras.

El objetivo es encontrar una asignación que minimice los costos y que satisfaga todas las restricciones del problema.

Si bien este problema cuenta con algunos predecesores, fue propuesto con estas características específicas en el contexto del desafío ROADEF/EURO en conjunto con Google del año 2012 [6]. A continuación se describirán las distintas partes del problema, mostrando, además, algunos problemas relacionados y variantes de éste.

2.1. Elementos del problema

Los distintos elementos y conceptos que son necesarios para entender el problema son los siguientes:

- Procesos: Son los elementos que deben ser asignados, consumen ciertos recursos y cada uno pertenece a un servicio.
- Servicios: Pueden ser vistos como conjuntos de procesos. Existen relaciones entre servicios, que se traducen en restricciones del problema.
- Máquinas: Son los elementos a los cuales serán asignados los procesos. Cada máquina tiene recursos que definen su capacidad y que son usados cuando un

proceso es asignado a ella. Cada máquina pertenece a una localización y a un vecindario.

- Localizaciones: Se definen como un conjunto de máquinas. Dependiendo del servicio al que pertenezcan los procesos asignados a cada máquina, existen restricciones que definen las localizaciones distintas en las que tienen que estar los procesos de un servicio. Es importante señalar que todas las localizaciones son conjuntos disjuntos.
- Vecindarios: Se definen como un conjunto de máquinas. Dependiendo de las relaciones entre los servicios a los que pertenezcan los procesos asignados a cada máquina, existen restricciones que indican en qué vecindarios deben ir los procesos de un servicio que depende de otro. Es necesario mencionar que se considera que todos los vecindarios son conjuntos disjuntos.

2.2. Restricciones del problema

Las restricciones del problema se agrupan en cinco grandes categorías, las cuales se definen a continuación:

1. **Restricciones de capacidad:** Este tipo de restricciones garantiza que el uso de los recursos de una máquina no exceda su capacidad. Esto se debe a que no es posible asignar recursos que no existen o bien, que no están disponibles en una máquina. De esta manera un proceso puede ser ejecutado en una máquina sólo si esta aún tiene capacidad suficiente disponible para satisfacer todas sus demandas.
2. **Restricciones de conflictos:** Estas restricciones señalan que dos procesos que pertenecen a un mismo servicio no pueden ser asignados a la misma máquina. Con esto, se busca minimizar los problemas causados a un servicio específico en caso de que una máquina falle, afectando sólo un proceso de cada servicio.
3. **Restricciones de dispersión:** En cada servicio existe un valor asociado llamado valor numérico mínimo de dispersión (*spreadmin*), el cual indica que todos los procesos de ese servicio deben ser asignados a máquinas de a lo menos esa cantidad de localizaciones distintas. En la Figura 2.1 puede verse un ejemplo de esta restricción, en donde teniendo dos servicios s_1 y s_2, s_1 cumple con la restricción de dispersión ya que sus procesos están siendo asignados a dos localizaciones distintas (igual al valor de su *spreadmin*). Por otro lado, el servicio s_2 no está cumpliendo con la restricción, ya que de acuerdo a su *spreadmin* sus procesos deben estar asignados a lo menos a dos localizaciones distintas, y todos están siendo asignados a la misma.

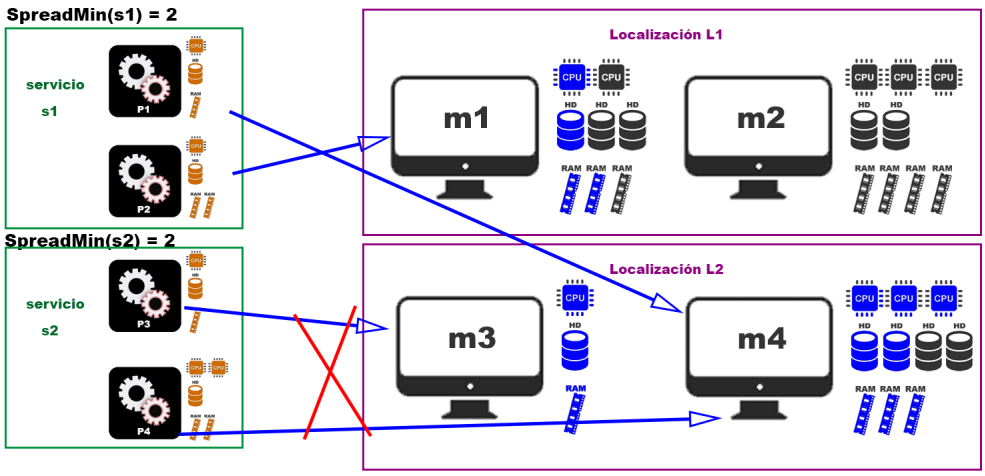


Figura 2.1: Ejemplo de restricción de dispersión.

4. **Restricciones de dependencia:** Cuando un servicio s_2 depende de otro s_1, esta restricción indica que cada uno de los procesos del servicio s_2 debe estar asignado a máquinas cuyo vecindario sea el mismo de alguna máquina a la cual están asignados los procesos de s_1. En la Figura 2.2 puede verse un ejemplo de la aplicación de esta restricción, en donde teniendo dos servicios s_1 y s_2 el segundo depende del primero. Puede apreciarse que la restricción no se está cumpliendo ya que el proceso p_4 del servicio s_2 se asigna a la máquina m_3, la cual no pertenece al vecindario de ninguno de los procesos de s_1.

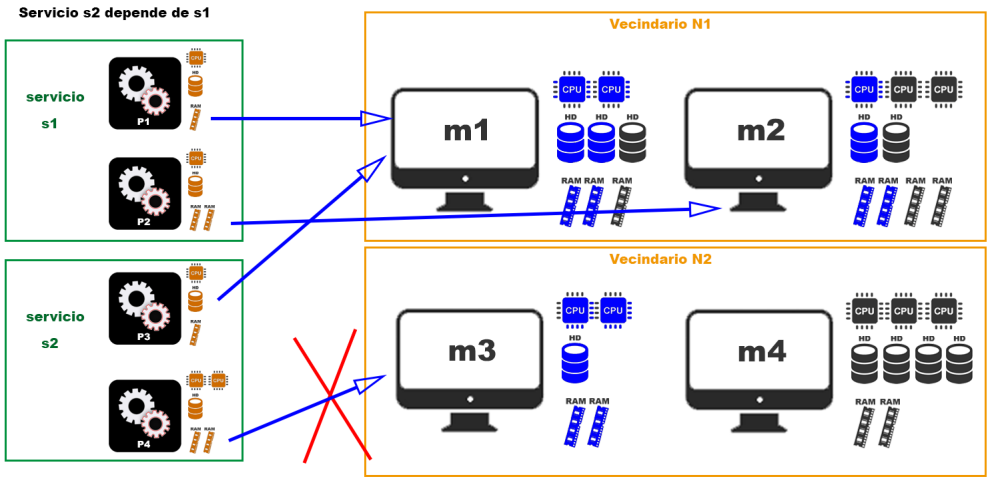


Figura 2.2: Ejemplo de restricción de dependecia.

5. **Restricciones de uso transitorio:** Existe la posibilidad de que, si un proceso se mueve de una máquina a otra, algunos recursos utilizados por el proceso se mantengan utilizados en ambas máquinas. A los recursos que tienen este tipo de características se les llama recursos transitorios. A partir de esto, para la restricción de capacidad asociada a la máquina, en la cual el recurso estaba asignado, deberá tomarse en cuenta que ese recurso aún está siendo usado por dicho proceso. En la Figura 2.3 se muestra un ejemplo de estas restricciones, en donde $TR = \{HD\}$ indica que el recurso de disco duro es transitorio. En la primera imagen puede verse como el proceso p_2 está asignado en una primera instancia a la máquina m_2. Luego, en la segunda imagen se observa que p_2 fue asignado a la máquina m_3, utilizando recursos de CPU y disco duro. Si bien el recurso de CPU ha sido liberado de m_2, el recurso de disco duro aún sigue usándose. Finalmente, en la última imagen se aprecia que tratar de asignar el proceso p_1 a m_2 incurrirá en una violación de la restricción de capacidad, ya que no existe suficiente disco duro. Es muy importante señalar que en el MRP no existe la dimensión del tiempo, es decir, que todos los movimientos son realizados exactamente en el mismo instante de tiempo.

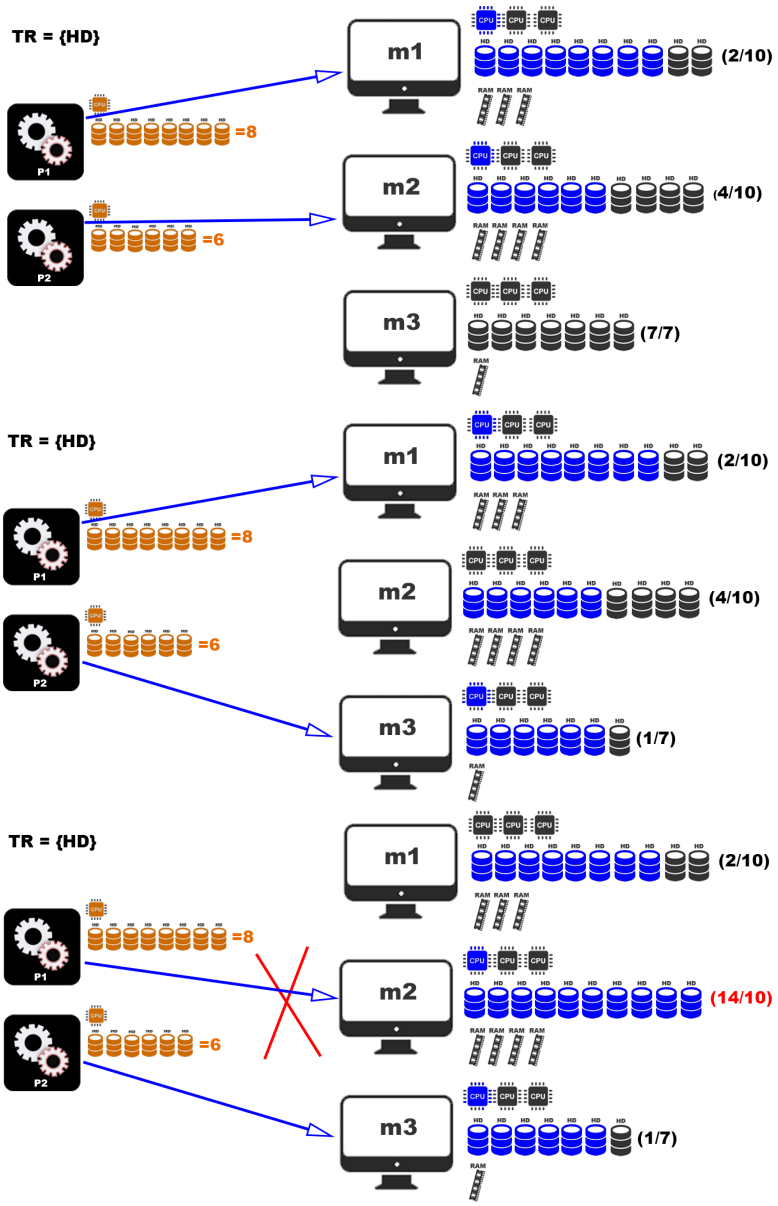


Figura 2.3: Ejemplo de restricción de uso transitorio.

2.3. Objetivos del problema

El objetivo general del problema es lograr una mejor distribución de los procesos de las máquinas. A partir de esto, existe una función de costos a minimizar, la cual está

compuesta por distintos costos que sumados y ponderados conforman el costo total.

- **Costo de carga:** Para esto se define la "capacidad segura de carga", que es un límite para la ocupación de cada recurso. Si son asignados más procesos a esa máquina y se sobrepasa ese margen, se incurrirá en costos asociados. Si bien pueden haber muchas razones para la consideración de este costo, una podría ser que las máquinas que están demasiado utilizadas podrían consumir más energía. Por otro lado, mantenerlas utilizadas por debajo de sus capacidades seguras, también llevará a una distribución más uniforme de las asignaciones. El costo de carga de cada recurso corresponde a la suma de los costos de carga en todas las máquinas, medido en el número de unidades en que se supera la capacidad segura de cada una para ese recurso. Un ejemplo de esto se puede ver en la Figura 2.4, donde en las máquinas m_1 y m_4 se está sobrepasando la capacidad segura de carga. Cada recurso tiene asociado además un peso, el cual pondera su aporte al costo de carga total.

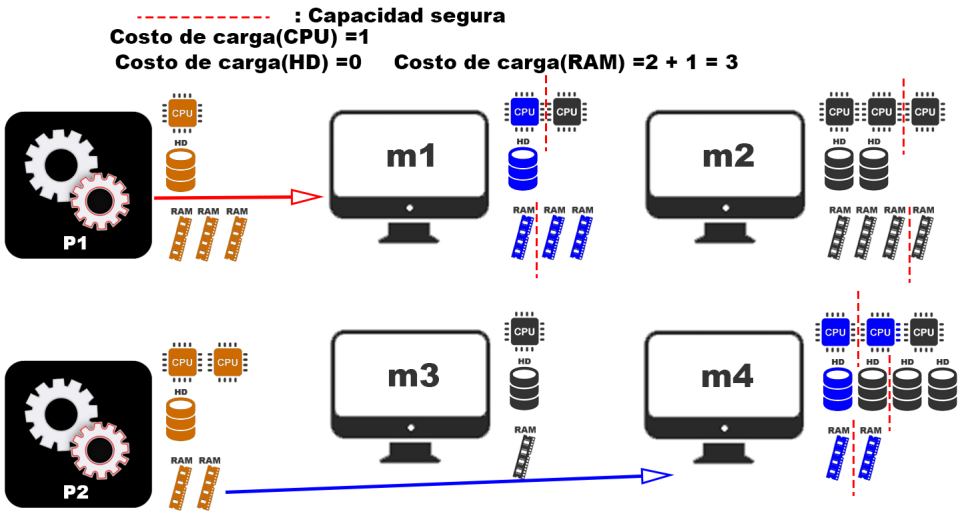


Figura 2.4: Ejemplo de costo de carga

- **Costo de balance:** En este caso, se quiere que los recursos disponibles sean utilizados de forma balanceada dentro de cada máquina, de manera que el uso de cada uno este equilibrado. Este equilibrio será de vital importancia en futuras asignaciones. De acuerdo a lo anterior, un mal uso de una máquina estaría asociado, por ejemplo, al exceso de memoria RAM a la vez que no hay suficiente disco duro, ya que esto traerá consigo que no sea posible asignar procesos por violar la restricción de capacidad. Para definir las distintas relaciones de balance se utiliza

un valor entero por cada par de recursos. Por ejemplo $\{CPU, RAM, 2\}$ indicaría que por cada unidad de CPU disponible, deberá haber el doble de unidades disponibles de RAM. En caso de no cumplirse esta relación, el costo asociado será la suma de las unidades desbalanceadas en cada máquina. Un ejemplo de lo anterior puede observarse en la Figura 2.5, en el que se tiene la relación $\{CPU, RAM, 2\}$, destacándose lo siguiente lo siguiente:

- En la máquina m_1 hay 6 unidades disponibles de CPU. De acuerdo a la relación, deberían haber $2{*}6 = 12$ unidades de RAM disponibles. Por esto, se incurre en un costo de $12-7 = 5$ por solamente tener 7 disponibles. Por otro lado, en la máquina m_2, a partir de la CPU utilizada se exigen 6 unidades de RAM disponibles, y como existen 11 unidades de RAM disponibles no se incurre en costo alguno.

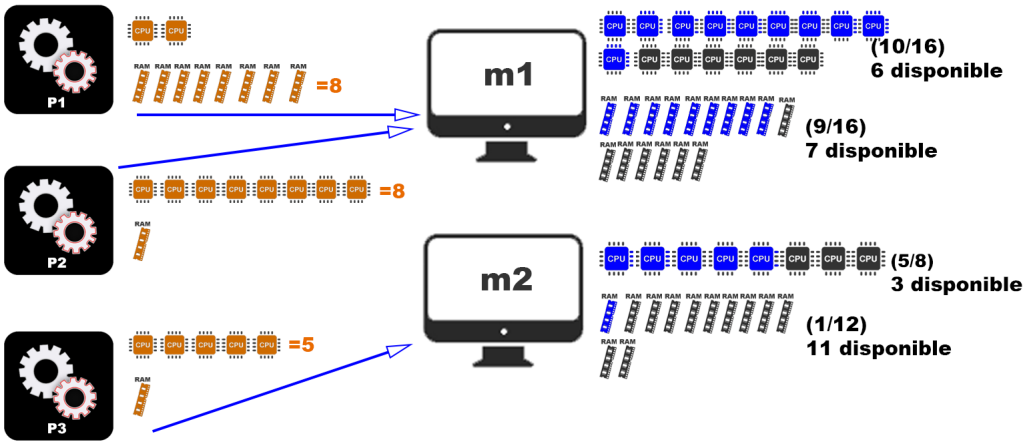


Figura 2.5: Ejemplo de costo de balance

De esta manera, se define un conjunto de tripletas $\{Recurso_1, Recurso_2, N\}$ que expresan las distintas relaciones de balance entre cada par de recursos. Por otro lado, cada tripleta tiene asociado un peso que pondera su aporte al costo de balance total.

- **Costo de movimiento de procesos:** Puede suceder que mover algunos procesos sea una tarea muy costosa, por lo que se define un costo para su movimiento. De esta manera, el costo de movimiento de todos los procesos es igual a la suma de los costos de aquellos procesos que han sido trasladados. También existe un peso asociado que pondera su aporte a la función de costos totales.

- **Costo de movimiento de servicios:** Este costo se define para como la máxima cantidad de procesos movidos entre todos los servicios, buscando equilibrar los movimientos realizados a los procesos de cada uno.
- **Costo de movimiento entre máquinas:** Este costo está asociado a transferir procesos de una máquina a otra, para lo cual se define una matriz de costos dependiendo de las máquinas de origen y destino. Así, el costo de movimiento entre máquinas es el costo de mover cada proceso, desde su máquina de origen, a la máquina de destino. Un ejemplo de la matriz de costos puede verse en la tabla siguiente:

x	m_1	m_2	m_3
m_1	0	25	125
m_2	25	0	5
m_3	125	5	0

  Este costo también es ponderado por un peso en la función de costos totales.
- **Función de costos totales:** Esta función es la suma de todos los costos anteriormente mencionados, ponderados por sus respectivos pesos.

2.4. Problemas relacionados

Si bien el Machine Reassignment Problem fue propuesto en 2012, existen problemas que comparten características en común y que son más antiguos, algunos remontan incluso a los años 90. A continuación, se presentan algunos problemas similares y que han sido propuestos anteriormente, así como también su descripción general y características principales. Es importante destacar que una revisión de estos problemas es igualmente importante, ya que algunos de los mejores resultados para el MRP en la literatura fueron obtenidos usando heurísticas y movimientos inspirados en problemas anteriores.

1. Generalized Assignment Problem (GAP) [32]: Este problema consiste en la asignación de tareas a agentes que incurren en costos y consumen recursos que varían dependiendo de la asignación tarea-agente. Además, cada tarea debe ser asignada sólo a un agente y cada agente tiene una cantidad de recursos disponible, teniéndose que la suma de las tareas asignadas no puede superar esta cantidad. Seguidamente, el objetivo es buscar una asignación que minimice el costo incurrido en todas las asignaciones. Sin duda alguna, este problema guarda similitudes con el MRP, en el que cada máquina puede ser vista como agente y cada proceso como una tarea. En [4] se propone un algoritmo cooperativo multi-hilo basado en Variable Neighborhood Search y una Hiperheurística que utilizando los mejores movimientos conocidos del GAP Generalized Assignment Problem busca resolver el MRP.

2. Multi-dimensional Resource Scheduling for Parallel Queries: Este problema se refiere a la asignación óptima de tareas a procesadores, considerando un beneficio de asignación asociado, la cantidad de recursos necesarios respecto de la asignación y la asignación total disponible de recursos de los procesadores. En [12] se presenta el problema de asignación multidimensional de recursos, aplicado al agendamiento de queries paralelas en bases de datos. De acuerdo a lo que se menciona, la tarea de agendar ejecuciones de forma paralela se vuelve particularmente complicada en ambientes de ejecución de tipo *share-nothing*[1], en donde cada nodo del sistema representa un conjunto de recursos tales como CPU, disco duro, RAM, programado idénticamente entre sí o de manera similar en el manejo de sus recursos. En este sentido, se busca ganar rendimiento en la ejecución de queries por medio de un mayor grado de paralelismo, el cual es logrado al asignar de buena forma las tareas independientes de cada query a los recursos disponibles. A partir de lo anterior, cada nodo puede ser asociado con una máquina en MRP, con recursos independientes que deberán ser asociados de forma correcta a las tareas o procesos.
3. Vector Bin Packing Problem (VBPP)[11]: El Bin Packing Problem es un común reductor de muchos problemas en la literatura, siendo MRP uno de ellos. El problema de asignar un conjunto de procesos a un conjunto de máquinas en servidores es básicamente un problema tipo bin packing. En el VBPP cada item tiene múltiples dimensiones mientras que cada bin tiene una capacidad C en cada una de las dimensiones. De esta manera, el VBPP puede verse como una simplificación del MRP, en donde cada máquina es un recipiente $d-dimensional$ y cada proceso es un ítem, donde los tamaños de cada ítem vienen dados por los tamaños de sus múltiples recursos (dimensiones). Se señala que es una simplificación ya que si bien cada proceso y máquina poseen múltiples dimensiones o recursos, las máquinas tienen la misma capacidad para todos ellos. Una asignación factible para este problema es una partición $P_1, P_2, ..., P_n$ tal que para cada una, la suma de los items en dicha partición no exceda el valor C, para cada dimension.
4. Multi Capacity Bin Packing Problem (MCBPP): Este problema es similar al Vector Bin Packing Problem, pero además se considera que cada bin tiene diferentes capacidades en cada dimensión y cada máquina. A partir de esto, el objetivo es asignar la mayor cantidad de procesos dentro de cada recipiente, mientras se respetan sus respectivas restricciones de capacidad . Por otro lado, en el caso del MRP se requiere optimizar el uso de las máquinas reasignando los procesos, de modo que se reduzcan distintos costos asociados.
   En [23] se propone una metaheurística multi-start basada en búsqueda local para problemas bin packing, y adaptada para problemas MRP de gran escala y problemas bin packing con múltiples recursos.

[1] Arquitectura distribuida en donde cada nodo es independiente y autosuficiente, teniendo sus propios recursos y no compartiendo ninguno.

En [10] este problema es denominado Vector Bin Packing with Heterogeneous Bins (VBPHB), y los autores proponen diferentes heurísticas basadas en greedy para la generación rápida de soluciones de dicho problema. Además, examinan diversas propiedades estructurales del MRP, lo cual les permite adaptar dichas heurísticas y poder generar también soluciones rápidas para este.

5. Virtual Machine Placement Problem [5]: Este problema busca determinar el host (o máquina física en el data center) donde se instanciará cada máquina virtual. Dado un conjunto máquinas físicas y una cola de peticiones de clientes para instanciar máquinas virtuales, se busca conseguir al menos uno de los siguientes objetivos:

   a) Minimizar el consumo eléctrico del Data Center.

   b) Maximizar la tasa de colocación, que está definida como el número de instanciaciones total respecto al tamaño de la cola de peticiones.

   c) Minimizar la degradación de rendimiento total del sistema debido a la migración de las máquinas virtuales.

   En [9] se muestra cómo el problema clásico Bin Packing Problem sirve para solucionar el Virtual Machine Placement Problem y el Machine Reassignment Problem. A grandes rasgos, el objetivo general es encontrar una asignación factible, dado un conjunto de elementos, una capacidad y una función de tamaño, minimizando el número de recipientes. Dado que cada recipiente tiene una capacidad, la suma de los pesos de cada elemento asignado a cada recipiente no debe sobrepasar su capacidad máxima.

2.5. Variantes del problema

Existen en la actualidad diversas variantes del MRP, que incorporan nuevos elementos no considerados en su formulación original. Las variantes más destacadas se muestran a continuación:

1. Machine Reassignment Problem con modelamiento de tiempos de respuesta: En [7], aun cuando el concurso estaba en proceso, se presentó una variación del MRP en el que se incorpora el modelamiento de tiempos de respuesta *end-to-end* y restricciones asociados a éstos. En líneas generales, los autores señalan que el problema descrito en el desafío Google no considera dos aspectos esenciales:

   a) No toma en consideración los tiempos de respuesta *end-to-end*: Esto se refiere a que una posible solución en el problema original podría llevar al incremento de tiempos de respuesta, aún cuando la función de costo penaliza la alta utilización de recursos (y por ende los altos tiempos de respuesta)

por medio de los costos de carga. Lo anterior se debe a que los tiempos de respuesta no están siendo modelados ni restringidos explícitamente.

b) Cuando se mueven procesos entre máquinas no se toma en consideración que esto puede aumentar el consumo de recursos: Esto hace referencia a que, si dos servicios s_1 y s_2 están relacionados por restricciones de dependencia (por ejemplo s_1 depende de s_2), es posible que uno de ellos, por ejemplo s_1, envíe tareas al otro (s_2), tal como sucede en el caso de los despachadores de solicitudes en los servidores webs. A partir de esto y de acuerdo a la restricción de dependencia, s_1 deberá tener sus procesos en alguno de los vecindarios donde existe algún proceso de s_2. Según lo que señalan los autores, la cantidad de tareas que los procesos de s_1 enviarán a los procesos de s_2 aumentará si es que algunos procesos de s_2 son reasignados a vecindarios en los que no hay procesos de s_1. Por ejemplo, si inicialmente s_1 tiene todos sus procesos asignados al vecindario v_1, y s_2 tiene 3 procesos asignados al vecindario v_1 y 3 procesos asignados al vecindario v_2, y luego dos procesos de s_2 son reasignados y ahora quedan en el vecindario v_2, la condición de dependencia sigue cumpliéndose, pero se tendrá que como s_1 manda trabajo sólo a los procesos de s_2 que están en el mismo vecindario, todo el trabajo que envíe ahora estará siendo realizado por un proceso en lugar de tres. Este cambio de asignaciones tiene como efecto un cambio en el enrutamiento de las tareas entre procesos y por consiguiente, la carga asociada a cada proceso.

A partir de lo anterior, extienden el modelo para tratar de dar solución a los problemas antes mencionados, modificando la definición de servicios e introduciendo "probabilidades de ruteo" y número de visitas. Además, desarrollan un modelo paramétrico de encolamiento para la estimación de tiempos de respuesta.

2. Multi-Objective Machine Reassignment Problem: En [30] se presenta una generalización del MRP en la que se agregan múltiples objetivos. Mientras en el MRP original se busca encontrar una nueva asignación para cada proceso inicialmente asignado a una máquina $M_0(p)$, cumpliendo con las restricciones y evaluando con una función mono-objetivo ponderada, en el MRP multi-objetivo se busca encontrar aquellas soluciones no dominadas o pertenecientes al frente de Pareto. Su estudio se enfoca en los tres objetivos que consideran más importantes: confiabilidad, migración y costos de electricidad para los cuales definen funciones de costo asociadas. Cada costo se define de la siguiente manera:

   a) Confiabilidad: Una máquina m es confiable si no está cargada más que un valor límite de confianza. Así, si la capacidad segura de carga de una máquina m para el recurso r es más grande que la suma de las demandas, entonces no impactará en la confiabilidad de la máquina. Esta definición está inspirada en el concepto de capacidad segura de carga del MRP.

b) Migración: Representa el tiempo necesario para preparar un proceso p para su migración desde una máquina a otra. Todos estos costos son dependientes de parámetros de cada proceso (por ej. tamaño de los datos guardados en el disco duro y la RAM, complejidad de la instalación) y parámetros de topología (por ej. número de saltos, ancho de banda). A partir de lo anterior se define un costo de migración.

c) Electricidad: Este costo es de gran importancia, ya que el 50 % de los costos operativos de las máquinas se deben al costo asociado a su consumo de electricidad [8]. Los autores modelan la función de costo eléctrico, la cual si bien es compleja de evaluar, generalmente depende de forma lineal del uso de CPU. Este costo se define en función de dos parámetros, uno lineal y otro fijo dependiente de la carga de la CPU para cada máquina m.

