UNIVERSIDAD DEL BÍO BÍO
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INDUSTRIAL

TESIS

Presentada para la obtención del grado de

Magíster en Ingeniería Industrial

Por:

Felipe Tomás Muñoz Valdés

"EXTENCIONES DE META-RAPS AL PROBLEMA DE MÁQUINAS PARALELAS NO RELACIONADAS"

Presentada en Concepción el 15 de Enero de 2006 ante la Comisión Examinadora compuesta por:

Director de Tesis:

- Sr. Felipe Baesler Abufarde — Profesor, Facultad de Ingeniería, Universidad del Bío Bío.

Correctores:

- Sr. Oscar Cornejo Zúñiga — Profesor, Facultad de Ingeniería, Universidad Católica de la Santísima Concepción.
- Sr. Luis Cevallos Araneda — Profesor, Facultad de Ingeniería, Universidad del Bío Bío.

- RESUMEN.

En este estudio se presenta el diseño de una aplicación de Meta-RaPS (Meta-heuristic for Randomized Priority Search) para resolver el problema de programación de la producción en máquinas paralelas no-relacionadas, con tiempos de preparación dependientes de la secuencia, con el objetivo de minimizar el makespan. El cual consiste en programar n trabajos (sin interrupción), disponibles en el tiempo cero, en m máquinas en paralelo (R_m), que procesan los trabajos con tiempos de procesamiento arbitrarios. Cada trabajo debe ser asignado a una máquina, y cada máquina puede procesar un trabajo a la vez. Si el trabajo j es programado en la máquina k, el tiempo necesario para procesar ese trabajo es P_{jk}, que depende del trabajo j, y de la máquina k. Siempre que un nuevo trabajo se inicia, se requiere un tiempo de preparación en la máquina, ese tiempo de preparación es dependiente de la secuencia y de la máquina (S_{ijk}), donde el tiempo de preparación en la máquina k necesario para realizar el trabajo j después del trabajo i puede ser diferente al tiempo de preparación para el trabajo i después del trabajo j. El objetivo del problema es encontrar el programa de producción que minimice el máximo tiempo de completación o makespan (C_{max}). En términos de la notación de scheduling introducida por Graham *et al.* (1979), el problema en cuestión puede ser representado mediante $R_m|S_{ijk}|C_{max}$.

El problema de scheduling en estudio es un problema fuertemente NP-hard (Garey & Johnson, 1979). Por lo que es necesario el uso de meta-heurísticas para encontrar soluciones de buena calidad en tiempos de respuesta factibles.

En el Capítulo 1 se realiza una introducción del estudio realizado, planteando las hipótesis, objetivos, justificación y el alcance del estudio. En el Capítulo 2 se presenta una revisión bibliográfica del problema, y se describen los modelos matemáticos asociados al problema $R_m|S_{ijk}|C_{max}$. Luego, en el Capítulo 3 se realiza una descripción de Meta-RaPS, donde se detallan los aspectos más importantes de esta meta-heurística.

El diseño de la aplicación de Meta-RaPS se dividió en tres etapas:

- Fase de construcción
- Fase de mejoramiento
- Ajuste de parámetros

Para evaluar el desempeño de la meta-heurística diseñada en cada una de estas etapas, se consideró la mejor aplicación encontrada en la literatura (propuesta por Rabadi *et al.*, 2006), llamada Meta-RaPS SAPSL. También se utilizó la librería de problemas propuesta por Rabadi (2005).

En el Capítulo 4 se diseñó la fase de construcción, donde se presenta una heurística constructiva visionaria (look-ahead) para encontrar soluciones al problema $R_m|S_{ijk}|C_{max}$, la cual es comparada satisfactoriamente con una heurística constructiva glotona (greedy). Se presenta un ejemplo numérico de su aplicación, y la adición de aleatoriedad controlada mediante los parámetros de Meta-RaPS (*%prioridad* y *%restricción*).

La fase de mejoramiento se abordó en el Capítulo 5, donde se diseñó una heurística de mejoramiento local para el problema $R_m|S_{ijk}|C_{max}$, la cual es comparada satisfactoriamente con la heurística propuesta por Rabadi *et al.* (2006).

El problema de ajuste de parámetros presentó en los Capítulos 6 y 7. En el Capítulo 6 se aplica un ajuste de parámetros off-line, en el cual el ajuste de parámetros se realiza previamente a la resolución del problema. En el Capítulo 7 se diseñan técnicas de auto-ajuste de parámetros (ajuste on-line o parámetros auto-adaptables), en el cual el ajuste de parámetros se realiza durante la resolución del problema. En los Capítulos 6 y 7 se comparan satisfactoriamente los resultados obtenidos de la aplicación propuesta contra la aplicación propuesta por Rabadi *et al.* (2006) (Meta-RaPS SAPSL), para ajuste off-line y on-line respectivamente. Donde se puede observar que la aplicación propuesta reporta en promedio mejores resultados que Meta-RaPS SAPSL, además de encontrar mejores soluciones en la mayoría de los problemas de prueba. En el Capítulo 7 se comparan los resultados obtenidos por la aplicación propuesta (Meta-RaPS LACH) utilizando ajuste de parámetros off-line y on-line, mostrando que los resultados obtenidos por ajuste on-line son mejores que los que se obtienen con un ajuste off-line.

Finalmente en el Capítulo 8 se detallan las conclusiones obtenidas de este estudio y las futuras investigaciones a realizar. Mostrando que el enfoque de solución propuesto (Meta-RaPS LACH), permite obtener mejores resultados que los encontrados en literatura.

El impacto de este estudio, es diseñar alternativas para resolver problemas complejos de programación de la producción. Específicamente en el diseño propuesto de Meta-RaPS, el cual es modificado para facilitar la tarea de fijación o selección de parámetros. Además de permitir la obtención de mejores resultados que los encontrados en literatura.

INDICE

CAPITULO 1: INTRODUCCIÓN.

En este trabajo se presenta una aplicación de Meta-RaPS (Meta-heuristic for Randomized Priority Search) para resolver el problema de minimizar el máximo tiempo de completación en un ambiente de máquinas paralelas no-relacionadas con tiempos de preparación dependientes de la secuencia en que se procesan los trabajos.

1.1 Introducción.

En general, el problema de programación de la producción en máquinas paralelas considerado en este estudio es una generalización del problema representado por $P_m||C_{max}$, en el cual n trabajos deben ser procesados por m máquinas paralelas idénticas (P_m) para minimizar el máximo tiempo de completación (también llamado makespan, C_{max} o tiempo de completación del último trabajo). Una variedad bastante grande de problemas de programación de la producción en máquinas paralelas pueden ser derivados desde ese problema básico, dependiendo del tipo de máquinas, características de procesamiento, restricciones, y función o funciones objetivo en consideración. El entorno de máquinas puede ser idénticas (P_m), uniformes (Q_m), o no-relacionadas (R_m). Una cantidad de m máquinas son consideradas idénticas si tienen exactamente los mismos tiempos de procesamiento para los trabajos. Se consideran uniformes si difieren sólo en su velocidad, y se consideran no-relacionadas si son completamente diferentes en todos sus tiempos de procesamiento, es decir, los tiempos de procesamiento son arbitrarios para todos los

trabajos y máquinas. Las características de procesamiento se refieren, por ejemplo, si los trabajos pueden ser interrumpidos (*prmp*) o si tienen tiempos de preparación o setup dependientes de la secuencia (S_{ij}). Ejemplos de restricciones incluyen restricciones de elección de máquinas (M_j) y de precedencia (*prec*).

Muchos objetivos pueden ser de interés como la minimización del costo de producción, del máximo tiempo de completación o makespan (C_{max}), el tiempo de completación total (ΣC_j), la máxima tardanza (L_{max}), la tardanza total (ΣT_j), y el adelantamiento y tardanza total (ET_j). Algunas funciones objetivo también pueden considerar el uso de ponderadores, donde cada trabajo tiene una importancia o ponderador (w_j) para ser considerado en la función objetivo (Pinedo, 2002).

Según Pinedo (2002), el máximo tiempo de completación (makespan) como objetivo de optimización tiene varias características de importancia, entre ellas: permite terminar el total de los trabajos lo más tempranamente posible, también permite lograr un mejor balance de cargas de trabajo en las máquinas y con ello una mejor utilización de los recursos (máquinas).

El problema de programación de la producción en máquinas paralelas no-relacionadas con tiempos de preparación dependientes de la secuencia, con el objetivo de minimizar el máximo tiempo de completación (makespan) consiste en programar *n* trabajos (sin interrupción), disponibles en el tiempo cero, en *m* máquinas en paralelo (R_m), que procesan los trabajos con tiempos de procesamiento arbitrarios. Cada trabajo debe ser asignado a una máquina, y cada máquina puede procesar un trabajo a la vez. Si el trabajo *j* es programado en la máquina *k*, el tiempo necesario para procesar ese trabajo es P_{jk}, que depende del trabajo *j*, y de la máquina *k*. Siempre que un nuevo trabajo se inicia, se requiere

un tiempo de preparación en la máquina, ese tiempo de preparación es dependiente de la secuencia y de la máquina (S_{ijk}), donde el tiempo de preparación en la máquina k necesario para realizar el trabajo j después del trabajo i puede ser diferente al tiempo de preparación para el trabajo i después del trabajo j. El objetivo del problema es encontrar el programa de producción que minimice el máximo tiempo de completación o makespan (C_{max}). En términos de la notación de scheduling introducida por Graham *et al.* (1979), el problema en cuestión se representa mediante por $R_m|S_{ijk}|C_{max}$.

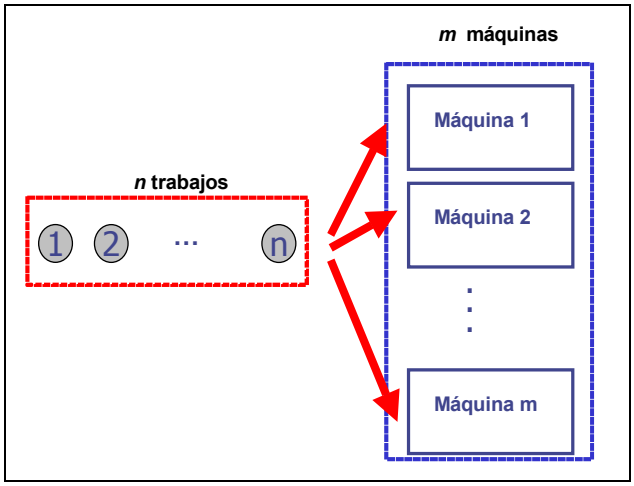


Figura Nº1.1: Problema de máquinas paralelas.

El problema de programación de trabajos en máquina paralelas idénticas (Figura Nº1.1), minimizando el máximo tiempo de completación ($P_m||C_{max}$), es un conocido problema NP-hard incluso cuando se consideran dos máquinas (Garey & Jonson, 1979). Luego, como el problema en estudio es una generalización de ese problema, éste es también NP-hard. Esto significa que no existe un algoritmo de tiempo computacional polinomial que pueda resolver el problema en forma óptima. Para analizar este aspecto, consideremos

el clásico problema del agente viajero (TSP), el cual es un problema fuertemente NP-hard. El TSP es equivalente al problema de minimización del máximo tiempo de completación (C_{max}) en una máquina, con tiempos de setup dependientes de la secuencia, con la consideración de que los tiempos de preparación son idénticos si se programa el trabajo *i* antes del trabajo *j* o viceversa ($S_{ij} = S_{ji}$). Haciendo un símil con programación de la producción (scheduling), este problema se representaría mediante $1|S_{ij}=S_{ji}|C_{max}$. Si se considera el caso donde los tiempos de preparación no cumplen la condición anterior ($S_{ij} \neq S_{ji}$), el problema sería equivalente al problema del agente viajero asimétrico (ATSP), el cual es un problema fuertemente NP-hard. Por otro lado, si para el problema $1|S_{ij}=S_{ji}|C_{max}$ consideramos varias máquinas idénticas, se estaría en presencia del problema $P_m|S_{ij}=S_{ji}|C_{max}$, el cual es similar al problema de ruteo de vehículos (VRP), que también es un problema NP-hard. A pesar de que se pueden hacer similitudes entre los problemas de programación de la producción con los clásicos problemas de ruteo de vehículos y del agente viajero, no se puede hacer lo mismo con el problema $R_m|S_{ijk}|C_{max}$ debido a su enorme complejidad.

Desarrollar algoritmos que permitan resolver el problema $R_m|S_{ijk}|C_{max}$, es importante porque el problema de máquinas paralelas es común en la industria. Además existen diversos factores de importancia, entre ellos: permiten optimizar procesos, mejorar sistemas de producción y lograr un mejor aprovechamiento de los recursos. En la industria el tamaño de estos problemas es grande, y la aplicación de métodos optimales no es factible debido a sus largos tiempos de respuesta. En cambio los algoritmos metaheurísticos ofrecen una vía factible para resolver estos problemas.

Lo que se busca en este estudio, es desarrollar un procedimiento meta-heurístico que permita encontrar soluciones de buena calidad en un corto tiempo de respuesta para el problema $R_m|S_{ijk}|C_{max}$. En este estudio se aplicará Meta-RaPS, debido a su gran capacidad para resolver problemas combinatorios.

1.2 Hipótesis del estudio.

Los autores de Meta-RaPS proponen que esta metaheurística podría utilizar en su fase constructiva heurísticas glotonas (greedy) y visionarias (look-ahead). Sin embargo en la literatura no existen aplicaciones de Meta-RaPS donde se utilizaran heurísticas visionarias. Por lo que la primera hipótesis consiste en validar este supuesto.

