Inteligencia Artificial

Informe Final: *Machine Reassignment Problem*

Iván E. González López.

24 de junio de 2015

Evaluación

Mejoras 1ra Entrega (10 %):	_____
Código Fuente (10 %):	_____
Representación (15 %):	_____
Descripción del algoritmo (20 %):	_____
Experimentos (10 %):	_____
Resultados (10 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100):	_____

Resumen

El *Machine Reassignment Problem* consiste en la reasignación de un conjunto de procesos a una serie de equipos donde han de ejecutarse. Dichos equipos disponen de un conjunto de recursos, tales como: CPU, RAM y almacenamiento en disco. El objetivo es conseguir una asignación que permita mejorar la utilización de los equipos, respetando una serie de restricciones. Con la colaboración de Google, el problema fue propuesto como un desafío abierto por la ROADEF/EURO, el año 2012 [1].
En este documento se hace una descripción del problema, una revisión de la literatura con las distintas técnicas desarrolladas hasta la fecha para resolverlo y la presentación de distintos modelos matemáticos del problema.

1. Introducción

Los centros de datos (*Data centers*), compuestos de servidores y otros equipos, se han convertido en parte sustancial del funcionamiento de las organizaciones modernas, ya sea desde pequeñas o medianas empresas, hasta gigantes corporativos como Amazon, Facebook y Google entre otros, que ofrecen sus servicios basados en el concepto de la "nube". Esto viene de la mano con un incremento explosivo del contenido digital, el comercio electrónico y el tráfico en Internet, reflejo de que la carga de trabajo sobre los *Data centers* se ha incrementado. Sin embargo, el funcionamiento de éstos últimos demandan una gran cantidad de electricidad, convirtiéndolos en uno de los consumidores de energía con mayor crecimiento, especialmente en los Estados

Unidos. Este escenario se enmarca además, en la mayor preocupación de la sociedad por la sustentabilidad y el cuidado del medio ambiente, en función de los gases de efecto invernadero emitidos a la atmósfera.
Uno de los factores que presenta los más grandes problemas y oportunidades para el ahorro de la energía, es el de la sub-utilización de los servidores en los *Data centers*. Cuando un servidor realiza más trabajo, más eficiente es, tal como un bus que utiliza menos bencina por pasajero cuando lleva cincuenta personas, en comparación a cuando lleva solo unos cuantos. No obstante, el servidor promedio opera a no más del 12 o 18 % de su capacidad. Incluso en estado ocioso, se produce un consumo importante de energía dado al esquema 24/7 de funcionamiento. Para poner esto en perspectiva, gran parte de la energía consumida en los Estados Unidos por los *Data centers* es usada para abastecer a más de 12 millones de servidores, que hacen poco o nada de trabajo durante la mayor parte del tiempo. Aunque los proveedores de la nube a gran escala logran tasas de utilización más altas (entre el 40 a 70 %), incluso estos no son consistentes en alcanzar esas tasas. Nuevas investigaciones desde Google indican que los clusters de servidores típicos promedian en cualquier lugar desde un 10 a un 50 % de uso [3].
En ese contexto, se define en el año 2012 el *Machine Reassignment Problem*, desafío propuesto por la French Operational Research and Decision Support Society (ROADEF) y la European Operational Research Society (EURO), en colaboración con Google [1]. El objetivo de este desafío es poder optimizar la utilización de los recursos (unidades de procesamiento CPU, memoria RAM, ancho de banda y almacenamiento en disco entre otros) de un conjunto de máquinas o servidores, que serán asignados para ejecutar un conjunto de procesos.

Este documento tiene como objetivo presentar un estudio profundo acerca del *Machine Reassignment Problem*, dividiéndose de la siguiente manera:

- En la sección 2 se realizará una descripción del problema, tomando en cuenta las variables involucradas, restricciones y objetivos.
- En la sección 3 se presenta el "estado del arte" en torno al problema, realizando una inspección de las distintas técnicas desarrolladas hasta la fecha para resolverlo, poniendo hincapié en las que han presentado mejores resultados.
- En la sección 4 se presentan dos modelos matemáticos, uno de programación entera mixta y otro de programación con restricciones.
- En la sección 9 se presenta un resumen y algunas consideraciones del presente trabajo, haciendo referencia a las mejores técnicas de resolución del problema, considerando además el trabajo futuro alrededor del *Machine Reassignment Problem*.

2. Definición del Problema

Se dispone de un conjunto $\mathcal{M}$ de máquinas o servidores y un conjunto $\mathcal{P}$ de procesos. El objetivo del *Machine Reassignment Problem* es encontrar una asignación de mínimo costo, de cada proceso a una máquina, que optimice la utilización de los recursos que disponen estas últimas, respetando ciertas restricciones.

2.1. Restricciones

- **Restricciones de capacidad:** Para un conjunto $\mathcal{R}$ de recursos r, cada máquina m puede disponer de una capacidad $C(m, r)$ máxima de cada uno de ellos. Los procesos que se ejecutan en cada máquina, consumen un cierto nivel de esos recursos. La utilización $U(m, r)$ de un recurso r en una máquina m no debe exceder la capacidad de éste.