2.6. Complejidad del problema

Dado que el Machine Reasignment Problem es reducible a un Bin Packing, tiene complejidad fuertemente NP-Hard [29], lo cual significa que en la práctica para cualquier función objetivo computable en tiempo polinomial, la cantidad de tiempo computacional y número de operaciones o cálculos necesarios para resolverlo crecerán exponencialmente a medida que crezca el tamaño de la instancia a resolver.

2.7. Resumen

En este capítulo se presenta el Machine Reassignment Problem, sus componentes, restricciones y objetivos. Además, se hizo una revisiónde algunos problemas relacionados presentes en la literatura. Se presentaron las variantes del MRP, que incorporan nuevos componentes y traen consigo mayores complejidades y desafíos. En el siguiente capítulo se hará una revisión de avances y técnicas que se han utilizado en la literatura, así como también los modelos matemáticos propuestos con sus respectivas restricciones asociadas al problema.

3 Estado del Arte

Tal como se ha mencionado anteriormente existen distintos problemas de reasignación, los cuales, en líneas generales, son tareas que deben ser realizadas por distintos servidores o agentes. Dichos problemas son anteriores al MRP y han sido revisados brevemente en la Sección 2.4 de problemas relacionados. Por otro lado, el Machine Reassignment Problem, fue propuesto como tal en el marco de la competencia ROADEF/EURO en conjunto con Google el año 2012 [6]. En la definición del problema presentada en el sitio web se describe el MRP en detalle junto con el formato de las instancias, sus características y el output esperado para el software que intente solucionarlo.

3.1. Competencia EURO/ROADEF 2012

La competencia contó con un total de 82 equipos competidores registrados, de los cuales sólo 48 enviaron sus algoritmos para calificación. Dichos equipos fueron divididos en las categorías de Junior, para jóvenes investigadores, y Senior, para investigadores calificados. Cada uno de los algoritmos presentados fue ejecutados por 300 segundos, una sola vez. Luego fue definida una métrica de puntaje para cada resultado utilizando la siguiente función:

$$Puntaje(I) = \frac{Costo(S) - Costo(B)}{Costo(R)} \cdot 100$$

Aquí, I es la instancia actual en evaluación, R la solución inicial, S la solución encontrada por el algoritmo, y B la mejor solución encontrada entre todos los algoritmos para esa instancia. Tal como se ha señalado en [22] esta función tiene el beneficio asociado de no ser influenciada por algunas instancias que tienen costos más altos que otras, ya que está centrada en un valor normalizado que sirve para puntuar cada algoritmo.

A continuación, se mostrará el puntaje total de cada equipo para cada una de las dos etapas de la competencia, el cual se definió como la suma de los puntajes asociados a cada instancia:

1. Etapa de clasificación: Se ejecutó cada algoritmo con instancias de tamaño medio y fueron seleccionados los 30 mejores puntajes de cada categoría. Los resultados de esta primera etapa, así como también los puntajes asociados a los cinco primeros equipos pueden verse en la Figura 3.1.

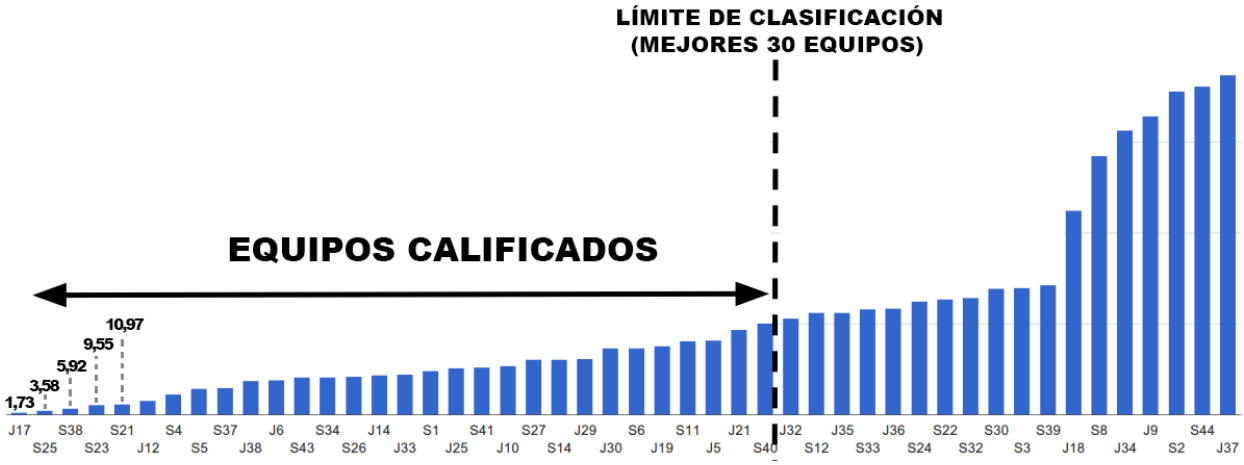


Figura 3.1: Resultados de la fase de clasificación. Elaborado a parir de resultados mostrados en [14].

2. Etapa final: Se ejecutó cada algoritmo con instancias de gran tamaño, desde 5000 hasta 50.000 procesos y desde 100 hasta 5.000 máquinas. Los resultados de la categoría junior, categoría senior y clasificación general se muestran a continuación (un mayor detalle de los resultados de la competencia puede ser visto en [25]).

   a) Categoría Junior:

      1) Primer lugar: Equipo J12, con puntaje de 1,72 ([18]).
      2) Segundo lugar: Equipo J25, con puntaje de 2,60 ([3]).
      3) Tercer lugar: Equipo J33, con puntaje de 4,66.

   b) Categoría Senior:

      1) Primer lugar: Equipo S41, con puntaje de 0,47 ([14, 13]).
      2) Segundo lugar: Equipo S38, con puntaje de 0,62 ([24, 22]).
      3) Tercer lugar: Equipo S14, con puntaje de 3,91.

   c) Clasificación General (Junior y Senior):

      1) Primer lugar: Equipo S41, con puntaje de 0,47 ([14, 13]).
      2) Segundo lugar: Equipo S38, con puntaje de 0,62([24, 22]).
      3) Tercer lugar: Equipo J12, con puntaje de 1,72 ([18]).
      4) Cuarto lugar: Equipo J25, con puntaje de 2,60 ([3]).

3.2. Instancias y benchmarks

En el sitio web de la competencia [1] han sido publicadas las instancias del problema, las cuales son 30 en total y están divididas en tres grupos en orden creciente de dificultad, estas instancias han sido ampliamente utilizadas para diversas pruebas de modelos y diferentes algoritmos. Cada una consta de dos archivos, uno que contiene la asignación inicial y otro con los demás datos. Por otro lado, también se presenta un checker de soluciones correctas junto con los resultados para cada una de las instancias.

Las características generales de cada instancia, así como sus mejores resultados encontrados en la competencia ROADEF se presentan a continuación en la Tabla 3.1. En el caso de las instancias de los grupos A1 y A2, los resultados fueron obtenidos a partir de la fase de clasificación, para las cuales también están disponibles para descargar las soluciones que dan origen a estos valores. Por otro lado, para las instancias B y X los resultados fueron obtenidos a partir de la fase final.

[1] http://challenge.roadef.org/2012/en/instances.php

Instancia	N^oRec.	N^oMáq.	N^oServ.	N^oProc.	N^oLoc.	N^oVec.	Mejor Solución
A1_1	2	4	79	100	4	1	44.306.501
A1_2	4	100	980	1000	4	2	777.532.896
A1_3	3	100	216	1000	25	5	583.005.717
A1_4	3	50	142	1000	50	50	252.728.589
A1_5	4	12	981	1000	4	4	727.578.309
A2_1	3	100	1.000	1000	1	1	198
A2_2	12	100	170	1000	25	5	816.523.983
A2_3	12	100	129	1000	25	5	1.306.868.761
A2_4	12	50	180	1000	25	5	1.681.353.943
A2_5	12	50	153	1000	25	5	336.170.182
B_1	12	100	2.512	5000	10	5	3.339.186.879
B_2	12	100	2.462	5000	10	5	1.015.553.800
B_3	6	100	15.025	20000	10	5	156.835.787
B_4	6	500	1.732	20000	50	5	4.677.823.040
B_5	6	100	35.082	40000	10	5	923.092.380
B_6	6	200	14.680	40000	50	5	9.525.857.752
B_7	6	4000	15.050	40000	50	5	14.835.149.752
B_8	3	100	45.030	50000	10	5	1.214.458.817
B_9	3	1000	4.609	50000	100	5	15.885.486.698
B_10	3	5000	4.896	50000	100	5	18.048.515.118
X_1	12	100	2.529	5000	10	5	3.100.852.728
X_2	12	100	2.484	5000	10	5	1.002.502.119
X_3	6	100	14.928	20000	10	5	211.656
X_4	6	500	1.190	20000	50	5	4.721.629.497
X_5	6	100	34.872	40000	10	5	93.823
X_6	6	200	14.504	40000	50	5	9.546.941.232
X_7	6	4000	15.273	40000	50	5	14.253.273.178
X_8	3	100	44.950	50000	10	5	42.674
X_9	3	1000	4.871	50000	100	5	16.125.612.590
X_10	3	5000	4.615	50000	100	5	17.816.514.161

Tabla 3.1: Datos de las instancias publicadas en ROADEF.

Además de lo anterior, en [22] añaden un dato destacable, el cual señala que para cada instancia la mejor solución reduce en promedio, la solución inicial en un $65,78\,\%$.

3.2.1. Formato de instancias de prueba

El formato de instancias de pruebas está especificado en detalle en [6]

3.3. Modelo Matemático

Tal como se ha señalado anteriormente, el problema ha sido cubierto por varios autores en la literatura, los cuales han propuesto diferentes modelos. A continuación se describen las notaciones y variables de dos de los modelos existentes, así como también una breve descripción de cada uno.

3.3.1. Conjuntos relacionados

Los siguientes conjuntos relacionados con el MRP han sido utilizados con una nomenclatura similar en distintas publicaciones:

$\mathcal{M}$: Conjunto de todas las máquinas.

$\mathcal{R}$: Conjunto de todos los recursos.

$\mathcal{TR}$: Conjunto de todos los recursos considerados transitorios.

$\mathcal{P}$: Conjunto de todos los procesos.

$\mathcal{S}$: Conjunto de todos los servicios.

$\mathcal{L}$: Conjunto de todas las localizaciones.

$\mathcal{N}$: Conjunto de todos los vecindarios.

$\mathcal{B}$: Conjunto de todas las tripletas de costo de balance.

3.3.2. Modelo base propuesto en la definición del problema

Este modelo fue propuesto por ROADEF/EURO en la definición y presentación del problema ([6]), y ha sido utilizado en [14]. Se destaca porque es relativamente sencillo y consta de las siguientes partes:

3.3.2.1. Parámetros de entrada

Se definen los siguientes parámetros de entrada:

1. $C(m, r)$: Es la capacidad de cada recurso $r \in \mathcal{R}$ en cada máquina $m \in \mathcal{M}$.
2. $R(p, r)$: Es la cantidad del recurso $r \in \mathcal{R}$ que requiere cada proceso $p \in \mathcal{P}$.
3. $SpreadMin(s)$: Indica el número mínimo de localizaciones $l \in \mathcal{L}$ distintas en las que deben ser distribuidos los procesos $p \in s$ para cada servicio $s \in \mathcal{S}$.
4. $SC(m, r)$: Safety capacityEs la capacidad segura de carga de cada recurso $r \in \mathcal{R}$ en cada máquina $m \in \mathcal{M}$.

5. $PMC(p)$: Es el costo de mover cada proceso $p \in \mathcal{P}$ desde su máquina de origen $M_0(p)$ a cualquier otra.
6. $MMC(m_0(p), m_1(p))$: Son los costos de mover un proceso $p \in \mathcal{P}$ desde la máquina m_0 a cualquier otra máquina m_1.
7. $target_{r_1,r_2}$: Es la proporción entre los recursos r_1 y r_2 pertenecientes a una tripleta en el cálculo del costo de balance. Cada tripleta tiene la forma $b = \{Recurso_1, Recurso_2, N\}$.
8. $weight_{loadCost}(r)$: Son los pesos del costo de carga para cada recurso r específico.
9. $weight_{balanceCost}(b)$: Son los pesos de los costos de balance para cada tripleta b específica.
10. $weight_{processMoveCost}$: Es el peso del costo de movimiento de procesos, es el mismo para todos los procesos.
11. $weight_{serviceMoveCost}$: Es el peso del costo de movimiento de servicios, es el mismo para todos los servicios.
12. $weight_{machineMoveCost}$: Es el peso del costo de movimiento de máquinas, es el mismo para todas las máquinas.

3.3.2.2. Variables de decisión

Se tienen las siguientes variables de decisión:

1. $M(p) = m$: Representa una asignación del proceso p a la máquina m, con $M : [1, p] \to [1, m]$.
2. $U(m, r)$: Es el uso del recurso $r \in \mathcal{R}$ para la máquina $m \in \mathcal{M}$, con $U : [1, m][1, r] \to \mathbb{R}^+$. En términos matemáticos :

$$U(m, r) = \sum_{p \in \mathcal{P} | M(p) = m} R(p, r)$$

3.3.2.3. Restricciones

Las restricciones que deben cumplirse son las siguientes:

1. **Restricciones de capacidad:** La cantidad de recursos $r \in \mathcal{R}$ requeridos por los procesos asignados a la máquina $m \in \mathcal{M}$ no debe superar la capacidad de la máquina para ese recurso.

$$\forall m \in \mathcal{M},\, r \in \mathcal{R},\, U(m, r) \leq C(m, r)$$

2. **Restricciones de capacidad con recursos transitorios:** Aseguran que se cumplan las restricciones de capacidad tomando en cuenta los recursos transitorios, esto es, tomando en cuenta que existen algunos recursos que a pesar de haber sido asignados a una máquina $m \in \mathcal{M}$ distinta de m_0, igualmente siguen ocupando recursos en m_0.

$$\forall m \in \mathcal{M},\, r \in \mathcal{TR}, \sum_{p \in \mathcal{P} | M_0(p)=m \vee M(p)=m} R(p,r) \leq C(m,r)$$

3. **Restricciones de conflictos:** Los procesos $p \in \mathcal{P}$ que pertenecen a un mismo servicio $s \in \mathcal{S}$ no pueden ser asignados a la misma máquina.

$$\forall s \in \mathcal{S},\, (p_i, p_j) \in s^2,\, p_i \neq p_j \Rightarrow M(p_i) \neq M(p_j)$$

4. **Restricciones de dispersión:** Los procesos $p \in \mathcal{P}$ que pertenecen a un mismo servicio $s \in \mathcal{S}$ deben ser asignados a un número mínimo de localizaciones, el cual está determinado por el valor $SpreadMin$ del servicio.

$$\forall s \in \mathcal{S}, \sum_{l \in \mathcal{L}} min\left(1, \left|\{p \in s \mid M(p) \in l\}\right|\right) \geq SpreadMin(s)$$

5. **Restricciones de dependencia:** Si un servicio $s_a \in \mathcal{S}$ depende de otro $s_b \in \mathcal{S}$, entonces el primero debe tener al menos un proceso que esté en el mismo vecindario que alguno de los procesos del segundo. Es decir, debe tener al menos un proceso cuya máquina asignada pertenezca al mismo vecindario que la máquina asignada de alguno de los procesos de s_b.

$$\forall p_a \in s_a,\, \exists p_b \in s_b \wedge n \in \mathcal{N} \mid M(p_a) \in n \wedge M(p_b) \in n$$

3.3.2.4. Función Objetivo

Está compuesta por varias funciones de costos que representan los objetivos señalados en la Sección 2.3:

1. **Costo de carga:** Suma de los costos de cada máquina $m \in \mathcal{M}$ por exceder sus capacidades seguras de carga. Busca no llevar la máquina a sus límites.

$$loadCost(r) = \sum_{m \in \mathcal{M}} max\left(0, U(m,r) - SC(m,r)\right)$$

2. **Costo de balance:** Es el costo por no usar uniformemente los recursos en cada máquina. Busca una configuración más robusta en cuanto a la incorporación de nuevos procesos a futuro.

$$balanceCost(b) = \sum_{m \in \mathcal{M}} max\left(0, target \cdot A(m, r_1) - A(m, r_2)\right)$$

   Teniéndose que: $A(m, r) = C(m, r) - U(m, r)$

3. **Costo de movimiento de procesos:** Es la suma de los costos de mover un proceso desde la máquina original. Estos costos **siempre se calculan a partir de $M_0(p)$, ya que en el MRP no existe dimensión del tiempo**, lo cual asume que todos los movimientos son en el mismo instante de tiempo.

$$processMoveCost = \sum_{p \in \mathcal{P} \mid M(p) \neq M_0(p)} PMC(p)$$

4. **Costo de movimiento de servicios:** Es el costo de asignar procesos de un servicio específico a máquinas distintas de la original.

$$serviceMoveCost = \max_{s \in \mathcal{S}} \left(\left| \{p \in s | M(p) \neq M_0(p)\} \right| \right)$$

5. **Costo de movimiento de máquinas:** Es la suma de los costos de cambiar un proceso desde su máquina inicial.

$$machineMoveCost = \sum_{p \in \mathcal{P}} MMC(M_0(p), M(p))$$

Función de costo Final: Es la suma de todos los costos antes mencionados, ponderados por sus pesos respectivos, la cual busca minimizarse:

$$\begin{aligned} totalCost = & \\ & \sum_{r \in \mathcal{R}} weight_{loadCost}(r) \cdot loadCost(r) \\ & + \sum_{b \in \mathcal{B}} weight_{balanceCost}(b) \cdot balanceCost(b) \\ & + weight_{processMoveCost} \cdot processMoveCost \\ & + weight_{serviceMoveCost} \cdot serviceMoveCost \\ & + weight_{machineMoveCost} \cdot machineMoveCost \end{aligned}$$

3.3.3. Modelo de Programación Lineal Entera Binaria

Este modelo fue utilizado por Renaud Masson et. al. en [23], está basado principalmente en variables binarias para especificar la asignación de los procesos a las máquinas, para saber si un proceso está en un vecindario (i.e. asignado a una máquina de ese vecindario) y saber si un proceso está en una localización (i.e. está asignado a una máquina de esa localización).

3.3.3.1. Conjuntos

Además de los conjuntos mencionados en la Sección 3.3.1 en esta publicación se añadió el siguiente:

1. D^s: Conjunto de servicios de los cuales depende el servicio s, con $D^s \subset \mathcal{S}$, $s \in \mathcal{S}$.

3.3.3.2. Parámetros de entrada

Se utilizan los siguientes parámetros de entrada:

1. $M_0(p)$: Es cada máquina a la cual esta asignado cada proceso $p \in \mathcal{P}$.
2. $C_{m,r}$: Es la capacidad la capacidad de cada recurso $r \in \mathcal{R}$ en cada máquina $m \in \mathcal{M}$.
3. $SC_{m,r}$: Es la capacidad segura de carga de cada recurso $r \in \mathcal{R}$ en cada máquina $m \in \mathcal{M}$.
4. $R_{p,r}$: Indica la cantidad del recurso $r \in \mathcal{R}$ que requiere cada proceso $p \in \mathcal{P}$.
5. SM_s: Indica el $SpreadMin$ o el número mínimo de localizaciones $l \in \mathcal{L}$ distintas en las que deben ser distribuidos los procesos $p \in s$ para cada servicio $s \in \mathcal{S} \subset \mathcal{P}$.
6. Φ_r: Es el peso del costo de carga de cada recurso $r \in \mathcal{R}$.
7. Φ_b : Es el peso del costo de balance asociado a cada tripleta $b \in \mathcal{B}$. Cada una de ellas es de la forma $\{r_1^b, r_2^b, T^b\}$,$\forall r_1^b, r_2^b \in \mathcal{R}$; como fue definida en la Sección 2.3.
8. T^b: Es la proporción entre los recursos pertenecientes a una tripleta $b \in \mathcal{B}$, para el cálculo del costo de balance.
9. Φ_p: Es el costo real de movimiento de procesos, obtenido a partir del costo de movimiento de cada proceso $p \in \mathcal{P}$ multiplicado por el peso del costo de movimiento de procesos (este último es el mismo para todos los procesos), $\forall p \in \mathcal{P}$.
10. Φ_s: Es el peso del costo de movimiento de servicios, es el mismo para todos los servicios, $\forall s \in \mathcal{S}$.

11. $\Phi_{m_i m_j}$: Es el costo real de mover un proceso de una máquina m_i a otra m_j, se obtiene multiplicando cada costo por el peso del costo de movimiento de máquinas (este último es el mismo para todos los movimientos de máquinas), $\forall m_i, m_j \in \mathcal{M}$, con $\Phi_{m_i,m_i} = 0$.

3.3.3.3. Variables de decisión

Se tiene las siguientes variables de decisión:

1. $x_{m,p} = \begin{cases} 1 & \text{si el proceso } p \text{ está asignado a la máquina } m; \forall\, m \in \mathcal{M},\ p \in \mathcal{P} \\ 0 & \text{en caso contrario} \end{cases}$
2. $y_{l,s} = \begin{cases} 1 & \text{si al menos un proceso del servicio } s \text{ se ejecuta en} \\ & \text{la localización } l;\ l \in \mathcal{L},\ s \in \mathcal{S} \\ 0 & \text{en caso contrario} \end{cases}$
3. $z_{n,p} = \begin{cases} 1 & \text{si el proceso } p \text{ está en el vecindario } n;\ n \in \mathcal{N},\ p \in \mathcal{P} \\ 0 & \text{en caso contrario} \end{cases}$

3.3.3.4. Restricciones

Las restricciones que deben cumplirse son las siguientes:

1. **Restricciones de variables binarias:**

   a) Las siguientes restricciones establecen los valores de $x_{m,p}$: Cada proceso sólo puede ser asignado a una máquina.

   $$\sum_{m \in \mathcal{M}} x_{m,p} = 1 \quad \forall p \in \mathcal{P}$$

   b) Las siguientes restricciones establecen los valores de $y_{l,s}$: Para cada servicio $s \in \mathcal{S}$ y para cada localización $l \in \mathcal{L}$, la suma de todos los procesos de un servicio asignados a todas las máquinas de una localización, es siempre mayor o igual que la variable que indica si hay al menos un proceso de ese servicio en esa localización.

   $$\sum_{p \in \mathcal{S}} \sum_{m \in l} x_{m,p} \geq y_{l,s} \quad \forall l \in \mathcal{L},\ s \in \mathcal{S}$$

c) Las siguientes restricciones establecen los valores de $z_{n,p}$: Para cada vecindario $n \in \mathcal{N}$ y para cada proceso $p \in \mathcal{P}$, la variable $z_{n,p}$ es la suma de las variables $x_{m,p}$ (Esta suma siempre es 1 ya que el proceso p sólo puede ser asignado a una de las máquinas).

$$z_{n,p} = \sum_{m \in n} x_{m,p} \quad \forall n \in \mathcal{N},\, p \in \mathcal{P}$$

2. **Restricciones de Capacidad:** La suma de los requerimientos de todos los procesos para cada recurso de una máquina, no debe superar la capacidad de la máquina para ese recurso.

$$\sum_{p \in \mathcal{P}} x_{m,p} R_{p,r} \leq C_{m,r} \quad \forall m \in \mathcal{M},\, r \in \mathcal{R} \backslash \mathcal{TR}$$

3. **Restricciones de capacidad con recursos transitorios**: Aseguran que se cumplan las restricciones de capacidad tomando en cuenta los recursos transitorios, los cuales siempre añadirán un requerimiento de recurso $R_{p,r}$ en las máquinas que estaban asignados inicialmente.

$$\sum_{p \in \mathcal{P}, M_o(p) \neq m} x_{m,p} R_{p,r} + \sum_{p \in \mathcal{P}, M_0(p) = m} R_{p,r} \leq C_{m,r} \quad \forall r \in \mathcal{TR},\, m \in \mathcal{M}$$

4. **Restricciones de conflictos:** Cada proceso de un servicio debe ser asignado a una máquina distinta.

$$\sum_{p \in \mathcal{S}} x_{m,p} \leq 1 \quad \forall m \in \mathcal{M},\, s \in \mathcal{S}$$

5. **Restricciones de dispersión:** para cada servicio la suma de localizaciones distintas en que están sus procesos debe ser mayor que SM_s.

$$\sum_{l \in \mathcal{L}} y_{l,s} \geq SM_s \quad \forall s \in \mathcal{S}$$

6. **Restricciones de dependencia:** Asegura que se cumplan las restricciones de dependencia.

$$z_{n,p} \leq \sum_{k \in s_b} z_{n,k} \quad \forall n \in \mathcal{N},\, p \in \mathcal{P},\, s_b \in D^{s(p)}$$

3.3.3.5. Función Objetivo

Está compuesta por varias funciones de costos que representan los objetivos señalados en la Sección 2.3:

1. **Costo de carga total:** Es la suma de todos los costos de carga ponderados, esto es, para cada recurso $r \in \mathcal{R}$, sumar todas las cargas por sobre la capacidad segura de carga de todas las máquinas $m \in \mathcal{M}$, y ponderar dicha suma por el peso Φ_r asociado al recurso. Notar que si la capacidad segura es mayor al uso de la máquina no se incurre en ningún costo.

$$\sum_{r \in \mathcal{R}} \Phi_r \sum_{m \in \mathcal{M}} max \left\{ 0, \left(\sum_{p \in \mathcal{P}} x_{p,m} R_{p,r} \right) - SC_{m,r} \right\}$$

2. **Costo de balance total:** Es la suma de todos los costos de balance ponderados por su respectivo peso Φ_b. Para entender de mejor manera esta ecuación es preciso recordar que cada tripleta $b = \{r_1^b, r_2^b, T^b\}$ señala que por cada unidad disponible de r_1^b deberían haber T^b unidades disponibles de r_2^b, en una máquina cualquiera. En términos algebraicos, $C_{m,r_i^b} - \sum_{p \in \mathcal{P}} x_{p,m} R_{p,r_i^b}$ representa la cantidad del recurso r_i^b que está disponible en la máquina m. A partir de esto, para cada tripleta b, se restan los recursos disponibles de r_1^b y r_2^b, multiplicando dicha diferencia por T^b. Finalmente dicha suma es ponderada por Φ_b asociado a esa tripleta.

$$\sum_{b \in \mathcal{B}} \Phi_b \sum_{m \in \mathcal{M}} max \left\{ 0, T^b \left(C_{m,r_1^b} - \sum_{p \in \mathcal{P}} x_{p,m} R_{p,r_1^b} \right) - \left(C_{m,r_2^b} - \sum_{p \in \mathcal{P}} x_{p,m} R_{p,r_2^b} \right) \right\}$$

   Cabe destacar que para la tripleta $\{r_1^b, r_2^b, T^b\}$ es r_2^b el que debe tener T^bunidades disponibles por cada r_1^b disponible, **y no al revés**. Esto se ve reflejado en que si la diferencia entre $T^b r_1^b$ y lo disponible respecto a r_2^b resulta negativa, el valor no es tomado en cuenta, ya que significa una holgura de r_2^b y no se penaliza (de ahí que se utiliza la función max para calcular el máximo entre 0 y la diferencia).

3. **Costo de movimiento de procesos y máquinas total:** Es la suma de todos los costos de movimientos de procesos y movimientos entre máquinas. Los factores Φ_p y Φ_{m_i,m_j} no representan pesos sino costos reales, los cuales se calculan multiplicando el costo asociado al respectivo movimiento por el peso de mover procesos o servicios, según sea el caso. Cada vez que se encuentra un proceso que ha sido movido de su máquina inicial se suma Φ_p y $\Phi_{M_0(p),m}$, ya que se ha

incurrido en un costo de mover un proceso y otro al cambiarlo de una máquina específica a otra.

$$\sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M} \setminus M_0(p)} \Big(\Phi_p + \Phi_{M_0(p),m} \Big) x_{m,p}$$

4. **Costo de movimiento de servicios total:** Es la suma de los costos de movimiento de servicios ponderados por su peso Φ_s asociado. Para esto, se calcula en cada servicio el máximo de procesos movidos de su máquina inicial $M_0(p)$.

$$\Phi_s max \left(\sum_{p \in \mathcal{S}} \sum_{m \in \mathcal{M} \setminus M_0(p)} x_{m,p} \right)$$

5. **Función de costo Final:** Es la suma de todos los costos antes mencionados, en la cual se busca como objetivo su minimización.