Hipótesis 1: Utilizar una heurística constructiva visionaria en la fase constructiva de Meta-RaPS permite obtener mejores resultados en comparación si se utilizara una heurística glotona.

En general, una heurística constructiva glotona utiliza reglas simples para elegir el orden en que se deben agregar las actividades a la solución, por ejemplo la regla SPT, que consiste en programar la actividad que tenga el menor tiempo de procesamiento primero. En cambio una heurística constructiva visionaria considera una mayor cantidad de factores e información, lo cual permite tomar mejores decisiones.

Muchos autores en temas de aplicación de metaheurísticas proponen que estudiar la estructura del problema es beneficioso porque permite diseñar heurísticas de mejoramiento o búsqueda de buena calidad. En general, una heurística constructiva considera la estructura del problema, en cambio una heurística de mejoramiento local no la considera. Por lo que

es razonable pensar que considerar las características del problema es beneficioso para lograr un buen diseño.

Hipótesis 2: Considerar la estructura del problema en el diseño de heurísticas de mejoramiento local, permite obtener mejores resultados y un ahorro en tiempos de corridas.

Diseñar metaheurísticas implica seleccionar los mejores valores para los parámetros que controlan el desempeño de esta. Este problema recibe el nombre de "problema de fijación de parámetros". Mientras mejor es ese ajuste de parámetros, mejores resultados pueden ser encontrados por la aplicación de la metaheurística. Este problema de fijación de parámetros es complejo y existen dos alternativas para realizarlo: fijación previa (off-line) a la resolución del problema y fijación durante la resolución de problema (on-line).

Hipótesis 3: Un ajuste de parámetros basado en control de parámetros (ajuste on-line o parámetros auto-adaptados) reporta mejores resultados que el ajuste off-line.

Una consecuencia de esta hipótesis es que si el ajuste on-line entrega resultados satisfactorios, se mostraría que es posible facilitar la tarea de ajuste de parámetros para el usuario final de la aplicación.

1.3 Objetivos del estudio.

El objetivo principal de este estudio es desarrollar una aplicación metaheurística eficiente y eficaz para resolver el problema de programación de la producción en máquinas paralelas no relacionadas con tiempos de preparación dependientes de la secuencia

($R_m|S_{ijk}|C_{max}$). Considerando tres enfoques para evaluar su desempeño: la calidad de los resultados obtenidos, el tiempo de respuesta de la aplicación y la facilidad de su uso.

Los objetivos secundarios son:

- Encontrar mejores resultados que los reportados en la literatura (Rabadi *et al.,* 2006).
- Desarrollar técnicas de ajuste de parámetros on-line para Meta-RaPS, con el fin de facilitar la utilización.
- En la literatura existe solo una heurística constructiva, y es ésta es de tipo glotón. Por lo que será necesario diseñar una heurística constructiva visionaria para el problema $R_m|S_{ijk}|C_{max}$.

1.4 Justificación.

Los problemas de programación de la producción son muy comunes en plantas de producción y es necesario resolverlos varias veces en un día, por ejemplo al comienzo de cada turno. Por lo que existe una necesidad de desarrollar una aplicación para programar la producción, que genere buenos resultados en cortos periodos de tiempo, y que además sea fácil de utilizar. Para el problema $R_m|S_{ijk}|C_{max}$ la posibilidad de resolverlo en forma óptima en tiempos de respuesta factibles es imposible, por lo que la utilización de metaheurísticas es necesaria.

Las metaheurísticas ofrecen la posibilidad de resolver problemas de este tipo mediante el mejor balance entre tiempos de respuesta y calidad de la solución. Meta-RaPS es una eficiente y eficaz metaheurística que permite resolver problemas combinatorios

(Moraga *et al.*, 2004) como el problema $R_m|S_{ijk}|C_{max}$. Según Rabadi *et al.* (2006), Meta-RaPS es una estrategia maestra que utiliza heurísticas de construcción y mejoramiento para generar soluciones de buena calidad en tiempos de respuesta cortos, de tal forma que es una estrategia genérica de alto nivel usada para modificar heurísticas de construcción (o reglas de prioridad) basándose en la inserción del elemento aleatorio. De tal forma que Meta-RaPS integra reglas de prioridad, aleatoriedad y muestreo. En cada iteración, Meta-RaPS construye y mejora soluciones factibles a través de la utilización de reglas de prioridad utilizadas de manera aleatorizada. Después de un número dado de iteraciones, Meta-RaPS reporta la mejor solución encontrada.

En la literatura existe una aplicación de Meta-RaPS para el problema $R_m|S_{ijk}|C_{max}$ (Rabadi *et al.*, 2006), que utiliza en su fase constructiva una heurística glotona. Esta aplicación reporta los mejores resultados para una librería de problemas de prueba propuesta por Rabadi (2005).

Todas las metaheurísticas controlan su desempeño mediante sus parámetros, los cuales pueden tomar valores en un rango preestablecido por el diseño de la metaheurística. Es por ello, que el ajuste de parámetros es clave en el desempeño, y esto se debe a la alta correlación existente entre precisión en ajuste de parámetros y calidad de los resultados generados. Por lo tanto mientras más precisa sea la fijación de parámetros mejores resultados pueden ser obtenidos de la aplicación de Meta-RaPS. El ajuste de los parámetros es una tarea muy importante y complicada, por lo que es necesario desarrollar metodologías que eviten o faciliten esta tarea al usuario.

1.5 Alcances del estudio.

En este estudio, se considera la aplicación de Meta-RaPS para resolver el problema $R_m|S_{ijk}|C_{max}$, ya que es la metaheurística que ha reportado los mejores resultados según la literatura. Además su estudio permitirá validar la hipótesis que hace referencia al uso de heurísticas visionarias.

El diseño de la metaheurística Meta-RaPS se dividirá en tres etapas: fase de construcción, fase de mejoramiento y ajuste de parámetros.

Para evaluar el diseño de cada una de estas etapas se utilizará la librería de problemas propuesta por Rabadi (2005), y se compararán los resultados obtenidos con los obtenidos por (Rabadi *et al.*, 2006).

CAPITULO 2: REVISIÓN BIBLIOGRÁFICA.

En este capítulo se presenta una revisión bibliográfica del problema $R_m|S_{ijk}|C_{max}$. Luego se presenta la formulación matemática del problema.

2.1 Revisión bibliográfica.

Casos prácticos de problemas de máquinas paralelas son muy comunes en la vida real. En literatura se pueden encontrar aplicaciones en las áreas de producción de textiles (Guinet, 1991), transporte (Guinet, 1993), programación de actividades de oficina (Herrmann *et al.,* 1997), metalurgia de aluminio (Dhaenens-Flipo, 2001), producción de semiconductores (Kim *et al.,* 2002), producción de plásticos por moldeo (Lin *et al.,* 2002), producción de placas de circuito impresas (Yu *et al.,* 2002).

Revisiones del estado del arte sobre problemas de programación de la producción en máquinas paralelas pueden ser encontradas en Graves (1981), Cheng & Sin (1990), Lawler *et al*. (1993), y más recientemente en Mokotoff (2001).

En general, las investigaciones que consideran el problema $R_m|S_{ijk}|C_{max}$ son escasas. Sin embargo la literatura relacionada con variantes del problema es bastante extensa. En la Tabla N°2.1 (al final del capítulo) se muestra un resumen de la revisión bibliográfica realizada.

Horowitz & Sahni (1976), Ibarra & Kim (1977) han estudiado el problema sin tiempos de preparación, y proponen heurísticas y algoritmos de aproximación.

Investigaciones recientes han estudiado la resolución para problemas de máquinas paralelas con tiempos de preparación dependientes de la secuencia (Franca *et al.,* 1996; Lee & Pinedo, 1997; Anagnostopoulos & Rabadi, 2002; Helal & Hosni, 2003; Al-Salem, 2004; Moraga, 2004 y Rabadi *et al.,* 2006).

Algunas metaheurísticas han sido propuestas para problemas de máquinas paralelas: Tabú Search (Franca *et al.,* 1996; Lee & Pinedo, 1997; Srivastava, 1998; Armentano & Yamashita, 2000 y Helal & Hosni, 2003), Simulated Annealing (Herrmann *et al.,* 1997; Kim *et al.,* 2002 y Anagnostopoulos & Rabadi, 2002), Algoritmos Genéticos (Min & Cheng, 1999), Meta-RaPS (Moraga, 2004 y Rabadi *et al*., 2006), Redes Neuronales (Hamad *et al*., 2003) y GRASP (Laguna & González, 1991 y Rojanasoonthon & Bard, 2005).

Armentano & Yamashita (2000), presentan una aplicación de Tabú Search para resolver el problema de máquinas paralelas, donde el objetivo es minimizar la tardanza media ($P_m||\sum T_j$). El procedimiento de mejoramiento local, se inicia con una solución obtenida a partir de una heurística constructiva denominada KPM, realizando movidas de inserción e intercambio. Lee & Pinedo (1997), estudiaron el mismo problema pero en presencia de tiempos de preparación dependientes de la secuencia y ponderación de los trabajos ($P_m|S_{ij}|\sum w_j T_j$) y propusieron una heurística de tres fases para resolver el problema, que considera una regla de despacho para construir una solución inicial y un procedimiento de búsqueda basado en Simulated Annealing.

Hurink & Knust (2001), estudiaron el problema de máquinas paralelas en presencia de restricciones de precedencia y tiempos de preparación, donde el objetivo es minimizar el máximo tiempo de completación ($P_m|prec,\ S_{ij}|C_{max}$). Los autores consideran el problema

como una combinación de problemas de particionamiento y secuenciamiento, donde en el problema de particionamiento se seleccionan los trabajos que cada máquina realizará, luego esos trabajos son secuenciados en el problema de secuenciamiento. Para resolver el problema de programación de la producción los autores proponen un algoritmo pseudo-polinomial basado en programación dinámica y fijación de cotas superiores para encontrar una cota inferior del máximo tiempo de completación (makespan). Guinet (1993), estudió el mismo problema pero en ausencia de restricciones de precedencia, y consideró dos objetivos por separado, la minimización del máximo tiempo de completación (makespan) y la minimización del tiempo de flujo medio ($P_m|S_{ij}|C_{max}$ y $P_m|S_{ij}|\sum C_j$). Guinet propone un algoritmo de asignación para resolver el problema, como una extensión del método Húngaro para el uso de múltiples recursos, que ha sido aplicado al problema de ruteo de vehículos. Yalaoui & Chu (2003) estudiaron el problema ($P_m|S_{ij}|C_{max}$) y utilizaron un enfoque de solución que consistió en generar, a partir del problema original, un problema de máquina única con tiempos de setup dependientes de la secuencia para cada máquina. Luego cada subproblema fue resuelto mediante un algoritmo para resolver el problema del agente viajero (TSP). Luego de eso la solución obtenida es modificada en una etapa de mejoramiento.

Srivastava (1998) presenta un algoritmo de Tabú Search para el problema de máquinas paralelas, donde la función objetivo es minimizar el máximo tiempo de completación (makespan) ($R_m||C_{max}$) y reporta que el algoritmo puede entregar soluciones de buena calidad para problemas de tamaño práctico en una razonable cantidad de tiempo computacional. Mokotoff & Jimeno (2002) estudiaron el mismo problema y presentan varios algoritmos heurísticos basados en enumeración parcial, que consisten básicamente

en la construcción de subproblemas entero-mixto, considerando la integralidad de algún subgrupo de variables binarias obtenidas de la solución del problema lineal relajado. Presentan experimentos computacionales para varios problemas de prueba, reportando en algunos problemas mejores resultados que otros enfoques de aproximación.

Liaw *et al*. (2003) desarrollaron un algoritmo de ramificación y acotamiento (R&A) para encontrar soluciones óptimas para el problema de máquinas paralelas no-relacionadas sin tiempos de preparación, donde la función objetivo es minimizar la tardanza total ponderada ($R_m||\sum w_j T_j$). Para el algoritmo de R&A utilizan una heurística de dos fases basada en el problema de asignación, mediante la cual obtienen cotas inferiores y superiores. Esas cotas son utilizadas para incorporar reglas de dominancia. También definen las características de la solución óptima del problema. Vredeveld & Hurkens (2002) presentan una comparación empírica de algoritmos de aproximación de tiempo polinomial y heurísticas basadas en búsqueda local para el problema de minimización del tiempo de completación total ponderado en máquinas paralelas no-relacionadas ($R_m||\sum w_j C_j$).

Yu *et al.* (2002), estudiaron el problema de máquinas paralelas no-relacionadas, donde el objetivo es minimizar los costos totales de los trabajos procesados en presencia de restricciones de elegibilidad de máquina ($R_m|M_j|\sum Costo$). Los autores estudian el problema dentro de un marco mayor de un flexible flow shop y proponen una metodología de solución basada en relajación lagrangeana, donde utilizan un procedimiento heurístico para relajar las restricciones y determinar los multiplicadores lagrangeanos. Para los autores la resolución del problema está fuertemente influenciada por la necesidad de una empresa electrónica que produce placas de circuitos integradas, de lograr una mejora en la

productividad y un mejor balance de las cargas de trabajo en las máquinas, y consideran que el tiempo de respuesta que debe tener la metodología de solución es de a lo más un minuto. El algoritmo propuesto por los autores es comparado con un modelo de redes y una regla FIFO modificada, logrando mejores resultados que ambos.