- **Restricciones de conflicto:** Un servicio s es un conjunto de procesos. Sea $\mathcal{S}$ el conjunto de todos los servicios (disjuntos) que particionan $\mathcal{P}$. Los procesos de un servicio $s \in S$ deben ejecutarse en distintas máquinas.
- **Restricciones de dispersión:** Una localización l es un conjunto de máquinas. Sea $\mathcal{L}$ el conjunto de todas las localizaciones (disjuntas) que particionan $\mathcal{M}$. Para cada servicio, se define un número de dispersión, que indica la cantidad mínima de localizaciones en las que los procesos del servicio deben repartirse para su ejecución.
- **Restricciones de dependencia:** Un vecindario n es un conjunto de máquinas. Sea $\mathcal{N}$ el conjunto de todos los vecindarios (disjuntos) que particionan $\mathcal{M}$. Sea un servicio s_a que depende de otro s_b. Entonces cada proceso de s_a debe ejecutarse en el vecindario de algún proceso de s_b.
- **Restricciones de uso transitorio:** Cuando un proceso es movido de una máquina m_a a otra m_b, hay ciertos recursos que son requeridos en ambas máquinas durante el proceso. Estos son los llamados recursos de uso transitorio. Para cada uno de estos últimos, su utilización no debe exceder la capacidad máxima dispuesta por las máquinas.

2.2. Costos

- **Costos de carga:** En cada máquina m, se establece la capacidad segura $SC(m, r)$ de un recurso r. El costo de carga es la sobre-utilización de r por sobre la capacidad segura. Para cada recurso r, este se define como $\sum_{m \in M} max(0, U(m, r) - SC(m, r))$.
- **Costos de balance:** Hace referencia a la proporción t que se quiere alcanzar, entre la cantidad disponible de dos recursos distintos (r_a y r_b). Para ello, se define la tripla $b =< r_a, r_b, t >$. Sea $\mathcal{B}$ el conjunto de todas las triplas. Para una tripla dada, el costo de balance se define como $\sum_{m \in M} max(0, t \times A(m, r_a) - A(m, r_b))$ donde $A(m, r) = C(m, r) - U(m, r)$.
- **Costos de movimiento de procesos:** Se define como el costo de mover un proceso p desde una máquina inicial cualquiera.
- **Costos de movimiento de servicios:** Se define como el máximo número de procesos movidos según servicio.
- **Costos de movimientos de máquina:** Se define como el costo de mover cualquier proceso p, desde una máquina m_a en particular a otra m_b distinta. El movimiento de un proceso dentro de una misma máquina tiene un costo cero. En consecuencia, por cada proceso p, se define una matriz de costos entre dos pares de máquinas m_a y m_b.

2.3. Objetivo

La función objetivo es el costo total CT, que se define como la suma de todos los costos anteriormente especificados, con sus respectivos *pesos* o ponderaciones.

2.4. Problemas similares

Algunos problemas similares al *Machine Reassignment Problem* son el Generalized Assignment Problem (GAP), que puede ser representado a través de un conjunto de n trabajos y de m máquinas. El objetivo es encontrar una asignación de mínimo costo, donde para cada trabajo, se debe realizar la asignación de una sola máquina respetando una restricción de recursos para esta última. Asignar un trabajo j a una máquina m implica un costo de $c_{m,j}$, consumiendo una cantidad $a_{m,j}$ de algún recurso, donde en cada máquina, existe una capacidad máxima de este

recurso. Este problema a demostrado ser *NP-duro*.
Otro problema similar es el Bin Packing Problem (BPP), donde el objetivo es básicamente depositar una serie de objetos dentro de ciertos recipientes, de tal forma que empaque sea factible respecto de ciertas restricciones, como por ejemplo de la capacidad de los depósitos, tratando de optimizar alguna función objetivo, como la utilización de la menor cantidad posible de recipientes para almacenar todos los ítemes disponibles.
Para cualquiera de los casos anteriores, el *Machine Reassignment Problem* se considera como una instancia más restringida de problema.

3. Estado del Arte

El *Machine Reassignment Problem* fue publicado como parte de la ROADEF/EURO del año 2012, en colaboración con Google, desafío en el cual participaron diferentes equipos de todo el mundo. El desafío propiamente tal tuvo instancias de evaluación, consistentes en problemas que iban desde un tamaño pequeño (4 máquinas, 100 procesos y 2 recursos), hasta problemas cercanos a un escenario real, como el que se puede apreciar en los data centers de las grandes corporaciones (5000 máquinas, 50000 procesos, 12 recursos). El tiempo máximo de ejecución de los algoritmos para cada instancia era de 5 minutos. Se comenzaba desde una solución inicial factible, generada aleatoriamente.
Considerando las anteriores condiciones, el desarrollo de las distintas técnicas de solución se han basado en metaheurísticas que tienen en su core, procedimientos de búsqueda local que se aplican iterativamente, tratando de obtener una asignación de menor costo en cada repetición. Comúnmente se opera solo con un subconjunto de las máquinas y procesos, debido especialmente al gran tamaño de algunas instancias del problema, las cuales son las que presentan mayor relación con la realidad de las grandes corporaciones como Google. Dada la naturaleza de la búsqueda local, la generación de los vecindarios de soluciones se originan, principalmente, en base a dos tipos de movimientos: *shift*, que implica trasladar un proceso de una máquina a otra; y el de *swap*, que implica el intercambio de la asignación de las máquinas distintas de dos procesos. Algunas de las técnicas a ver, incorporan mecanismos de perturbación, que modifican de forma aleatoria partes de la solución actual, con el fin incorporar más zonas del espacio de búsqueda. Otras incorporan técnicas de reinicio, para escapar de óptimos locales. En general (especialmente con grandes instancias), para evaluar la calidad de una solución, se hace en base a la comparación con los llamados *lower bounds* o límites inferiores, especialmente respecto de los costos de carga y de balance, que por lo que puede desprenderse de la literatura, se consideran los más importantes.
A continuación se presentan los trabajos más importante desarrollados hasta la fecha, en función de la resolución del *Machine Reassignment Problem*. Se nombraran algunas metaheurísticas notables que han sido utilizadas, proveyendo una pequeña definición de cada una.