3.4. Algoritmos para MRP

En esta sección se hace una revisión de los principales algoritmos propuestos en la literatura para resolver el MRP.

3.4.1. Variable Neighborhood Search

En el año 2012, Gavranovic et. al. [14] del equipo S41, que obtuvo el lugar 19 en la etapa de clasificación y primer lugar en la etapa final (clasificación general), presentan un método de resolución basado en búsquedas de vecindarios variables, en el cual combinan y examinan diferentes vecindarios buscando lograr diversificación e intensificación de manera eficiente al ordenar procesos y ejecutar la búsqueda en cada vecindario.

En primer lugar, obtienen un límite inferior de la función de costos tomando sólo en cuenta los costos de carga y de balance, para los cuales calculan los límites inferiores, quedando cada costo y la función objetivo final de la siguiente forma (utilizando la nomenclatura de la Sección 3.3.2.1):

- Límite inferior del costo de carga :

$$\begin{aligned}\sum_{m\in\mathcal{M}} max\,(0, U(m,r) - SC(m,r)) &= \sum_{m\in\mathcal{M}} (U(m,r) - SC(m,r)) \\ &= \sum_{m\in\mathcal{M}} U(m,r) \; - \sum_{m\in\mathcal{M}} SC(m,r)) \\ &= U(r) - SC(r) \\ &= R(r) - SC(r)\end{aligned}$$

Por lo tanto:

$$LB_{loadCost} = \sum_{r\in\mathcal{R}} weight_{loadCost}(r) \cdot (R(r) - SC(r))$$

Esto es, si se toma para cada recurso, la suma de los requerimientos en todos los procesos y la suma de las capacidades seguras en todas las máquinas, el exceso de requerimientos de recursos es un límite inferior del costo de carga.

- Límite inferior del costo de balance: Siendo $A(m,r) = C(m,r) - U(m,r)$ la capacidad disponible del recurso r en la máquina m. Se tiene que:

$$\begin{aligned}\sum_{m\in\mathcal{M}} max\,(0, tarjet \cdot A(m,r_1) - A(m,r_2)) &= \sum_{m\in\mathcal{M}} (tarjet \cdot A(m,r_1) - A(m,r_2)) \\ &= tarjet \cdot \sum_{m\in\mathcal{M}} A(m,r_1) - \sum_{m\in\mathcal{M}} A(m,r_2) \\ &= tarjet \cdot A(r_1) - A(r_2)\end{aligned}$$

Luego el límite inferior del costo total de balance es:

$$LB_{balanceCost} = \sum_{b\in\mathcal{B}} weight_{balanceCost}(b) \cdot (tarjet \cdot A(r_1) - A(r_2))$$

Dado que $A(r) = \sum_{m\in\mathcal{M}} C(m,r) - \sum_{p\in\mathcal{P}} R(p,r)$, Esto es similar al límite anterior, es decir, en vez de analizar cada máquina por separado y verificar la diferencia de espacio disponible para cada recurso, se considera el espacio disponible en todas las máquinas y los requerimientos de todos los procesos.

- Límite inferior para la función de costo total (no se toman en cuenta los demás costos): $LBC = LB_{loadCost} + LB_{balanceCost}$.

A continuación, en su método exploran cuatro tipos de vecindarios:

1. $Neighborhood_{Shift}$: Se genera a partir de aplicar el movimiento $shift$, que es tomar un proceso de una máquina en la solución actual y asignarlo a otra distinta.
2. $Neighborhood_{Swap}$: Se genera a partir de aplicar el movimiento $swap$, que es intercambiar todos los procesos de dos máquinas distintas en la solución actual.
3. $Neighborhood_{Chain}$: Se genera a partir de aplicar el movimiento $chain$, que es hacer $shift$ a l procesos tal que al proceso p_1 se le asigne la máquina de p_2, el de p_2 a p_3 y el de p_3 se le asigne la máquina de p_4 y así sucesivamente hasta p_l asignándole la máquina de p_1 en la solución actual.

   Este vecindario es explorado mediante una estructura de grafo dirigido, en el que cada nodo representa un proceso, y dos procesos unidos por una arista (p_i, p_j) están siempre asignados a máquinas distintas. A partir de esto, el peso de la arista es el cambio en la función objetivo producto de asignar p_i a la máquina de p_j y remover p_j de su máquina. Si se encuentra un ciclo en el grafo cuyo valor completo sea negativo, significa que se ha encontrado un movimiento $chain$ de tamaño l. Finalmente destacan que por motivos de rendimiento sólo puede mantenerse el grafo con 30-100 procesos, manteniendo además sólo las aristas negativas.
4. $Neighborhood_{BPR}$: Se genera a partir de aplicar el movimiento BPR, que es hacer $shift$ de un proceso y asignarlo a una máquina y quitar de esta varios procesos a la vez, reasignándolos en distintas máquinas en la solución actual.

   Este vecindario destaca de los demás en que es el único que permite mover procesos pesados (con alta exigencia de recursos) ya que cada uno de los otros, al quitar sólo un proceso de cada máquina, no es capaz de hacer el espacio suficiente como para que procesos pesados puedan asentarse.

En base a lo anterior, destacan que reasignar procesos pequeños mientras se busca mejorar la función objetivo tiene como consecuencia que en cada máquina se estará cada vez más cerca de superar la capacidad segura de carga, con lo que mover procesos grandes será difícil. Por esta razón, aplican dos medidas:

- Explorar los vecindarios en el orden $Neighborhood_{BPR} \rightarrow Neighborhood_{Shift} \rightarrow Neighborhood_{Swap} \rightarrow Neighborhood_{Chain}$
- Ordenar la lista de procesos por tamaño y comenzar tomando en cuenta un proceso para generar los vecindarios, aumentando la cantidad a medida que transcurren las iteraciones.

Las dos medidas anteriores ayudan a mover los procesos más pesados primero, mientras que la segunda permite además controlar el tamaño de los vecindarios que generan y evitar perder tiempo generando vecindarios demasiado grandes. En el Algoritmo 3.1, puede verse el pseudocódigo del funcionamiento de su técnica.

Algoritmo 3.1 Algoritmo General LS

```
1  Ordenar la lista de procesos por la suma de sus requerimientos
2  N_ITERACIONES=5
3  FOR i=1 HASTA N_ITERACIONES DO
4  BEGIN
5      FOR r=1 HASTA NMR_RECURSOS_USAR DO
6      BEGIN
7          LocalSearch()
8          Aumentar peso del recurso Recursos[r]
9          LocalSearch() (*con función objetivo modificada*)
10     END
11     Incrementar número de procesos a considerar
12 END
```

Este algoritmo, trata de reemplazar la asignación anterior por una que tenga menor costo de función objetivo, en donde en cada iteración, el método LocalSearch() explora cada vecindario sólo una vez teniendo como criterios de término un tiempo máximo o hasta lograr una mínima mejora. Además, utilizan una técnica de diversificación de búsqueda basada en un método de agitación, el cual cambia el peso del costo de carga de los recursos en el cálculo de la función objetivo, cambiando la función objetivo con la finalidad de escapar de óptimos locales. La selección de los recursos, para aumentar su costo, se hace en base a la distancia de su costo de carga respecto de su límite inferior asociado.

El algoritmo fue ejecutado en una máquina Intel core I7 con un procesador de 2.66GHz, 8MB de caché y 6 GB de RAM, teniendo su algoritmo un paralelismo de dos núcleos. Los valores fueron obtenidos a partir de correr el programa con 100 semillas aleatorias distintas, con un tope de tiempo de 300 segundos en los grupos de instancias A y B. Los resultados junto a los mejores resultados obtenidos en la competencia ROADEF para cada instancia se muestran en la Tabla 3.2.

En la Tabla 3.2 los mejores resultados de todos los algoritmos de la competencia se muestran en la columna “Mejor ROADEF”. Puede verse que en todas las instancias mostradas el algoritmo pudo obtener valores muy cercanos o mejores a los mejores valores de ROADEF. Además, la media de los valores muestra que a lo largo de todas las 100 ejecuciones el algoritmo tendió siempre a obtener valores bajos. Por otro lado, también se aprecian en negrita aquellos valores en los cuales el algoritmo encontró valores iguales o mejores que los mejores encontrados en la competencia, cuyos puntajes correspondientes son negativos. Cabe destacar aquí que los resultados más significativos fueron los encontrados en las instancias del grupo A, en donde en la competencia el equipo S41 obtuvo una suma de 51,15. Esta diferencia puede explicarse tomando en cuenta que en la competencia los algoritmos fueron ejecutados sólo una vez. En el caso de las instancias B, sus resultados promedio fueron muy similares a los de

Instancia	Mejor ROADEF	Media	Mejor	Puntaje (media)	Puntaje(mejor)
A1_1	44.306.501	**44.306.501**	**44.306.501**	**0,000**	**0,000**
A1_2	777.532.896	778.265.189	777.536.907	0,069	0,000
A1_3	583.005.717	583.006.320	583.005.818	0,000	0,000
A1_4	252.728.589	260.903.327	**251.524.763**	1,292	**-0,190**
A1_5	727.578.309	727.578.312	727.578.310	0,000	0,000
A2_1	198	333	199	0,000	0,000
A2_2	816.523.983	**748.528.290**	**720.671.548**	**-3,623**	**-5,107**
A2_3	1.306.868.761	**1.218.013.414**	**1.190.713.414**	**-3,910**	**-5,111**
A2_4	1.681.353.943	**1.680.740.350**	**1.680.615.425**	**-0,019**	**-0,023**
A2_5	336.170.182	**317.804.454**	**309.714.522**	**-2,333**	**-3,360**
Suma(A)				**-8,523**	**-13,792**
B_1	3.339.186.879	3.345.152.832	**3.307.124.603**	0,078	**-0,419**
B_2	1.015.553.800	1.015.561.513	**1.015.517.386**	0,000	**-0,001**
B_3	156.835.787	157.737.166	156.978.411	0,014	0,002
B_4	4.677.823.040	4.677.981.438	4.677.961.007	0,002	0,001
B_5	923.092.380	923.905.512	923.610.156	0,007	0,004
B_6	9.525.857.752	9.525.934.654	9.525.900.218	0,001	0,000
B_7	14.835.149.752	14.835.328.102	**14.835.031.813**	0,000	0,000
B_8	1.214.458.817	1.214.510.885	**1.214.416.705**	0,000	0,000
B_9	15.885.486.698	15.885.693.227	15.885.548.612	0,001	0,000
B_10	18.048.515.118	18.048.711.483	**18.048.499.616**	0,000	0,000
Suma(B)				0,103	**-0,412**

Tabla 3.2: Resultados de [14] para 100 ejecuciones del algoritmo comparados con ROADEF.

la competencia, en donde obtuvieron un puntaje de 0,43. Por otro lado, los mejores puntajes obtenidos presentan un valor que en suma es negativo, lo cual es debido casi exclusivamente por instancia B_1, en donde se encontraron mejores resultados que los mejores de ROADEF.

3.4.2. Large Neighborhood Search (LNS)

El mismo año 2012 de Deepak Mehta et. al. [24] del equipo S38, que obtuvo el tercer lugar en la etapa de clasificación y segundo en la etapa final (clasificación general), presentan dos modelos para el MRP, uno basado en Programación Lineal Entera Mixta (MIP) y otro en Satisfacción de Restricciones con restricciones globales (CP). En el caso de MIP, este es resuelto mediante Large Neighborhood Search (LNS) usando el

software CPLEX[2] con los algoritmos implementados en JAVA. Por otro lado, en el caso de CP el problema es resuelto también con LNS con un programa en C, buscando más flexibilidad y beneficiarse de las restricciones globales añadidas. Es destacable que esta última alternativa logró tener muy buenos resultados, estando muy cerca de obtener el primer lugar de la competencia. En líneas generales, el algoritmo LNS utilizado para resolver MRP para ambos casos está dividido en las partes que se muestran en la Figura 3.2.

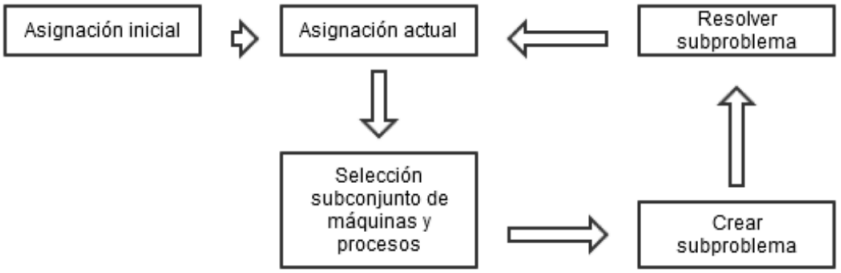


Figura 3.2: Funcionamiento general del algoritmo LNS.

Partiendo de una asignación inicial, la cual es establecida como la solución inicial dada en el problema base, el algoritmo procede a resolver el problema de manera iterativa. En cada iteración, se selecciona un subconjunto de máquinas de las cuales se toma un conjunto de procesos a ser reasignados, siendo sólo estos los que serán considerados en el subproblema. A continuación se resuelve el subproblema dando un tiempo límite y se conserva la mejor solución obtenida como la solución actual.

En el caso de LNS para CP específicamente, se agregan más parámetros y mayor complejidad, siendo necesario explicar su estrategia en tres partes:

1. Selección de cada subproblema: En esta parte se explica cómo se seleccionan las máquinas y los procesos que conformarán el subproblema, de forma que en cada iteración se tendrán k_m máquinas, de las cuales se seleccionarán k_p procesos en cada iteración. Como heurística se utiliza inicialmente $k_m = 1$, incrementándose mientras la búsqueda aumenta y reiniciándose a 1 cuando excede las 10 máquinas. Además, las máquinas son ordenadas a intervalos regulares de tiempo, de acuerdo a los costos en los que incurre cada una, sirviendo este ordenamiento para seleccionar una máquina, mientras que las restantes $k_m - 1$ son seleccionadas al azar.

   Por otro lado, se analiza que si k_p resulta ser un número muy grande, podría ocurrir que luego de resolver el subproblema generado algunos procesos vuelvan a sus máquinas iniciales, es por esto que se establece que k_p debería ser menor o igual que la mitad del número de procesos por máquina, teniéndose que $k_p \leq$

[2]http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

$\frac{1}{2}\frac{\#Procesos}{\#Máquinas}$. Se añade cota adicional a k_p, para los casos en que el promedio de procesos por máquina sea muy grande, estableciéndose además que $k_p \leq 10$. También se establece una cota superior para el producto entre k_m y k_p, teniéndose que $k_m \cdot k_p \leq 40$, ofreciendo más variabilidad para el valor de k_p a medida que cambia k_m en cada iteración.

2. Creación de cada subproblema: Aquí se evalúa la forma más eficiente en términos de memoria y procesamiento de cómo crear cada subproblema. Dado que el tamaño de las instancias puede ser muy grande (hasta 50.000 procesos y 5.000 máquinas) y el tiempo para resolver el MRP es restringido (300 segundos en la competencia), la selección eficiente de una forma de crear y resolver los subproblemas, es un tema importante. Aquí, una de las mayores preocupaciones es la gestión del dominio de las variables a medida que se crean los subproblemas, al respecto se evalúan tres formas de hacerlo:

   a) Crear todo el problema en memoria: Esto involucraría además reinicializar y recomputar los dominios de todas las variables en cada iteración. Se descarta esta opción debido a que el gran tamaño de las instancias podría generar que el costo en memoria necesario fuese demasiado.

   b) Crear cada subproblema en memoria: Esto involucraría crear cada subproblema en memoria e inicializar sus variables y dominios en cada iteración. A simple vista parece una opción viable, pero teniéndose en cuenta que cada instancia del MRP es generalmente grande, y que en cada iteración sólo unos pocos procesos son seleccionados (el tamaño del subproblema es considerablemente más pequeño que el tamaño del problema completo), serán necesarias muchas iteraciones para dar la oportunidad a cada proceso de ser reasignado. Lo anterior tiene como consecuencia que el inicializar y recomputar los dominios en cada subproblema traerá consigo un considerable costo en tiempo computacional, es por esto que esta opción también es descartada.

   c) Crear un subproblema a partir de una solución ya existente: Para esto, la creación de un subproblema es vista como la desasignación de un conjunto de procesos de un conjunto de máquinas. Cada vez que un proceso es extraído de una máquina se actualizan los dominios involucrados. Asimismo, cada vez que se asigne un proceso a una máquina también se actualizan los dominios involucrados, la simplicidad y precisión de estas operaciones tiene como consecuencia que se actualiza sólo lo necesario. Esta alternativa es la propuesta por los investigadores **y es la seleccionada por presentar la mejor eficiencia**.

3. Reoptimización de cada subproblema: Para resolver cada problema utilizan búsqueda sistemática con un criterio de término en base al número de fallas. Luego de esto, definen tres componentes para la búsqueda:

a) Heurísticas de ordenamiento de variables: Está basada en información mantenida sobre cada proceso respecto a tres puntos: máximo incremento de la función objetivo cuando se asigna la mejor máquina al proceso p, suma de todos los recursos de un proceso ponderados por su peso y número de máquinas disponibles para un proceso.

b) Heurísticas de ordenamiento de valores: Es usada para seleccionar una máquina para un proceso dado y está basada en el mínimo costo.

c) Reglas de filtrado: En cada nodo de la búsqueda se hace filtrado vía propagación de restricciones. Además, mientras se realiza la resolución de cada subproblema se aplica filtrado basado en el uso y el costo. Para esto, de manera similar a lo realizado en [14] (ver Sección 3.4.1), definen un límite inferior para el costo de carga y el costo de balance, esta vez asociados sólo a la asignación de un conjunto de procesos $\mathcal{Q}$. Luego los límites quedan de la siguiente manera (utilizando la nomenclatura de la Sección 3.3.2.1):

- Límite inferior para el costo de carga de asignar todos los procesos de Q: Corresponde a sumar la demanda total para cada recurso y compararla con la capacidad segura de carga total para ese recurso en todas las máquinas, ponderando por su peso de costo de carga asociado:

$$LB_{loadCost}(Q) = \sum_{r \in \mathcal{R}} \left(\sum_{p \in \mathcal{Q}} R(p,r) - \sum_{m \in \mathcal{M}} max(0, SC(m,r) - U(m,r)) \right) \cdot weight_{loadCost}$$

- Límite inferior para el costo de balance de asignar todos los procesos de $\mathcal{Q}$: De forma similar al punto anterior se obtiene este límite interior considerando las diferencias de los requerimientos totales, ponderando por su costo de balance asociado.

$$LB_{balanceCost}(Q) = \sum_{b \in \mathcal{B}} \left(-tarjet \cdot \sum_{p \in \mathcal{Q}} R(p,r_1) - \sum_{p \in \mathcal{Q}} R(p,r_2) \right) \cdot weight_{BalanceCost}$$

Finalmente, el límite inferior de la función de costo asociada al subproblema relacionado con el conjunto de procesos $\mathcal{Q}$ es:

$$LW(\mathcal{Q}) = LB_{loadCost}(\mathcal{Q}) + LB_{loadCost}(\mathcal{Q})$$

Los algoritmos fueron ejecutados sobre los conjuntos de instancias de A y B de ROADEF durante 300 segundos, usando un core cada vez pues su algoritmo no tiene paralelismo. Los mejores valores de función de costos del algoritmo LNS para ambos modelos MIP y CP en los grupos de instancias A y B se muestran en la Tabla 3.3 comparados con los mejores resultados de ROADEF. Sólo se muestran los puntajes asociados al modelo CP, que fue el que obtuvo los mejores resultados.

Instancia	Mejor ROADEF	MIP mejor	CP mejor	Puntaje(CP)
A1_1	44.306.501	**44.306.501**	**44.306.501**	**0,000**
A1_2	777.532.896	792.813.766	778.654.204	0,106
A1_3	583.005.717	583.006.527	583.005.829	0,000
A1_4	252.728.589	258.135.474	**251.189.168**	**-0,243**
A1_5	727.578.309	727.578.310	727.578.311	0,000
A2_1	198	273	**196**	**0,000**
A2_2	816.523.983	836.063.347	**803.092.387**	**-0,716**
A2_3	1.306.868.761	1.393.648.719	**1.302.235.463**	**-0,204**
A2_4	1.681.353.943	1.725.846.815	1.683.530.845	0,068
A2_5	336.170.182	359.546.818	**331.901.091**	**-0,542**
Suma(A)				**-1,532**
B_1	3.339.186.879	-	**3.337.329.571**	**-0,024**
B_2	1.015.553.800	-	1.022.043.596	0,125
B_3	156.835.787	-	157.273.705	0,007
B_4	4.677.823.040	-	**4.677.817.475**	**0,000**
B_5	923.092.380	-	923.335.604	0,002
B_6	9.525.857.752	-	9.525.867.169	0,000
B_7	14.835.149.752	-	14.838.521.000	0,009
B_8	1.214.458.817	-	1.214.524.845	0,000
B_9	15.885.486.698	-	15.885.734.072	0,001
B_10	18.048.515.118	-	18.049.556.324	0,002
Suma(B)				0,123

Tabla 3.3: Resultados de [24] del algoritmo LNS con modelos MIP y CP comparados con ROADEF.

En negritas se muestran los resultados en los cuales alguno de los algoritmos obtuvo un valor de la función objetivo igual o inferior al mejor de ROADEF. En base a esto se aprecia, que si bien el algoritmo con modelo MIP pudo obtener resultados positivos en las instancias del conjunto A, sólo pudo obtener resultados mejores que ROADEF en dos oportunidades y sólo pudo superar al algoritmo con el modelo CP en una oportunidad (y una sola unidad). Por otro lado CP obtuvo resultados iguales o mejores que ROADEF en seis de las 10 instancias del grupo A, sumando un puntaje de -1,532. En el conjunto de instancias B no se obtuvieron resultados para MIP mientras que CP si bien pudo superar a ROADEF en sólo dos oportunidades, en las otras instancias obtuvo resultados cuyo puntaje asociado fue muy cercano a 0, lo cual da cuenta de su escalabilidad. De acuerdo a los autores, Large Neighborhood Search fue escalable al aplicarse basado en el modelo CP debido a tres factores principales: primero, seleccionar un problema desde sólo un conjunto más pequeño de máquinas funciona mucho mejor que seleccionar los procesos desde el conjunto total. Segundo, la aplicación de

la estrategia de actualización de dominios eficiente permitió no incurrir en costos adicionales, esto permite incrementar la cantidad de iteraciones que se realizan en los 300 segundos de ejecución y, en consecuencia, mejora la exploración del espacio de búsqueda. Finalmente, usar chequeos de restricciones baratos y heurísticas de selección de máquinas que incurren en el menor costo para un proceso dado tiene como resultado una exploración eficiente del algoritmo.

3.4.3. Simulated Annealing

Como último trabajo encontrado del año 2012 se muestra lo presentado por Gabriel Portal [29], quien estuvo en el equipo S23 y obtuvo el cuarto lugar en la etapa de clasificación y octavo en la etapa final (clasificación general). En su trabajo muestra un análisis del MRP desde distintas perspectivas, partiendo por un modelamiento con el software CPLEX en Programación Lineal Entera (ILP), luego aplicando búsqueda local aleatorizada y finalmente generando un algoritmo basado en Simulated Annealing, que fue el algoritmo que lo llevó a obtener el cuarto lugar. Del modelamiento y resolución del problema con ILP obtuvo resultados similares a los encontrados por [24], en este caso se pudieron resolver tres instancias con resultados excelentes, con límite de tiempo de 1 hora de ejecución (instancias del grupo A). Asimismo, al aumentar el límite de tiempo a 10 horas se encontró que en sólo dos instancias pudieron mejorarse los resultados a niveles comparables con los mejores de ROADEF. Además de lo anterior, el modelo ILP con CPLEX no pudo resolver ninguna de las instancias del conjunto B, reafirmando la intratabilidad del problema con algoritmos del tipo branch and bound.

Los autores además calculan límites inferiores para los costos de carga y de balance exactamente de la misma forma que lo hace Gavranovic en [14] (ver Sección 3.4.1). Además mediante su modelo ILP obtiene límites inferiores usando CPLEX y los compara para ver la calidad de los límites inferiores calculados mediante la fórmula desarrollada. La conclusión a la que llega es que si bien los límites del modelo ILP tienen mayor precisión (ya que toman en cuenta más costos para su cálculo), en la mayoría de las instancias ambos límites son cercanos, comprobando que los límites calculados con la fórmula, además de ser poco costosos, se presentan como una alternativa bastante cercana a los valores óptimos de cada instancia.

El autor enfrenta además el problema desde la perspectiva de la Búsqueda Local, aplicando primero Búsqueda Local Aleatorizada y Simulated Annealing, la primera sin muy buenos resultados mientras que la segunda, mediante un esquema de exploración de vecindarios combinados, logra obtener buenos resultados.

El algoritmo presenta un movimiento llamado sa-move que está compuesto internamente de dos tipos de movimientos, move y swap, los cuales se utilizan para cambiar un proceso de máquina e intercambiar las máquinas de dos procesos, respectivamente. Para esto, el algoritmo utiliza una probabilidad de elegir un movimiento u otro para

generar el vecindario en una iteración cualquiera, de esta manera, cada movimiento ofrece ventajas que el otro no tiene y el complemento de ambos permite explorar y explotar de mejor manera el espacio de búsqueda.

Para escapar de óptimos locales, como es habitual en algoritmos basados en Simulated Annealing, el algoritmo utiliza la temperatura, que permite aceptar movimientos que empeoran la función objetivo, la cual comienza en un valor alto y disminuye geométricamente en base a la cantidad de iteraciones realizada. También se aumenta la temperatura en caso de que el algoritmo detecte que no se ha mejorado en una cierta cantidad de iteraciones.

En el Algoritmo 3.2, se detalla la sección principal del programa. Entre las líneas 18 y 30 se aplica el movimiento y se actualiza la mejor solución, en caso de requerirse. Además, puede verse cómo entre estas líneas se cumplen n ciclos en los cuales la temperatura se mantiene constante, luego de esto se enfría multiplicándola por el factor r. Tal como señala el autor, esto tiene como consecuencia que la temperatura se enfríe geométricamente siguiendo la ecuación $t_k = t_0 r^k$, donde r es el radio de la progresión geométrica y k la $k-ésima$ vez que se ejecuta el FOR. En caso de que la mejor solución no haya sido actualizada en $20n$ iteraciones y que la tasa de movimientos aceptados sea menor que el $0,1\,\%$, la solución se considera "congelada" y es recalentada aumentando su temperatura a $t_0/100$, ayudando al algoritmo a escapar de óptimos locales. La estructura del movimiento sa_move se presenta en el Algoritmo 3.3.

El vecindario *move* es escogido manualmente con una probabilidad p y el movimiento *swap* es escogido con probabilidad $1 - p$, donde p es igual a $0,7$ de acuerdo a los mejores resultados de los experimentos. En el caso del movimiento *move*, se selecciona un proceso al azar, de una máquina k también al azar. Luego, como candidatos de destino del proceso seleccionado se consideran las $(k+i)\,\%|M|$ máquinas, con $0 \leq i \leq c$, siendo c una constante definida. Posteriormente, la primera asignación exitosa en las máquinas del conjunto definido es la elegida. En caso de no encontrarse ninguna, el proceso falla.

En el caso del movimiento *swap* se usa una idea similar, en donde el movimiento escogido es el primero en donde un proceso seleccionado al azar pueda intercambiarse con otro dentro de una secuencia de c procesos. La constante c fue establecida en 100 en los experimentos realizados.