Kim *et al*. (2002) consideraron el problema de máquinas paralelas no-relacionadas con tiempos de preparación dependientes de la secuencia, para minimizar la tardanza total ($R_m|S_{ijk}|\sum T_j$) y desarrollan heurísticas para ser incluidas en un algoritmo de Simulated Annealing (SA), que considera seis tipos de movida en el vecindario. También se realizan comparaciones con un SA convencional y con búsqueda local.

Weng *et al.* (2001) estudiaron el problema de máquinas paralelas no-relacionadas, para minimizar el tiempo de completación ponderado medio e incluyen tiempos de preparación dependientes de la secuencia, pero independientes de las máquinas ($R_m|S_{ij}|\sum w_j C_j$). Ellos presentan y prueban siete heurísticas para resolver el problema. En sus algoritmos, ellos asignan trabajos a las máquinas con el menor costo de contribución, o a la máquina en la que el trabajo tiene el menor tiempo de procesamiento. También prueban una estrategia donde asignan primero el trabajo con la menor tasa de tiempo de procesamiento y preparación ponderado; esta estrategia deja fuera de desempeño al resto de las estrategias significativamente. Los autores proponen que una de las principales características de sus algoritmos es que son extremadamente rápidos y que resuelven problemas por sobre 120 trabajos y 12 máquinas.

Anagnostopoulos & Rabadi (2002) proponen un algoritmo de Simulated Annealing para el problema de máquinas paralelas no-relacionadas con tiempos de preparación dependientes de la secuencia, minimizando el máximo tiempo de completación (makespan)

($R_m|S_{ijk}|C_{max}$). Los autores realizan pruebas con problemas de pequeño tamaño logrando resultados óptimos. Helal & Hosni (2003), estudiaron el mismo problema y desarrollan un algoritmo basado en Tabú Search, también proponen una modificación de la regla SPT (shortest procesing time) para construir una solución inicial, y dos esquemas de perturbaciones para mejorar la solución. Por otro lado, Moraga (2004) y Rabadi *et al.* (2006) proponen la aplicación de Meta-RaPS para resolver el problema, utilizando en la fase constructiva una regla glotona llamada SAP-SL.

Al-Salem (2004), propone una heurística de particionamiento, la cual utiliza tres fases. La primera de ellas es una fase constructiva que asigna trabajos a las máquinas, la segunda fase es una fase de mejoramiento al resultado entregado en la primera fase. La última fase es donde se realiza el secuenciamiento basado en técnicas para resolver el problema del agente viajero (TSP).

Armacost & Al-Salem (1999) estudiaron el mismo problema, pero consideraron además restricciones de elegibilidad de máquina, denotado por ($R_m|S_{ijk},M_j|C_{max}$), y propusieron una heurística multi-fase, que considera una metodología de descomposición, y luego aplica varias heurísticas relacionadas con el problema de ruteo de vehículos con múltiples almacenes. El procedimiento es sencillo y consta de dos fases, primero se utiliza la heurística constructiva para asignar los trabajos a las máquinas, y luego se utiliza una heurística de mejoramiento. Herrmann *et al.* (1997) consideran el problema en ausencia de tiempos de preparación, pero consideran la restricción de precedencia ($R_m|prec|C_{max}$), y abordan el problema desde el caso práctico de programación de actividades de oficina, donde los trabajadores tienen diferentes habilidades, y proponen una heurística para asignar las tareas a los trabajadores, definiendo el programa de producción simultáneamente.

Guinet (1991) propone varios algoritmos basados en métodos de asignación y ruteo de vehículos, para solucionar un problema de programación de la producción en empresas textiles, con el objetivo de minimizar el tiempo de completación total en un ambiente de máquinas paralelas no-relacionadas con tiempos de preparación dependientes de la secuencia, pero independientes de las máquinas ($R_m|S_{ij}|\sum C_j$).

Dhaenens-Flipo (2001), propone un procedimiento bi-criterio de solución para el problema de programación de la producción de lotes de aluminio en máquinas paralelas no-relacionadas, en presencia de fechas de entrega común y tiempos de preparación dependientes de la secuencia, donde el objetivo es minimizar los costos en un horizonte de planificación determinado ($R_m|S_{ij}\ ,\ d_j=D|\sum Costo$). El procedimiento bi-criterio, se refiere a un tipo de optimización multi-objetivo, donde se ponderan los objetivos en la función objetivo. Su funcionamiento consta de una fase constructiva y una fase de mejoramiento local.

Adamopoulos & Pappis (1998), estudiaron el problema de máquinas paralelas no-relacionadas con objetivo de minimizar los adelantos y tardanzas ponderadas totales unidas a los costos, donde la fecha de entrega de todos los trabajos es una variable de decisión ($R_m|d_j=D|\sum(\alpha E_j+\beta T_j+Costo$). Desarrollan un procedimiento heurístico de tiempo polinomial para encontrar soluciones eficientes al problema. Hamad *et al*. (2003) proponen una aplicación de redes neuronales para resolver el mismo problema pero sin considerar el objetivo adicional de costo, y utilizan el modelo de perceptrón multicapa, logrando mostrar que es una efectiva y robusta metodología para encontrar soluciones cercanas a la óptima en problemas de tamaño modesto.

Lin *et al.* (2002) estudiaron un problema de planificación y programación de la producción en el área de producción de plásticos por moldeo con el objetivo de minimizar el máximo tiempo de completación (makespan), considerando un ambiente de máquinas paralelas no-relacionadas, y restricciones de fechas de entrega, tamaño de lote variable, tiempos de preparación dependientes de la secuencia, límites en capacidad de inventarios. Presentan el modelo general, el cual permite resolver problemas de tamaño muy pequeño, y proponen una metodología de descomposición para resolver problemas de mayor tamaño, el cual considera problemas de selección de máquinas, asignación de moldes a las máquinas y programación de la producción en las máquinas.

Rabadi *et al.* (2004) analizaron el problema de programación de la producción en una máquina, con tiempos de preparación dependientes de la secuencia y fecha de entrega común, donde el objetivo es minimizar las tardanzas y adelantos totales ($1|S_{ij}|\sum(E_j+T_j)$), y proponen un algoritmo de ramificación y acotamiento, que parte con una solución inicial (cota superior) obtenida a partir de una regla heurística (SAPT), basada en programar primero los trabajos con el menor tiempo de procesamiento ajustado.

La literatura vinculada al problema en estudio es escasa, y la mejor aplicación encontrada en la literatura para resolver el problema es la aplicación de Meta-RaPS propuesta por Rabadi *et al.* (2006). Por lo tanto es necesario desarrollar aplicaciones metaheurísticas que generen mejores resultados que los encontrados en literatura. A diferencia de Rabadi *et al.* (2006), en este estudio se considera el uso de una heurística constructiva visionaria (look-ahead) para construir soluciones en la fase constructiva de Meta-RaPS.

2.2 Formulación matemática del problema.

La formulación matemática del problema $R_m|S_{ijk}|C_{max}$ puede desarrollarse desde dos perspectivas. Ellas son:

- Minimización de la máxima carga de trabajo en las máquinas (Muñoz *et al.,* 2005).
- Minimización del máximo tiempo de completación de los trabajos (Guinet, 1991).

Para la formulación se utilizará la siguiente notación:

Parámetros:

n: cantidad de trabajos.

m: cantidad de máquinas.

P_j^k: tiempo de procesamiento del trabajo j en la máquina k.

S_{ij}^k: tiempo de preparación dependiente de la secuencia para procesar el trabajo j después del trabajo i en la máquina k.

S_{0j}^k: tiempo de preparación inicial si el trabajo j es el primero programado en la máquina k.

M: es un número positivo suficientemente grande.

Variables:

$X_{ij}^k = 1$, si el trabajo j es procesado directamente después del trabajo i en la máquina k y 0 en otro caso.

$X_{0j}^k = 1$, si el trabajo j es el primer trabajo procesado en la máquina k y 0 en otro caso.

$X_{i0}^k = 1$, si el trabajo i es el último trabajo en ser procesado en la máquina k y 0 en otro caso.

WL^k: carga de trabajo en la máquina k.

C_j: tiempo de completación del trabajo j.

La formulación desde el punto de vista de las máquinas es la siguiente (Muñoz *et al.* 2005):

$$Minimizar \quad C_{\max} \tag{2.1}$$

Sujeto a:

$$\sum_{\substack{i=o \\ i\neq j}}^{n}\sum_{k=1}^{m} X_{ij}^{k} = 1; \qquad \forall\, j = 1,...,n \tag{2.2}$$

$$\sum_{j=1}^{n} X_{0j}^{k} = 1; \qquad \forall\, k = 1,...,m \tag{2.3}$$

$$\sum_{i=1}^{n} X_{i0}^{k} = 1; \qquad \forall\, k = 1,...,m \tag{2.4}$$

$$\sum_{\substack{i=0 \\ i\neq h}}^{n} X_{ih}^{k} = \sum_{\substack{j=0 \\ j\neq h}}^{n} X_{hj}^{k}; \qquad \forall\, h = 1,...,n \quad \forall\, k = 1,...,m \tag{2.5}$$

$$WL^k = \sum_{i=0}^{n}\sum_{\substack{j=1 \\ j\neq i}}^{n} (P_j^k + S_{ij}^k) X_{ij}^k; \qquad \forall\, k = 1,...,m \tag{2.6}$$

$$C_{\max} \geq WL^k; \qquad \forall\, k = 1,...,m \tag{2.7}$$

$$WL^k \geq 0; \qquad \forall\, k = 1,...,m \tag{2.8}$$

$$X_{ij}^k \in \{0,1\}; \qquad \forall\, i = 1,...,n \quad \forall\, j = 1,...,n \quad \forall\, k = 1,...,m \tag{2.9}$$

En el modelo previo la función objetivo (2.1) es minimizar el máximo tiempo de completación (makespan); el grupo de restricciones (2.2) asegura que cada trabajo es programado sólo una vez y es procesado por una máquina; el grupo (2.3) asegura que cada máquina tiene sólo un trabajo inicial; el grupo (2.4) asegura que cada máquina tiene sólo un trabajo final; el grupo (2.5) asegura que cada trabajo tiene un predecesor y un sucesor en la

máquina. El grupo de restricciones (2.6) determina la carga de trabajo en cada máquina. El grupo (2.7) es usado para calcular el máximo tiempo de completación (makespan); y el grupo (2.8) asegura la no-negatividad de las cargas de trabajo. Mientras, el grupo (2.9) especifica que cada variable de decisión X_{ij}^{k} es binaria.

La formulación desde el punto de vista de los trabajos (Guinet, 1991) es:

$$Minimizar \quad C_{\max} \tag{2.10}$$

S*ujeto a:*

$$\sum_{\substack{i=o \\ i\neq j}}^{n}\sum_{k=1}^{m} X_{ij}^{k} = 1; \qquad \forall\, j = 1,...,n \tag{2.11}$$

$$\sum_{\substack{i=0 \\ i\neq h}}^{n} X_{ih}^{k} = \sum_{\substack{j=0 \\ j\neq h}}^{n} X_{hj}^{k}; \qquad \forall\ h = 1,...,n \quad \forall\ k = 1,...,m \tag{2.12}$$

$$C_j \geq C_i + \sum_{k=1}^{m} X_{ij}^{k}\left(S_{ij}^{k} + P_j^{k}\right) + M\left(\sum_{k=1}^{m} X_{ij}^{k} - 1\right); \quad \forall i = 0,...,n \quad \forall j = 1,...,n \tag{2.13}$$

$$C_j \leq C_{\max}; \qquad \forall\, j = 1,...,n \tag{2.14}$$

$$\sum_{j=1}^{n} X_{0j}^{k} = 1; \qquad \forall\ k = 1,...,m \tag{2.15}$$

$$C_j \geq 0; \qquad \forall\, j = 1,...,n \tag{2.16}$$

$$C_0 = 0 \tag{2.17}$$

$$X_{ij}^{k} \in \{0,1\}; \qquad \forall\ i = 1,...,n \quad \forall\, j = 1,...,n \quad \forall\ k = 1,...,m \tag{2.18}$$

En el modelo previo la función objetivo (2.10) es minimizar el máximo tiempo de completación (makespan); el grupo de restricciones (2.11) asegura que cada trabajo es programado sólo una vez y es procesado por una máquina; el grupo (2.12) asegura que cada trabajo tiene un predecesor y un sucesor en la máquina. El grupo de restricciones (2.13) es usado para calcular el tiempo de completación de cada trabajo y asegura que ningún trabajo

puede preceder y suceder al mismo trabajo; el grupo (2.14) es usado para calcular los tiempos de completación. El grupo de restricciones (2.15) asegura que cada máquina tiene sólo un trabajo inicial; el grupo (2.16) asegura la no-negatividad de los tiempos de completación. La restricción (2.17) establece que el tiempo de completación para el trabajo nulo es cero. Mientras el grupo (2.18) especifica que cada variable de decisión X_{ij}^{k} es binaria.

Ambos modelos matemáticos permiten resolver en forma óptima el problema en estudio, pero su aplicación es viable para problemas de tamaño pequeño debido a los largos tiempos de respuesta. El modelo cuya formulación se realiza desde el punto de vista de las máquinas tiene una menor cantidad de variables y restricciones que el modelo propuesto por Guinet (1991), por lo que es recomendable su utilización.

Tabla N°2.1: Resumen revisión bibliográfica.