3.0.1. Large Neighborhood Search

La metaheurística Large Neighborhood Search (LNS), fue propuesta en 1998 por Shaw [14]. En LNS, se mejora una solución inicial de forma gradual, destruyendo y reparando alternadamente la solución.
Consiguiendo el segundo lugar en la ROADEF/EURO del año 2012, Mehta et al. [11], proponen dos soluciones. Una primera, utilizando un modelo basado en Mixed Integer Linear Programming (MIP); una segunda basada en Constraint Programming (CP). En ambos casos, se utilizó la metaheurística LNS para la búsqueda de la solución. En cada iteración del LNS, se selecciona un subconjunto de máquinas. Desde estas últimas, se escoge un conjunto de procesos. Se forma así, un subproblema de tamaño más pequeño que el original, para luego ser optimizado y obtener una asignación que mejore la inicial. En [11], los resultados muestran que las soluciones obtenidas por el algoritmo basado en CP, superan a las obtenidas por la versión MIP, escalando mejor

para problemas de mayor envergadura, con un consumo de memoria menor y con una mejor calidad en las soluciones. Se deja entrever que el uso de computadores con múltiples núcleos, puede mejorar el rendimiento de los algoritmos. Posteriormente, los mismo autores en [9] desarrollan métodos automáticos para la generación de parámetros de entrada para el LNS, para así mejorar la performance de este último. Entre esos parámetros se encuentran el número máximo de procesos que pueden ser seleccionados de una máquina para ser reasignados, el máximo del total de procesos que pueden ser seleccionados para ser reasignados, el máximo de máquinas que se seleccionan para crear un subproblema, entre otros. Las pruebas muestran que los resultados son positivos a la hora de obtener mejores asignaciones, a pesar del costo computacional adicional que implica el ejecutar estos métodos automáticos, aunque se ve rápidamente solventado cuando se considera un espectro de tiempo más amplio.

3.0.2. Variable Neighborhood Search

La metaheurística Variable Neighborhood Search (VNS), consiste en cambiar sistemáticamente la estructura de los vecindarios, con el fin de poder escapar de óptimos locales en los cuales se quede estancado. Básicamente, el proceso consiste en obtener una solución s' nueva del vecindario de la actual s^*. Luego, a partir del vecindario de s' se ejecuta una búsqueda local hasta alcanzar un nuevo óptimo. Si este último es mejor que s^*, lo reemplaza. En caso contrario, modifica la estructura del vecindario.

Ganando el primer lugar [2] del desafío de la ROADEF/EURO el 2012, Buljubasic et al. [6], proponen un método híbrido, consistente en la realización de una búsqueda local, originando los vecindarios s' de soluciones en base a la realización de cuatro tipos de movimientos (en el orden en que están presentados más abajo), a partir de una solución inicial s y así encontrar una mejor asignación. Se pone especial énfasis en que el costo más importante de la solución es el de carga, por lo que los procesos de mayor tamaño (atributo definido en función del total de requerimientos considerando todos los recursos), son los primeros en ser reasignados. Movimientos:

- ***BPR***: Se mueve un proceso p a una máquina m y desplazando algunos procesos desde esta a otras máquinas. Movimiento especialmente útil para la resignación de procesos grandes.
- ***Shift***: Se desplaza un proceso p desde una máquina a otra.
- ***Swap***: Intercambio de la asignación de dos procesos, originalmente asignados a dos máquinas distintas.
- ***Chain***: Se desplazan simultáneamente l procesos $p_1, p_2, ...p_l$ de tal forma que:

$$s'(p_k) = s(p_{k+1})\ k \in \{1, 2, 3, ..., l-1\}$$
$$s'(p_l) = s(p_0)$$

Para el proceso de diversificación, es decir, explorar otras zonas del espacio de búsqueda y así obtener soluciones de mejor calidad, se genera ruido o *noise* sobre la función objetivo, eligiendo un recurso r e incrementando su ponderación o peso de costo de carga, creando una función objetivo modificada. El proceso global se repite hasta que límite del tiempo de ejecución termine o se obtenga un mínimo de mejora en la asignación. Según los autores, las máquinas y procesos elegidas en las primeras etapas de ejecución del algoritmo, inciden de gran manera en la calidad final de la solución. Se pone hincapié en que la posibilidad de poder construir una solución desde cero, y no partiendo desde una inicial como lo establece el problema original, permitiría obtener mejores resultados en el proceso de búsqueda local.

3.0.3. Bin Packing

En Gabay et al. [5], proponen una solución basada en la heurística Variable size vector bin packing (VSVBP), generalización del Vector Bin Packing y este a su vez, una generalización del Bin Packing, donde el objetivo es cargar una serie ítemes (procesos) dentro de ciertos recipientes (máquinas), de tal forma que el empaque de cada uno de estos sea factible con respecto a determinadas restricciones, ya sea límites de capacidad o balanceo, optimizando alguna función objetivo, como por ejemplo el número de recipientes utilizados. En el caso de VBP, los tamaños de los ítemes son representados por un vector d-dimensional (requerimientos tipo de recurso) y en el caso de VSVBP, los recipientes además tienen su propio vector de capacidades (para cada tipo de recurso). El problema VSVBP es considerado un subproblema del *Machine Reassignment Problem* considerando para este último como adicionales, las restricciones de conflicto, uso transitorio, de dispersión y de dependencia. Este método se centra principalmente en la factibilidad más que en la optimilidad de las soluciones, de tal forma que se utilicen como input inicial para heurísticas de búsqueda local.

3.0.4. Acercamientos híbridos

Cuando se habla de acercamientos híbridos, se refiere a la combinación de las ventajas de varios métodos de optimización combinatorial en un solo algoritmo.