El programa presentado anteriormente fue ejecutado con dos procesadores, en los cuales se eligió ejecutar dos hilos distintos y retornar la mejor solución. Para esto, la elección de los parámetros fue escogida mediante la ejecución del programa en las instancias consideradas como las más difíciles para el algoritmo, las cuales fueron: A1-4, A2-2, A2-3, A2-5, B-1 y B-3. Los valores de los parámetros probados fueron $n \in \{10^4, 10^5, 10^6\}$, $r \in \{0,91; 0,95; 0,97\}$ y $t_0 = \{10^7, 10^8, 10^9\}$, ejecutándose cada combinación de parámetros 5 veces para cada instancia. Los mejores parámetros encontrados fueron $n = 10^5$, $r = 0,97$ y $t_0 = 10^8$. En el otro thread se consideró que los

Algoritmo 3.2 Algoritmo Simulated Annealing

```
1  Entrada: Solución inicial sol
2  Entrada: Temperatura inicial t_0
3  Entrada: Tiempo límite para finalización del algoritmo TIME_LIMIT
4  Entrada: Número de iteraciones con igual temperatura n
5  Entrada: Factor de decrecimiento de temperatura r
6  Entrada: Mínima tasa de movimientos aceptados MIN_PERCENT
7  Entrada: Número de iteraciones máximo sin actualizar la mejor solución
       antes de recalentar FREEZE_LIM
8
9  SALIDA: Solución Final
10
11 mejorSolucion ← sol
12 T ← t_0
13 freeze ← 0
14 WHILE time() ≤ TIME_LIMIT DO
15 BEGIN
16      it ← 0
17      ac ← 0
18      FOR i = 1 TO n DO
19      BEGIN
20          IF sa_move(sol,T) THEN
21          BEGIN
22              ac ← ac+1
23          END
24          it ← it + 1
25          IF sol.cost < bestSolution.cost THEN
26          BEGIN
27              bestSolution ← sol
28              freeze ← 0
29          END
30      END
31      T ← r · T
32      IF ac/it < MIN_PERCENT THEN
33      BEGIN
34          freeze ← freeze +1
35          IF freeze = FREEZE_LIM THEN
36          BEGIN
37              freeze ← 0
38              T ← t_0/100
39          END
40      END
41 END
```

parámetros anteriores, al decrementar muy lentamente la temperatura, podrían hacer que el algoritmo se ejecutara muy lento para algunas instancias (diferentes de las mencionadas anteriormente) y se eligieron los parámetros $n = 70000$, $r = 0,95$ y $t_0 = 10^8$.

Algoritmo 3.3 Algoritmo sa-move

```
1  Entrada: Solución actual sol
2  Entrada: Temperatura actual T
3  Entrada: Probabilidad p de elegir cada vecindario
4
5  SALIDA: booleano que indica si el movimiento fue aplicado
6
7  IF random() < p THEN
8  BEGIN
9       move = getMoveNeighbor(sol)
10      delta = getCost(sol,move)
11      IF delta ≤ 0 OR random ≥ e^(-delta/T) THEN
12      BEGIN
13          makeMove(sol,move)
14          RETURN TRUE
15      ELSE
16          RETURN FALSE
17      END
18 ELSE
19      move = getSwapNeighbor(sol)
20      delta = getCost(sol,move)
21      IF delta ≤ 0 OR random ≥ e^(-delta/T) THEN
22      BEGIN
23          makeMove(sol,move)
24          RETURN TRUE
25      ELSE
26          RETURN FALSE
27      END
28 END
```

Los mejores resultados de la ejecución del algoritmo en los grupos de instancias A y B se presentan en la Tabla 3.4(utilizando dos núcleos).

Instancia	Mejor ROADEF	Mejor resultado SA	Puntaje
A1_1	44.306.501	44.306.935	0,001
A1_2	777.532.896	777.533.311	0,000
A1_3	583.005.717	583.009.439	0,001
A1_4	252.728.589	260.693.258	1,259
A1_5	727.578.309	727.578.311	0,000
A2_1	198	222	0,000
A2_2	816.523.983	877.905.951	3,271
A2_3	1.306.868.761	1.380.612.398	3,245
A2_4	1.681.353.943	**1.680.587.608**	**-0,024**
A2_5	336.170.182	**310.243.809**	**-3,293**
Suma(A)			4,460
B_1	3.339.186.879	3.455.971.935	1,528
B_2	1.015.553.800	1.015.763.028	0,004
B_3	156.835.787	215.060.097	0,919
B_4	4.677.823.040	4.677.985.338	0,002
B_5	923.092.380	923.299.310	0,002
B_6	9.525.857.752	9.525.861.951	0,000
B_7	14.835.149.752	14.836.763.304	0,004
B_8	1.214.458.817	1.214.563.084	0,001
B_9	15.885.486.698	15.886.083.835	0,003
B_10	18.048.515.118	18.049.089.128	0,001
			2,463

Tabla 3.4: Resultados del algoritmo Simulated Annealing.

En los resultados anteriores ambos movimientos se consideran en cada iteración. Respecto a esto, el autor también muestra resultados para la aplicación del movimiento *move* solamente, el cual en general tiene peores resultados por quedarse atrapado en mínimos locales. En general puede verse que los resultados fueron muy cercanos a los mejores de ROADEF, llegando a superarlos en las en algunas de las instancias A. Por otro ladose aprecia que el alguritmo sólo instancias específicas tiene tiene problemas, las cuales representan la mayor parte de su puntaje (A1_4, A2_2, A2_3, B_1 y B_3).

3.4.4. Iterated Local Search with Multi-start and shaking moves.

En 2013, Renaud Masson et. al. [23] del grupo J6, que obtuvo el lugar número 11 en etapa de calificación y número 19 en la etapa final (clasificación general), presentan una heurística basada en búsqueda local para problemas del tipo bin-packing, en donde también se considera el MRP con uno de ellos (para más detalle, ver el punto 3 de

la sección Sección 2.4), logrando así obtener un algoritmo versátil que además de ser robusto encuentra buenas soluciones. Su heurística utiliza un sistema multi-start que le ayuda a salir de óptimos locales. Está basada en el framework de búsqueda local iterada presentado en [21], el cual aplica iterativamente un procedimiento de búsqueda local para mejorar las soluciones junto con movimientos de agitación (shaking moves) para escapar de óptimos locales. Además, con el fin de mejorar los tiempos de cómputo, dichos movimientos de agitación junto con los restarts son realizados de forma dinámica cuando se estima que la mejora es muy pequeña en comparación con el costo total de la solución.

Además de lo anterior, presentan un modelo de Programación Lineal Entera donde se trabaja con el problema relajado (para más detalle, ver el modelo en la sección Sección 3.3.3). Por otro lado, de forma similar a [14, 24, 29] estiman lower bounds para la función de costos en base a lower bounds de las funciones que calculan los costos de carga y balance, lo cual les permite en conjunto con su modelo, relajar el problema original y computar su función objetivo en tiempo relativo al tamaño del problema.

Su algoritmo consta principalmente de la combinación de tres componentes: búsqueda local, movimientos de agitación especialmente creados para el problema y movimientos de restart especializados. El pseudocódigo de la estructura general de la búsqueda local se presenta en el Algoritmo 3.4.

Puede verse entre las líneas 10 y 15 que la búsqueda local se aplica con igual probabilidad a la mejor solución o a la solución actual. En el primer caso, con la intención de intensificar la búsqueda en un área cercana de buenas soluciones, mientras que la segunda con la finalidad de encontrar soluciones de buena calidad en otra localidad. Entre las líneas 16 y 19 se guarda la mejor solución en caso de que la búsqueda local haya encontrado un valor de mejor calidad. Entre las líneas 20 y 25 se selecciona un movimiento de sacudida. Puede verse que esto último se hace utilizando el contador de rondas It_{SHAK}, por lo cual se irá seleccionando el movimiento $Shaking1$ y $Shaking2$ alternadamente. Se destaca que cada movimiento está diseñado para modificar la solución actual, de tal manera que los cambios no puedan ser revertidos directamente aplicando un sólo movimiento de búsqueda local. Cada movimiento se describe a continuación:

- $Shaking1$ o Home realocate: Selecciona al azar ξ procesos, para los cuales su máquina asignada no sea su máquina de origen, y los devuelve a sus máquinas de origen.
- $Shaking2$ o K-Swap shaking: Selecciona ξ veces un par de máquinas al azar y hace swap al azar entre 3, 4 ó 5 procesos.

En conjunto, la búsqueda local y los movimientos de sacudida son aplicados hasta que la mejora en la mejor solución en las últimas 5 rondas es mayor que un parámetro $\gamma_{RESTART}$, que denota cuán significativa es dicha mejora. En caso de que la mejora no sea lo suficientemente significativa, se hace restart y se vuelve a comenzar el bucle

Algoritmo 3.4 Algoritmo general de Búsqueda Local Multistart para MRP de [23]

```
1  S_BESTALL ← S_INIT
2  WHILE time()<T_max DO
3  BEGIN
4      /*En Cada restart se comienza  desde la solución inicial */
5      S_CURR ← S_INIT
6      S_BEST ← S_INIT
7      It_SHAK = 0 /* Contador de las rondas de agitación */
8      WHILE mejora relativa de las últimas 5 rondas de agitación respecto a
           la mejor solución > γ_RESTART DO
9      BEGIN
10         IF (RANDOM(0,1) <0,5)
11         BEGIN
12             S_CURR ← LocalSearch(S_BEST)
13         ELSE
14             S_CURR ← LocalSearch(S_CURR)
15         END
16         IF Cost(S_CURR) < Cost(S_BEST) THEN
17         BEGIN
18             S_BEST ← S_CURR
19         END
20         IF It_SHAK mod 2 = 0
21         BEGIN
22             S_CURR ← Shaking1(S_CURR)
23         ELSE
24             S_CURR ← Shaking2(S_CURR)
25         END
26     END
27     IF Cost(S_BEST < Cost(S_BESTALL) THEN
28     BEGIN
29         S_BESTALL ← S_BEST
30     END
31 END
32 RETURN S_BESTALL
```

desde la solución inicial. El algoritmo completo termina al alcanzar el tiempo máximo de cómputo. A partir de lo anterior, puede verse que el procedimiento de búsqueda local utiliza, al igual que en [14, 29], vecindarios obtenidos a partir de movimientos de swapping de procesos (intercambiando uno o dos procesos) y movimientos de un proceso de una máquina a otra (designados en este caso con los nombres swap y relocate, respectivamente), sólo que en este caso la búsqueda local utiliza ambos vecindarios, pero sólo sobre un subconjunto de las máquinas. El pseudocódigo del algoritmo de búsqueda local puede verse en el Algoritmo 3.5.

Algoritmo 3.5 Algoritmo general del procedimiento Búsqueda Local para mejorar la solución ([23]).

```
1  WHILE La mejora relativa de la solución de los últimos 5 loops es mayor
       que γ_SHAKING DO
2  BEGIN
3       Sort(MachineList)
4       fractionMachines ← |M| · [0,1 + 0,9 · random(0,1)^2]
5       FOR ALL i ∈ {1,...,fractionMachineList} DO
6       BEGIN
7            m ← MachineList[i]
8            exploreRealocateMoves(m)
9            exploreSwapMoves(m)
10           S_CURR ← performBestMove(S_CURR)
11      END
12 END
13 RETURN S_CURR
```

Se aprecia que al igual que en [14] también se efectúa ordenamiento sobre los datos, en este caso las máquinas (las cuales son ordenadas en cada loop decrecientemente en base a su costo), teniendo como función de costos la relajación de la función objetivo mencionada anteriormente que sólo considera los costos de carga y de balance. Por otro lado, al igual que en [14] donde sólo se considera una porción de los procesos para sus movimientos y que [24], que sólo considera una porción de máquinas y procesos para realizar sus subproblemas, este algoritmo considera una fracción de las máquinas, obtenidas seleccionando las primeras $|M| \cdot [0,1 + 0,9 \cdot random(0,1)^2]$ de la lista anterior en cada loop, esto con el fin de realizar una búsqueda más eficiente y no perder tiempo explorando vecindarios que son excesivamente grandes en las instancias de gran tamaño. En esta misma línea cada vecindario no es explorado totalmente, sino que sólo se evalúan en promedio $\phi/3$ movimientos para una máquina específica, siendo ϕ un parámetro del algoritmo. A continuación, por cada una de las máquinas del subconjunto obtenido se explora su vecindario de movimientos de relocate y swap, guardando en la solución actual el mejor movimiento de ambos vecindarios en caso de existir alguno. Finalmente, el procedimiento de Búsqueda Local continúa hasta que se detecta que la mejora total de la solución en las últimas 5 iteraciones es mayor que el umbral $\gamma_{SHAKING}$.

El algoritmo fue programado en C++ y ejecutado sobre una máquina Opteron de 2.4 GHz con 4 GB de RAM y sistema operativo OpenSuse 11.1. Para medir la calidad de la solución se definió una función de ganancia de costo $G(X) = S_{INIT} - S_{FINAL}(X)$. Por otro lado, se definió un límite superior para la ganancia de costo G_{MAX} como la diferencia entre la solución inicial y el límite inferior de la función de costos, esto es $G_{MAX} = S_{INIT} - LB$. Considerando lo anterior, la función de puntaje utilizada fue $C(X) = 100 \cdot \frac{G_{MAX} - G(X)}{S_{INIT}} = 100 \cdot \frac{S_{FINAL} - LB}{S_{INIT}}$. Así, un puntaje de 1 significa que no

es posible, en el mejor de los casos, obtener una mejora adicional de más del 1 % de la solución inicial, esta idea de puntuación es equivalente a usar el puntaje ROADEF pero sustituyendo el mejor de la competencia con el Límite Inferior.

Cada instancia fue ejecutada 40 veces con un límite de tiempo de 300 segundos. Los resultados de la aplicación del algoritmo mostrado anteriormente sobre las instancias ROADEF se muestran en el Tabla 3.5.

En la Tabla 3.5, se presentan el promedio de las 40 ejecuciones y el mejor resultado obtenido del algoritmo. Luego, además del puntaje ROADEF se muestran los puntajes obtenidos por el promedio y mejor resultado utilizando la función $C(X)$ definida anteriormente. Del puntaje ROADEF obtenido se observa que las mayores dificultades del algoritmo están en las instancias del grupo A, en donde la suma de los puntajes ROADEF alcanzó los 35,922 y 25,905 para le media y mejor respectivamente. Estos valores fueron muy parecidos a los obtenidos en la competencia, en donde el equipo J6 obtuvo un puntaje de 37,32 en la fase de calificación. Un caso especial es el de la instancia A2_1, en donde el algoritmo logró obtener mejores resultados que los de ROADEF.

De los puntajes asociados a la función $C(X)$ puede apreciarse que en 13 de las 20 instancias el algoritmo encontró una solución que, en el mejor de los casos, no podría seguir mejorándose más de un 1 % de la solución inicial. Además, comparando ambas columnas se aprecia que los resultados promedio no se alejan mucho de los mejores resultados obtenidos, lo cual señala que el algoritmo es robusto.

En las instancias de los grupos B y X el algoritmo tuvo mejores resultados, logrando obtener puntajes menores a 1 en la mayoría de los casos, para estos grupos la suma de los puntajes promedio y mejor de los grupos B (6,217 ; 3,761) y X (5,831 ; 3,132) también fue muy similar en la competencia (6,66 y 6,24 para B y X respectivamente). Lo anterior señala que el algoritmo si bien no logra competir contra los mejores resultados, tiene más facilidad para resolver instancias de mayor tamaño.

3.4.5. Mejoras del equipo S38

En el año 2013 en [22], Malitsky y Mehta del equipo S38, presentan un trabajo en el cual incluyen mayor parametrización y sintonizan los parámetros de su algoritmo Large Neighborhood Search publicado en [24] (ver Sección 3.4.2). A continuación, proceden a hacer un estudio del impacto de los parámetros en los resultados de su algoritmo, estableciendo un modelo de dos fases para la resolución de cada instancia: una offline y otra de resolución. Recordando que su algoritmo consistía esencialmente en la generación de subproblemas, en donde para generar cada uno se seleccionaban un subconjunto de máquinas y de ellas un subconjunto de procesos a ser reasignados, se presenta un resumen de los parámetros establecidos para el algoritmo LNS en la Tabla 3.6. Dado que la mitad de los parámetros son continuos, tienen grandes dominios y hay dependencia

Inst.	Mejor ROADEF	Media ILS	Mejor ILS	Puntaje Media	Puntaje Mejor	$C(X)_{media}$	$C(X)_{Mejor}$
A1_1	44.306.501	44.306.501	44.306.501	0,000	0,000	0,00	0,00
A1_2	777.532.896	787.593.254	780.499.081	0,948	0,279	0,95	0,28
A1_3	583.005.717	583.006.343	583.006.015	0,000	0,000	0,00	0,00
A1_4	252.728.589	263.449.642	258.024.574	1,695	0,837	3,33	2,47
A1_5	727.578.309	727.578.639	727.578.412	0,000	0,000	0,00	0,00
A2_1	198	515.038	**167**	0,132	0,000	**0,13**	**0,00**
A2_2	816.523.983	1.006.392.553	970.536.821	10,117	8,206	25,00	23,09
A2_3	1.306.868.761	1.488.682.473	1.452.810.819	8,001	6,422	20,12	18,54
A2_4	1.681.353.943	1.721.276.657	1.695.897.404	1,238	0,451	1,27	0,49
A2_5	336.170.182	444.758.671	412.613.505	13,792	9,709	17,45	13,36
Suma(A)				35,922	25,905		
B_1	3.339.186.879	3.643.207.680	3.516.215.073	3,977	2,316	4,61	2,95
B_2	1.015.553.800	1.034.641.974	1.027.393.159	0,368	0,228	0,38	0,24
B_3	156.835.787	165.037.128	158.027.548	0,129	0,019	0,13	0,02
B_4	4.677.823.040	4.677.962.023	4.677.940.074	0,002	0,001	0,00	0,00
B_5	923.092.380	926.221.613	923.857.499	0,025	0,006	0,03	0,01
B_6	9.525.857.752	9.525.923.099	9.525.913.044	0,001	0,000	0,00	0,00
B_7	14.835.149.752	15.372.961.904	15.244.960.848	1,417	1,080	1,42	1,08
B_8	1.214.458.817	1.220.521.241	1.214.930.327	0,043	0,003	0,05	0,01
B_9	15.885.486.698	15.885.635.887	15.885.617.841	0,001	0,001	0,00	0,00
B_10	18.048.515.118	18.155.878.491	18.093.202.104	0,254	0,106	0,26	0,11
Suma(B)				6,217	3,761		
X_1	3.100.852.728	3.348.966.927	3.209.874.890	3,343	1,469	4.38	2.51
X_2	1.002.502.119	1.026.504.981	1.018.646.825	0,470	0,316	0.49	0.34
X_3	211.656	3.151.834	1.965.401	0,048	0,029	0.05	0.03
X_4	4.721.629.497	4.721.814.589	4.721.786.173	0,002	0,002	0.00	0.00
X_5	93.823	696.174	615.277	0,005	0,004	0.01	0.00
X_6	9.546.941.232	9.547.005.000	9.546.992.887	0,001	0,000	0.00	0.00
X_7	14.253.273.178	14.893.088.510	14.701.830.252	1,694	1,188	1.70	1.19
X_8	42.674	369.803	309.080	0,003	0,002	0.00	0.00
X_9	16.125.612.590	16.125.775.950	16.125.753.242	0,001	0,001	0.00	0.00
X_10	17.816.514.161	17.928.266.694	17.867.789.754	0,265	0,122	0.27	0.12
Suma(X)				5,831	3,132		

Tabla 3.5: Resultados de [23] del Iterated Local Search con multi start.

entre algunos ellos, es que utilizan dos algoritmos para su sintonización: Gender-Based Genetic Algorithm (GGA) [2] y el algoritmo ISAC [19]. A partir de lo anterior, en la

Notación	Tipo	Rango	Descripción
u_p	Entero	$[0; 50]$	Límite superior del número de procesos que pueden ser seleccionados de una máquina para reasignar. Usado en $k_p \leq u_p$
t_p	Entero	$[0; 100]$	Límite superior del número de procesos totales que pueden ser seleccionados para reasignar. Usado en $k_m \cdot k_p \leq t_p$
r_p	Continuo	$[0, 1; 1]$	Fracción de la media del número de procesos que se seleccionan para reasignar. Usada antes como $k_p \leq \frac{1}{2}\frac{\#Procesos}{\#Máquinas}$, ahora $k_p \leq r_p\frac{\#Procesos}{\#Máquinas}$
t_m	Entero	$[2; 25]$	Límite superior para el número de máquinas seleccionadas para la selección de un subproblema. Usado en $k_m \leq t_m$
r_m	Continuo	$[0, 1; 10]$	Radio entre el límite superior de las iteraciones consecutivas que no mejoran y la media del número de procesos en una máquina. Usado en $Iteraciones_{sinMejora} \leq r_m\frac{\#Procesos}{\#Máquinas}$
t_f	Continuo	$[0, 1; 10]$	Radio entre el umbral en el número de fallas y el número total de procesos seleccionados para reasignar. Usado en $\#Fallas \leq t_f \cdot k_p$

Tabla 3.6: Parámetros del algoritmo LNS [22].

fase offline propuesta se utiliza clusterización para organizar las distintas instancias en grupos, a partir de los cuales se aplica una sintonización de parámetros a cada grupo. Posteriormente, en la fase online cuando se dispone a resolver una nueva instancia usando LNS, primero se encuentra el grupo de instancias más cercano y luego se obtienen los parámetros específicos obtenidos por cada sintonizador para ese grupo. Además, con el fin de ampliar sus resultados experimentales, generan 1245 instancias, las cuales son variaciones del conjunto B de ROADEF obtenidas realizando perturbaciones al número de recursos, a los pesos en los costos de balance, pesos de movimiento de procesos y máquinas, y costos de movimientos de servicios. Estas instancias están disponibles en el sitio web de uno de los autores [3].

El algoritmo fue ejecutado en un computador con sistema operativo Linux 2.6.5 con arquitectura de 64 bits y procesador Dual Quad Core Xeon de 2.66 GHz de velocidad y 12 GB de memoria RAM. La función de puntaje utilizada fue la misma establecida en la competencia, disminuyendo el tiempo máximo de ejecución a 100 segundos y, dado que el algoritmo fue paralelizado, se probó su rendimiento usando 1, 2 y 4 cores. Se analizaron los resultados obtenidos del algoritmo con tres parámetros distintos: los

[3] `http://4c.ucc.ie/~ymalitsky`

mismos parámetros de la competencia, los parámetros encontrados mediante el uso de GGA y usando los obtenidos por el algoritmo ISAC. Cada uno sobre las instancias B de ROADEF y sobre el conjunto de datos de testing de las instancias que generaron, el cual estuvo compuesto de 500 instancias.

En los resultados obtenidos en los 100 segundos de ejecución sobre las instancias B se destaca que el algoritmo siempre logró encontrar buenos resultados, teniéndose especialmente que los resultados obtenidos con los parámetros de ISAC fueron los mejores, teniendo una media de puntaje (distancia a la mejor solución encontrada) siempre cercano a cero, y teniéndose que para 1, 2 y 4 cores las medias fueron $0,137$; $0,109$ y $0,065$ respectivamente. Por otro lado, en las 500 instancias pudo concluirse que los parámetros obtenidos con ISAC siempre superaron a los demás, obteniendo valores muy cercanos a cero en la mayor parte de los grupos de instancias generados.

Finalmente se concluye, dada la gran cantidad de pruebas que realizaron sobre instancias grandes, que este algoritmo tiene un buen comportamiento en instancias de gran tamaño. También que los parámetros obtenidos mediante la sintonización de parámetros automática tuvieron mucho mejor puntaje que los enviados a la competencia (establecidos mediante pruebas manuales), y dicha mejora en los resultados es tan buena como para considerar un gasto adicional de procesamiento en la sintonización. Respecto a esta última, cabe destacar que el algoritmo más apropiado es ISAC, del cual sus parámetros lograron buenos resultados ampliamente en todas las instancias testeadas.

3.4.6. Mejoras equipo S41

En 2014, Gavranović y Buljubašić del equipo S41 presentan en [13] varias mejoras del trabajo y las estrategias presentadas en [14] (ver Sección 3.4.1). Utilizando en esencia el mismo método de búsqueda local, añaden la capacidad de poder hacer restart y tomar semillas de aleatoriedad distintas. Por otro lado, eliminan la exploración del vecindario generado por los movimientos de tipo chain, a la vez que mantienen el orden de exploración $BPR \rightarrow shift \rightarrow swap$.

Si bien el movimiento BPR se presenta como una alternativa importante para poder reasignar procesos grandes (que a menudo no es posible de realizar sólo con $shift$ y $swap$), puede resultar computacionalmente muy costoso si se utiliza demasiado. La razón de lo anterior es porque intenta encontrar alguna máquina entre las más pesadas (las que tienen más costo de carga y costo de balance) y asignar un proceso al mismo tiempo que elimina varios procesos de ella. Así, mientras no se encuentre alguna solución que mejore a la actual y que sea factible, el algoritmo estará intentando mediante fuerza bruta. Para contrarrestar esto los autores utilizan un parámetro que controla la cantidad de iteraciones explorando el vecindario BPR, además de establecer la

cantidad de máquinas "más pesadas" a $\frac{|M|}{20}$. Por otro lado, también evalúan si es importante la exploración o no del vecindario BPR, ejecutando el algoritmo con y sin dicho movimiento sobre las instancias más complicadas. A partir de esto, concluyen que si bien la exploración aumenta el tiempo de cómputo del algoritmo entre 3 y 7 veces, la ganancia obtenida es suficiente como para utilizarse dentro del límite acostumbrado de 300 segundos de ejecución.

El algoritmo resultante es muy similar al utilizado en su publicación anterior [14]. Reutilizan el método de ruido que incrementa el peso de un recurso con la finalidad de diversificar e intensificar, en el que añaden un parámetro para limitar la cantidad de recursos que son considerados. También reutilizan su método de tomar inicialmente pocos procesos, incrementando la cantidad de procesos considerados en cada iteración en δ, siendo este valor un parámetro del algoritmo. En esta línea, además de ordenar las máquinas por su tamaño, también son ordenados los procesos. Por último, también se estableció un sistema multi start que añadió variabilidad al utilizar diferentes semillas aleatorias en cada nuevo inicio.

Las instancias de ROADEF fueron ejecutadas en un computador core 2 duo $E8500$ de 4 GB de memoria RAM y sistema operativo Debian con arquitectura de 64 bits (utilizando los dos núcleos del procesador), el tiempo máximo de ejecución fue establecido en 300 segundos, cada instancia se ejecutó 100 veces y el algoritmo fue programado en C++, publicándose su código online con licencia de código abierto. Los resultados obtenidos lograron superar los de su trabajo anterior, tanto en valores promedio de las 100 ejecuciones como en los mejores valores obtenidos.

La suma de sus puntajes ROADEF en las instancias A fueron de -9,501 y -15,847 para los resultados promedio y mejores respectivamente. Por otro lado, para las instancias B y X la suma de los puntajes ROADEF fueron -0,382 y -1,486 para los resultados promedio y mejor respectivamente.

Las instancias con mayores diferencias respecto de su trabajo pasado fueron A2_2, A2_3, destacándose que, según los propios autores, las instancias del grupo A son las más difíciles de resolver óptimamente para su algoritmo. Por otro lado, en las instancias del grupo B y X, el algoritmo se comportó de manera similar a la publicación anterior, en donde todas las soluciones generadas tenían una distancia de menos del $0,01\,\%$ del límite inferior de cada instancia. Un caso particular fue el de la instancia X_1, en donde los valores promedio y mejor fueron -0,482 y -0,951 respectivamente, en donde estos resultados fueron junto la mayor influencia para obtener la suma de puntajes negativa.

Otro punto que se destaca es que si bien su algoritmo obtiene buenos resultados, su comportamiento es muy sensible a los parámetros para la elección de los procesos y las máquinas que intervienen en los movimientos, debiéndo establecer el valor del parámetro BPR diferentemente para instancias pequeñas y grandes. Además de lo anterior, el algoritmo obtiene buenas soluciones finales sólo si al comienzo pudo reasignar procesos

grandes de manera apropiada. Esto fue contrarrestado con las estrategias de optimizar la reasignación de grandes procesos y usar sólo algunas semillas aleatorias en el resto de la búsqueda, pero aún así señalan que este es un punto por mejorar.