Autor	Ambiente	Restricción	Objetivo(s)	Técnica
Rabadi *et al.* (2004)	1	S_{ij}, $d_j=D$	$\sum(E_j+T_j)$	Ramificar & Acotar
Min & Cheng (1999)	P_m		C_{max}	Algoritmos Genéticos
Armentano & Yamashita (2000)	P_m		$\sum T_j/n$	Tabú Search
Yalaoui & Chu (2003)	P_m	S_{ij}	C_{max}	Descomposición al TSP y fase de mejoramiento
Lee & Pinedo (1997)	P_m	S_{ij}	$\sum w_j T_j$	Reglas de despacho y Tabú Search
Guinet (1993)	P_m	S_{ij}	C_{max}, $\sum C_j/n$	Algoritmo de Asignación y Ruteo de Vehículos
Hurink & Knust (2001)	P_m	*prec*, S_{ij}	C_{max}	Heurística basada en Programación Dinámica
Vredeveld & Hurkens (2002)	R_m		$\sum w_j C_j$	Heurísticas basadas en Programación Lineal y Búsqueda Local
Liaw *et al.* (2003)	R_m		$\sum w_j T_j$	Ramificar & Acotar
Horowitz & Sahni (1976)	R_m		C_{max}	Algoritmos exactos y aproximados
Ibarra & Kim (1977)	R_m		C_{max}	Algoritmos Heurísticos
Srivastava (1998)	R_m		C_{max}	Tabú Search
Mokotoff & Jimeno (2002)	R_m		C_{max}	Heurísticas basada en enumeración parcial
Adamopoulos & Pappis (1998)	R_m	$d_j=D$	$\sum(\alpha E_j+\beta T_j+Costo)$	Heurística Constructiva
Hamad *et al.* 2003	R_m	$d_j=D$	$\sum(\alpha E_j+\beta T_j)$	Redes Neuronales
Dhaenens-Flipo (2001)	R_m	$d_j=D$, S_{ij}	$\sum Costo$	Multiobjetivo, heurísticas de construcción y búsqueda local
Yu *et al.* (2002)	R_m	M_j	$\sum Costo$	Relajación Lagrangeana
Herrmann *et al*. (1997)	R_m	*prec*	C_{max}	Heurística constructiva y Simulated Annealing
Lin *et al.* (2002)	R_m	d_j, S_{ijk}	C_{max}	Descomposición en subproblemas
Guinet, A. (1991)	R_m	S_{ij}	$\sum C_j$	Algoritmos de Asignación y Ruteo de Vehículos
Weng *et al.* (2001)	R_m	S_{ij}	$\sum w_j C_j/n$	7 Heurísticas Constructivas
Rabadi *et al.* (2006)	R_m	S_{ijk}	C_{max}	Meta-RaPS: fase constructiva glotona
Muñoz *et al.* (2005)	R_m	S_{ijk}	C_{max}	Meta-RaPS: fase constructiva visionaria
Helal & Hosni (2003)	R_m	S_{ijk}	C_{max}	Tabú Search
Anagnostopoulos & Rabadi (2002)	R_m	S_{ijk}	C_{max}	Simulated Annealing
Al-Salem (2004)	R_m	S_{ijk}	C_{max}	Heurística Multi-Fase
Kim *et al.* (2002)	R_m	S_{ijk}	$\sum T_j$	Simulated Annealing
Armacost & Salem (1999)	R_m	S_{ijk}, M_j	C_{max}	Heurística Multi-Fase

CAPITULO 3: META-RAPS.

El objetivo de los métodos de optimización es encontrar algoritmos que garanticen soluciones óptimas. El principal interés en la práctica, es encontrar soluciones cercanas al óptimo o soluciones de buena calidad en una cantidad razonable de tiempo, es decir, en un tiempo polinomial con respecto al tamaño del problema. La mayoría de los métodos de optimización combinatoria pueden ser clasificados en: exactos (por ejemplo ramificación y acotamiento, y programación dinámica) y heurísticos. Los algoritmos heurísticos tienen la remota posibilidad de encontrar la solución óptima, pero su fortaleza radica en lograr el mejor trade-off entre tiempo de respuesta y calidad de la solución generada.

Meta-RaPS (Meta-heuristic for Randomized Priority Search) surgió como consecuencia de una investigación desarrollada por las Universidades de Louisville y Central Florida, sobre la aplicación de un enfoque modificado de COMSOAL (Computer Method of Sequencing Operations for Assembly Lines) para resolver diversos problemas combinatorios. COMSOAL es una heurística originalmente introducida por Arcus (1966) como enfoque de solución al problema de balanceo de líneas de ensamble. A pesar del hecho de que cada versión modificada de COMSOAL conserva en esencia las ideas básicas de Arcus, en la práctica, las versiones discutidas por DePuy & Whitehouse (2000, 2001) y DePuy *et al.* (2000) difieren considerablemente de la versión original de COMSOAL. Meta-RaPS puede ser considerado una forma más general de COMSOAL. Sin embargo, Meta-RaPS es también la forma general de otra metaheurística llamada GRASP (Greedy Randomized Adaptive Search Procedure; Feo & Resende, 1989, 1995; Resende &

González, 2003). GRASP construye soluciones introduciendo aleatoriedad a una heurística de construcción glotona (greedy) mediante el uso de un parámetro (porcentaje de restricción) en la misma forma como lo hace Meta-RaPS, pero no considera la combinación aleatoria de reglas. GRASP también incluye mejoramiento mediante técnicas búsqueda local aplicada a todas las soluciones construidas. Meta-RaPS en cambio aplica técnicas de búsqueda local sólo a aquellas soluciones que ofrecen mejor potencial de mejoramiento. De modo que Meta-RaPS puede imitar GRASP cuando no existe combinación de heurísticas y todas las soluciones pasan por mejoramiento.

Meta-RaPS ofrece mayor flexibilidad que COMSOAL y GRASP debido a que permite fijaciones de parámetros definidas por el usuario. Adicionalmente, GRASP usa mayormente heurísticas glotonas mientras que Meta-RaPS puede ser usado con heurísticas glotonas (greedy) y con otras más inteligentes o visionarias (look-ahead).

GRASP a sido aplicado a una variedad de problemas de scheduling (Feo *et al*., 1991; Laguna & González, 1991; Feo *et al.*, 1996; Aiex *et al.*, 2003; & Rojanasoonthon & Bard, 2004), sin embargo hasta la fecha no se a utilizado GRASP para resolver el problema $R_m|S_{ijk}|C_{max}$.

Según Rabadi *et al.* (2006), Meta-RaPS es una estrategia maestra que utiliza heurísticas de construcción y mejoramiento para generar soluciones de alta calidad, de tal forma que es una estrategia genérica de alto nivel usada para modificar heurísticas de construcción (o reglas de prioridad) basándose en la inserción de aleatoriedad controlada. De tal forma que Meta-RaPS integra reglas de prioridad, aleatoriedad y muestreo. En cada iteración, Meta-RaPS construye y mejora soluciones factibles a través de la utilización de

reglas de prioridad utilizadas de manera aleatorizada. Después de un número dado de iteraciones, Meta-RaPS reporta la mejor solución encontrada.

Meta-RaPS es una metaheurística de tipo multi-arranque. Marti & Moreno (2003) ofrecen una buena descripción de métodos multi-arranque, tanto para problemas combinatorios como para problemas de optimización global.

Meta-RaPS toma ventaja del hecho que aplicando aleatoriedad controlada en una regla de construcción, por sencilla que esta sea, se encuentran mejores soluciones. Además de tomar ventaja de la bondad de las heurísticas de construcción y diversificarlas usando aleatoriedad, Meta-RaPS es una heurística rápida en términos computacionales. Para Meta-RaPS, como otras metaheurísticas, la aleatoriedad representa un dispositivo para evitar quedar atrapado en una solución óptima local (Rabadi *et al.,* 2006).

La filosofía detrás de Meta-RaPS considera dos ideas básicas que han sido demostradas empíricamente:

1. La incorporación de aleatoriedad en una heurística particular puede mejorar la calidad de las soluciones obtenidas.
2. La combinación aleatoria de heurísticas puede llevar a la obtención de mejores resultados que cada heurística individualmente.

Meta-RaPS controla su desempeño mediante el uso de cuatro parámetros:

- *%p* (porcentaje de prioridad): mide el número porcentual de veces que una regla de prioridad (heurística) es usada. Su valor puede ser fijado en el rango [0-100%].

- *%r* (porcentaje de restricción): mide porcentualmente el tamaño de una lista de candidatos, es decir, mide que tan comprimida o expandida será la lista. Su valor puede ser fijado en el rango [0-100%].
- *%i* (porcentaje de mejoramiento): factor que mide el aumento o reducción de la posibilidad de que soluciones construidas pasen por un mejoramiento local. Su valor puede ser fijado en el rango [0-100%].
- *I* (cantidad de iteraciones): mide el número de veces que se repite el proceso completo (construcción y mejoramiento).

Durante la ejecución de Meta-RaPS, el número de iteraciones determina el número de soluciones factibles construidas, y en general, la probabilidad de acercarse al óptimo global aumenta a medida que la cantidad de iteraciones aumenta, a un costo computacional mayor. Comúnmente, una heurística constructiva construye una solución agregando sistemáticamente actividades factibles a la solución parcial. En el problema $R_m|S_{ijk}|C_{max}$, el trabajo con mejor prioridad es agregado al programa de producción parcial. Meta-RaPS modifica la forma general en que una heurística de construcción elige el próximo trabajo a ser agregado a la solución, mediante la elección ocasional de un trabajo que no tiene la mejor prioridad. En la fase de construcción, el parámetro *%p* es usado para determinar el porcentaje de veces que la próxima actividad es agregada a la solución mediante el uso de la regla de prioridad (heurística constructiva). El porcentaje remanente de veces (100%-*%p*), la próxima actividad a ser agregada a la solución es aleatoriamente elegida desde un subconjunto de actividades factibles, cuyo valor de prioridad es a lo menos *%r* menor que el mejor valor de prioridad. Siguiendo en esta vía, Meta-RaPS garantiza un proceso de

diversificación cuando se construye una solución, mientras conserva la calidad producida por la regla de la prioridad usada (Rabadi *et al.,* 2006).

Normalmente una heurística de generación de soluciones funciona mediante una lista (regla de prioridad). Esta lista puede ser construida usando un criterio glotón o "regla greedy", o un mecanismo más visionario o "regla look-ahead".

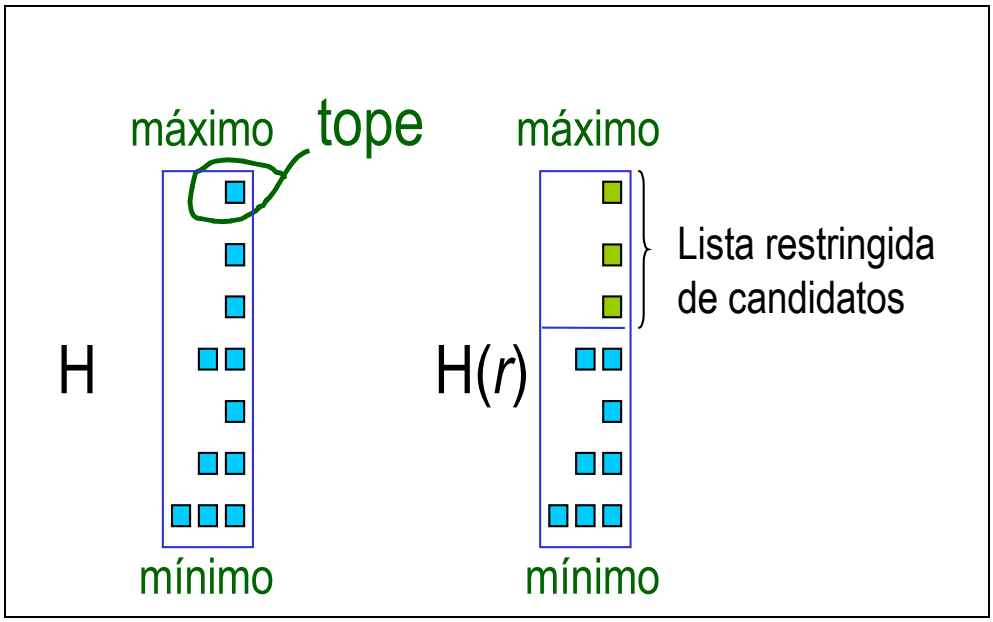


Figura Nº3.1: Regla de construcción aleatorizada.

Al usar una heurística de construcción H, los elementos son priorizados de acuerdo a algún criterio y se selecciona aquél que está en el tope de la lista (Figura Nº3.1). Si se aplica aleatoriedad usando el parámetro *%r*, se modifica la regla y se construye una sublista (lista restringida de candidatos) con un grupo de mejores candidatos, lo que da origen a H(*r*), en la Figura Nº3.1 se muestran 3 candidatos con la misma posibilidad de ser seleccionados. Algunos casos particulares de valores para el parámetro *%r* son:

- *%r* = 0, asignación basada en la regla H original y no existe variabilidad, con lo que se obtiene una buena solución subóptima.
- *%r* = 100, asignación altamente aleatorizada, se tiene alta variabilidad y soluciones pobres.

Una forma muy utilizada para crear la lista restringida de candidatos (LRC) es la siguiente:

$a_j \in LRC \Leftrightarrow prioridad(a_j) \geq \left(mayor\_prioridad \times (1-r) + menor\_prioridad \times r\right)/100$,

para problemas donde lo que se busca son elementos con mayor prioridad.