Obteniendo el tercer lugar en la ROADEF/EURO 2012, Jaśkowski et al. [7], proponen una técnica híbrida compuesta potencialmente de 2 partes (puede realizarse una de las dos o ambas). En la primera, a partir de una asignación inicial, se realiza una búsqueda local basada en Hill-Climbing, generando los vecindarios de soluciones en base al desplazamiento de un proceso desde una máquina a otra distinta. El Hill-Climbing acepta la primera mejora que encuentra. Si no se encuentra una mejor solución en el vecindario, el algoritmo termina. Con este método se provee de una forma rápida de conseguir soluciones de calidad aceptable. El Hill-Climbing además, mantiene una lista tabú, donde se encuentran las máquinas que han sido utilizadas para el desplazamiento de los procesos. En la segunda parte del algoritmo, el desarrollo se basa en un procedimiento LNS. Se selecciona iterativamente un subproblema más pequeño que el original, escogiendo el subconjunto de máquinas de forma aleatoria o según el nivel de mejora en el costo de la solución que producen. Posteriormente, mediante un modelo MIP (*Mixed integer programmming*), se utiliza un solver como el CPLEX de IBM, para obtener una nueva solución de mejor calidad. Sobre esta última, es posible aplicar nuevamente Hill-Climbing. Según los resultados provistos en [7], la mejor combinación del algoritmo es HC-LNS-HC, es decir, aplicar primero Hill-Climbing, luego LNS con selección de subproblemas basados en una heurística de mejora optimista, para finalmente culminar con un nuevo Hill-Climbing.

Otro algoritmo híbrido es GENEPI, propuesto el 2014 por Saber et al. [13], el cual concibe al *Machine Reassignment Problem* como un problema multi-objetivo, definiendo tres metas para el problema original: confiabilidad, basada en el castigo que se da a las asignaciones que sobrecargan a las máquinas; migración, relativo a la penalización de las asignaciones que mueven en demasía los procesos, especialmente a lugares remotos; y electricidad, relativo a obtener las asignaciones minimicen el consumo de electricidad por parte del *Data Center*. La cuantificación de esta última, se realiza en base a la utilización de ciertas constantes de consumo y costo de electricidad, sobre las máquinas que se encuentran en funcionamiento. El algoritmo propiamente tal, se divide en la aplicación de tres diferentes técnicas. Primero se aplica una adaptación de la metaheurística GRASP. Esta última significa Greedy Randomized Adaptive Search Procedures (GRASP) y fue propuesta por Resende et al. [4], consiste en un método iterativo de dos fases, donde primero se construye una solución factible. Si no lo es, se implementa un proceso reparador para alcanzar la factibilidad. Si esta última no puede ser lograda, la solución se descarta y se

crea una nueva. Una vez lograda la factibilidad, se pasa a la segunda fase, donde a partir de un proceso de búsqueda local se explora el vecindario para encontrar una mejor solución. Durante el transcurso de las iteraciones, se guarda la mejor solución encontrada. donde en cada iteración los procesos son ordenados según sus dependencias y requerimientos, para luego reasignar una determinada fracción de procesos a máquinas distintas a las iniciales y seleccionar aleatoriamente una máquina de entre las factibles y más útiles. El objetivo con la aplicación de GRASP es encontrar soluciones de forma rápida, ya sean de buena o mala calidad, para servir como semillas para la aplicación de la siguiente técnica. En segundo lugar se aplica un algoritmo genético llamado Non-dominated Sorting Genetic Algorithm-II (NSGA-II), donde el operador de cruzamiento se aplica sobre los servicios con tal de minimizar las violaciones de las restricciones de dependencia. El operador de mutación es la reasignación de un proceso a una máquina factible. El objetivo de esta segunda etapa es encontrar una gran cantidad de buenas soluciones, explorando la mayor parte del espacio de búsqueda. Finalmente, se aplica un proceso de búsqueda local, calculando los vecindarios usando operadores *swap*, para intercambiar la asignación de las máquinas de dos procesos, *1-exchange* para mover un proceso de una máquina a otra, y *shift*, para mover un proceso a otra máquina perteneciente al mismo servicio.

3.0.5. Iterated Local Search

Lopes et al. [8], proponen una representación basada en MIP utilizando una heurística de tipo *Iterated Local Search* (ILS), para búsqueda de mejores soluciones. ILS toma una solución inicial s^* a la cual realiza pequeños cambios o perturbaciones para generar una nueva solución s'. Luego aplica búsqueda local sobre s' para producir un nuevo óptimo local s'^*. Esta ultima reemplazará a s^* si es mejor de acuerdo algún criterio de aceptación. El proceso se repite hasta que se cumpla alguna condición de término. Respecto de las perturbaciones, estas no deben ser muy pequeñas, con tal de generar buenos niveles de exploración, pero tampoco muy grandes como para permitir el explotar la información de las iteraciones anteriores.
En el caso de [8], se utilizan dos tipos de ILS, según la forma en que se genera el vecindario de soluciones:

- ***Heurísticas ILS no restringidas*:**
  - *Shift*: Se genera y explora el vecindario completo de soluciones, producto del desplazamiento de cada uno de los procesos a cada máquina, evaluando el costo de todos.
  - *Swap*: Se genera y explora el vecindario completo de soluciones, producto del intercambio realizado entre cada par de procesos.
- ***Heurísticas ILS restringidas*:** En la búsqueda de la eficiencia y considerando que las instancias del desafío de la ROADEF/EURO podían ser considerablemente grandes, se proponen los siguientes métodos de exploración, que poseen una menor tiempo de ejecución:
  - *Shift machine load sort*: Se evalúa un subconjunto de las soluciones originidas con el *shift*. Los vecinos que podrían originar menores costos son evaluados primero.
  - *Swap in service*: Para una solución s, define un vecindario compuesto por las soluciones que difieren de s por el intercambio de máquina de dos procesos que pertenecen al mismo servicio.
  - *Restricted swap*: Se evalúa un subconjunto del vecindario de soluciones originadas por *swap*. Para realizar el intercambio, solo se seleccionan los procesos que han cambiado de máquinas en la solución perturbada del ILS.