3.4.7. Enfoque de Hiperheurística

En 2015 Rodolfo Hoffmann et al. [16] presentan la primera hiperheurística aplicada al MRP, que está inspirada en el algoritmo Simulated Annealing. Consta de una estrategia de auto adaptación en base a dos heurísticas, las cuales usa para combinar exploración y explotación del espacio de búsqueda. Su esquema se basa en tres etapas fundamentales. En la primera etapa sólo se ejecuta la primera heurística, que busca explorar el espacio de búsqueda, reduciendo iterativamente la temperatura del algoritmo. Para controlar la ejecución de esta etapa utilizan un parámetro que especifica el tiempo, luego del cual se da paso a la siguiente etapa. A continuación en la etapa 2 se utiliza la solución obtenida en la etapa anterior, para poner a competir ambas heurísticas estableciendo como constante la temperatura del algoritmo. Cada una se ejecuta durante tiempos especificados por dos parámetros del algoritmo. Luego en la etapa 3 se selecciona la mejor solución y la heurística que la produjo. En esta etapa se le da más tiempo a la heurística ganadora para que se ejecute por un tiempo especificado por otro parámetro. Luego de ese paso, la solución obtenida se utiliza para volver a ejecutar la etapa número 2, y comenzar el ciclo nuevamente, teniéndose que se repetirá periódicamente la selección de la heurística más prometedora dependiendo de sus resultados en la etapa 2.

Para confeccionar las heurísticas utilizadas definen 4 tipos de movimientos, dos de ellos son los más utilizados, Swap e Insert (llamado shift en algunas publicaciones). Los otros dos se describen a continuación:

1. Double Insert: Este movimiento es muy parecido al movimiento BPR publicado en [14] y consiste en tratar de migrar un proceso a otra máquina. Si esta operación no puede realizarse por falta de capacidad, entonces se saca un proceso de esa máquina y se inserta en otra para hacer espacio.
2. Backtracking: Este movimiento es similar a la estrategia de generación de subproblemas de [24], y consiste en elegir de manera aleatoria n máquinas y m procesos y ejecutar backtracking para resolver este subproblema.

A partir de lo anterior, las heurísticas generadas son las siguientes:

1. Heurística H0: Esta heurística utiliza los movimientos Insert, Swap y Backtracking. Dependiendo de un parámetro escoge con cierta probabilidad Swap o Insert durante una cierta cantidad de veces. Si la solución encontrada es mejor que la actual se acepta; de lo contrario se acepta o no dependiendo de la temperatura

asociada a Simulated Annealing. Por último, se ejecuta Backtracking para mejorar la solución candidata, seleccionando un número pequeño de variables al azar.

2. Heurística H1: Esta heurística usa los movimientos Insert, Swap y Double Insert. Es similar a la heurística H0 pero en vez de utilizar Backtracking como proceso final, ejecuta Double Insert.

El algoritmo de la propuesta fue programado en C++ y ejecutado en una máquina Linux QuadCore AMD FX-4300 con 4GB de memoria RAM, sobre 20 instancias de ROADEF, cada una ejecutada 5 veces de manera independiente. Dada la gran cantidad de parámetros de los cuales depende, se utilizó el sintonizador de parámetros ParamILS [4].

Los experimentos realizados fueron aplicados utilizando las instancias B de ROADEF, y fueron divididos en dos partes: una para comparar el desempeño de las heurísticas por si solas y la hiperheurística, y la segunda para comparar los resultados de las 10 instancias con los mejores resultados obtenidos en la competencia. Los resultados de la primera parte mostraron que la hiperheurística obtuvo los mejores resultados en 9 de 10 instancias testeadas, con lo cual pudo concluirse que esta es capaz de auto adaptar las heurísticas de bajo nivel y utilizar una selección de cada estrategia dinámicamente. Por otro lado, los resultados de la segunda parte fueron comparados con los tres mejores lugares de la competencia Senior ROADEF, en donde se encontró que la hiperheurística propuesta muestra en promedio $0,24\,\%$ peor rendimiento que el algoritmo del equipo S41 (primer lugar en la categoría Senior), $0,02\,\%$ peor rendimiento que el algoritmo del equipo S38 (segundo lugar) y mejor que el equipo S14 (tercer lugar).

3.4.8. Búsqueda Híbrida

En el año 2015, W. Jaśkowski et. al. presentó una metaheurística híbrida para el MRP [18], con la que junto al equipo J12 logró obtener el sexto lugar en la etapa de calificación, y tercer lugar en la etapa final (clasificación general). El algoritmo consta de dos componentes principales: un Hill Climbing (HC) y un Large Neighborhood Search (LNS), que usa MIP para resolver subproblemas. Partiendo de la solución original, se ejecuta HC de forma iterativa para mejorar rápidamente la solución, y luego usa LNS basado en MIP para realizar mejoras adicionales.

El objetivo del HC es mejorar una asignación dada tan rápido como sea posible. Para esto, se genera el vecindario mediante un movimiento *shift*, que reasigna un proceso de una máquina a otra. El algoritmo es determinista, ya que acepta el primer movimiento

[4]Implementación de Frank Hutter's para el sintonizador ParamILS, disponible en: http://www.cs.ubc.ca/labs/beta/Projects/ParamILS

que genere una solución factible con un costo menor al de la mejor solución hasta el momento. Si no se encuentra mejora en el vecindario, el algoritmo se detiene.

Para mejorar el rendimiento del algoritmo se aplican varias técnicas. Una de ellas, denominada *evaluación delta*, consiste en calcular sólo la diferencia entre los costos de dos vecindarios, en lugar de calcular el costo de la nueva solución generada desde cero. Esto se justifica con el hecho de que una de las mayores limitaciones del algoritmo está en la implementación del movimiento, al tener que computar el costo de cada solución del vecindario. La segunda técnica propuesta, consiste en explorar el espacio de búsqueda, de manera que se puedan encontrar más rápidamente las soluciones con mejor valor. Para esto, los procesos se ordenan de manera decreciente según su *potencial*, que mide cuánto disminuiría el valor del costo total de la solución si el proceso no existiera. A mayor potencial, más probabilidad tiene ese proceso de disminuir el costo de la solución. Así, el algoritmo evalúa los procesos con mayor potencial primero, y aquellos con cero potencial no se consideran, ya que el costo de la solución no disminuye al reasignar aquellos procesos.

Otra mejora es el uso de una lista tabú para el HC, que contiene las máquinas que son ignoradas al buscar la mejor máquina para un proceso dado. Inicialmente, la lista tabú esta vacía, a medida que se reasignan procesos el algoritmo recuerda si es que se han desasignado o insertado procesos en cada máquina, guardando en la lista tabú aquellas máquinas en las que no se ha desasignado o insertado ningún proceso. La máquina se elimina de la lista tabú cuando se saca un proceso de ella. Esto se basa en el supuesto de que si no es posible mejorar el valor de la función objetivo insertando cualquier proceso en esta máquina, sólo remover un proceso puede revertir esta situación. Este supuesto en general no se cumple, pero no empeora mucho el resultado de HC y permite mejorar considerablemente la velocidad del algoritmo. Finalmente, el algoritmo considera mejor mejora en cada iteración, pero va almacenando adicionalmente aquellas máquinas que generan alguna mejora.

La segunda fase del algoritmo busca mover varios procesos a la vez. El algoritmo puede ser clasificado como un LNS debido a que el tamaño del vecindario crece exponencialmente con el número de procesos y máquinas. De esta manera, explorar un vecindario puede ser visto como resolver un subproblema del problema original con una heurística o algún método exacto. El algoritmo selecciona subproblemas iterativamente e intenta resolverlos de manera óptima. Este subproblema se genera a partir del problema original, seleccionando un subconjunto de máquinas $M_X \subset M$, entre las que es posible mover procesos a partir de la mejor solución hasta el momento. En las máquinas que no están en M_X no se realizan movimientos. Para su resolución se respetan todas las restricciones del problema original, por lo que siempre se producen soluciones factibles. El criterio de aceptación también es mejor mejora, es decir, se acepta la solución encontrada sólo si supera a la mejor solución actual. En esta fase se utilizan heurísticas para mejorar el rendimiento, que incluyen heurísticas para la selección de los subproblemas, para la adaptación dinámica del tamaño de los mismos, y búsqueda local en

HC, que considera primero las máquinas que fueron modificadas en esta etapa. Para elegir las máquinas componentes de los subproblemas se evaluaron dos enfoques: selección aleatoria o selección en base a un puntaje asignado que refleja el potencial de cada máquina de disminuir el costo de la solución. Los subproblemas generados se resuelven usando IBM CPLEX versión 12.5.

Para evaluar la calidad del algoritmo se usaron las instancias de ROADEF, y las pruebas se ejecutaron en un equipo Linux 64bit con procesador Intel Core i7 950 3.07 GHz (usando sólo un core) y 6GB de RAM. El algoritmo se implementó en Java y fue ejecutado usando Java 1.7.0_04. Se probaron varias combinaciones de los algoritmos planteados anteriormente, entre las cuales están HC, LNS, LNSHC, LNSR. Algunas variantes comienzan mejorando la solución inicial con HC, otras comienzan directamente aplicando LNS, otras mejoran el resultado de LNS con HC, además de probar las heurísticas mencionadas anteriormente. La estructura de las variantes evaluadas se muestra en la Figura 3.3.

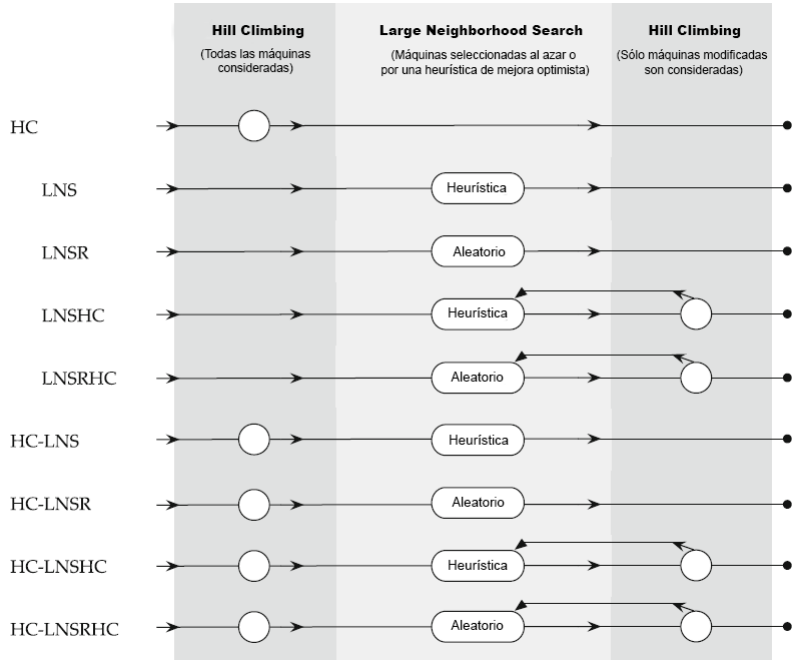


Figura 3.3: Diagrama de flujo de las variantes evaluadas del algoritmo.

En general, se pudo concluir que las variantes que comenzaron con HC tuvieron mejor rendimiento que los que empezaron directamente con LNS en las instancias más grandes (grupo B). Además se observó que las primeras mejoraron más rápidamente desde el inicio. Por otro lado, las variantes que usan LNS con la heurística para generar los subproblemas obtuvieron mejores resultados que las con selección aleatoria. Si bien

LNS puro no domina estrictamente a LNSR puro, se concluyó que el primero es más robusto. Finalmente, se logró determinar que la variante con mejor rendimiento en ambos datasets fue HC-LNSHC.

La mejor variante elegida, que corresponde a una simplificación del algoritmo que fue realmente presentado en ROADEF, fue comparada con los resultados obtenidos por los dos primeros lugares de la competencia, los equipos S41 y S38. A pesar de que para ambos grupos de instancias el algoritmo de S41 es mejor en promedio, ningún algoritmo supera a los otros en todas las instancias, y en la mayoría de los casos las diferencias en los resultados son mínimas. Al evaluar cada algoritmo en función del tiempo se logró concluir que la principal ventaja del algoritmo propuesto se encuentra en su robustez, que se justifica con la baja varianza de los resultados, haciéndolo menos sensible a factores aleatorios.

3.4.9. Heurísticas: Vector Bin packing y Machine Reassignment Problem

En 2015, Michael Gabay et al. muestran un trabajo que analiza el MRP desde la perspectiva del Vector Bin Packing (VBP) [10]. Presentando una perspectiva muy similar a la de Renaud Mason et. al. en [23], dan cuenta de que si bien el VBP trata sobre la asignación de items de múltiples dimensiones en bins o contenedores, dichos contenedores poseen todos la misma capacidad entre si y en cada una de sus dimensiones. Es por esto que proponen la introducción de una generalización de dicho problema, denominado Vector Bin Packing with Heterogeneous Bins (VBPHB), en el cual se propone que cada bin tenga capacidades distintas para cada dimensiones (resultando ser el mismo multi-capacity bin packing al que se refiere Mason en [23]) . A partir de esto y en base a conocimiento de la literatura existente del VBP y el BP, ellos proponen varias familias de heurísticas que permiten crear soluciones para el VBPHB. A continuación, analizan las propiedades estructurales del MRP que permiten descomponerlo en problemas más pequeños, y muestran que las heurísticas son suficientemente flexibles como para ser adaptadas y así poder generar nuevas soluciones factibles rápidamente, de modo que puedan servir como distintos puntos de algoritmos de inicio de búsqueda local como los ya publicados en años anteriores.

A partir de lo anterior implementan un algoritmo, que adapta las conocidas heurísticas first fit y best fit de los problemas del BP, y que busca generar soluciones distintas desde cero, sin tomar en cuenta la función objetivo. Además

Los resultados obtenidos muestran que sus algoritmos logran asignar rápidamente (en menos de 5 segundos) más del 90 % de los procesos en todas las instancias, sin embargo no logran encontrar soluciones en todas estas, obteniendo soluciones en 2 las 10 instancias del grupo A y 6 de las 10 instancias del grupo B. Si bien estos resultados no se muestran como una alternativa confiable de generación de soluciones iniciales

factibles que puedan ser utilizadas como alternativas a la solución inicial entregada por ROADEF, sus heurísticas y algoritmos son lo suficientemente flexibles como para ser extendidas y adaptadas en trabajos futuros, por lo que sientan un precedente en la obtención de soluciones creadas completamente desde cero.

3.4.10. Algoritmo Evolutivo Cooperativo: Simulated Annealing

En 2017, Turky et al. [31] muestran un algoritmo para resolver el MRP con un enfoque evolutivo paralelo, el cual toma ventaja de las arquitecturas paralelas existentes como multi-core o computación en la nube. Basado en la cooperación de varios algoritmos Simulated Annealing heterogéneos, cada uno parte desde individuos de la población y configuraciones de parámetros diferentes, y es ejecutado paralelamente utilizando distintos hilos del procesador.

Partiendo de la solución inicial suministrada por ROADEF, se crea una población de soluciones, las cuales se obtienen a partir de mutaciones de la solución inicial. Dado que el objetivo de la creación de la población inicial a partir de mutaciones es diversificar la búsqueda, cada una de las soluciones es obtenida mediante la aplicación de un operador de mutación repetidamente. Los siguientes operadores de mutación son seleccionados aleatoriamente para aplicarse en distintas soluciones:

1. Swap: Seleccionar e intercambiar dos procesos de diferentes máquinas.
2. Doble Swap: Seleccionar e intercambiar cuatro procesos de diferentes máquinas.
3. Shift: Seleccionar un proceso de una máquina y moverlo a otra distinta.
4. Doble Shift: Seleccionar dos procesos de dos máquinas y moverlos a máquinas distintas.

Su algoritmo no incluye cruzamiento, por ser costoso computacionalmente y guiar mayoritariamente a soluciones infactibles, de acuerdo a sus experimentos preliminares.

Luego de obtenerse la población inicial, su enfoque procede a la ejecución de varios algoritmos Simulated Annealing, que constan de los siguientes pasos:

1. Inicialización: Se inicializan los parámetros de cada SA, los cuales son temperatura inicial, radio de enfriamiento (para un enfriamiento geométrico) y temperatura final.
2. Inicialización de la solución de cada SA: Se asocia una solución de la población a cada SA, asignándolas aleatoriamente.
3. Cada algoritmo SA se ejecuta efectuando los siguientes pasos:
   a) Evaluación: Se calcula el valor fitness para cada solución.

b) Se genera una solución nueva modificando la solución actual utilizando un operador de vecindario. Los operadores de vecindario son los mismos cuatro operadores de mutación (Swap, Doble Swap, Shift y Doble Shift), pero asignados de manera distinta. De acuerdo a la arquitectura de 4 hilos, 4 soluciones y 4 SA de su algoritmo, las configuraciones utilizadas fueron:

1) SA1: Operador de mutación: Doble Swap - Operador de vecindario: Swap
2) SA2: Operador de mutación: Swap - Operador de vecindario: Doble Swap
3) SA3: Operador de mutación: Doble Shift - Operador de vecindario: Shift
4) SA4: Operador de mutación: Shift - Operador de vecindario: Doble Shift

c) Se reemplaza la solución actual en caso de que la nueva solución sea mejor o que se cumpla la condición de elección por temperatura.

Con $P(aceptar_{SolucionCandidata}) = e^{\frac{-\Delta f}{T}}$.

d) Se actualiza la temperatura, disminuyéndola geométricamente según la ecuación: $t_{i+1} = t_i \cdot \alpha$

e) Condición de término: Si la temperatura es mayor que la temperatura final, el algoritmo sigue a la siguiente iteración (genera nueva solución), en caso contrario termina y entrega la mejor solución encontrada.

Puntos de restart

El algoritmo propone la búsqueda de soluciones cerca de las que ya han sido exploradas. Esto se hace modificando la mejor solución mediante el operador de mutación asociado al individuo. Este mecanismo se activa cuando un la solución actual no ha sido mejorada por una cierta cantidad de iteraciones.

Estrategia cooperativa

En cada generación, se comparan las mejores soluciones encontradas por cada SA, guardándose la mejor de ellas como punto de restart.

Resultados experimentales

Para la ejecución de su algoritmo utilizaron los grupos de instancias A, B y X provistos por ROADEF. Para cada ejecución se estableció un tiempo máximo de 5 minutos, al igual que en la competencia. Su enfoque propuesto consideró una población de

4 soluciones, la cual es equivalente al número de hilos disponibles, y los distintos parámetros fueron establecidos de manera manual analizando varias configuraciones. Cada instancia fue ejecutada 31 veces con la misma configuración de parámetros.

En sus primeros experimentos determinaron que la configuración de varios SA cooperativos y con distintas configuraciones (CHSA), obtiene mejores resultados que aquellos que no cooperan o aquellos que tienen configuraciones homogéneas. Respecto a los resultados obtenidos por CHSA para las 30 instancias de ROADEF, los autores muestran sólo los mejores resultados, los cuales son mostrados en la Tabla 3.7. Puede verse que los mejores valores son muy bajos, obteniendo puntajes negativos en 6 de las 10 instancias del grupo A, en todas las instancias del grupo B y en 6 instancias del grupo X.

Finalmente, realizaron comparaciones de CHSA con la literatura, en donde se destaca que su algoritmo logró obtener resultados más bajos que los mejores algoritmos de la literatura, en términos de mejores resultados obtenidos. Los resultados fueron corroborados con los test de hipótesis de Friedman y Iman-Davenport con significancia de 95 %.

Inst.	Mejor ROADEF	Mejor CHSA	Puntaje Mejor
A1_1	44.306.501	44.306.501	0,0000
A1_2	777.532.896	777.532.181	-0,0001
A1_3	583.005.717	583.005.717	0,0000
A1_4	252.728.589	244.875.201	-1,2416
A1_5	727.578.309	727.578.309	0,0000
A2_1	198	162	0,0000
A2_2	816.523.983	720.671.541	-5,1073
A2_3	1.306.868.761	1.190.908.042	-5,1028
A2_4	1.681.353.943	1.680.368.567	-0,0306
A2_5	336.170.182	307.150.825	-3,6857
Suma(A)			-15,168
B_1	3.339.186.879	3.291.245.601	-0,6272
B_2	1.015.553.800	1.015.482.891	-0,0014
B_3	156.835.787	156.757.941	-0,0012
B_4	4.677.823.040	4.677.811.387	-0,0001
B_5	923.092.380	922.944.510	-0,0012
B_6	9.525.857.752	9.525.851.397	0,0000
B_7	14.835.149.752	14.834.456.193	-0,0018
B_8	1.214.458.817	1.214.291.143	-0,0012
B_9	15.885.486.698	15.885.437.301	-0,0002
B_10	18.048.515.118	18.048.271.541	-0,0006
Suma(B)			-0,635
X_1	3.100.852.728	3.044.418.078	-0,7603
X_2	1.002.502.119	1.002.390.081	-0,0022
X_3	211.656	493.938	0,0046
X_4	4.721.629.497	4.721.586.145	-0,0005
X_5	93.823	521.050	0,0035
X_6	9.546.941.232	9.546.956.909	0,0001
X_7	14.253.273.178	14.252.476.502	-0,0021
X_8	42.674	80.107	0,0003
X_9	16.125.612.590	16.125.531.251	-0,0004
X_10	17.816.514.161	17.815.981.144	-0,0013
Suma(X)			-0,758

Tabla 3.7: Resultados de CHSA [31].

3.5. Resumen

En este capítulo se realizó una revisión de los avances más importantes que se han logrado en la resolución de MRP. Además, se presentaron las principales instancias propuestas en la literatura, sus características y mejores resultados obtenidos, los cuales servirán para realizar futuras comparaciones de rendimiento. Se mostró dos de los modelos propuestos en la literatura para resolver el problema, que corresponden al modelo propuesto por ROADEF en la definición del problema y al modelo de Programación Lineal Entera Mixta, los cuales han sido utilizados en publicaciones con muy buenos resultados. En general, los métodos presentados han sido muy diversos, pero han tenido una clara tendencia hacia la Búsqueda Local y Large Neighborhood Search (LNS). Se destaca que en la competencia ninguno de los competidores pudo dominar en todas las instancias, al igual que si bien las publicaciones posteriores ofrecen muy buenos resultados, generalmente logran obtener mejores resultados en las instancias más grandes que en las pequeñas. Si bien estos métodos han servido para solucionar el MRP en tiempos bajos y con resultados cercanos a los mejores encontrados, es preciso notar que el ambiente tecnológico del problema propone grandes desafíos como su resolución en menor tiempo que los 300 segundos utilizados en la competencia. Tal es el caso de problemas en que los procesos a reasignar son generados en tiempo real o por ráfagas, en donde el tiempo para resolver un problema de gran envergadura podría reducirse considerablemente. Por otro lado las últimas publicaciones han sido cada vez más competitivas y buscan no solo presentar resultados cercanos a los óptimos, sino también algoritmos robustos que obtengan buenos resultados independientemente de las instancias o de las semillas aleatorias con que se ejecutan, desafío que sigue presente.

4 Descripción del algoritmo

En este capítulo, se presenta el algoritmo utilizado para resolver el problema MRP mediante una colaboración de las metaheurísticas de búsqueda Hill Climbing (HC) y Simulated Annealing (SA). Para esto, se muestra un esquema de funcionamiento del algoritmo general, para después definir las distintas partes de este, también las estrategias usadas para abordar el problema, las heurísticas utilizadas, y las estrategias a seguir para lograr diversificación e intensificación en la búsqueda de soluciones.

4.1. Esquema general del algoritmo propuesto: SMaRT

El algoritmo propuesto, denominado como SMaRT (Simulated Annealing for Machine Reassignement with Temperature Control), consta de cuatro partes fundamentales, las cuales son las siguientes:

1. Preprocesamiento de datos: En esta etapa se busca precomputar datos útiles o podar dominios de variables, con la finalidad de aumentar la eficiencia del algoritmo y acelerar el procesamiento.
2. Hill Climbing 1 (HC1): En esta etapa se busca mejorar rápidamente el valor de la función objetivo enfocándose, esencialmente, en disminuir la sobrecarga de algunas máquinas, buscando mover los procesos de mayor tamaño de las máquinas que más costo añaden a la función objetivo. Esta fase finaliza cuando se alcanza el tiempo límite o al alcanzar un óptimo local.
3. Hill Climbing 2 (HC2): En esta etapa se busca mover los procesos mediante movimientos simples shift y swap con el objetivo de intensificar en el espacio de búsqueda. Esta fase finaliza cuando se alcanza el tiempo límite o al alcanzar un óptimo local.
4. Simulated Annealing (SA): Esta etapa busca escapar del óptimo local encontrado en HC2 mediante su mecanismo de temperatura, con la finalidad de diversificar la búsqueda hacia otras regiones. Esta fase finaliza cuando se alcanza el tiempo límite.

Un diagrama de flujo general del algoritmo se muestra en la Figura 4.1.

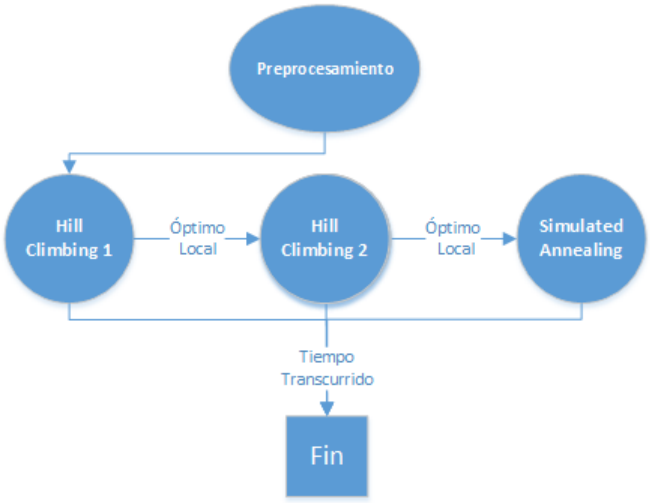


Figura 4.1: Esquema general del algoritmo SMaRT

4.2. Preprocesamiento de datos

Se han añadido las siguientes operaciones de preprocesamiento de datos:

1. Filtrar dominio de procesos: En esta sección se analiza, para cada proceso, si puede ser asignado a cada máquina, eliminando aquellas que no son factibles de su dominio. Esto se logra tomando en cuenta los recursos transitorios, los cuales seguirán siendo utilizados en una máquina, aún cuando todos los procesos de ésta han sido movidos a otras máquinas. En base a lo anterior, se eliminan del dominio de cada proceso aquellas máquinas en las cuales el espacio disponible, considerando los recursos transitorios, es insuficiente para efectuar la asignación.
2. Computar procesos fijos a su máquina inicial: En base al filtrado anterior, se calculan aquellos procesos que debido a que tienen grandes requerimientos de recursos transitorios, sólo pueden ser asignados a su máquina inicial. Estos procesos no son considerados para generar nuevas soluciones, sino que estarán fijos durante toda la ejecución del algoritmo. Además, se elimina la máquina en la cual el proceso quedó asignado, de los dominios de los procesos que están en el mismo servicio (propagando la restricción de conflicto), los cuales son marcados para revisar en caso de que dicha eliminación haya causado que sólo sean asignables a una máquina. Lo anterior permite reducir el espacio de búsqueda al considerar como procesos asignables un subconjunto del total.
3. Computar límite inferior de la función de costo: Se calcula el límite inferior de la función de costo en base a los límites inferiores del costo de carga y balance, tal como ha sido propuesto por [29, 14, 28, 13, 24].

4. Computar factor de tamaño de recursos: Cada vez que los procesos son ordenados, lo hacen en base a una noción de tamaño. Esta noción se obtiene usando la suma de sus requerimientos, ponderados por un factor que expresa la escasez de los recursos que requieren, y que es calculado de acuerdo a la capacidad total de las máquinas para ese recurso. Esta forma de dimensionar cada proceso ha sido propuesta en [10], y se presenta como una mejor alternativa a la idea de tamaño de un proceso como la suma de sus requerimientos. La fórmula que indica el tamaño es la siguiente:

$$Tamaño(p) = \sum_{r \in \mathcal{R}} R(p,r)\beta_r \ ; \ donde\, \beta_r = \frac{\sum_{p \in \mathcal{P}} R(p,r)}{\sum_{m \in \mathcal{M}} C(m,r)} \tag{4.1}$$

   En definitiva, el tamaño de un proceso es la suma de sus requerimientos para cada recurso, en donde cada requerimiento está ponderado por el factor β_r, que se calcula dividiendo el requerimiento total, por la capacidad total, para ese recurso dado. Esta medida permite dar mayor prioridad en los ordenamientos a los procesos cuyos requerimientos no solo son de gran tamaño, sino que además involucra a los recursos más escasos.

4.3. Algoritmo

A continuación, se describirá la representación utilizada, la función objetivo y la función de evaluación, que son comunes para todas las fases. Luego se dará paso a la explicación de los algoritmos HC1, HC2 y SA individualmente, junto con la implementación de cada una de sus componentes en el contexto del problema.