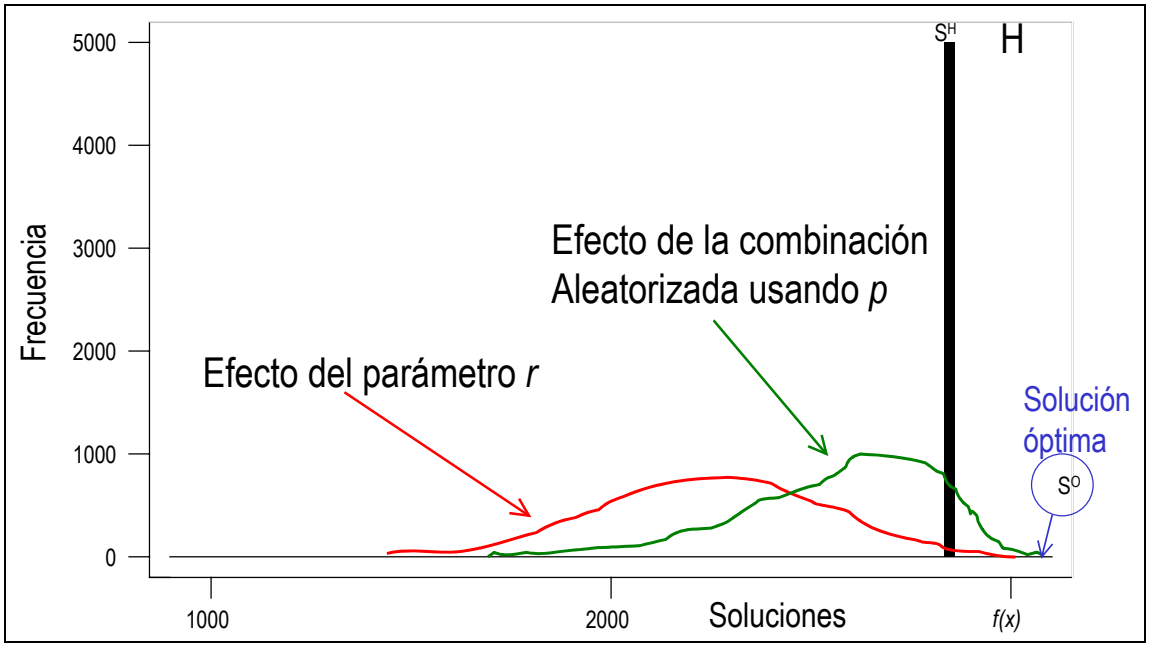


Figura Nº3.2: Efecto de los parámetros %p y %r en la regla H.

En la Figura Nº3.2 se muestra un gráfico (hipotético) del efecto de los parámetros *%p* y *%r* en un problema donde se desea maximizar la función objetivo. Si la regla H se

usara durante 5000 iteraciones, se obtendría 5000 veces el resultado S^H, de existir algún empate se observaría un gráfico multi-modal. El objetivo del parámetro *%r* es crear una regla aleatorizada H(*r*), cuyo efecto es modificar la regla H para poder encontrar algunas soluciones mejores que S^H, de tal forma que permita acercarse a la solución óptima del problema. El objetivo del parámetro *%p* es mezclar aleatoriamente las reglas, su efecto es desplazar la población potencial de soluciones en la dirección de la solución óptima, de tal forma que logra mejorar las posibilidades de encontrar soluciones más cercanas al óptimo.

Adicionalmente, Meta-RaPS considera una fase de mejoramiento de soluciones, la cual es controlada mediante el parámetro *%i*. El objetivo y efecto del parámetro *%i* es acercar las soluciones construidas a la solución óptima. Esto se realiza aplicando una técnica de búsqueda local para un número seleccionado de soluciones, es importante destacar que aquí también se puede aplicar aleatorización de reglas, y que las técnicas de mejoramiento permiten generalmente encontrar un óptimo local en el vecindario de una solución construida.

Una vez que un programa de producción es construido, Meta-RaPS puede proceder a mejorarlo con técnicas de búsqueda en la vecindad (mejoramiento local). El parámetro *%i* determina cuales soluciones pasarán por el algoritmo de mejoramiento. De tal forma que una determinada función $\Psi(\%i)$, determinará la calidad exigida para que una solución pase a la rutina de mejoramiento. Finalmente, la mejor solución encontrada entre todas las iteraciones es entregada como resultado. Una forma muy utilizada para la función $\Psi(\%i)$ en problemas de maximización es la siguiente:

$$\Psi(\%i) = \left(mayor \times (1-i) + menor \times i\right)/100 \qquad (3.1)$$

De tal forma que si *x* es una solución construida, y se cumple que la función objetivo de la solución construida es superior al umbral $\Psi(\%i)$, la solución *x* es aceptada para pasar por mejoramiento. Esta condición se representaría de la siguiente forma:

x es aceptada, si y solo si, función_objetivo(*x*) ≥ $\Psi(\%i)$.

Si el problema es de minimización, la función $\Psi(\%i)$ es la siguiente:

$$\Psi(\%i) = \left(menor \times (1-i) + mayor \times i\right)/100 \qquad (3.2)$$

De tal forma que si *x* es una solución construida, y se cumple que la función objetivo de la solución construida es inferior al umbral $\Psi(\%i)$, la solución *x* es aceptada para pasar por mejoramiento. Esta condición se representaría de la siguiente forma:

x es aceptada, si y solo si, función_objetivo(*x*) ≤ $\Psi(\%i)$.

En general, el problema de fijación de parámetros para Meta-RaPS es muy importante, sobre todo para los parámetros que influyen en la fase de construcción de soluciones (*%prioridad* y *%restricción*). Por otra parte los parámetros *iteraciones* y *%mejoramiento*, pueden ser fijados con mayor facilidad. Esto se debe a que mientras mayor sean sus valores fijados, mayores posibilidades tendrá el algoritmo de acercarse a la solución óptima, a un costo computacional mayor. Para lograr una buena fijación de estos parámetros (*iteraciones* y *%mejoramiento*) se considera comúnmente el mejor compromiso entre calidad de la solución generada y el tiempo de corrida.

En la Figura Nº3.3 se muestra el algoritmo genérico de Meta-RaPS para un problema de minimización (fase integral) y en la Figura Nº3.4 se muestra en algoritmo genérico para la fase de construcción.

```
mejor_objetivo = infinito;                          % caso de minimización
repetir I veces
        x = Construcción(%p, %r);                   % construye solución
        si función_obj(x) ≤ Ψ(%i),                  % regla filtro de mejoramiento
            entonces x = mejoramiento(x);           % búsqueda mejor solución
        si función_obj(x) ≤ mejor_objetivo,         % actualización
            entonces mejor_objetivo ← función_obj(x); x* ←x;
        fin
reportar x*;
fin
```

Figura Nº3.3: Algoritmo genérico de Meta-RaPS para minimización.

```
Construcción(%p, %r)                    % construye una solución
    x = ϕ
    para j=1, 2, ..., n
        si n_ aleatorio(#) ≤ p          % regla para decir el uso de regla H o H(r)
            entonces a_j=H;             % se agrega elemento usando regla H
        de otro modo a_j=H(r);          % se agrega elemento usando regla H(r)
        x ← x ∪ a_j                     % se agrega el elemento a la solución
    fin
fin
```

Figura Nº3.4: Algoritmo genérico de la fase constructiva de Meta-RaPS.

Usando Meta-RaPS con parámetros *%p*= 0, *%r*= 100%, y *%i*= 0 se logra imitar a COMSOAL. Usando Meta-RaPS con parámetros *%p*= 0, y *%i*= 100% se logra imitar a GRASP.

Meta-RaPS ofrece gran flexibilidad comparado con COMSOAL y GRASP, ya que permite al usuario definir los tres parámetros involucrados (*%p*, *%r*, y *%i*). DePuy *et al*. (2005[1]) muestran que la flexibilidad adicional producida por la forma general de Meta-RaPS parece beneficiosa, demostrando que los resultados obtenidos por Meta-RaPS

mediante un ajuste de parámetros son mejores que los resultados que entregan un ajuste de parámetros para COMSOAL y GRASP.

La aplicación de Meta-RaPS a cualquier problema combinatorio como una metodología general considera los siguientes pasos (Moraga *et al*., 2002, 2004 y DePuy *et al.,* 2005[1]):

1. Estudiar la estructura del problema a ser resuelto. Conseguir una buena comprensión del tipo de variables, restricciones, y la optimización requerida para solucionar el problema.
2. Encontrar reglas de prioridad (heurísticas) apropiadas que construyan soluciones factibles.
3. Modificar las reglas de prioridad usando aleatoriedad.
4. Construir soluciones factibles usando reglas de prioridad y aleatoriedad.
5. Mejorar localmente las soluciones.
6. Mantener la mejor solución después de determinado número de iteraciones.

Como metaheurística, Meta-RaPS es un prometedor enfoque para resolver problemas de optimización combinatoria (DePuy *et al.* 2001, 2002, 2003 y Moraga *et al.* 2004) y a sido aplicado exitosamente a varios problemas de optimización combinatorial, logrando muy buenos resultados, en algunos casos sus resultados han sido mejores que otros enfoques meta-heurísticos tales como Algoritmos Genéticos, Simulated Annealing, Tabú Search y otros enfoques basados en características particulares del problema. Algunas aplicaciones de Meta-RaPS son:

- Problema de programación de proyectos con recursos restringidos (DePuy *et al*. 2005[2]),
- Problema del agente viajero (DePuy *et al*. 2005[1]),
- Problema de set covering (Lan *et al.* 2005),
- Problema de la mochila multidimensional 0-1 (Moraga *et al.* 2005),
- Problema de ruteo de vehículos (Moraga *et al.* 2002).

CAPITULO 4: META-RAPS, FASE CONSTRUCTIVA.

Meta-RaPS utiliza una fase constructiva, para la cual es necesario buscar en la literatura heurísticas constructivas de tipo glotón (greedy) o visionarias (look-ahead). En general, una heurística constructiva, permite encontrar buenas soluciones factibles rápidamente. Meta-RaPS ha sido aplicado al problema $R_m|S_{ijk}|C_{max}$ utilizando una heurística constructiva glotona (Moraga 2004 y Rabadi *et al.* 2006), reportando buenos resultados.

En éste capítulo, se presenta una heurística constructiva visionaria (look-ahead) para resolver el problema de máquinas paralelas no-relacionadas con tiempos de preparación dependientes de la secuencia, donde el objetivo es minimizar el máximo tiempo de completación o makespan ($R_m|S_{ijk}|C_{max}$). Se explica su funcionamiento mediante un ejemplo numérico, y se comparan sus resultados con la heurística constructiva (SAP-SL) propuesta por Moraga (2004) y Rabadi *et al.* (2006). También se muestra como esa heurística puede ser modificada y mejorada agregando aleatoriedad controlada. Esa inclusión de aleatoriedad se realiza mediante los parámetros de porcentajes de prioridad y restricción que influyen en la fase constructiva de Meta-RaPS.

4.1 Heurística constructiva Look-Ahead para $R_m|S_{ijk}|C_{max}$.

Desarrollar una heurística constructiva para el problema $R_m|S_{ijk}|C_{max}$ es necesario porque el funcionamiento de muchas metaheurísticas requiere de soluciones iniciales, las cuales pueden ser entregadas por una heurística constructiva. Además, las heurísticas

glotonas construyen soluciones rápidamente, pero son miopes para evaluar el siguiente trabajo a ser programado, por lo tanto se espera que un enfoque visionario logre mejores resultados.

La heurística constructiva visionaria propuesta (llamada LACH, Look-Ahead Constructive Heuristic) se basa en criterios de ahorro. En la cual se asignan y secuencian iterativamente los trabajos a las máquinas, de manera de aumentar las cargas de trabajo de las máquinas lo menos posible. Lo cual tiene como resultado un programa de producción que logra un mejor balance de cargas de trabajo en las máquinas y un menor máximo tiempo de completación (*makespan o* C_{max}).

Sean m la cantidad de máquinas y n la cantidad de trabajos. Denotaremos por P^k_j al tiempo de procesamiento del trabajo j en la máquina k, y a S_{ij}^k como el tiempo de preparación que necesita la máquina k para procesar el trabajo j si anteriormente se realiza el trabajo i.

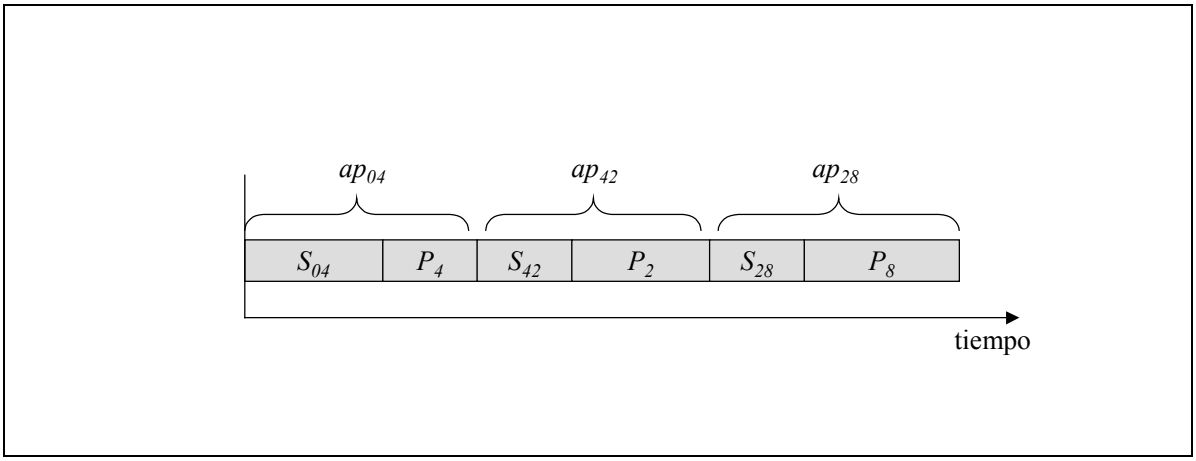


Figura N°4.1: Tiempos de procesamiento ajustado.