Se definen cuatro tipos de movimientos para realizar las perturbaciones: el primero, basado en un movimiento de $k - shift$, donde a partir de una solución s, se genera su vecindario realizando k movimientos sucesivos de desplazamiento, para luego seleccionar una solución s' aleatoria;

un segundo, que se basa en seleccionar un subconjunto de las máquinas de forma aleatoria, construyendo un MRP similar al original, para luego obtener una solución s', utilizando una estrategia de ramificación y poda usando el solver CPLEX; un tercero, llamado "*k-swap in service*", seleccionando una solución s' de forma aleatoria, desde el vecindario originado al realizar k sucesivos swap in service; y el último llamado "*restricted IP perturbation*", también basada en crear un MRP a partir de la selección aleatoria de un subconjunto de máquinas, pero esta vez seleccionando también el vecindario del cual se extraerán tales máquinas, de forma aleatoria. Los experimentos computacionales mostraron que, la combinación de perturbación y búsqueda local, que dieron mejores resultados fue la de *restricted IP perturbation* con heurísticas ILS restringidas. Esta combinación obtuvo el lugar 14 en la ROADEF/EURO 2012.

Otra solución basada en ILS, es la propuesta por Mason et al. [10], donde iterativamente se realiza una búsqueda local a partir de una solución factible inicial. Se selecciona un subconjunto de máquinas, donde a cada miembro m de este se evalúan dos tipos de movimientos. El que presente la mayor efectividad en mejorar la solución actual, es el que será realizado. El primero de ellos se basa en originar el vecindario de soluciones que se obtiene a partir de re-localizar un proceso desde una máquina m a otra distinta. El segundo de los movimientos, se basa en originar el vecindario de soluciones que se obtiene al intercambiar uno o dos procesos de m con otros pertenecientes a otras máquinas. En función de la eficiencia y dado que las instancias de los problemas pueden ser muy grandes (50000 procesos y 5000 máquinas), solo se explora una fracción aleatoria de los vecindarios originados, en ambos movimientos. Respecto de las perturbaciones, propias de la metaheurística ILS, se cuentan con dos versiones. La primera de ellas es el *home relocate*, donde un subconjunto aleatorio de procesos son re-localizados a su máquina inicial, con el fin de reducir el costo por movimiento de procesos. El otro tipo de perturbación es el *k-swap*, que aleatoriamente selecciona durante cierto número de veces, un par de máquinas desde las cuales se intercambian grupos aleatorios de procesos. Estas perturbaciones o *shaking moves* tienen la función de hacer escapar de "valles" u óptimos locales, desde donde se quede estancado el algoritmo. Si no se observan mejoras significativas respecto de las iteraciones anteriores, se aplicará un *Restart*. Los resultados obtenidos de las pruebas computacionales indican que la calidad de las soluciones provistas por el algoritmo, son de un alto nivel, donde en 21 de 30 instancias testeadas, el 99 % de las posibles mejoras son alcanzadas.

3.0.6. Simulated Annealing

El trabajo de Portal et al. [12], desarrolló un algoritmo basado en la metaheurística *Simulated Annealing* (SA), basada en principios de búsqueda local aplicados iterativamente. Permite la aceptación de soluciones nuevas, que empeoran el valor de la función objetivo, pero de una forma probabilista, con el fin de poder escapar a óptimos locales. El SA propuesto utiliza dos tipos de movimientos para generar los vecindarios. El primero es un *shift*, moviendo un proceso de una máquina a otra. El segundo es un *swap*, intercambiando dos procesos en máquinas distintas. Para el SA, se comienza con una temperatura t_0 que se mantiene constante por n iteraciones y luego se reduce en una proporción r. Si la mejor solución encontrada no genera una optimización en su valor dentro de $20n$ iteraciones y el número de de movimientos aceptados es menor al 0,1 %, se aumenta la temperatura para generar la suficiente perturbación como para escapar del óptimo local y aumentar la exploración. Según los resultados reportados de los experimentos computacionales, con pruebas hechas sobre el dataset de instancias oficial del desafío, se buenos resultados para las versiones A de instancias y comportándose de forma robusta para las instancias B, que eran la de mayor tamaño.

4. Modelos Matemáticos

Para ambos modelos presentados a continuación, la notación referente a los nombres de los conjuntos de máquinas, procesos, requerimientos, servicios, localizaciones, vecindarios y triplas, será igual a la presentada en la sección 2.

4.1. Primer Modelo

El siguiente modelo está basado en MIP (Mixed Integer Programming) o Programación entera mixta, extraído del trabajo realizado por Mason et al. [10].

4.1.1. Variables

- $x_{m,p}$: Toma el valor de uno si el proceso p se ejecuta en la máquina m. Cero en caso contrario.
- $y_{l,s}$: Toma el valor de uno si al menos un proceso del servicio s se ejecuta en la localización l. Cero en caso contrario.
- $z_{n,p}$: Toma el valor de uno si el proceso p está en el vecindario n.

4.1.2. Parámetros

- $C_{m,r}$: Indica el la capacidad del recurso r en la máquina m.
- $SC_{m,r}$: Indica el la capacidad segura del recurso r en la máquina m.
- $R_{p,r}$: Indica la cantidad del recurso r que requiere el proceso p.
- SM_s: *Spreadmin*, indica la cantidad mínima de localizaciones distintas en que los procesos del servicio s deben ejecutarse.
- Φ_r: Es el peso del costo de carga del recurso r.
- Φ_b: Es el peso del costo de carga de balance de la tripla b.
- T^b: Es la proporción entre los recursos pertenecientes a una tripla b, para el cálculo del costo de balance.
- Φ_p: Es el peso del costo de mover un proceso p.
- Φ_s: Es el peso del costo referente al máximo número de procesos movidos según servicio s.
- Φ_{m_1,m_2}: Es el peso del costo de movimiento de máquina, al mover un proceso p entre las máquinas m_1 y m_2.
- D^s: es el conjunto de servicios de los cuales depende el servicio s.