4.3.1. Representación

Dado un conjunto P de n procesos $p_1, p_2, ..p_n$ y un conjunto M de k $m_1, \ldots, m_k$ máquinas, se define una representación del MRP como un arreglo A de tamaño n donde $a_i = m_j$ indica que el proceso p_i está asignado a la máquina m_j. Esto se muestra en la Figura 4.2.

1	**2**	**3**	**4**	**...**	**...**	**...**	$\boldsymbol{p \in P}$
5	5	4	3	...	...	...	$m \in M$

Figura 4.2: Representación de una solución como arreglo.

Esta representación permite identificar fácilmente qué procesos están asignados a qué máquinas, además de satisfacer la restricción en la que un proceso sólo debe ser asignado a una máquina específica. Es necesario recorrer todo el arreglo para chequear que todos los procesos hayan sido asignados a una máquina.

4.3.2. Función de evaluación

La función de evaluación es la misma que la establecida por ROADEF [6] y descrita en Sección 3.3.2, la cual corresponde a la suma de los costos ponderados, y debe minimizarse.

$$
\begin{aligned}
totalCost = & \\
& \sum_{r \in \mathcal{R}} weight_{loadCost}(r) \cdot loadCost(r) \\
& + \sum_{b \in \mathcal{B}} weight_{balanceCost}(b) \cdot balanceCost(b) \\
& + weight_{processMoveCost} \cdot processMoveCost \\
& + weight_{serviceMoveCost} \cdot serviceMoveCost \\
& + weight_{machineMoveCost} \cdot machineMoveCost
\end{aligned}
$$

Es importante destacar que el cómputo de la función de evaluación se lleva a cabo calculando las variaciones de los sub-costos debido a los movimientos, y no recomputando el costo total completamente. Esto permite reducir considerablemente los costos computacionales asociados a evaluar los distintos movimientos.

4.3.3. Hill Climbing en MRP

La metaheurística Hill Climbing busca mejorar iterativamente una solución inicial. En cada iteración genera vecinos mediante algún movimiento, y se cambia a alguno que presente alguna mejora, o a aquel que presenta la mejor mejora de entre todos, dependiendo de su variante. En caso de no encontrar algún vecino que presente mejora, el algoritmo se encuentra estancado en un óptimo local y es posible incorporar un mecanismo de restart, que permite iniciar desde un punto de partida distinto, con la finalidad de explorar otra región del espacio de búsqueda. Un pseudocódigo de Hill Climbing con alguna mejora para un problema de minimización se muestra en el Algoritmo 4.1

Algoritmo 4.1 Algoritmo Hill Climbing para minimización

```
1  s_actual ← SolucionInicial()
2  s_mejor ← s_actual
3  WHILE Criterio_Termino DO
4  BEGIN
5      s_actual ← GenerarSiguienteVecinoMejora()
6      IF s_actual = ∅
7      BEGIN
8          s_actual ← GenerarNuevoPuntoPartida()
9      ELSE IF f(s_actual) < f(s_mejor)
10         s_mejor ← s_actual
11     END
12 END
```

A continuación se presentan los algoritmos HC de SMaRT.

4.3.3.1. Hill Climbing 1 (HC1)

El propósito general de este algoritmo es lograr una mejora de la función de evaluación, buscando mover los procesos más grandes de las máquinas más sobrecargadas (es decir, cuyo costo de carga es mayor). Antes de presentar el algoritmo se describe cada uno de sus componentes.

Movimiento: Size Discriminant Excluding Shift (SDES) El movimiento escogido tiene como base el hecho de que se aplica sobre una solución factible obtenida con anterioridad, su idea general es seleccionar las primeras $max_{máquinas}$ máquinas, ordenadas por costo total, más sobrecargadas, y de cada una los $max_{procesos}$ procesos, ordenados por suma de requerimientos ponderados, de mayor tamaño (ver Sección 4.2, Elemento 4). Con estos procesos se hace shift a otra máquina, que produzca la mejor mejora de la función de evaluación de entre todos los shifts factibles. Además, el algoritmo tiene la restricción de que los procesos escogidos para mover no deben ser asignados a ninguna de las otras $max_{máquinas} - 1$ máquinas consideradas, en la iteración actual. Una vez hecho el shift de un proceso de la lista de máquinas, se reordenan las máquinas y se vuelve a iterar.

Criterio de selección El criterio de selección de soluciones candidatas para el algoritmo HC1 es: de la primera máquina, dentro de las $max_{máquinas}$ más pesadas, de la que puedan moverse procesos, buscar el shift que produzca la mejor mejora de entre los $max_{procesos}$ procesos con mayor tamaño de esa máquina. Una condición adicional es que para mover cualquier proceso se debe excluir del destino a cualquiera de las $max_{máquinas}$ consideradas en esa iteración.

Algoritmo 4.2 Algoritmo HC 1 para minimización en MRP

```
1  max_máquinas //Parámetro
2  max_procesos //Parámetro
3  s_actual ← SolucionInicial()
4  s_mejor ← s_actual
5  WHILE time() < Límite_Tiempo DO
6  BEGIN
7      mejorado ← false
8      OrdenarPorCostoCargaDesc(máquinas)
9      máquinas_sobrecargadas ← máquinas[1:max_máquinas]
10     máquinas_asignables ← máquinas[max_máquinas + 1 : end]
11     FOR m IN máquinas_sobrecargadas
12     BEGIN
13         vectorPAsignados ← ProcessosAsignados(m)
14         OrdenarPorSumReqPonderadosDesc(vectorPAsignados)
15         TopProcesos_pesados ← vectorPAsignados[1:max_procesos]
16         s_actual ← GenerarMejorMejoraShift(TopProcesos_pesados, máquinas_asignables)
17         IF s_actual ≠ ∅
18         BEGIN
19             s_mejor ← s_actual
20             mejorado ← true
21             break
22         END
23     END
24     IF !mejorado
25     BEGIN
26         break
27     END
28 END
29 devolver s_mejor
```

Criterio de término El criterio de término definido es cuando ya no existan más vecinos que produzcan mejora respecto de la solución actual, o cuando se alcance el límite de tiempo señalado por el parámetro $Límite_{Tiempo}$.

Estructura del Algoritmo HC1 propuesto En el Algoritmo 4.2 se muestra el pseudocódigo de HC1 para MRP.

Entre las líneas 1 y 4 se muestran los parámetros y variables del algoritmo, siendo $max_{máquinas}$ y $max_{procesos}$ parámetros que señalan cuántas de las máquinas más sobrecargadas y cuántos procesos pesados de cada una serán tomados en cuenta, respectivamente. En la línea 7 se comienza con una nueva iteración, estableciendo la variable booleana *mejorado*, la cual, en caso de mantenerse con el valor *false* luego de la revisión de todas las máquinas, indicará que un óptimo local se ha alcanzado.

A continuación, en la línea 8 se ordenan las máquinas por costo de carga en forma

descendente. En las líneas 9 y 10 se particionan las máquinas ordenadas en dos conjuntos: $máquinas_{sobrecargadas}$, que serán aquellas a tomar en cuenta para sacar procesos, y $máquinas_{asignables}$, que serán aquellas a las cuales será posible asignar procesos. Luego, entre las líneas 11 y 23, se itera sobre cada máquina sobrecargada m buscando quitar un proceso de alguna de ellas. En las líneas 13 y 15 se ordenan por tamaño descendente los procesos asignados de la máquina sobre la cual se itera, y se genera el conjunto $TopProcesos_{pesados}$, con el cual se buscará aquel shift a una máquina de $máquinas_{asignables}$ que produzca la mejor mejora, para la máquina sobrecargada seleccionada. En la función de la línea 16 se generan los mejores shifts para cada proceso en $TopProcesos_{pesados}$, actualizando s_{actual} con el mejor shift de todos los procesos. Si se ha encontrado, realizando un shift, una mejor solución con respecto a la actual, la variable *mejorado* se establecerá en *true*.

Luego, entre las líneas 18 y 22 se revisa si en la iteración actual existe una mejora respecto de la mejor solución hasta el momento. En caso de existir, se actualiza la mejor solución y se termina con las iteraciones en $máquinas_{sobrecargadas}$, por cumplirse el criterio de alguna mejora en las máquinas revisadas y se comienza con una nueva iteración.

Finalmente, entre las líneas 24 y 27 se revisa la variable *mejorado* para salir del algoritmo si es que se ha alcanzado un óptimo local, en caso contrario se pasa a la siguiente iteración.

Se destaca que el algoritmo, desde una perspectiva general, buscará que los conjuntos de $máquinas_{sobrecargadas}$ en cada iteración tengan menos procesos asignados, esto hace que estas máquinas queden con espacio disponible para que puedan realizarse cambios en las siguientes etapas del algoritmo.

4.3.3.2. Hill Climbing 2 (HC2)

El algoritmo HC2 se ejecutará luego de que el algoritmo HC1 haya quedado estancado en un óptimo local. Dado que HC2 opera con movimientos menos restringidos que HC1, la solución factible entregada por HC1 se utiliza como solución inicial. El propósito principal de este algoritmo es seguir explotando la solución con los movimientos simples shift y swap. El algoritmo termina cuando se ha llegado al límite de tiempo o cuando un óptimo local ha sido encontrado.

Movimiento y Criterio de selección El movimiento a utilizar es aplicado sobre una solución factible generada con anterioridad, y se basa en los siguientes dos movimientos.

1. **Shift de procesos:** Se toma un proceso de una máquina y se inserta en otra. Este movimiento tiene la ventaja de brindar variabilidad, ya que permite aumentar o disminuir la cantidad de procesos de una máquina.

2. **Swap de procesos:** Tomar dos procesos asociados a distintas máquinas y asignar cada uno a la máquina contraria. Si bien este movimiento no cambia la cantidad total de procesos de las máquinas involucradas, tiene la ventaja de posibilitar cambios más grandes en la solución en un solo paso, y gracias a esto posibles mejoras más sustanciales en la función de evaluación.

Puede verse que ambos vecindarios son complementarios. Mientras shift permite conocer el mejor vecino cambiando un proceso de lugar, swap permitirá mover dos procesos a la vez. Ambos movimientos han sido ampliamente utilizados en la literatura obteniendo buenos resultados.

El mecanismo para generar una nueva solución candidata es el siguiente:

1. Se genera, si es posible, un nuevo vecino que tenga alguna mejora mediante shift.
2. Se genera, si es posible, un nuevo vecino que tenga alguna mejora mediante swap.
3. Criterio de selección: Se comparan las mejoras en la función de evaluación asociadas a ambos vecinos, aceptándose el de menor valor; en caso de empate se elige el vecino generado con swap, ya que aporta más en diversidad.

Considerando un análisis de complejidad algorítmica de peor escenario, la cantidad de vecinos generados antes de encontrar alguna mejora usando shift podría ascender a $N°Procesos \cdot (N°Máquinas - 1)$, la cual en las instancias más grandes sería de $50{,}000 \cdot 4{,}999 \approx 2,5 \cdot 10^8$ vecinos. Por otro lado, considerando el movimiento swap, la cantidad de vecinos generados podría llegar a $\frac{(N°Procesos)!}{(N°Procesos-2)!*2!} = \frac{(N°Procesos \cdot N°Procesos-1)}{2}$, que en las instancias más grandes sería de $\frac{50{,}000 \cdot 49{,}999}{2!} = 25{,}000 \cdot 49{,}999 \approx 1,25 \cdot 10^9$, teniéndose además que como swap cambia la asignación de dos procesos es más costoso y difícil calcular si su intercambio satisface las restricciones. Es por esto que en el caso de la generación de un vecino con alguna mejora usando swap, el número de intentos ha sido limitado usando una cantidad máxima de procesos a considerar para realizar los swaps, especificada por el parámetro $MaximosVecinosSwap$.

Además de lo anterior, cada vez que el algoritmo encuentra una solución mejor que la solución actual, las listas de procesos para generar los vecindarios shift y swap son aleatorizadas. Lo anterior permite cambiar el orden en que los procesos son considerados, y en el caso del vecindario swap, brinda una oportunidad a procesos que no la tuvieron por estar más allá del límite de procesos considerados.

Esta aleatorización de las listas de procesos también es ejecutada de manera independiente en el peor de los casos, es decir, cada vez que un vecindario se ha recorrido completamente (hasta el máximo de procesos en el caso de swap).

Criterio de término El algoritmo finaliza en caso de haber alcanzado el límite de tiempo, o que no haya sido posible encontrar ningún vecino shift o swap que presente mejora, simultáneamente.

Algoritmo Hill Climbing 2 propuesto El Algoritmo 4.3 presenta un pseudocódigo del HC2 propuesto:

Algoritmo 4.3 Algoritmo HC2 para minimización de MRP

```
1  MaximoProcesosSwap //Parámetro
2  s_mejor ← HC1()
3  WHILE time() < Límite_Tiempo DO
4  BEGIN
5      s_candidata ← ObtenerVecinoShiftSwapMejora(MaximoProcesosSwap, s_mejor)
6      IF s_candidata = ∅
7      BEGIN
8          retornar s_mejor
9      ELSE
10         s_mejor ← s_candidata
11         AleatorizarProcesosShiftSwap()
12     END
13 END
14 retornar s_mejor
```

En la línea 2 se establece la mejor solución a partir de la obtenida con HC1. En la línea 5 se obtiene el siguiente vecino a evaluar mediante los movimientos y criterios descritos anteriormente.

A continuación, entre las líneas 6 y 8 se establece que si no se ha encontrado ningún vecino que mejore la función de evaluación, se finaliza con el algoritmo entregando la mejor solución hasta el momento. Por el contrario, entre las líneas 9 y 12 se actualiza la solución actual en caso de haberse encontrado algún vecino que mejore y se aleatorizan las listas de procesos para la siguiente iteración. El algoritmo también finaliza cuando se alcanza el límite de tiempo, entregando la mejor solución encontrada.

4.3.4. Algoritmo Simulated Annealing

La metaheurística Simulated Annealing (SA) [20] está basada en una analogía con las técnicas de calentamiento de los metales. Estos son calentados a altas temperaturas, lo cual permite a las moléculas acomodarse en estados de mínima energía. Posteriormente, los metales son enfriados de manera controlada para reducir los defectos en el material. De manera similar, SA dispone de un mecanismo basado en temperatura que permite aceptar soluciones de peor calidad, y gracias a esto, evita quedar estancado en óptimos locales, logrando así una mejor exploración.

Partiendo de una solución inicial, en cada iteración SA genera un vecino candidato al cual es posible moverse, si la calidad del vecino es mejor que la de la solución actual, entonces el algoritmo acepta la nueva solución. En caso de que la calidad del

vecino sea peor, el algoritmo decide si aceptarla o no en base a una probabilidad de aceptación, que depende tanto de la diferencia de calidad como de la temperatura actual. La probabilidad de aceptación está dada por la siguiente fórmula, para el caso de minimización:

$$P(aceptar_{SolucionCandidata}) = e^{\frac{-\Delta f}{T}}$$

Donde Δf representa el cambio en la función de evaluación entre la solución candidata y la actual, mientras que T es la temperatura actual del algoritmo.

Cuando la temperatura es muy alta, la probabilidad de aceptación de soluciones que empeoran tiende a ser alta, por lo que el algoritmo se comporta como una búsqueda aleatoria, y es posible explorar. Por otro lado, cuando la temperatura es muy baja, la probabilidad de aceptación es muy cercana a cero y el algoritmo tiende a aceptar sólo soluciones con calidad mejor que la actual, comportándose como Hill Climbing.

El Algoritmo 4.4 muestra el pseudocódigo genérico del algoritmo SA.

Se comienza definiendo los valores iniciales de las variables principales además de recibir los parámetros entre las líneas 1 y 5, en las cuales $Iteraciones_{enfriar}$ y T son parámetros. Luego iterativamente se obtiene una solución candidata, si esta tiene una mejor calidad que la solución actual, es aceptada; en caso contrario se evalúa su aceptación en base a la temperatura entre las líneas 12 y 17. Luego entre las líneas 18 y 21 se evalúa si es necesario actualizar la mejor solución. A continuación, entre las líneas 22 y 26 se evalúa si es necesario disminuir la temperatura. El algoritmo realiza las operaciones anteriores hasta finalizar con un criterio de término. Por último, se entrega la mejor solución encontrada.

Para la construcción de un algoritmo SA se deben tener en cuenta los siguientes aspectos:

1. Una solución inicial o punto de partida.
2. Un movimiento para generar las soluciones candidatas: El cual puede ser seleccionado aleatoriamente o aplicando un mecanismo en particular.
3. Un criterio de selección de las soluciones.
4. Temperatura y mecanismos de disminución y recalentamiento: La temperatura será utilizada para la aceptación de soluciones en caso de que estas tengan peor calidad que la solución de ese momento. Por otro lado, el enfriamiento será el mecanismo mediante el cual SA irá disminuyendo la temperatura una vez que ciertas condiciones se cumplan. En el Algoritmo 4.4 $Iteraciones_{enfriar}$ representa la cantidad de iteraciones antes de disminuir la temperatura, la cual es disminuida en la línea 25. El recalentamiento es una opción alternativa, mediante la cual el algoritmo aumenta la temperatura en caso de detectar que está estancado en óptimo local.

Algoritmo 4.4 Algoritmo Simulated Annealing para minimización

```
1  Iteraciones_enfriar //Parámetro
2  T ← TemperaturaInicial //Parámetro
3  s_actual ← SolucionInicial()
4  s_mejor ← s_actual
5  IteracionesDesdeEnfriamiento ← 1
6  WHILE CriterioTermino DO
7  BEGIN
8       s_candidata ← ObtenerSolucionCandidata()
9       IF f(s_candidata) < f(s_actual)
10      BEGIN
11           s_actual ← s_candidata
12      ELSE
13           IF RANDOM(0,1)< e^(-Δf/T)
14           BEGIN
15                s_actual ← s_candidata
16           END
17      END
18      IF f(s_actual) < f(s_mejor)
19      BEGIN
20           s_mejor ← s_actual
21      END
22      IteracionesDesdeEnfriamiento ← IteracionesDesdeEnfriamiento+1
23      IF IteracionesDesdeEnfriamiento = Iteraciones_enfriar
24      BEGIN
25           DisminuirTemperatura(T)
26           IteracionesDesdeEnfriamiento ← 1
27      END
28 END
29 RETURN s_mejor
```

A continuación, se describirán cada una de las partes del algoritmo SA propuesto para resolver el MRP.

4.3.4.1. Simulated Annealing en MRP

El algoritmo SA se ejecutará luego de que el algoritmo HC2 haya quedado estancado en un óptimo local. Dado que SA utilizará los mismos movimientos que HC2, será necesario partir escapando del óptimo local aceptando movimientos que empeoran la función de evaluación.

Adaptando la estructura genérica mostrada del Algoritmo 4.4 para MRP, en el algoritmo SA propuesto, al igual que con HC, se trabajará siempre con soluciones candidatas factibles. El propósito general de este algoritmo es principalmente utilizar la temperatura para salir de zonas del espacio de búsqueda en las cuales HC2 quedaría estancado.

Para esto, la temperatura siempre es establecida usando una relación de distancia entre el valor de la función de evaluación de la solución actual, y un valor óptimo teórico aproximado, determinado por el límite inferior de la función de costo.

Este algoritmo utiliza un mecanismo de enfriamiento y otro de recalentamiento de la temperatura. El enfriamiento se realiza cada cierta cantidad de iteraciones determinada por un parámetro. Por otro lado, el recalentamiento se realiza luego de una cierta cantidad de iteraciones sin que haya ocurrido una mejora en la solución actual, también asociada a un parámetro del algoritmo.

Movimiento El algoritmo usa los mismos movimientos de HC2, es decir, está compuesto por shift y swap de procesos. En cada iteración se generan dos vecinos de la solución actual de manera aleatoria , uno con shift y el otro con swap, ambos sin repetición, y el vecino escogido para pasar al criterio de selección es el mejor de los dos, o swap en caso de empate.

Criterio de selección Luego de generar los vecinos mediante el mecanismo descrito en la Sección 4.3.4.1 se analiza el valor del cambio en la función de evaluación del vecino escogido:

1. Si el cambio es negativo o cero, es decir, mejora o mantiene el valor de la función de evaluación, el vecino es aceptado inmediatamente.
2. Si el cambio es positivo, el vecino es aceptado o no dependiendo de la temperatura, es decir, utilizando la ecuación $P(aceptar_{SolucionCandidata}) = e^{\frac{-\Delta f}{T}}$.

Temperatura, enfriamiento y recalentamiento La temperatura es controlada de la siguiente manera:

1. Temperatura inicial: Dado que SA explora casi el mismo vecindario que HC2, la solución de partida muy probablemente será un óptimo local. Por esto se establece el valor de la temperatura en base a un parámetro $T_{aumento}$ con la siguiente fórmula:

$$T_{i+1} = T_i + T_{aumento} \cdot PuntajeROADEF_{LB}(SoluciónActual)^2$$

   Donde el puntaje ROADEF utilizado es el mismo propuesto en la competencia, pero el "mejor valor obtenido en la competencia" es reemplazado por el límite inferior de la función de costo, es decir:

$$PuntajeROADEF_{LB}(SoluciónActual)^2 = \left[\frac{f(SoluciónActual - (LB_{CostoCarga} + LB_{CostoBalance})}{f(SoluciónInicial)} \cdot 100\right]^2$$

Esta fórmula también es utilizada cada vez que es necesario recalentar. Esto significa que la temperatura siempre aumentará tomando como base el parámetro $T_{aumento}$, el cual será amplificado o disminuído dependiendo de cuán buena sea la solución respecto del límite inferior de la función de costo y de su solución inicial. En caso de que la función de evaluación esté muy cerca del límite inferior, el puntaje ROADEF tomará valores $\in [0, 1]$, con lo cual su valor al cuadrado disminuirá, y $T_{aumento}$ será disminuída. Con esto el algoritmo aceptará soluciones de peor calidad muy cercanas a la solución actual, buscando seguir explotando en lugar de dar grandes saltos a otras regiones del espacio de búsqueda.

Por otro lado, a medida que la función de evaluación de la solución actual tome valores mucho mayores que el límite inferior, el puntaje será varias veces superior a 1, con lo que $T_{aumento}$ se multiplicará varias veces, y la temperatura aumentará sustancialmente, permitiendo de esta manera explorar otras regiones del espacio de búsqueda.

Es importante destacar que la disminución o amplificación de la temperatura $T_{aumento}$ en base al puntaje ROADEF hace que el algoritmo se adapte dinámicamente, independientemente de la instancia que se está tratando de resolver. Esto gracias a que el puntaje ROADEF representa un valor normalizado, el cual no es influido por las diferencias entre los valores de costos de distintas instancias.

2. Recalentamiento: El algoritmo también dispone de un parémetro que señala la cantidad de iteraciones sin mejora en la solución actual, luego de la cual deberá recalentarse. El objetivo del recalentamiento es permitir al algoritmo diversificar la búsqueda hacia otras regiones del espacio.

3. Condición para enfriar: El enfriamiento está controlado por un parámetro llamado $Iteraciones_{enfriar}$, el cual señala la cantidad de iteraciones de SA antes de enfriar la temperatura. Para esto, se utiliza un factor de disminución de la temperatura, especificado por un parámetro llamado $FactorEnfriamiento$, el enfriamiento se lleva a cabo usando la siguiente ecuación de enfriamiento geométrica:

$$T_{i+1} = T_i \cdot FactorEnfriamiento$$

Esta ecuación también ha sido utilizada por [29, 28].

Criterio de término El criterio de término de SA es hasta alcanzar el límite de tiempo especificado por el parámetro $Límite_{tiempo}$.

Algoritmo propuesto A partir de las partes explicadas anteriormente, el Algoritmo 4.5 muestra un pseudocódigo del algoritmo SA propuesto.

Algoritmo 4.5 Algoritmo SMaRT para minimización de MRP

```
1  MaximoProcesosSwap //Parámetro
2  T_aumento                    //Parámetro
3  IteracionesSinMejora_recalentar //Parámetro
4  Iteraciones_enfriar //Parámetro
5  FactorEnfriamiento //Parámetro
6  T ← 0
7  s_actual ← HC2()
8  s_mejor ← s_actual
9  IteracionesDesdeEnfriamiento ← 1
10 IteracionesSinMejora ← 1
11 WHILE time() < Límite_Tiempo DO
12 BEGIN
13     s_candidata ← ObtenerVecinoShiftSwap(MaximoProcesosSwap, s_actual)
14     IF f(s_candidata) < f(s_actual)
15         s_actual ← s_candidata
16         IteracionesSinMejora ← 1
17     ELSE
18         IF RANDOM(0,1) < e^(-Δf/T)
19         BEGIN
20             s_actual ← s_candidata
21             IteracionesSinMejora ← 1
22         END
23         IteracionesSinMejora ← IteracionesSinMejora+1
24         IF IteracionesSinMejora = IteracionesSinMejora_recalentar
25         BEGIN
26             T = T + T_aumento · PuntajeROADEF_LB(s_actual)^2
27             IteracionesSinMejora ← 1
28         END
29     END
30     IF f(s_actual) < f(s_mejor)
31     BEGIN
32         s_mejor ← s_actual
33     END
34     IteracionesDesdeEnfriamiento ← IteracionesDesdeEnfriamiento+1
35     IF IteracionesDesdeEnfriamiento = Iteraciones_enfriar
36     BEGIN
37         T = T · FactorEnfriamiento
38         IteracionesDesdeEnfriamiento ← 1
39     END
40 END
41 RETURN s_mejor
```

Entre las líneas 1 y 5 pueden verse los parámetros del algoritmo, mientras que entre

las líneas 6 y 10 se establecen los valores iniciales de las variables que serán utilizadas.

En la línea 13 se llama a la función $ObtenerVecinoShiftSwap()$, la cual obtendrá el mejor de los vecinos entre shift y swap, y utilizará el parámetro $MáximosVecinosSwap$ para limitar la búsqueda en el vecindario swap.

Por otro lado, entre las líneas 14 y 16 se acepta automáticamente la nueva solución si es que mejora, y entre las líneas 18 y 21 se decide si se aceptará o no si es que la solución candidata no mejora la solución actual.

Entre las líneas 24 y 28 se decide si recalentar el algoritmo en base a las iteraciones sin mejora. Finalmente, entre las líneas 30 y 33 se verifica si se ha mejorado la mejor solución, mientras que entre las líneas 34 y 39 se decide si enfriar o no en base a la cantidad de iteraciones desde el último enfríamiento. El algoritmo finaliza cuando ha transcurrido el límite de tiempo, retornando la mejor solución encontrada.

4.4. Conclusiones

En este capítulo se ha presentado un algoritmo compuesto de dos fases de HC y otra de SA, denominado SMaRT. HC1 tiene como objetivo mejorar una solución inicial dada rápidamente enfocándose en mover los procesos más grandes desde las máquinas más sobrecargadas, mientras que HC2 busca mejorar lo más posible la función de evaluación mediante movimientos simples shift y swap. SA, por otro lado, tiene la finalidad de escapar del óptimo local obtenido en HC2 mediante su mecanismo de temperatura y, usando el mismo movimiento, intenta intensificar la búsqueda en otras regiones. Todos los algoritmos trabajan sólo con soluciones factibles, por lo que se definieron movimientos especialmente diseñados para esto.

Para la realización de todas las fases se ha definido la representación de las soluciones, estrategias de generación de estas, movimientos utilizados y criterios de aceptación. Seguidamente, se ha presentado la función de evaluación utilizada, que permite guiar a los algoritmos en los criterios de selección de soluciones. Específicamente para el algoritmo basado en Simulated Annealing se han especificado además los mecanismos de enfriamiento y recalentamiento de la temperatura, destacándose este último por permitir la adaptabilidad dinámica de la temperatura en base al valor de la función objetivo de la solución actual, y cuán distante está del límite inferior de la función de costo, medido utilizando el valor normalizado de puntaje ROADEF establecido en la competencia. Dicho límite se presenta como un buen indicador de esta distancia, reduciendo de esta forma la influencia por diferencias de costos entre distintas instancias.

En el siguiente capítulo, se mostrarán los resultados experimentales al aplicar el algoritmo propuesto para las instancias de los grupos A, B y X publicadas por ROADEF, además de presentar las comparaciones de dichos resultados con los expuestos en la competencia y sus versiones más recientes.