Para abordar el problema $R_m|S_{ijk}|C_{max}$, se utilizará el concepto de *Matriz de Tiempos de Procesamiento Ajustado* [AP^k] para cada máquina *k* propuesto por Rabadi *et al.* (2006) (Figura N°4.1). En general las *Matrices de Tiempos de Procesamiento Ajustado* [AP^k] para cada máquina se construyen utilizando la siguiente ecuación:

$$ap_{ij}^k = S_{ij}^k + P_j^k;\ \forall k = 1,...,m;\ \forall i = 1,...,n;\ \forall j = 1,...,n. \quad (4.1)$$

Donde:

ap_{ij}^k : tiempo de procesamiento ajustado, es decir, al tiempo necesario para realizar el trabajo *j* después del trabajo *i* en la máquina *k*.

Con lo anterior se pueden construir *m* matrices de orden $(n+1)\times n$, en las cuales se encontraran los tiempos de procesamiento ajustados, de tal forma que el elemento *(i,j)-ésimo* sea el tiempo de procesamiento ajustado si se realiza el trabajo *j* después del trabajo *i* en la máquina *k*. En esas matrices, los elementos de la primera fila, llamada fila cero (ap_{0j}^k), indican el tiempo necesario para procesar el trabajo *j* como el primero en el programa de trabajo de la máquina *k*.

En la Figura N°4.2 se muestra un diagrama de la heurística constructiva LACH.

Dada las características de la heurística, su aplicación estará limitada para problemas de tamaño pequeño, es decir, la heurística puede ser aplicada a problemas de gran tamaño. Específicamente a los problemas donde se cumpla la inecuación $n \geq 3m$, que indica que la cantidad de trabajos del problema debe ser mayor o igual a tres veces la cantidad de máquinas.

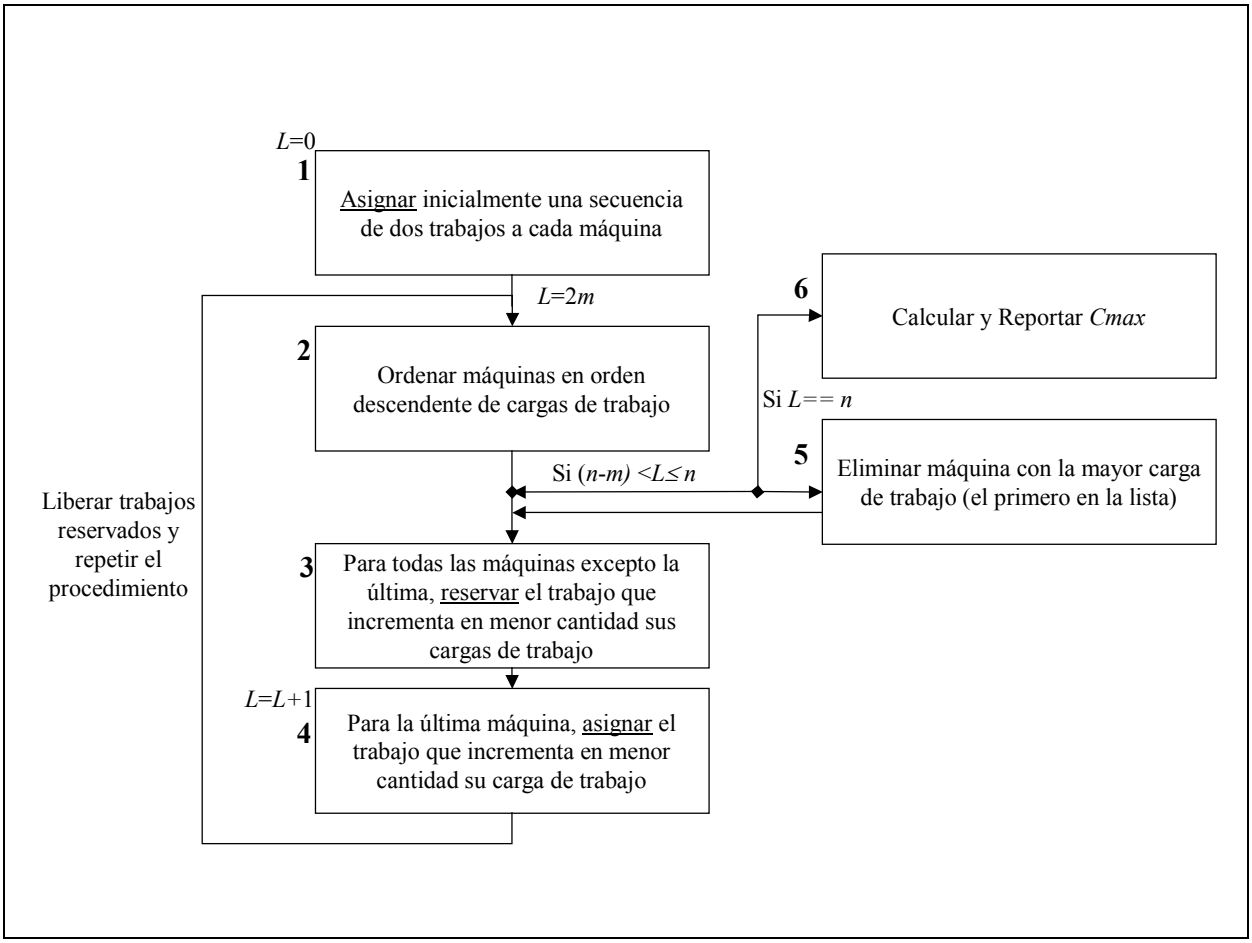


Figura N°4.2: Diagrama de LACH

A continuación se describe cada parte del diagrama detalladamente.

Parte 1: Asignar inicialmente una secuencia de dos trabajos a cada máquina.

Esta es la parte inicial de la heurística, y su objetivo es asignar una secuencia de dos trabajos a cada una de las *m* máquinas. Esa secuencia inicial no indica que el primer trabajo sea el trabajo inicial en el programa de producción. Para ello es importante hacer una diferencia entre el programa de producción final y el programa de producción parcial. Por lo tanto al finalizar esta parte, cada máquina tendrá un programa parcial compuesto por dos trabajos. En las siguientes etapas de la heurística, ese programa parcial de cada máquina

será completado, de tal manera que los demás trabajos serán secuenciados antes del *primer* trabajo y después del *segundo* trabajo del programa parcial.

La variable L es un contador de los trabajos que han sido programados. Por lo tanto, cuando se finalice esta etapa, L tendrá un valor igual a $2m$ (dos trabajos para cada máquina).

Un criterio glotón para seleccionar los dos trabajos del programa parcial asignado a cada máquina, podría considerar solamente que el tiempo de procesamiento ajustado asociado sea mínimo, por lo que bastaría buscar en las matrices de tiempos de procesamiento ajustado aquel ap_{ij}^{k} menor para una determinada máquina k. Sin embargo, utilizar ese criterio podría llevar a programas parciales muy malos. Eso se debe principalmente a que no se considera que trabajo puede ser asignado antes del trabajo i o después del trabajo j. Para considerar esto LACH utiliza un estimador, que indica que tan prometedora es la asignación de dos trabajos en una máquina en términos de aumentar en mínima forma su carga de trabajo. En la Figura N°4.3 se muestra gráficamente una evaluación de dos trabajos (x e y), donde $F(x)$ y $F(y)$ son estimadores del tiempo de procesamiento ajustado de los trabajos que podrían asignarse antes del trabajo x, y después del trabajo y respectivamente.

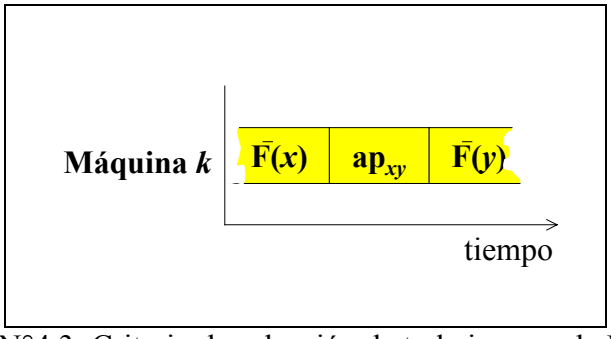


Figura N°4.3: Criterio de selección de trabajos para la Parte 1.

El estimador, llamado cota inferior (lower bound) se calcula mediante la siguiente expresión:

$$LB(x,y) = \min_{i \in J}\{ap_{i\,x}\} + ap_{x\,y} + \min_{j \in J}\{ap_{y\,j}\} \quad (4.2)$$

De donde se puede observar que los estimadores *F(x)* y *F(y)* son:

$$\bar{F}(x) = \min_{i \in J}\{ap_{i\,x}\} \quad (4.3)$$

$$\bar{F}(y) = \min_{j \in J}\{ap_{y\,j}\} \quad (4.4)$$

LACH utiliza los estimadores *F(x)* y *F(y)* para evitar la evaluación de una cantidad enorme de combinaciones que se pueden generar. Si *n* indica la cantidad de trabajos y *m* la cantidad de máquinas, la cantidad de combinaciones a evaluar sin el uso de los estimadores sería:

Combinaciones por búsqueda exhaustiva: $n(n-1)(n-2)(n-3)m \approx O(n^4m)$, para *n* suficientemente grande.

Mediante el uso de los estimadores *F(x)* y *F(y)*, la búsqueda se reduce a:

Combinaciones mediante uso de F(x) y *F(y):* $n(n-1)m \approx O(n^2m)$, para *n* suficientemente grande.

Supongamos un problema donde *n*=100 y *m*=10. Mediante búsqueda exhaustiva de combinaciones se deberían probar 941.094.000 combinaciones, mediante el uso del estimador esa búsqueda se reduce a 99.000, lo cual es significativamente menor.

Los estimadores *F(x)* y *F(y)* se obtienen buscando el menor elemento de la columna *x* y el menor elemento de la fila *y* en la matriz $[AP^k]$, sin considerar la fila cero. Esto es similar a buscar que trabajo programar antes de *x,* y después de *y* respectivamente, aumentando en menor cantidad la carga de trabajo de la máquina *k*.

En resumen, esta parte de la heurística consiste en asignar un par de trabajos mediante el estimador *LB* a cada máquina del problema.

Parte 2: Ordenar máquinas en orden descendente de cargas de trabajo.

Esta parte consiste simplemente en ordenar las máquinas en forma descendente según la carga de trabajo asociada a su programa parcial, de tal forma que la primera de la lista sea la que tiene la mayor carga de trabajo, y la última máquina de lista sea la que tiene la menor carga de trabajo. En la Figura N°4.4 se muestra un ejemplo, en el cual se observa coloreado el tiempo de procesamiento del programa parcial asociado a los trabajos asignados a cada máquina, y en achurado el tiempo de procesamiento asociado al trabajo inicial en la secuencia parcial.

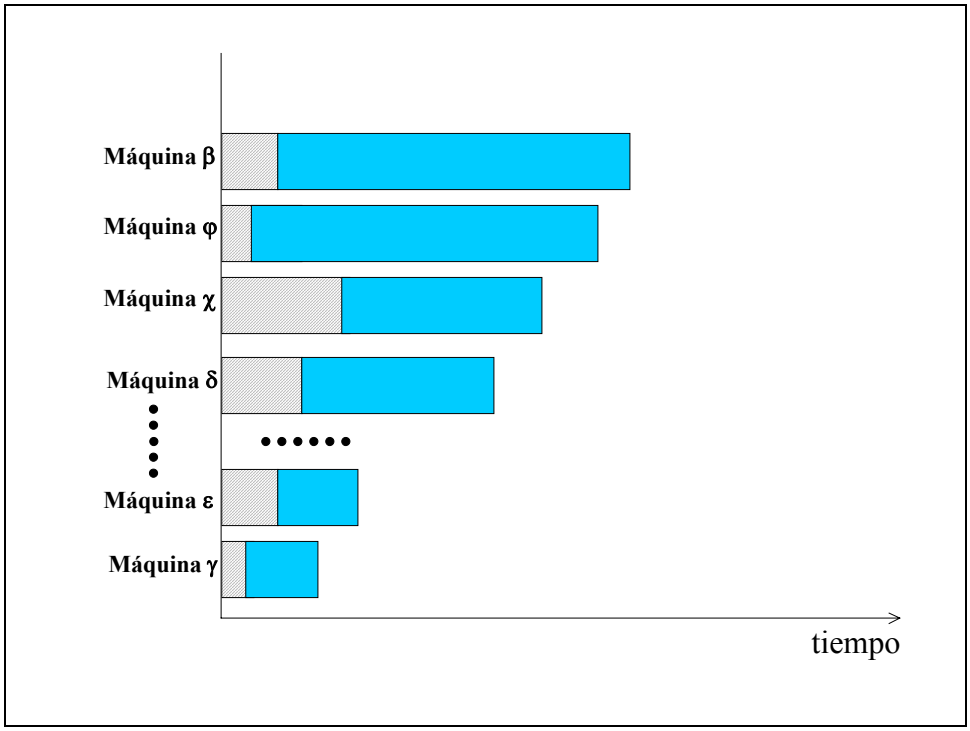


Figura N°4.4: Ordenamiento de las máquinas según su carga de trabajo parcial.