4.1.3. Función objetivo

La función objetivo es la correspondiente a la expresada en la sección 2, pero formalizada según el presente modelo MIP:

$$
\begin{aligned}
minimizar \quad & \sum_{r \in R} \Phi_r \sum_{m \in M} max\{0, (\sum_{p \in P} x_{p,m} R_{p,r}) - SC_{m,r}\} \\
& + \sum_{b \in B} \Phi_b \sum_{m \in M} max\{0, T^b (C_{m,r_1^b} - \sum_{p \in P} x_{p,m} R_{p,r_1^b}) - (C_{m,r_2^b} - \sum_{p \in P} x_{p,m} R_{p,r_2^b})\} \\
& + \sum_{p \in P} \sum_{m \in M \setminus M_0(p)} (\Phi_p + \Phi_{M_0(p),m}) x_{m,p} + \Phi_s max(\sum_{p \in s} \sum_{m \in M \setminus M_0(p)} x_{m,p}) \qquad (1)
\end{aligned}
$$

4.1.4. Restricciones

$$\sum_{m \in M} x_{m,p} = 1 \; ; \quad \forall p \in \mathcal{P} \qquad (2)$$

$$\sum_{p \in P} x_{m,p} R_{p,r} \leq C_{m,r} \; ; \qquad \forall m \in \mathcal{M}, r \in R \setminus TR \qquad (3)$$

$$\sum_{p \in P, M_0(p) \neq m} x_{m,p} R_{p,r} + \sum_{p \in P, M_0(p) = m} R_{p,r} \leq C_{m,r} \; ; \qquad \forall r \in TR, m \in \mathcal{M} \qquad (4)$$

$$\sum_{p \in P} \leq 1 \; ; \qquad \forall m \in \mathcal{M}, s \in \mathcal{S} \qquad (5)$$

$$\sum_{p \in s} \sum_{m \in l} x_{m,p} \geq y_{l,s} \; ; \qquad \forall l \in \mathcal{L}, s \in \mathcal{S} \qquad (6)$$

$$\sum_{l \in L} y_{l,s} \geq SM_s \; ; \qquad \forall s \in \mathcal{S} \qquad (7)$$

$$z_{n,p} = \sum_{m \in n} x_{m,p} \; ; \qquad \forall n \in \mathcal{N}, p \in \mathcal{P} \qquad (8)$$

$$z_{n,p} \leq \sum_{k \in s_b} z_{n,k} \; ; \qquad \forall n \in \mathcal{N}, p \in \mathcal{P}, s_b \in D^{s(p)} \qquad (9)$$

$$x_{m,p} \in \{0,1\} \; ; \qquad \forall m \in \mathcal{M}, p \in \mathcal{P} \qquad (10)$$

$$x_{l,s} \in \{0,1\} \; ; \qquad \forall l \in \mathcal{L}, s \in \mathcal{S} \qquad (11)$$

$$z_{n,p} \in \{0,1\} \; ; \qquad \forall n \in \mathcal{N}, p \in \mathcal{P} \qquad (12)$$

Explicación de las restricciones:

- (2) garantiza que cada proceso sea asignado a una sola máquina.
- (3) establece que el consumo total de cada recurso en cada máquina no debe superar la capacidad de ésta.
- (4) para cada recurso de uso transitorio, su utilización no debe exceder la capacidad dispuesta por las máquinas.
- (5) previene que dos procesos pertenecientes al mismo servicio, sean asignados a la misma máquina.

- (6) y (7), relativas al cumplimiento de las restricciones de dispersión.
- (8) y (9), relativas al cumplimiento de las restricciones de dependencia.
- (10), (11) y (12), relativas al dominio de las variables.

4.2. Segundo Modelo

A continuación se presenta un modelo basado en CP (Constraint Programming) o programación por restricciones, propuesto por Mehta et al. [11].

4.2.1. Variables

- x_p: Variable entera que indica cual máquina es asignada al proceso p. $x_p \in [0, |\mathcal{M}| - 1]$.
- $u_{m,r}$: Variable entera que indica el consumo del recurso r en la máquina m. $u_{m,r} \in [0, c_{m,r}]$ ($c_{m,r}$ capacidad del recurso r en la máquina m).
- $t_{m,r}$: Variable entera que indica la utilización transitoria de una máquina m para un recurso transitorio r. $t_{m,r} \in [0, c_{m,r}]$
- um_s: Indica el conjunto de máquinas que son asignadas a los procesos del servicio s.
- $yu_{s,l}$: Indica el número de máquinas pertenecientes a la localización l, que son asignadas a los procesos pertenecientes al servicio s.
- nul_s: Indica el número de localizaciones usadas por los procesos pertenecientes al servicio s.
- $zu_{s,n}$: Indica el número de máquinas del vecindario n que son usadas por el servicio s.
- $zm_{s,n}$: Indica el número de servicios que obligan a n ser el vecindario de s.
- nmp_s: Indica cuantos proceso en el servicio s son asignados a nuevas máquinas.
- mmp: Indica el máximo número de procesos de los servicios, que son movidos a nuevas máquinas.