5 Experimentos y Resultados

En este capítulo se presentan los experimentos realizados, que permiten observar los resultados del algoritmo presentado en el capítulo 4, aplicado a las instancias de los grupos B y X de la competencia. Por otro lado, se muestra un estudio de la función de variación de temperatura propuesta, adaptable en función del límite inferior de la función de costos, contrastándola con el mismo algoritmo utilizando una variación de temperatura no dinámica

Luego se muestra una comparación con los mejores resultados obtenidos en ROADEF. Finalmente, se presentan los resultados comparados con los mejores obtenidos por equipos de la competencia a la fecha y su evaluación con el mejor algoritmo de la competencia en su mejor vesión, utilizando el test de Wilcoxon [15].

5.1. Especificaciones del equipo de trabajo

Las pruebas fueron realizadas en un equipo con las siguientes características:

1. Sistema operativo: Ubuntu 14.04.5
2. Procesador: Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz.
3. Memoria RAM 15 GB.

5.2. Instancias, métodos de evaluación y experimentos

El algoritmo, ejecutado en las instancias del grupo A, obtiene resultados que le hubieran permitido clasificar. Por otro lado, las instancias de los grupos B y X de la fase final, fueron utilizadas en la comparación del algoritmo propuesto versus los resultados de ROADEF y la literatura (para más detalle de las instancias ver Sección 3.2). El algoritmo fue ejecutado con un tiempo límite de 300 segundos, tal como se exige en la competencia.

Como métrica de evaluación se utilizó, para un grupo específico, la suma de los promedios de los puntajes ROADEF asociados a cada instancia. De esta forma se evaluó

a cada equipo en la competencia. La ventaja de usar este valor es que no se ve influenciado por las diferencias de valores de función objetivo de instancias de características distintas (ver Sección 3.1). La función para determinar el puntaje de una instancia es la siguiente:

$$Puntaje(I) = \frac{Costo(S) - Costo(B)}{Costo(R)} \cdot 100$$

Donde, I es la instancia actual en evaluación, R la solución inicial, S la solución encontrada por el algoritmo, y B la mejor solución encontrada entre todos los algoritmos de la competencia. Esto tiene como consecuencia que en caso de encontrar un mejor resultado que en la competencia, el puntaje obtenido será negativo, ya que el costo de referencia $Costo(B)$ sigue siendo el de ROADEF. Este ajuste permite que a la vez de comparar el algoritmo propuesto con el de los equipos S41 y S38 (los dos mejores de la competencia), también se muestre implícitamente la diferencia respecto de la competencia, enriqueciendo de esta manera el análisis.

5.2.1. Sintonización de parámetros

En una primera instancia se utilizó el sintonizador de parámetros ParamILS [17] para calibrar los parámetros del algoritmo SMaRT, para luego utilizar estos parámetros y comparar los resultados con la competencia y literatura. Como métrica de evaluación se utilizó el puntaje y, dado que cada instancia debe ejecutarse durante 5 minutos, se decidió escoger de cada grupo aquellas que presentaron mayores desafíos. Para la ejecución de ParamILS se consideraron valores de parámetros cercanos a los ya obtenidos mediante prueba y error, por considerarse parámetros que ya obtenían resultados cercanos a los mejores de la literatura.

Las instancias consideradas para la sintonización fueron las siguientes:

- Grupo BX: Instancias B_1, B_2, B_3, B_7, X_1, X_2, X_3 y X_7.

Los valores de parámetros del algoritmo considerados para sintonizar fueron los siguientes:

- $max_{máquinas}$ (Asociado a HC1): 10, 16, 17, 25, 33, 41 y 45.
- $max_{procesos}$ (Asociado a HC1): 10, 16, 17, 25, 33, 41 y 45.
- $MaximoProcesosSwap$ (Asociado a HC2 y SA): 500, 700, 1.000, 1.100, 1.200 y 1.300.
- $T_{aumento}$ (Asociado a SA): 770.000, 790.000, 810.000. 830.000, 850.000 y 880.000.
- $IteracionesSinMejora_{recalentar}$ (Asociado a SA): 15.000, 17.500, 18.900, 19.500, 20.000 y 30.000.
- $Iteraciones_{enfriar}$ (Asociado a SA): 4.000, 4.200, 4.400, 4.600 y 4.800.

- $FactorEnfriamiento$ (Asociado a SA): 0,96; 0,965; 0,97; 0.975 y 0,98.

Finalmente los valores escogidos por paramILS fueron: los siguientes:

- $max_{máquinas}$ (Asociado a HC1): 16.
- $max_{procesos}$ (Asociado a HC1): 16.
- $MaximoProcesosSwap$ (Asociado a HC2 y SA): 1.000.
- $T_{aumento}$ (Asociado a SA): 810.000.
- $IteracionesSinMejora_{recalentar}$ (Asociado a SA): 18.900.
- $Iteraciones_{enfriar}$ (Asociado a SA): 4.400.
- $FactorEnfriamiento$ (Asociado a SA): 0,97.

5.2.2. Diseño de experimentos

Una vez escogidos los parámetros, se ejecutó el algoritmo propuesto (presentado en el Capítulo 4) con 31 semillas aleatorias y 300 segundos para cada instancia, promediándose los puntajes obtenidos para cada una.

Por otro lado, se evaluó cómo afecta al comportamiento de SMaRT la función de variación de temperatura propuesta, adaptable en función del límite inferior de la función de costos, contrastándolo con el mismo algoritmo utilizando una variación de temperatura no dinámica. Finalmente se ejecutó en el mismo equipo de trabajo, con las mismas 31 semillas y tiempo máximo de 300 segundos ejecución, los algoritmos del equipo S41[1] [13] y S38[2] [22], también promediándose los resultados en cada instancia, y se comparó con SMaRT usando los parámetros escogidos en la fase de sintonización.

Para el análisis mediante el test de hipótesis de Wilcoxon se utilizaron los resultados con cada semilla en las comparaciones uno a uno, comparando el mejor de los algoritmos propuestos en este trabajo con el del equipo S41.

5.3. Resultados experimentales

A continuación, se muestran los resultados de los experimentos realizados al algoritmo SMaRT. En primer lugar, se muestra una comparación entre el método dinámico propuesto con respecto a un enfoque estático tradicional. Posteriormente, se muestran las comparaciones respecto de los mejores resultados de la competencia ROADEF.

[1]Disponible bajo licencia opensource en: https://github.com/harisgavranovic/roadef-challenge2012-S41

[2]Disponible bajo licencia opensource en: https://sourceforge.net/projects/machinereassign/

Finalmente, se compara el algoritmo con los mejores propuestos en la literatura, correspondientes a los equipos S41 y S38, presentando los resultados del test de wilcoxon de a pares.

5.3.1. Evaluación de método de variación de temperatura

A continuación, se muestran los resultados sobre algunas de las instancias más representativas de los grupos B y X. Aquí, se puede apreciar de manera más directa cómo funciona el control adaptativo del parámetro de temperatura mediante el ajuste dinámico propuesto, en relación con un ajuste no dinámico más tradicional. Cada gráfico muestra como varía la temperatura en función del tiempo de ejecución. En las imágenes, los valores Sc y Sc_{LB} representan el puntaje en base al mejor valor de la competencia ROADEF y el puntaje en base al límite inferior de la función de costo total, respectivamente. Estos valores son los que fueron logrados antes del recalentamiento de temperatura. La temperatura dinámica se calculó usando la ecuación (Sección 4.3.4.1):

$$T_{i+1} = T_i + T_{aumento} \cdot PuntajeROADEF_{LB}(SoluciónActual)^2$$

En el caso del ajuste de temperatura no dinámico, la función de cálculo de temperatura se redujo a una más simple, buscando eliminar el efecto amplificador (o simplificador) que tiene el límite inferior del puntaje ROADEF sobre $T_{aumento}$:

$$T_{i+1} = T_i + T_{aumento}$$

Se estableció para ambas ecuaciones el valor de temperatura en T=810.000, que fue el obtenido por el sintonizador de parámetros. Las figuras han sido agrupadas de acuerdo a la similaridad del comportamiento del algoritmo, tomando en cuenta las características de las instancias. Los casos que se presentan son representativos de las instancias dos grupos (B y X).

En general, se puede ver que el primer recalentamiento se hace a una temperatura mucho mayor usando el enfoque dinámico. Esto se debe a que el puntaje en el primer recalentamiento, generalmente, es superior a 1, por lo que el puntaje elevado al cuadrado aumenta varias veces la temperatura en la fórmula de recalentamiento dinámico. En los siguientes recalentamientos, el enfoque dinámico buscará adaptar la temperatura en base a cuán distante se esté del puntaje calculado en base al límite inferior, buscando diversificar cuando esté lejos e intensificar cuando esté más cerca.

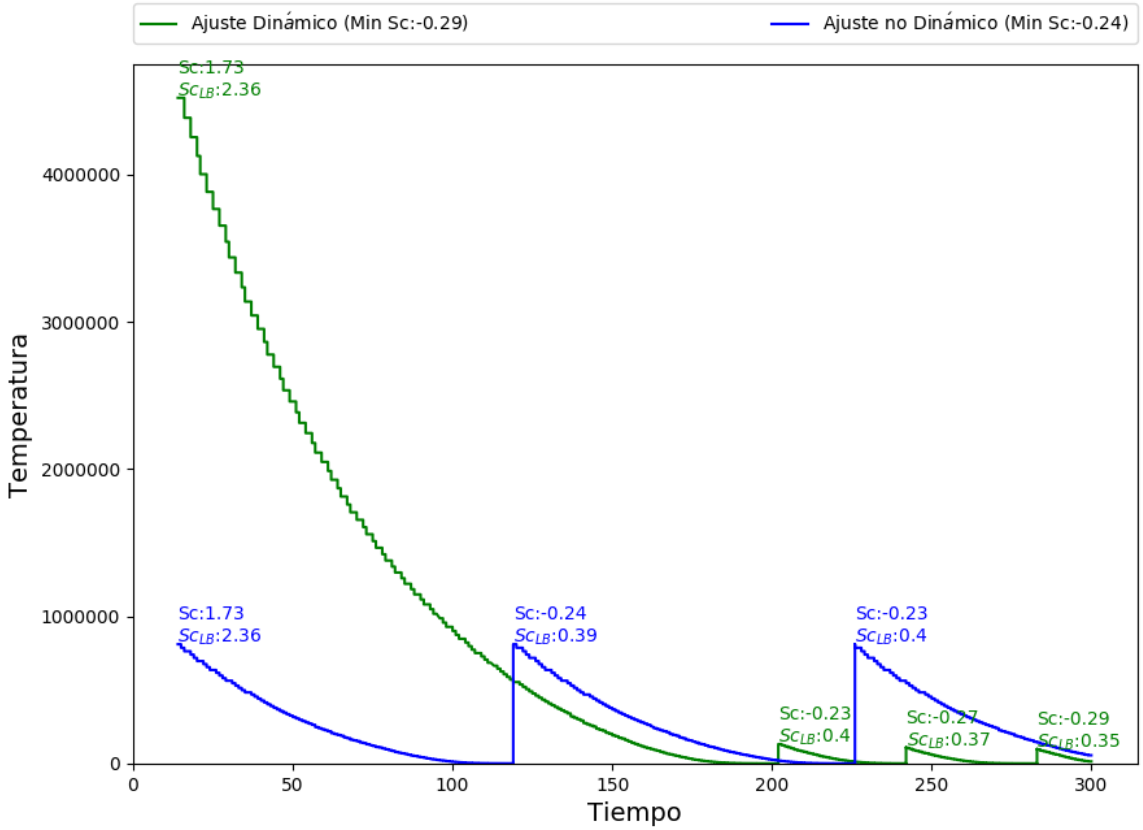


Figura 5.1: Variación de la temperatura respecto del tiempo de ejecución. Instancia B_1.

En la Figura 5.1 y la Figura 5.2 se muestran los resultados en las instancias B_1 y X_1, las cuales tienen características muy similares. En este caso, el límite inferior de la función de costo y el mejor valor ROADEF están en el mismo orden de magnitud, por lo que las temperaturas de ambos ajustes se mantienen en el mismo número de cifras significativas. Puede verse en ambas instancias, que aproximadamente en el segundo 20 es cuando terminó de ejecutarse HC2, ya que es aquí cuando se realiza el primer recalentamiento. Se aprecia que el primer incremento de temperatura es muy elevado en el caso del ajuste dinámico. Luego de esto, el puntaje respecto del límite inferior señala que se está muy cerca de los valores mínimos de la función objetivo, por lo que es más conveniente seguir intensificando. Así, en los siguientes recalentamientos se realizan pequeños aumentos de temperatura que permiten al algoritmo intensificar, y encontrar mejores valores de función objetivo. Por otro lado, si bien el ajuste no dinámico logra obtener un puntaje negativo, este no es tan bueno como el obtenido por el enfoque dinámico, diferencia que se acentúa mucho más si se observa la figura Figura 5.2.

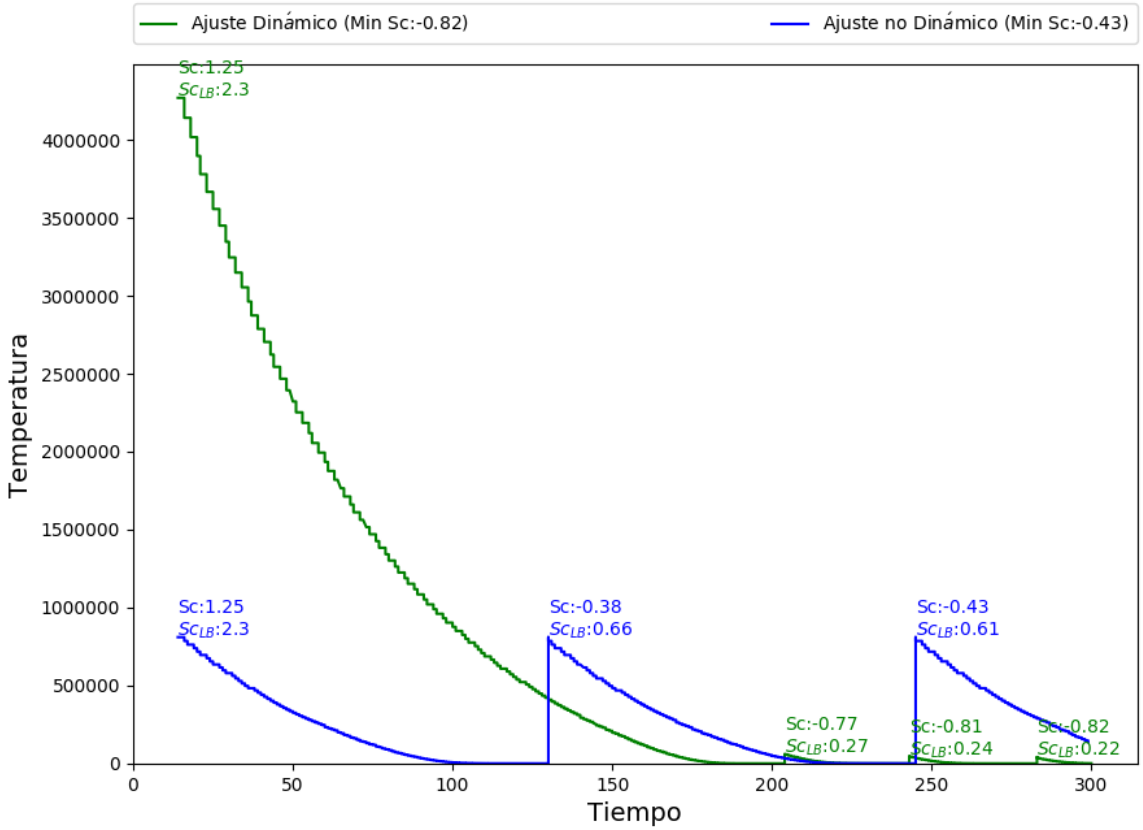


Figura 5.2: Variación de la temperatura respecto del tiempo de ejecución. Instancia X_1.

En la Figura 5.3, de manera similar a lo ocurrido en las instancias anteriores, puede verse que hasta el segundo 80, aproximadamente, el algoritmo HC2 estuvo en ejecución. Luego de esto, el valor de los puntajes en base al límite inferior y ROADEF era aproximadamente 0,27. En el caso del ajuste no dinámico, el incremento de temperatura es demasiado grande y lleva al algoritmo a explorar demasiado, por lo que entre el segundo 80 y 300 sólo logra mejorar el valor de los puntajes en 0,01, aproximadamente. Por otro lado en el ajuste dinámico, al tenerse un puntaje SC_{LB} menor que 1, la temperatura se multiplicará por $(SC_{LB})^2 \ll 1$, llevando al algoritmo a incrementos de temperatura muy pequeños que le ayudarán a tratar de explotar. Como resultado de esto, puede verse que los incrementos de temperatura del ajuste dinámico son cada vez más pequeños, y el Puntaje ROADEF logra mejorar desde 0,27 en el segundo 80 aproximadamente, hasta 0,04 en el segundo 300.

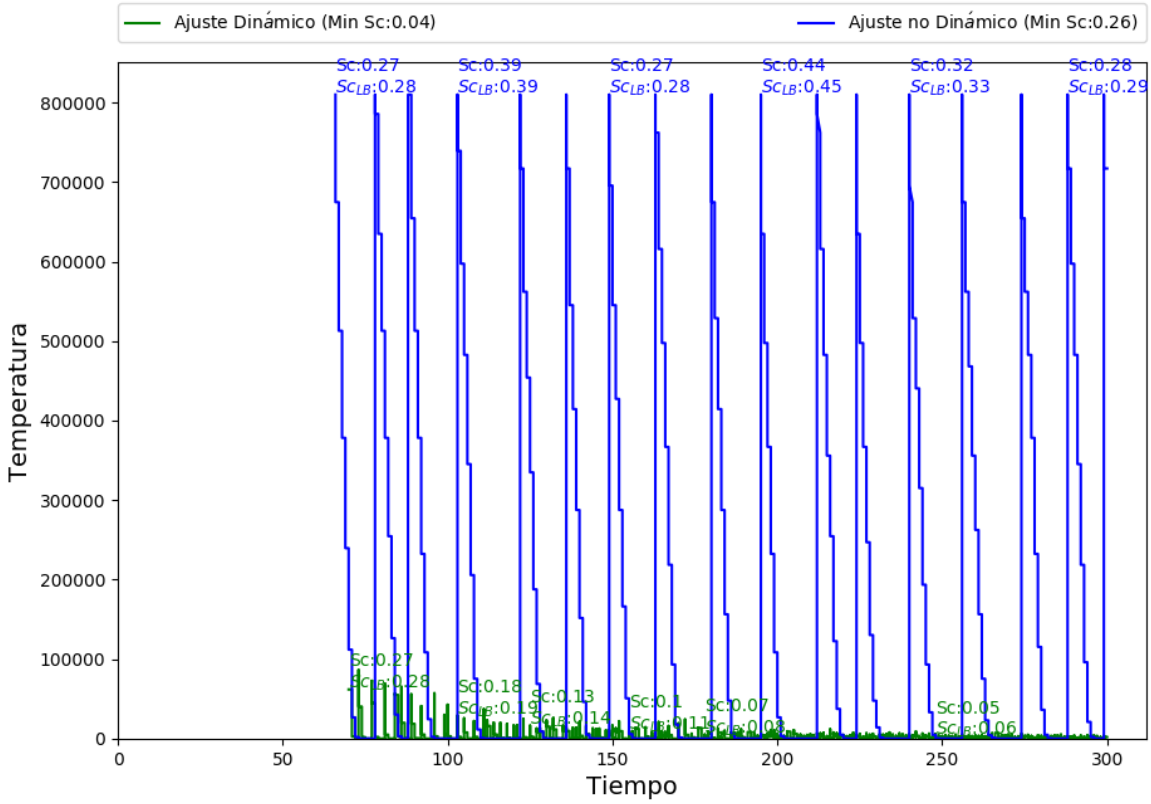


Figura 5.3: Variación de la temperatura respecto del tiempo de ejecución. Instancia B_2.

En la Figura 5.4 puede verse que, esta vez, el algoritmo HC2 ha terminado en el segundo 250 aproximadamente. Este comportamiento ocurre en 12 de las instancias más grandes del conjunto de instancias BX (con entre 20.000 y 50.000 procesos), en las cuales la combinación HC1+HC2 consume casi la totalidad del tiempo, y cuyos puntajes en cada una de estas 10 instancias no supera el valor 0,001. Es más, la suma de los puntajes promedio de 31 semillas de todas las instancias de la fase final fue sólo de 6,45, con lo cual se hubiese obtenido el 13° lugar.

La razón de que esta combinación de algoritmos obtenga buenos resultados, es debido a que el algoritmo HC1 logra una rápida mejora de la función objetivo, lo cual ayuda a HC2 a partir desde un valor mucho menor y contrarresta la lenta convergencia de este en las instancias más grandes.

En este tipo de casos, al terminar el algoritmo HC1+HC2 el valor de la función objetivo es muy bajo y cercano a los límites inferiores. En la Figura 5.4 se aprecia que tanto el ajuste dinámico como el no dinámico obtienen los mismos resultados.

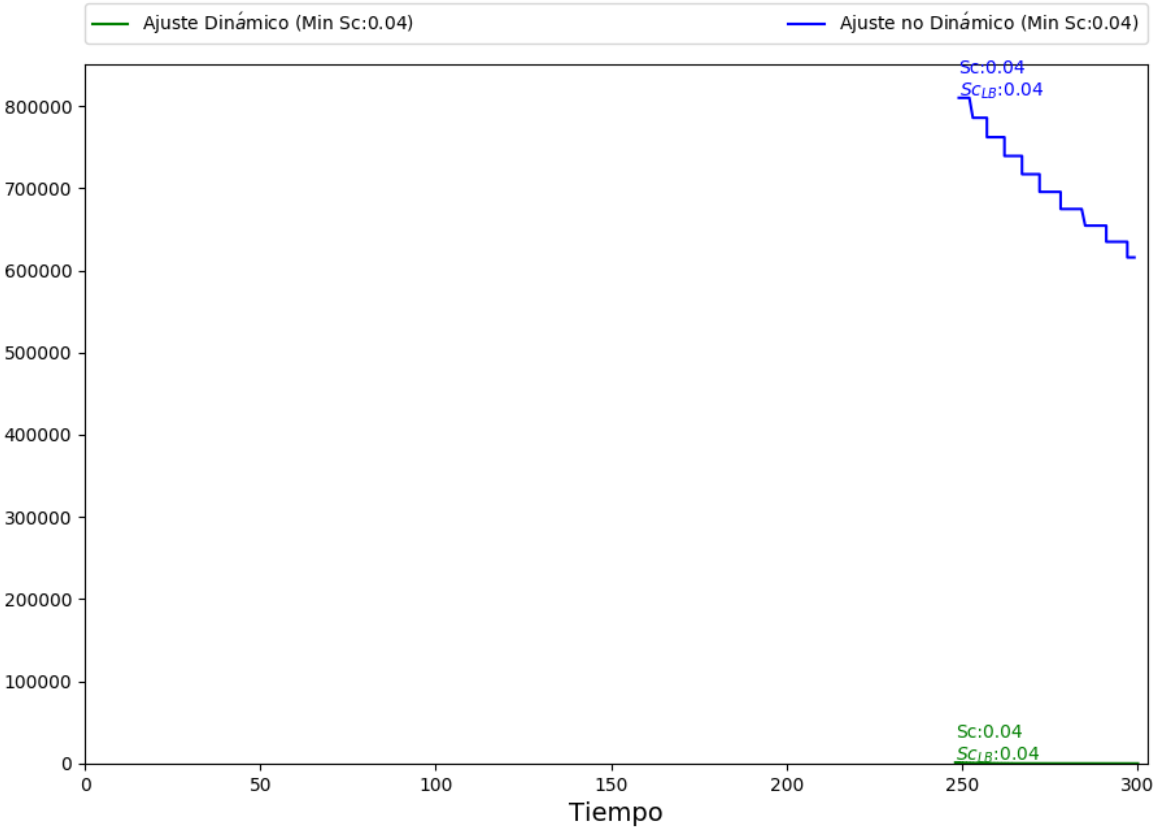


Figura 5.4: Variación de la temperatura respecto del tiempo de ejecución. Instancias B_5

5.3.2. Comparación con los mejores resultados de ROADEF

Los resultados de ROADEF tuvieron lugar en el año 2012, en que se realizó la competencia, y se presentan como un muy buen referente en calidad de soluciones.

Dado que ningún equipo dominó en todas las instancias, los mejores puntajes mostrados no corresponden todos a un solo equipo. Es por esto que también es de utilidad tomar como referencia el puntaje total de los primeros equipos en cada fase, que se muestran en la Tabla 5.1.

Inst.	1°	2°	3°	4°	5°
Fase Clasificación (Instancias A)	1,73 (J17)	3,58 (S25)	5,92 (S38[24])	9,55 (S23 [29])	10,97 (S21)
Fase Final (Instancias B y X)	0,47 (S41 [14])	0,62 (S38 [24])	1,72 (J12 [18])	2,60 (J25 [3])	3,91 (S14)

Tabla 5.1: Mejores 5 puntajes totales para cada fase.

	Mejor ROADEF	Promedio SMaRT		Mejor SMaRT		Peor SMaRT	
Inst.	FO	FO	Ptje	FO	Ptje	FO	Ptje
B_1	3.339.186.879	3.331.636.980	**-0,099**	3.307.834.020	**-0,410**	3.346.269.668	0,093
B_2	1.015.553.800	1.017.433.717	0,036	1.015.737.594	0,004	1.023.968.682	0,162
B_3	156.835.787	157.745.897	0,014	156.954.657	0,002	161.511.011	0,074
B_4	4.677.823.040	4.677.839.013	0,000	4.677.836.082	0,000	4.677.841.783	0,000
B_5	923.092.380	923.571.048	0,004	923.164.298	0,001	925.179.715	0,017
B_6	9.525.857.752	9.525.875.819	0,000	9.525.872.113	0,000	9.525.881.250	0,000
B_7	14.835.149.752	14.839.939.397	0,013	14.837.729.106	0,007	14.842.870.156	0,020
B_8	1.214.458.817	1.214.586.177	0,001	1.214.524.830	0,000	1.214.834.914	0,003
B_9	15.885.486.698	15.885.494.399	0,000	15.885.489.036	0,000	15.885.503.216	0,000
B_10	18.048.515.118	18.050.104.654	0,004	18.049.486.933	0,002	18.051.600.120	0,007
X_1	3.100.852.728	3.044.235.737	**-0,763**	3.036.116.625	**-0,872**	3.051.611.633	-0,663
X_2	1.002.502.119	1.004.505.802	0,039	1.003.177.420	0,013	1.011.746.985	0,181
X_3	211.656	1.168.585	0,016	362.736	0,002	3.059.316	0,047
X_4	4.721.629.497	4.721.651.064	0,000	4.721.644.289	0,000	4.721.656.075	0,000
X_5	93.823	165.875	0,001	150.492	0,000	184.078	0,001
X_6	9.546.941.232	9.546.959.553	0,000	9.546.955.937	0,000	9.546.962.780	0,000
X_7	14.253.273.178	14.265.996.983	0,034	14.255.873.223	0,007	14.275.325.679	0,058
X_8	42.674	77.520	0,000	68.961	0,000	83.875	0,000
X_9	16.125.612.590	16.125.628.337	0,000	16.125.623.667	0,000	16.125.636.321	0,000
X_10	17.816.514.161	17.820.028.260	0,008	17.817.696.217	0,003	17.823.791.123	0,017
		Suma BX	**-0,691**		**-1,240**		**0,018**

Tabla 5.2: Comparaciones de los mejores resultados con los mejores de ROADEF en los grupos de instancias B y X.

En la Tabla 5.2 se presenta el detalle de los resultados de la competencia, así como los resultados obtenidos con SMaRT sobre los grupos de instancias B y X, respectivamente. Se muestran los mejores valores de función objetivo presentados en ROADEF, los valores promedio, mínimo y máximo obtenidos con SMaRT, y sus respectivos puntajes. En general, se aprecia que SMaRT logra obtener buenos resultados, permitiendo mejorar el mejor valor encontrado para las instancias B_1 y X_1, obteniendo valores negativos. Observando los peores valores de SMaRT, se destaca que estos se mantuvieron cerca del promedio.