La idea fundamental de esta parte es ordenar los trabajos según una prioridad, de tal forma que la máquina que tenga la mayor carga de trabajo, es aquella que tiene mayores posibilidades de aumentar el máximo tiempo de completación (makespan) si se le asigna un trabajo adicional. Luego, la máquina que tiene la siguiente mayor carga de trabajo tiene la siguiente mayor prioridad y así sucesivamente. De tal forma que la máquina que tenga la menor carga de trabajo afectará en menor forma al máximo tiempo de completación (makespan) si se le asigna un trabajo adicional. Como se explicó en la Parte 1, ese trabajo puede asignarse antes del primer o después del último trabajo en la secuencia parcial.

El orden establecido en esta parte se utiliza para realizar el proceso de reserva de trabajos (Parte 3) y el proceso de asignación de trabajo (Parte 4).

Parte 3: Para todas las máquinas excepto la última, reservar el trabajo que incrementa en menor cantidad su carga de trabajo.

Para explicar más fácilmente este paso, consideraremos que la notación de una secuencia de trabajos parcial en cada máquina será dada por [*X*-...-*Y*], donde *X* e *Y* representan el primero y último trabajo respectivamente en la secuencia parcial de cada máquina. La heurística asigna o reserva trabajos al comienzo o al final de la secuencia parcial [*X*-...-*Y*] hasta completar el programa de producción (secuencia de trabajos completa) en cada máquina.

Se denotará por *Z* a un trabajo que es asignado o reservado antes de *X* (es decir, *Z*-*X*-...-*Y*); y de denotará por *W* a un trabajo que es asignado o reservado después de *Y* (es decir, *X*-...-*Y*-*W*).

En esta parte de la heurística, se reservan trabajos utilizando el orden establecido en la Parte 2. De tal forma que la máquina que tiene la mayor carga de trabajo busca dentro de los trabajos factibles, aquellos trabajos *Z* o *W* que aumentasen en menor cantidad su carga de trabajo. Aquel trabajo *Z* o *W* se saca temporalmente de la lista de trabajos factibles. Esto se repite de manera sucesiva con la siguiente máquina en la lista hasta llegar a la última máquina, para la cual no se realiza el proceso de reserva. En ese momento se sigue con el siguiente paso (proceso de asignación).

El proceso de reserva consiste básicamente en utilizar la información disponible para reservar un trabajo, utilizando un criterio basado en *"el peor de los casos"*. La filosofía detrás de esto es: *"si se fuese a asignar un trabajo en una máquina en particular, ese trabajo debiese afectar lo menos posible su carga de trabajo, afectando lo menos posible el máximo tiempo de completación (makespan)"*. Esta filosofía unida al orden entregado por la lista creada en el Paso 2, permiten usar el criterio basado en el peor de los casos. En la Figura N°4.5 se muestra esquemáticamente la forma como se realiza el proceso de reserva de trabajo en una máquina (máquina *k*), cuyo programa parcial es: *3-8-1-7-2-5*.

Las dos formas de reservar trabajo se denotarán por P1 y P2. Donde P1 consiste en asignar un trabajo (*W*) después del último trabajo (*Y*) en el programa parcial, y P2 consiste en asignar un trabajo (*Z*) antes del primer trabajo (*X*) en el programa parcial. Luego el trabajo reservado se obtiene del menor tiempo asociado a P1 o P2. Esto es lo mismo que buscar el menor elemento en la columna *X* de la matriz [AP^k] para obtener *Z*, y buscar el menor elemento en la fila Y de la matriz [AP^k] para obtener W (sin considerar la fila cero de cada matriz).

En la Figura N°4.5, el tiempo asociado a P2 es menor que el asociado a P1, por lo tanto el trabajo reservado para la máquina *k* es el trabajo *Z*.

En definitiva para cada máquina en el proceso de reserva es necesario evaluar la siguiente regla:

Calcular:

$P_1 = ap_{0\,X} + ap_{Y\,W}$ *W* pertenece a los trabajos factibles no reservados

$P_2 = ap_{0\,Z} + ap_{Z\,X}$ *Z* pertenece a los trabajos factibles no reservados

Si $P_1 < P_2$, reservar el trabajo *W*
Si $P_1 > P_2$, reservar el trabajo *Z*
En caso de empate, elegir arbitrariamente entre *W* o *Z* y reservarlo.

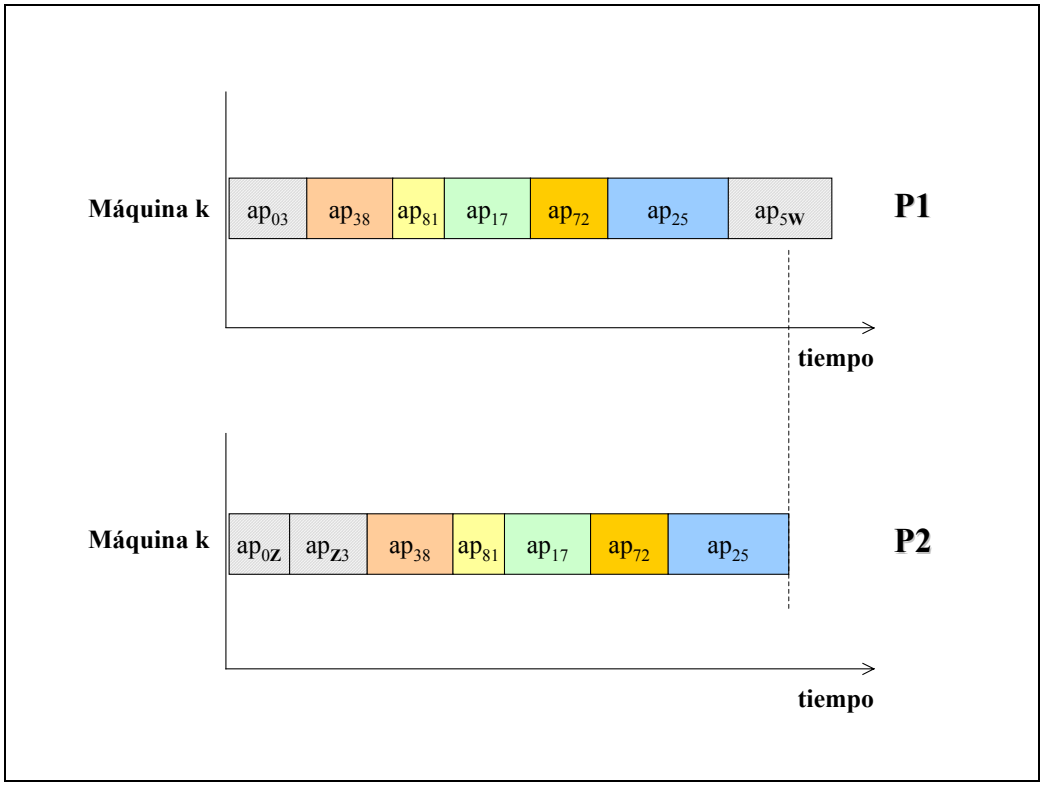


Figura N°4.5: Esquema del proceso de reserva de trabajo.

Parte 4: Para la última máquina, *asignar* el trabajo que incrementa en menor cantidad su carga de trabajo.

En esta parte, se considera solamente la máquina que tiene el programa de producción con la menor carga de trabajo, sin considerar los trabajos reservados en la Parte 3. En la máquina se realiza el proceso de asignación, que es exactamente igual que el proceso de reserva, con la salvedad que en esta parte el trabajo no es reservado, sino que asignado. Por lo tanto en esta parte de la heurística se altera el programa parcial de la máquina que tiene la menor carga de trabajo. Al terminar esta parte, el contador de trabajos asignados (L) debe incrementarse en 1, y los trabajos reservados en la Parte 3 son liberados, es decir, se devuelven al conjunto de trabajos factibles.

En la Figura Nº4.6 se muestra un ejemplo de como se incrementa el C_{max} a medida que se van asignando los trabajos a las máquinas (problema de 12 máquinas y 120 trabajos).

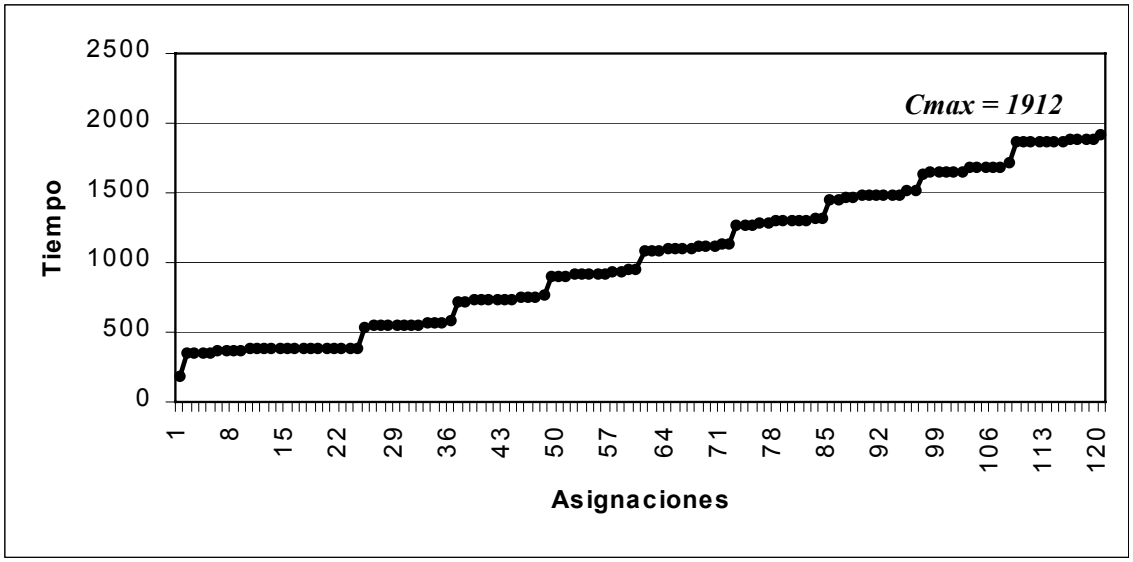


Figura Nº4.6: Variación del C_{max} en la resolución de un problema.

Parte 5: Eliminar máquina con la mayor carga de trabajo (el primero en la lista).

Esta parte se realiza siempre que la cantidad de trabajos factibles sea menor que la cantidad de máquinas por programar, y consiste en eliminar una máquina. La máquina eliminada es aquella que tiene la mayor carga de trabajo asociada al programa parcial (máquina β en Figura N°4.4), de tal forma que al continuar el proceso se tendrán la misma cantidad de trabajos que máquinas.

Parte 6: Calcular y Reportar C_{max}.

Esta parte se realiza cuando todos los trabajos han sido asignados, y consiste en calcular el máximo tiempo de completación (makespan) y reportar los programas de producción de cada máquina.

Como se mencionó anteriormente, el diseño de LACH no permite resolver problemas de tamaño pequeño, específicamente los problemas donde no se cumpla la inecuación $n \geq 3m$. Esto se debe a que LACH comienza con la asignación de dos trabajos a cada máquina (Parte 1). Donde, si m es la cantidad de máquinas, se asigna una cantidad de $2m$ trabajos del total de trabajos del problema. Adicionalmente LACH requiere de un proceso completo de reserva y asignación de trabajos (Parte 3 y 4), por lo que se necesita como mínimo una cantidad de trabajos igual a la cantidad de máquinas (m). Por lo tanto para que LACH funcione se requiere de al menos $3m$ trabajos.

4.1.1 Algoritmo LACH.

Notación
m: cantidad de máquinas del problema.

n: cantidad de trabajos del problema.

MC^k: vector que contiene todos los mínimos elementos de cada columna en la matriz $[AP^k]$ para la máquina k sin considerar la fila cero. mc_i^k es el *i-ésimo* elemento en el vector MC^k.

MF^k: vector que contiene todos los mínimos elementos de cada fila en la matriz $[AP^k]$ para la máquina k sin considerar la fila cero. mf_j^k es el *j-ésimo* elemento en el vector MF^k.

ap^k_{ij}: es el *(i,j)-ésimo* elemento de la matriz $[AP^k]$ de tiempos de procesamiento ajustado para la máquina k.

WL^k: carga de trabajo en la máquina k.

Se considera que en caso de cualquier empate, ese se rompe arbitrariamente.

El conjunto VM contiene todas las máquinas disponibles a cargar con trabajos y s es la cantidad de máquinas disponibles a cargar.

El conjunto J contiene los trabajos disponibles para programar y L es un contador de la cantidad de trabajos asignados en cualquier momento.

M, T y b son variables de la heurística constructiva.

El algoritmo de esta heurística constructiva visionaria se presenta en los siguientes pasos:

Paso 0. $VM= \{1,\ldots, m\}$ y $J= \{1,\ldots, n\}$.

Inicialmente, $M= VM$, $L= 0$, $s= m$ y $WL^k= 0$, para $k= 1,..., m$.

Paso 1. Para cada $k \in M$, calcular vectores MC^k y MF^k sin considerar la fila cero. De tal forma que:

$mc_i^k = \min_{i \in J} \left\{ ap_{ij}^k \right\}$ para $j \in J$

$mf_j^k = \min_{j \in J} \left\{ ap_{ij}^k \right\}$ para $i \in J$.