4.2.2. Restricciones

- *Restricciones de carga:* El uso de un recurso $r \in \mathcal{R}$ en una máquina m es la suma de los recursos requeridos por aquellos procesos p que son asignados a la máquina m:

$$u_{m,r} = \sum_{p \in \mathcal{P} \wedge x_p = m} r_{p,r} \; ; \qquad \forall m \in \mathcal{M}, r \in \mathcal{R} \tag{13}$$

- *Restricciones de uso transitorio:*La utilización de un recurso transitorio $r \in T$ (T conjunto de recursos de uso transitorio, $T \in \mathcal{R}$) en una máquina m es la suma de $u_{m,r}$ y los recursos requeridos por aquellos procesos p cuya máquina original o_p es m pero la actual es distinta.

$$t_{m,r} = u_{m,r} + \sum_{o_p = m \wedge x_p \neq o_p} r_{p,r} \; ; \qquad \forall m \in \mathcal{M}, r \in T \tag{14}$$

- *Restricciones de conflicto:* El conjunto de máquinas usadas por los procesos del servicio s es $um_s := x_p : p \in s$. Los procesos pertenecientes al mismo servicio s no pueden ejecutarse en la misma máquina:

$$|s| = |um_s| \; ; \qquad \forall s \in \mathcal{S} \tag{15}$$

- *Restricciones de dispersión:* El número de máquinas en una localización l usadas por un servicio s es $\forall s \in \mathcal{S}, l \in \mathcal{L}, yu_{s,l} := |\{x_p | p \in s \wedge |L(x_p) = l\}|$ ($L(x_p)$ es la localización de la máquina x_p). El número de localizaciones utilizadas por los procesos de un servicio s es $nul_s := |\{l | l \in \mathcal{L} \wedge yu_{s,l} > 0|$. El número de localizaciones utilizadas por un servicio s debe ser mayor o igual al mínimo de dispersión de s ($spreadmin_s$):

$$nul_s \geq spreadmin_s \tag{16}$$

- *Restricciones de vecindad:* El número de máquinas usadas por un servicio s en un vecindario n es $zu_{sn} := |\{x_p | p \in s \wedge N(x_p) = n\}|$ ($N(x_p)$ es el vecindario de la máquina x_p). El número de servicios que desean que n sea (obligatoriamente) el vecindario del servicio s es $zm_{sn} := ||\{s' | \langle s', s\rangle \in D \wedge zu_{s'n} = 0\}$. El número de vecindarios obligatorios del servicio s que no son usados por este es $nmn_s := |\{n \in N \wedge zm_{sn} > 0 \wedge zu_{sn} = 0\}|$. Cada vecindario obligatorio de un servicio s debería ser utilizado por s:

$$nmn_s = 0 \tag{17}$$

4.2.3. Función Objetivo

El costo de una máquina se define como la suma de los costos de carga ponderados para todos los recursos (con su correspondiente peso w_r) y la suma de todos los costos de balance ponderados para todas las triplas de balance (con su correspondiente peso v_b):

$$cost_m = \sum_{r \in \mathcal{R}} max(0, u_{m,r} - sc_{m,r}) \cdot w_r + \sum_{b \in \mathcal{B}} max(0, t_b a_{m,r_b^1} - a_{m,r_b^2}) \cdot v_b \tag{18}$$

En (18), $sc_{m,r}$ es la capacidad segura del recurso r en la máquina m, t_b es la proporción entre recursos de la tripla b y $a_{m,r}$ es la cantidad disponible del recurso r en la máquina m.
El costo de un proceso se define como la suma de los movimientos de proceso ponderados (con su correspondiente peso w^{pmc}):

$$cost_p = min(1, |x_p - o_p|) \cdot pmc_p \cdot w^{pmc} + mmc(o_p, x_p) \cdot w^{mmc} \tag{19}$$

En la ecuación (19), $mmc(m_1, m_2)$ es el costo de mover un proceso desde la máquina m_1 a la m_2 y $pmcp_p$ es el costo de mover el proceso p desde su máquina original a otra distinta.
El costo de un servicio se define como la suma ponderada del número de procesos movidos de un servicio:

$$cost_s = \sum_{p \in s \wedge x_p \neq o_p} w^{sms} \tag{20}$$

La función objetivo a minimizar es el costo total:

$$minimizar \qquad cost = \sum_{m \in \mathcal{M}} cost_m + (\sum_{p \in \mathcal{P}} cost_p) + \max_{s \in \mathcal{S}}(cost_s) \tag{21}$$

5. Representación

Representación matemática y estructura de datos que se usa (arreglos, matrices, etc.), por qué se usa, la relación entre la representación matemática y la estructura. **Utilizar figuras, tablas, etc. en lo posible, de modo que sea fácil entender las estructuras utilizadas, además describa lo más sencillo y claro que pueda.**

6. Descripción del algoritmo

Cómo fue implementando, interesa la implementación más que el algoritmo genérico, es decir, si se tiene que implementar SA, lo que se espera es que se explique en pseudo código la estructura general y en párrafo explicativo cada parte cómo fue implementada para su caso particular, si se utilizan operadores se debe explicar por qué se utilizó ese operador, si fuera el caso de una técnica completa, si se utiliza recursión o no, etc. **Recuerde utilizar pseudocódigo para mostrar cada algoritmo y separar la explicación de su algoritmo en secciones para que se logre un mejor entendimiento. Además, en este punto no se espera que se incluya código, eso va aparte.**

7. Experimentos

Se necesita saber cómo experimentaron, cómo definieron parámetros, cómo se fueron modificando, cuáles problemas se trataron, cuáles fueron las instancias utilizadas, cuáles fueron las características de las instancias, cuáles fueron los criterios de término del algoritmo. **En el caso de las técnicas incompletas, es importante que ejecute su programa varias veces con distintas semillas aleatorias, para obtener valores estadísticos de los resultados**.