Además de lo anterior, se muestra en la Tabla 5.3 las comparaciones de los primeros cinco equipos versus los resultados del algoritmo propuesto.

Inst.	S41	S38	J12	J25	S14	Prom SMaRT	Mejor SMaRT	Peor SMaRT
B_1	0,409	0,000	0,916	0,958	1,273	**-0,099**	**-0,410**	0,093
B_2	0,000	0,156	0,001	0,236	0,034	0,036	0,004	0,162
B_3	0,013	0,009	0,001	0,098	0,115	0,014	0,002	0,074
B_4	0,002	0,000	0,002	0,000	0,001	0,000	0,000	0,000
B_5	0,009	0,004	0,001	0,115	0,344	0,004	0,001	0,017
B_6	0,000	0,000	0,001	0,000	0,001	0,000	0,000	0,000
B_7	0,000	0,011	0,023	0,053	0,085	0,013	0,007	0,020
B_8	0,001	0,001	0,056	0,017	0,087	0,001	0,000	0,003
B_9	0,001	0,001	0,001	0,003	0,001	0,000	0,000	0,000
B_10	0,000	0,015	0,006	0,028	0,001	0,000	0,002	0,007
X_1	0,000	0,055	0,645	0,489	1,435	**-0,763**	**-0,872**	-0,663
X_2	0,019	0,242	0,018	0,310	0,046	0,039	0,013	0,181
X_3	0,003	0,008	0,000	0,065	0,063	0,016	0,002	0,047
X_4	0,002	0,000	0,003	0,001	0,001	0,000	0,000	0,000
X_5	0,001	0,000	0,000	0,002	0,168	0,001	0,000	0,001
X_6	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
X_7	0,005	0,098	0,032	0,127	0,000	0,034	0,007	0,058
X_8	0,001	0,000	0,000	0,001	0,259	0,000	0,000	0,000
X_9	0,001	0,001	0,001	0,002	0,000	0,000	0,000	0,000
X_10	0,000	0,024	0,013	0,094	0,001	0,008	0,003	0,017
Suma BX	0,47	0,62	1,72	2,60	3,91	**-0,691**	**-1,240**	**0,018**

Tabla 5.3: Comparaciones de los cinco primeros equipos con el algoritmo propuesto en los grupos de instancias B y X.

Puede verse que ningún algoritmo domina en todas las instancias. Las instancias más difíciles de resolver en general fueron: B_1, B_3, B_5, B_7, X_1, X_2, en las cuales sólo un grupo logró obtener el puntaje óptimo para esa instancia en la competencia (ninguno en el caso de la X_2). Las instancias más fáciles fueron: B_4, B_6, B_9, X_4, X_5, X_6, X_8 y X_9, en donde al menos tres grupos lograron obtener los puntajes óptimos para la competencia. En las instancias más fáciles, el algoritmo propuesto mostró estar a la par con los demás algoritmos, logrando en todas los puntajes mínimos. En las más difíciles logra en todas obtener puntajes por debajo de la mayoría de los equipos, y especialmente en la B_1 y X_1, que son las dos instancias más difíciles de cada grupo, en donde logra superar ampliamente a todos los equipos. Sin embargo, existen instancias en que el algoritmo propuesto en esta Tesis fue superado por los otros algoritmos.

5.3.3. Comparación con los mejores resultados de la literatura

5.3.3.1. Boxplots

En 2012 el equipo S41 fue el equipo ganador de la competencia. Su algoritmo basado en búsqueda de vecindarios variables ha logrado mostrar muy buenos resultados, logrando varios de los mejores puntajes encontrados en esa ocasión [14], con una suma de puntajes de los grupos de instancias BX de $0,47$. En 2014, los mismos autores publicaron otro trabajo, donde se muestran mejoras del algoritmo, además de un mejor manejo de las semillas aleatorias y sintonización de parámetros [13]. Esta propuesta trabaja con dos núcleos del procesador, estableciendo un mecanismo de búsqueda dual en donde gana la mejor de las dos soluciones encontradas. Otro equipo con buenos resultados es el equipo S38, que en la competencia presentó un algoritmo de búsqueda de vecindarios variables, proponiendo un enfoque de división en subproblemas [24]. Este equipo estuvo muy cerca de ganar en esa ocasión, obteniendo una suma de los puntajes de los grupos de instancias BX igual a $0,62$. Su propuesta trabaja con un solo núcleo del procesador. Posteriormente, estos autores presentaron mejoras a su algoritmo en [22]. Así, ambos equipos logran encontrar los mejores resultados presentados la literatura hasta el momento.

A continuación, se presentan los resultados para comparar el rendimiento del algoritmo propuesto, SMaRT, versus los del equipo S41 y S38 en los grupos de instancias B y X. Solo se muestran aquellas instancias en donde las diferencias en los puntajes fueron superiores a 0,003, ya que las diferencias en las demás se consideraron despreciables.

Puede verse en la Figura 5.5 que las mayores diferencias del grupo B se encuentran en la instancia B_1, en la cual el algoritmo del equipo S41 tuvo un puntaje promedio de $0,13$ con una varianza moderadamente grande, comparado con los demás resultados. Por su parte, en esta instancia el equipo S38 logró obtener un puntaje promedio de $-0,025$, con una varianza muy pequeña cercana a cero, mientras que el algoritmo SMaRT promedió $-0,099$. En las demás instancias se aprecia que en su mayoría los tres algoritmos logran resultados muy similares, especialmente en las instancias B_3, B_7 y B_10 todos obtuvieron valores muy cercanos a 0 con varianzas muy pequeñas. En la instancia B_2 pueden verse algunas diferencias, en donde el algoritmo del equipo S41 es el que logra los mejores resultados, SMaRT logra posicionarse muy cerca con varianza despreciable y promedio 0,036, mientras que algoritmo del equipo S38 obtiene resultados cercanos a $0,13$.

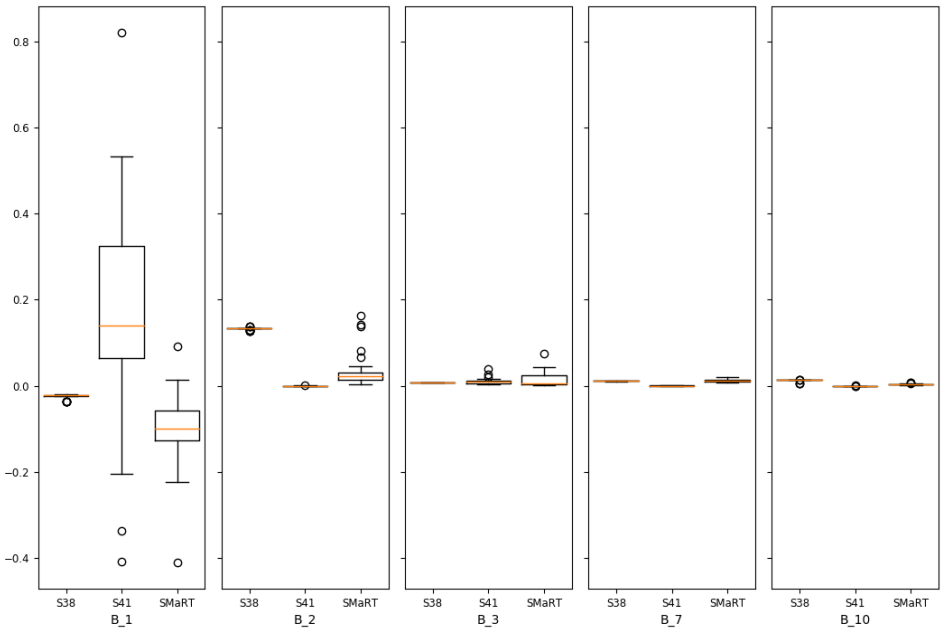


Figura 5.5: Comparación de resultados algoritmo SMaRT versus S41 y S38. Grupo B.

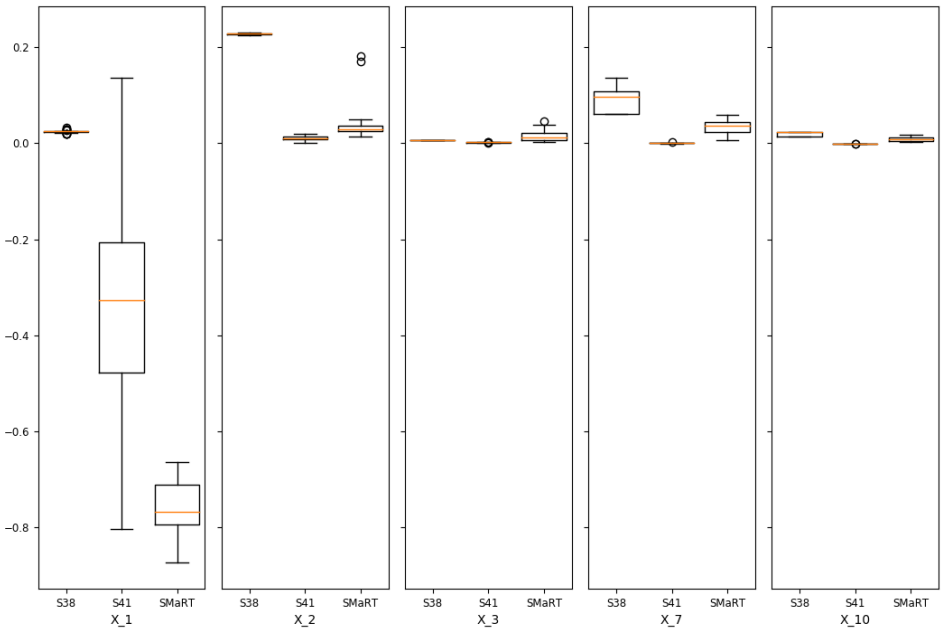


Figura 5.6: Comparación de resultados algoritmo SMaRT versus S41 y S38. Grupo X.

En la Figura 5.6, de manera muy similar a los resultados anteriores, se observa que las diferencias más significativas entre los algoritmos ocurren en la instancia X_1, en donde el equipo S38 obtiene resultados levemente superiores a 0 con varianza muy pequeña. El algoritmo del equipo S41, por su parte, obtiene valores cuyo promedio es de $-0,31$, pero al igual que sus resultados en la instancia B_1, su varianza es muy significativa en relación a los demás resultados. SMaRT obtiene un delta con varianza pequeña y promedio de $-0,763$. En las demás instancias se aprecia que los valores de X_3 y X_10 son muy similares y cercanos a cero, manteniendo todos varianzas muy pequeñas. En la instancia X_3 se puede ver que los algoritmos S41 y SMaRT obtienen resultados cercanos a cero, mientras que el del algoritmo S38 obtiene resultados promediando el valor $0,22$. Algunos resultados más diferenciados pueden verse para la instancia X_7, en donde los mejores valores los obtiene el equipo S41 rondando el valor cero. El algoritmo SMaRT se posiciona con una varianza un poco más grande con valores cercanos al $0,03$, mientras que el equipo S38 obtiene $0,09$ con una varianza ligeramente más grande.

5.3.3.2. Análisis Estadístico

Se utilizó el test de Wilcoxon de a pares para analizar si es que existen diferencias significativas entre los resultados obtenidos en sus valores de puntaje ROADEF por los tres algoritmos para los grupos de instancias B y X por separado.

Los resultados de la Tabla 5.4 muestran el test de Wilcoxon entre el algoritmo del equipo S41 y SMaRT. Puede verse que para los grupos B y X el p-value es $< 0,05$, por lo que con un 95 % de confianza se tiene que, dado que el número de rangos positivos es 157 y 158 respectivamente, SMaRT obtiene resultados significativamente superiores a S41.

Instancias	Algoritmos	Media	Desviación estándar	Min	Max
Grupo B	SMaRT	-0,003	0,046	-0,410	0,162
	S41	0,018	0,093	-0,409	0,820
Grupo X	SMaRT	-0,066	0,234	-0,872	0,181
	S41	-0,030	0,120	-0,804	0,137

Instancias	Comparación	S41 - SMaRT	N	Rango Promedio	Suma Rangos	Estadístico p
Grupo B	S41<SMaRT	Rangos negativos	153	185,55	28.389,00	0,007
	S41>SMaRT	Rangos positivos	157	126,22	19.816,00	
	S41=SMaRT	Empates	0			
		Total	310			
Grupo X	S41<SMaRT	Rangos negativos	152	183,11	27.833,00	0,018
	S41>SMaRT	Rangos positivos	158	128,94	20.372,00	
	S41=SMaRT	Empates	0			
		Total	310			

Tabla 5.4: Resultados del test de Wilcoxon para los algoritmo S41 y SMaRT

Instancias	Algoritmos	Media	Desviación estándar	Min	Max
Grupo B	SMaRT	-0,003	0,046	-0,410	0,162
	S38	0,014	0,041	-0,036	0,137
Grupo X	SMaRT	-0,066	0,234	-0,872	0,181
	S38	0,037	0,070	0,000	0,230

Instancias	Comparación	S38 - SMaRT	N	Rango Promedio	Suma Rangos	Estadístico p
Grupo B	S38<SMaRT	Rangos negativos	148	103,71	15.349,00	0,000
	S38>SMaRT	Rangos positivos	162	202,81	32.856,00	
	S38=SMaRT	Empates	0			
		Total	310			
Grupo X	S38<SMaRT	Rangos negativos	151	84,68	12.786,00	0,000
	S38>SMaRT	Rangos positivos	159	222,76	35.419,00	
	S38=SMaRT	Empates	0			
		Total	310			

Tabla 5.5: Resultados del test de Wilcoxon para los algoritmo S38 y SMaRT

En la Tabla 5.5 se aprecia el test de Wilcoxon para los algoritmos S38 y SMaRT. Puede verse que para los dos grupos de instancias el estadístico de prueba es $0,000 < 0,05$, lo que señala que de acuerdo a los rangos positivos 162 y 159, el algoritmo SMaRT obtiene resultados significativamente mejores que S38 en ambos grupos de instancias.

En la Tabla 5.6 se muestra como información adicional los resultados del test de Wilcoxon entre los algoritmos de los equipos S38 y S41, en donde al observar los estadísticos de prueba y rangos positivos, se puede concluir que el algoritmo de S41 obtiene significativamente mejores resultados en ambos grupos de instancias.

Instancias	Algoritmos	Media	Desviación estándar	Min	Max
Grupo B	S41	0,018	0,093	-0,409	0,820
	S38	0,014	0,041	-0,036	0,137
Grupo X	S41	-0,030	0,120	-0,804	0,137
	S38	0,037	0,070	0,000	0,230

Instancias	Comparación	S41 - S38	N	Rango Promedio	Suma Rangos	Estadístico p
Grupo B	$S41 < S38$	Rangos negativos	181	169,24	30.632,00	0,000
	$S41 > S38$	Rangos positivos	129	136,22	17.573,00	
	$S41 = S38$	Empates	0			
		Total	310			
Grupo X	$S41 < S38$	Rangos negativos	182	208,05	37.865,00	0,000
	$S41 > S38$	Rangos positivos	128	80,78	10.340,00	
	$S41 = S38$	Empates	0			
		Total	310			

Tabla 5.6: Resultados del test de Wilcoxon para los algoritmo S38 y S41

5.4. Conclusiones

En este capítulo se presentaron los resultados de los experimentos realizados sobre el algoritmo propuesto, SMaRT.

En primer lugar, mediante los análisis mostrados sobre las diferencias de recalentamiento dinámico y no dinámico, se pudo constatar que la estrategia de variación dinámica permite, en la mayoría de los casos, guiar al algoritmo hacia una mayor exploración y explotación en los momentos adecuados, previniéndole de gastar demasiado tiempo usando temperaturas excesivamente elevadas. Además, en 12 de las 20 instancias totales, la sola combinación de algoritmos HC1+HC2 logra excelentes resultados, mostrando tener una excelente sinergia para instancias de gran tamaño. También es destacable que esta sencilla combinación de algoritmos pueda obtener un puntaje de 6,45 en todas las instancias de la fase final. En las instancias más complicadas, la fase de SA logró hacer que el algoritmo obtuviese mejores resultados aún, escapando de óptimos locales y graduando la explotación y exploración mediante su mecanismo de temperatura.

El test de hipótesis de Wilcoxon realizado de a pares muestra que, frente a los algoritmos de los equipos S38 y S41, el algoritmo propuesto obtiene significativamente mejores resultados en ambos grupos de instancias.

En base a lo anterior, el algoritmo propuesto se presenta como una opción robusta, adaptable y simple.

En el próximo capítulo, se detallan las conclusiones generales y trabajo futuro de este trabajo de tesis.

6 Conclusiones

En esta tesis se presentó el Machine Reassignment Problem, problema de complejidad NP-Hard que fue presentado por Google para el ROADEF-Euro Challenge 2012. En el capítulo 2 se describió el problema en detalle, sus componentes, restricciones y objetivos. Además, se mostraron tanto problemas relacionados anteriores a este, como variaciones que han surgido luego de su presentación, mostrándose que no se trata de un problema aislado, sino que incluso posee variantes que añaden mayor complejidad, trayendo consigo nuevos desafíos.

En el capítulo 3, se presentaron los avances en la resolución del problema más relevantes, comenzando por aquellos presentados en la misma competencia, como aquellos que han sido realizados posteriormente. Además, se mostraron las diferentes técnicas de resolución existentes en la literatura y cuáles fueron algunos de los enfoques, heurísticas, algoritmos y modelos propuestos. Se destaca que aún existen publicaciones muy recientes, mostrando mejoras a los resultados obtenidos inicialmente en la competencia y a la vez proponiendo nuevos desafíos, como la búsqueda de soluciones en un marco de tiempo de 100 segundos de ejecución.

Los avances en la resolución del MRP desde su propuesta en 2012 han dado lugar a que las instancias propuestas por ROADEF sean resueltas en tiempos cada vez menores y con mejores resultados, lo que podría dar lugar en el futuro a la generación de nuevas instancias que presenten mayores desafíos tanto en complejidad como en tamaño. A partir de los métodos de resolución vistos en la literatura, se puede concluir que existe una tendencia hacia el uso de Búsqueda Local y Large Neighborhood Search.

En base a la metaheurística propuesta, que corresponde a un enfoque colaborativo entre dos Hill Climbing y un Simulated Annealing, se puede concluir que con este algoritmo es posible resolver las instancias de manera rápida y eficiente, logrando encontrar en 300 segundos, con un algoritmo secuencial, valores comparables con los mejores encontrados en la literatura. El algoritmo, además, es robusto, logrando obtener buenos resultados incluso en los peores casos. HC1 tiene como objetivo mejorar una solución inicial dada rápidamente, enfocándose en mover los procesos más grandes desde las máquinas más sobrecargadas con el movimiento SDES, mientras que HC2 busca mejorar lo más posible la función de evaluación mediante movimientos simples shift y swap. Resulta destacable el hecho de que en gran parte de las instancias probadas, la sola combinación de ambos HC es capaz de obtener resultados de muy buena calidad y altamente competitivos. Esto se debe principalmente a la utilización del movimien-

to de procesos pesados en HC1, que permite balancear la carga entre las máquinas, logrando una disminución muy rápida de la función objetivo. Con esto, HC2 cuenta con un punto de partida mejorado, con lo que se contrarresta su lenta convergencia en las instancias de mayor tamaño. Finalmente, la fase de SA permite encontrar mejoras adicionales, contrarrestando los problemas de estancamiento que tiene.

Para mejorar aún más el algoritmo SA, se utilizó un enfoque de temperatura dinámico, que utiliza el valor de la función objetivo y la distancia respecto al límite inferior de la función de costo, medido con el valor normalizado del puntaje ROADEF establecido en la competencia. Esto demostró tener un efecto positivo en los resultados, permitiendo así guiar al algoritmo hacia una mayor exploración y explotación en los momentos adecuados. Esto previene que se gaste demasiado tiempo usando temperaturas excesivamente elevadas, con lo que el algoritmo es capaz de converger con una mayor rapidez, logrando así encontrar excelentes resultados dentro del límite de tiempo establecido.

Con esto se ha logrado demostrar que el trabajo colaborativo entre tres algoritmos basados en metaheurísticas simples son suficientes en este problema para obtener resultados muy satisfactorios.

Trabajo Futuro

A partir del trabajo de investigación, los algoritmos y herramientas propuestos y los resultados obtenidos, es posible dar lineamientos para desarrollo futuro que consideren lo actualmente realizado. Algunas heurísticas posibles de implementar y que agregan variabilidad a la solución inicial, son las presentadas por [10], en donde se especifica que además de ordenar los procesos respecto de la suma de sus requerimientos, es posible hacerlo utilizando algunas funciones de la literatura existente del Bin Packing. Un área de desarrollo podría ser la utilización de un algoritmo para generar nuevas instancias que puedan ser utilizadas en nuevas investigaciones relacionadas a este problema.

Una característica que podría aportar grandes mejoras en los tiempos de cómputo del algoritmo, podría ser la utilización de la matriz de asignaciones factibles mencionada en la descripción del algoritmo para no solo descartar aquellos procesos que deberán ser fijados a su máquina inicial, sino también para ordenar la lista de procesos en base a la cantidad de máquinas a las cuales pueden ser asignados desde la perspectiva de la matriz, brindando así otro método para asignar aquellos procesos más conflictivos primero, y evitar regiones infactibles del espacio de búsqueda.

Una de las restricciones más difíciles de abordar es la que tiene relación con los vecindarios de los procesos asociados a un servicio. El movimiento de procesos se ve seriamente limitado por esta restricción, ya que a medida que el algoritmo avanza y se van llenando los vecindarios, se hace cada vez más difícil asignar otros procesos de un servicio al mismo vecindario en el que se asignaron los primeros, por causa de la

restricción de capacidad. Una idea es tomar en cuenta el nivel de capacidad disponible de cada vecindario, y priorizar la asignación de procesos a máquinas cuyos vecindarios tengan un mayor espacio disponible.

Finalmente, otra característica que podría estudiarse para mejorar los resultados es la utilización de "ventanas deslizantes", es decir, permitir la asignación de procesos a un subconjunto específico de máquinas, e ir "moviendo" este margen para ir considerando otras máquinas. Esto tiene la ventaja de que, además de dar la oportunidad a máquinas que quizás no se estaban utilizando, permite disminuir considerablemente los tiempos de ejecución al no considerar todas las máquinas.

Bibliografía

[1] EPA: US Environmental Protection Agency. Sources of greenhouse gas emissions. 2016. Accedido: 24-07-2016.

[2] Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *Principles and Practice of Constraint Programming-CP 2009*, pages 142–157. Springer, 2009.

[3] Felix Brandt, Jochen Speck, and Markus Völker. Constraint-based large neighborhood search for machine reassignment. *Annals of Operations Research*, 242(1):63–91, 2016.

[4] Franck Butelle, Laurent Alfandari, Camille Coti, Lucian Finta, Lucas Létocart, Gérard Plateau, Frédéric Roupin, Antoine Rozenknop, and Roberto Wolfler Calvo. Fast machine reassignment. *Annals of Operations Research*, pages 1–28, 2015.

[5] Ricardo Stegh Camati, Alcides Calsavara, and Luiz Lima Jr. Solving the virtual machine placement problem as a multiple multidimensional knapsack problem. *ICN 2014*, page 264, 2014.

[6] Google ROADEF/EURO CHALLENGE. 2011-2012: Machine reassignment problem. *Artificial Intelligence*, pages 9–47, 2011-2012. Accedido: 01-06-2015.

[7] Paolo Cremonesi and Andrea Sansottera. Modeling response times in the google roadef/euro challenge. *ACM SIGMETRICS Performance Evaluation Review*, 40(3):80–82, 2012.

[8] David Filani, Jackson He, Sam Gao, Murali Rajappa, Anil Kumar, Pinkesh Shah, and Ram Nagappan. Dynamic data center power management: Trends, issues, and solutions. *Intel Technology Journal*, 12(1), 2008.

[9] Michaël Gabay and Sofia Zaourar. Variable size vector bin packing heuristics - Application to the machine reassignment problem. Technical report, September 2013. HAL preprint hal-00868016, Tech. Rep. OSP.

[10] Michaël Gabay and Sofia Zaourar. Vector bin packing with heterogeneous bins: application to the machine reassignment problem. *Annals of Operations Research*, pages 1572–9338, September 2015.

[11] Michael R Garey, Ronald L Graham, David S Johnson, and Andrew Chi-Chih Yao. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory, Series A*, 21(3):257–298, 1976.

[12] Minos N Garofalakis and Yannis E Ioannidis. Multi-dimensional resource scheduling for parallel queries. In *ACM SIGMOD Record*, volume 25, pages 365–376. ACM, 1996.

[13] Haris Gavranović and Mirsad Buljubašić. An efficient local search with noising strategy for google machine reassignment problem. *Annals of Operations Research*, pages 1–13, 2014.

[14] Demirovic Emir Gavranovic Haris, Buljubasic Mirsad. Variable neighborhood search for google machine reassignment problem. *Electronic Notes in Discrete Mathematics*, 39, 2012.

[15] Edmund A Gehan. A generalized wilcoxon test for comparing arbitrarily singly-censored samples. *Biometrika*, 52(1-2):203–223, 1965.

[16] Rodolfo Hoffmann, Maria-Cristina Riff, Elizabeth Montero, and Nicolas Rojas. Google challenge: A hyperheuristic for the machine reassignment problem. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 846–853. IEEE, 2015.

[17] F. Hutter, H. Hoos, K. Leyton-Brown, and T. Stützle. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306, 2009.

[18] W Jaśkowski, M Szubert, and P Gawron. A hybrid mip-based large neighborhood search heuristic for solving the machine reassignment problem. *Annals of Operations Research*, pages 1–30, 2015.

[19] Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. Isac – instance-specific algorithm configuration. In *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 751–756, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press.

[20] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[21] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search: Framework and applications. In *Handbook of Metaheuristics*, pages 363–397. Springer, 2010.

[22] Yuri Malitsky, Deepak Mehta, Barry O'Sullivan, and Helmut Simonis. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 10th International Conference, CPAIOR 2013, Yorktown Heights, NY, USA, May 18-22, 2013. Proceedings*, chapter Tuning Parameters of Large Neighborhood Search for the Machine Reassignment Problem, pages 176–192. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[23] Renaud Masson, Thibaut Vidal, Julien Michallet, Puca Huachi Vaz Penna, Vinicius Petrucci, Anand Subramanian, and Hugues Dubedout. An iterated local

search heuristic for multi-capacity bin packing and machine reassignment problems. *Expert Systems with Applications*, 40(13):5266–5275, 2013.

[24] Deepak Mehta, Barry O'Sullivan, and Helmut Simonis. *Principles and Practice of Constraint Programming: 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, chapter Comparing Solution Methods for the Machine Reassignment Problem, pages 782–797. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[25] H. Murat Afsar, Christian Artigues, Eric Bourreau, and Safia Kedad-Sidhoum. Machine reassignment problem: the roadef/euro challenge 2012. *Annals of Operations Research*, 242(1):1–17, 2016.

[26] United Nations. Framework convention on climate change. 2016. Accedido: 24-07-2016.

[27] Intergovernmental Panel on Climate Change. *Climate change 2014: mitigation of climate change*, volume 3. Cambridge University Press, 2015.

[28] Gabriel M Portal, Marcus Ritt, Leonardo M Borba, and Luciana S Buriol. Simulated annealing for the machine reassignment problem. *Annals of Operations Research*, pages 1–22, 2015.

[29] Gabriel Marques Portal. An algorithmic study of the machine reassignment problem. Bachelor thesis, Universidade Federal do Rio Grande do Sul, 2012.

[30] Takfarinas Saber, Anthony Ventresque, Xavier Gandibleux, and Liam Murphy. Genepi: A multi-objective machine reassignment algorithm for data centres. *Hybrid Metaheuristics*, 8457:115–129, 2014.

[31] Ayad Turky, Nasser R Sabar, and Andy Song. Cooperative evolutionary heterogeneous simulated annealing algorithm for google machine reassignment problem. *Genetic Programming and Evolvable Machines*, pages 1–28, 2017.

[32] Mutsunori Yagiura, Toshihide Ibaraki, and Fred Glover. An ejection chain approach for the generalized assignment problem. *INFORMS Journal on Computing*, 16(2):133–151, 2004.