8. Resultados

Que fue lo que se logró con la experimentación, incluir tablas y parámetros, gráficos, etc. lo más explicativo posible. Además destacar puntos importantes del problema, características de las instancias que influyeron en los resultados, valores de parámetros que influyeron en los resultados, análisis de calidad de las soluciones encontradas a través del tiempo, análisis en profundidad de la técnica vs los resultados obtenidos, valores promedio, desviaciones, **comparaciones con resultados de la literatura**, etc. También debería discutir qué cosas podría haber agregado o quedan como desafíos para trabajo futuro en su algoritmo.

9. Conclusiones

En el presente informe se ha realizado una descripción en detalle del *Machine Reassignment Problem* con una revisión de la literatura referente a las distintos algoritmos y técnicas desarrollados para la resolución del problema, considerando las metaheurísticas y representaciones utilizadas para aquello.
Cabe destacar que a pesar de lo novel del problema -publicado en la ROADEF/EURO 2012-, la naturaleza o estructura de éste no es algo nuevo en la investigación científica, puesto que el *Machine Reassignment Problem* puede considerarse como una versión más restringida del *Generalized Assignment Problem* o del *Bin Packing Problem*. Sin embargo, dado el contexto actual en que los *data centers* son piedras angulares en el funcionamiento de las organizaciones modernas y su gestión se vuelve una tarea sumamente crítica en la búsqueda de la efectividad y eficiencia, el *Machine Reassignment Problem* por sí solo adquiere una relevancia importante. El hecho de que haya sido planteado por Google, corporación que basa la provisión de sus servicios a través del funcionamiento de extensas *granjas* de servidores, es un indicativo de que para la industria de la "nube", los objetivos planteados por el *Machine Reassignment Problem* son críticos.
Respecto de las técnicas de resolución vistas en la sección 3, todas intentan solucionar el problema original publicado por Google. En general, dado que la presentación del problema se puede dar a través de instancias de gran tamaño, considerando por ejemplo 5000 máquinas, 50000 procesos y 12 recursos (descripción bastante cercana de la realidad que afrontan los gigantes informáticos), los algoritmos presentados utilizan esquemas de búsqueda local, que se aplican

de forma iterativa sobre problemas de menor tamaño que el original, obtenidos a partir de un subconjunto de las máquinas y procesos involucrados, tratando de buscar asignaciones factibles y de bajo costo. A esto se suma que, el desafío original imponía un tiempo límite de ejecución para encontrar una solución de cinco minutos, por lo que los algoritmos en cuestión deben ser rápidos en la entrega de resultados, por lo que la utilización técnicas completas de búsqueda queda totalmente descartada. Esto se sustenta en el hecho de que los *data centers* funcionan 24/7, por lo que el tiempo disponible para generar asignaciones que mejoren el uso de los servidores, es bastante limitado. Otro aspecto relevante y relacionado con el tamaño de las instancias del problema, es el de la representación. Como lo dejó establecido el trabajo de Mehta et al. [11], la utilización de esquemas basados en *Mixed Integer Programming* es bastante ineficiente para instancias de gran tamaño (con un espacio de búsqueda mayor, definición de una mayor cantidad de restricciones), por lo que la utilización del *Constraint Programming* se es necesario para alcanzar una mayor escabilidad, especialmente cuando se utiliza una heurítica basada en LNS, combinación que entrega muy buenos resultados. Sin embargo, varios de los algoritmos vistos en el *estado del arte* utilizan una representación basada en MIP ([7], [8],[10]), valiéndose del hecho de que dentro de ejecución iterativa, se resuelven subproblemas de menor tamaño. Estos problemas MIP, son resueltos luego mediante algún solver como CPLEX.
En relación al trabajo futuro, es destacable las mejoras que puede proveer la utilización de múltiples procesadores a la hora de ejecutar los distintos algoritmos. También sería importante pensar en la configuración automática y en tiempo real de los distintos parámetros de entrada que necesitan los algoritmos para ejecutarse, como por ejemplo la proporción de máquinas que se utilizarán para crear un subproblema.

10. Bibliografía

Referencias

[1] Google roadef/euro challenge 2011-2012: Machine reassignment. 2012. `http://challenge.roadef.org/2012/files/problem_definition_v1`, visitado el 20-05-2015.

[2] Google roadef/euro challenge 2011-2012: Qualification and final results. 2012. `http://challenge.roadef.org/2012/files/Roadef%20-%20results.pdf`, visitado el 20-05-2015.

[3] Natural Resources Defense Council NRDC Anthesis. Data center efficiency assessment. 2014. `https://www.nrdc.org/energy/files/data-center-efficiency-assessment-IP.pdf`, visitado el 23-05-2015.

[4] ThomasA. Feo and MauricioG.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[5] Michaël Gabay and Sofia Zaourar. Variable size vector bin packing heuristics-application to the machine reassignment problem. *HAL preprint hal*, vol. 868016, 2013.

[6] Haris Gavranović, Mirsad Buljubašić, and Emir Demirović. Variable neighborhood search for google machine reassignment problem. *Electronic Notes in Discrete Mathematics*, 39:209–216, 2012.

[7] 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.

[8] Ramon Lopes, Vinicius WC Morais, Thiago F Noronha, and Vitor AA Souza. Heuristics and matheuristics for a real-life machine reassignment problem. *International Transactions in Operational Research*, 22(1):77–95, 2015.

[9] Yuri Malitsky, Deepak Mehta, Barry O'Sullivan, and Helmut Simonis. Tuning parameters of large neighborhood search for the machine reassignment problem. pages 176–192, 2013.

[10] 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.

[11] Deepak Mehta, Barry O'Sullivan, and Helmut Simonis. Comparing solution methods for the machine reassignment problem. pages 782–797, 2012.

[12] GabrielM. Portal, Marcus Ritt, LeonardoM. Borba, and LucianaS. Buriol. Simulated annealing for the machine reassignment problem. *Annals of Operations Research*, pages 1–22, 2015.

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

[14] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. pages 417–431, 1998.