Paso 2. Hacer: $\alpha_{i^* j^*}^{k^*} = \min_{\substack{k \in M \\ i \in J; j \in J \\ i \neq j}} \left\{ ap_{ij}^k + mc_i^k + mf_j^k \right\}$,

$L = L+2$, $WL^{k^*} = ap^{k^*}{}_{i^* j^*}$, $X^{k^*} = i^*$, $Y^{k^*} = j^*$, $J = J - \{i^*\} - \{j^*\}$, $M = M - \{k^*\}$.

Asignar la secuencia (i^*, j^*) a la máquina k^*. Eliminar fila i^*, columna j^* y elemento (j^*, i^*) de $[AP^{k^*}]$. Eliminar filas y columnas i^* y j^* de $[AP^k]$ para todo $k \neq k^*$.

Paso 3. Si $M = \phi$, ir al Paso 4.

En otro caso, ir a Paso 1.

Paso 4. Hacer $T = J$, $M = VM$ y $b = s$.

Si $b = 1$, ir al Paso 7.

Paso 5. Encontrar $k^* \in M$ tal que: $\lambda^{k^*} = \max_{k \in M} (WL^k + ap_{0X^k}^k)$

Para k^* encontrar los trabajos W^{k^*} y Z^{k^*} tal que $P_1{}^{k^*}$ y $P_2{}^{k^*}$ (costos de oportunidad) sean mínimos:

$$P_1^{k^*} = ap_{0X^{k^*}}^{k^*} + ap_{Y^{k^*}W^{k^*}}^{k^*} \qquad W^{k^*} \in T$$
$$P_2^{k^*} = ap_{0Z^{k^*}}^{k^*} + ap_{Z^{k^*}X^{k^*}}^{k^*} \qquad Z^{k^*} \in T$$

Hacer $M = M - \{k^*\}$

Si $P_1^{k^*} < P_2^{k^*}$, hacer $T = T - \{W^{k^*}\}$

En otro caso, hacer $T = T - \{Z^{k^*}\}$

Paso 6. Hacer $b = b - 1$

Si $b = 1$, ir al Paso 7.

En otro caso, ir al Paso 5.

Paso 7. Hacer $L = L + 1$. Para la máquina remanente (h) en M, encontrar los trabajos W^h y Z^h tal que P_1^h y P_2^h sean mínimos:

$$P_1^h = ap_{0X^h}^h + ap_{Y^hW^h}^h \qquad W^h \in T$$
$$P_2^h = ap_{0Z^h}^h + ap_{Z^hX^h}^h \qquad Z^h \in T$$

Si $P_1^h < P_2^h$ asignar el trabajo W^h después del trabajo Y^h en la máquina h.

Hacer $WL^h = WL^h + ap_{Y^hW^h}^h$.

Eliminar fila Y^h, columna W^h y elemento (W^h, X^h) de [AP^h]. Adicionalmente, eliminar fila y columna W^h de [AP^h] para todo $k \neq h$. Hacer $Y^h = W^h$ y $J = J - \{W^h\}$.

En otro caso, asignar el trabajo Z^h antes de X^h en la máquina h. Hacer $WL^h = WL^h + ap_{Z^hX^h}^h$.

Eliminar fila Z^h, columna X^h y elemento (Y^h, Z^h) de [AP^h]. Adicionalmente, eliminar fila y columna Z^h de [AP^h] para todo $k \neq h$. Hacer $X^h = Z^h$ y $J = J - \{Z^h\}$.

Paso 8. Si $L \leq (n - m)$, ir al Paso 4.

Si $L > (n - m)$ y $L < n$, ir al Paso 9.

Si $L = n$, ir al Paso 10.

Paso 9. Hacer $\beta^{k*} = \underset{k \in VM}{max}\{WL^{k*} + ap^{k*}_{0X^{k*}}\}$. También $VM = VM - \{k*\}$ y $s = s - 1$.

Ir al Paso 4.

Paso 10. Hacer $WL^k = WL^k + ap^k_{0X^k}$ para $k = 1,...,m$. Retornar $C_{\max} = \underset{k=1,...,m}{max}\{WL^k\}$

Para mostrar el funcionamiento de LACH, en la siguiente sección se presenta un ejemplo numérico.

4.1.2 Aplicación de LACH a un ejemplo numérico.

Se considerará un ejemplo de dos máquinas ($m = 2$) y siete trabajos ($n = 7$), donde los tiempos de procesamiento y preparación de los trabajos se muestran en las Tablas N°4.1 y N°4.2 respectivamente. Como este problema cumple con la condición $n \geq 3m$, se puede aplicar LACH.

En la Tabla N°4.2, la fila cero indica el tiempo de preparación de la máquina para realizar un trabajo, si anteriormente no se ha realizado ninguno. Por ejemplo, si el trabajo 5 es el primero en programarse en la máquina 1, el tiempo de preparación de la máquina es 10 unidades de tiempo (de la Tabla N°4.2), y el tiempo de procesamiento es 26 unidades de tiempo (de la Tabla N°4.1).

Tabla N°4.1: Tiempos de procesamiento.

Trabajo	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Máquina 1	18	24	20	15	26	29	14
Máquina 2	35	6	35	34	31	7	37

Tabla N°4.2: Tiempos de preparación dependientes de la secuencia.

Máquina 1

S_{ij}^1	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	6	17	5	10	10	13	13
1	--	11	9	4	6	16	7
2	9	--	8	16	12	17	10
3	3	4	--	3	13	12	3
4	5	11	5	--	7	13	14
5	4	4	10	13	--	13	14
6	6	14	17	14	13	--	5
7	11	4	3	6	10	5	--

Máquina 2

S_{ij}^2	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	9	8	8	6	6	16	16
1	--	8	7	17	9	7	13
2	12	--	12	11	8	15	12
3	15	10	--	12	16	6	11
4	8	16	6	--	4	14	14
5	5	12	5	17	--	12	14
6	9	17	7	6	11	--	3
7	3	11	3	9	11	9	--

Con los datos presentados anteriormente, se pueden obtener los tiempos de procesamiento ajustado que se muestran en la Tabla N°4.3.

Tabla N°4.3: Tiempos de procesamiento ajustado.

Máquina 1

AP^1	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	24	41	25	25	36	42	27
1	--	35	29	19	32	45	21
2	27	--	28	31	38	46	24
3	21	28	--	18	39	41	17
4	23	35	25	--	33	42	28
5	22	28	30	28	--	42	28
6	24	38	37	29	39	--	19
7	29	28	23	21	36	34	--

Máquina 2

AP^2	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	44	14	43	40	37	23	53
1	--	14	42	51	40	14	50
2	47	--	47	45	39	22	49
3	50	16	--	46	47	13	48
4	43	22	41	--	35	21	51
5	40	18	40	51	--	19	51
6	44	23	42	40	42	--	40
7	38	17	38	43	42	16	--

La forma como se resuelve un problema usando LACH, se muestra en los siguientes pasos:

Paso 0. VM={1, 2}, M= VM, y J={1, 2, 3, 4 ,5 ,6, 7}, L= 0, s= 2, WL^1= 0 y WL^2= 0.

Paso 1. Para la máquina 1: MC^1={21, 28, 23, 18, 32, 34, 17}, y MF^1={19, 24, 17, 23, 22, 19, 21}. Para la máquina 2: MC^2={38, 14, 38, 40, 35, 13, 40}, y MF^2={14, 14, 13, 21, 18, 23, 16}.

Paso 2. Para k*= 2, i*= 6 y j*= 2, $\alpha^2_{6,2} = ap^2_{62} + mc^2_6 + mf^2_2 = 23+13+14 = 50$ es el mínimo. Entonces L= 2, WL^2= 23, X^2= 6, Y^2= 2, J={1, 3, 4, 5, 7} y M={1}. Asignar la secuencia (6, 2) a la máquina 2. Eliminar fila 6, columna 2 y elemento (2, 6) de [AP^2]. Eliminar filas y columnas 6 y 2 de [AP^1].

Paso 3. Como $M \neq \phi$, ir al Paso 1.

Paso 1. Como M={1} y J={1, 3, 4, 5, 7}, MC^1={21, --, 23, 18, 32, --, 17} y MF^1={19, --, 17, 23, 22, --, 21}.

Paso 2. Para k*=1, i*=7 y j*=3, $\alpha^1_{7,3} = ap^1_{73} + mc^1_7 + mf^1_3 = 23+17+17 = 57$ es el mínimo. Entonces L= 2+2=4, WL^1= 23, X^1= 7, Y^1= 3, J={1, 4, 5} y M=ϕ. Asignar la secuencia (7, 3) a la máquina 1. Eliminar fila 7, columna 3 y elemento (3, 7) de [AP^1]. Eliminar filas y columnas 7 y 3 de [AP^2]. Hasta ahora, las matrices [AP^k] estarían de la siguiente forma:

Tabla N°4.4: Tiempos de Procesamiento Ajustado después de asignaciones iniciales.

Máquina 1

AP^1	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	24	--	--	25	36	--	27
1	--	--	--	19	32	--	21
2	--	--	--	--	--	--	--
3	21	--	--	18	39	--	--
4	23	--	--	--	33	--	28
5	22	--	--	28	--	--	28
6	--	--	--	--	--	--	--
7	--	--	--	--	--	--	--

Máquina 2

AP^2	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	44	--	--	40	37	23	--
1	--	--	--	51	40	14	--
2	47	--	--	45	39	--	--
3	--	--	--	--	--	--	--
4	43	--	--	--	35	21	--
5	40	--	--	51	--	19	--
6	--	--	--	--	--	--	--
7	--	--	--	--	--	--	--

Paso 3. Como $M=\phi$, ir al Paso 4.

Paso 4. $T=J=\{1, 4, 5\}$, $M=\{1, 2\}$ y $b= 2$.

Paso 5. Se encuentra que $\lambda^1 = WL^1 + ap^1_{0X^1} = WL^1 + ap^1_{07} = 23+27 = 50$ es el máximo.

Entonces, $k^*= 1$.

Dado que $k^*= 1$, $X^1= 7$, y $Y^1= 3$. Para $W^1= 4$ y $Z^1= 1$, se cumple que:

$P^1_1 = ap^1_{0X^1} + ap^1_{Y^1W^1} = ap^1_{07} + ap^1_{34} = 27+18 = 45$ y

$P^1_2 = ap^1_{0Z^1} + ap^1_{Z^1X^1} = ap^1_{01} + ap^1_{17} = 24+21 = 45$ son mínimos respectivamente.

Let $M=M-\{1\}=\{2\}$.

Como $P_1{}^1=P_2{}^1$, se elige arbitrariamente $P_2{}^1$, entonces: $T=T-\{1\}=\{4, 5\}$.

Paso 6. $b= 2-1=1$. Como $b= 1$, ir al Paso 7.

Paso 7. $L= 4+1=5$.

Dado que la máquina remanente es $h= 2$, $X^2= 6$ y $Y^2= 2$. Para $W^2=5$ y $Z^2=5$, se cumple que:

$P^2_1 = ap^2_{06} + ap^2_{25} = 23+39 = 62$ y

$P^2_2 = ap^2_{05} + ap^2_{56} = 37+19 = 56$ son mínimos respectivamente.

Como $P_2{}^2 < P_1{}^2$, asignar el trabajo 5 antes del trabajo 6 en la máquina 2.

$WL^2 = WL^2 + ap^2_{56} = 23+19 = 42$. Eliminar fila 5, columna 6 y elemento (2,5) de $[AP^2]$, y fila y columna 5 de $[AP^1]$. $X^2= 5$ y $J=\{1,4\}$.

Paso 8. Como se cumple que $L \le n - m$ ($5 \le 5$), ir al Paso 4.

Paso 4. $T=J=\{1, 4\}$, $M=VM=\{1, 2\}$ y $b= s= 2$.

Paso 5. Se encuentra que $\lambda^2 = WL^2 + ap^2_{05} = 42+37 = 79$ es el máximo. Entonces, $k^*=2$.

Dado que $k^*= 2$, $X^2= 5$, y $Y^2= 2$. Para $W^2= 4$ y $Z^2= 4$, se cumple que:

$P_1^2 = ap_{05}^2 + ap_{24}^2 = 37 + 45 = 82$ y

$P_2^2 = ap_{04}^2 + ap_{45}^2 = 40 + 35 = 75$ son mínimos respectivamente.

Hacer $M=M-\{2\}=\{1\}$.

Como $P_2^2 < P_1^2$, entonces $T=T-\{4\}=\{5\}$.

Paso 6. $b= 2-1=1$. Como $b= 1$, ir al Paso 7.

Paso 7. $L= 5+1=6$.

Dado que la máquina remanente es $h= 1$, $X^1= 7$, y $Y^1= 3$. Para $W^1= 1$ y $Z^1= 1$. se cumple que:

$P_1^1 = ap_{07}^1 + ap_{31}^1 = 27 + 21 = 48$ y

$P_2^1 = ap_{01}^1 + ap_{17}^1 = 24 + 21 = 45$ son mínimos respectivamente.

Como $P_2^1 < P_1^1$, asignar el trabajo 1 antes del trabajo 7 en la máquina 1.

$WL^1 = WL^1 + ap_{17}^1 = 23 + 21 = 44$. Eliminar fila 1, columna 7 y elemento (3,1) de $[AP^1]$, y fila y columna 1 de $[AP^2]$. $X^1= 1$ y $J= J-\{1\}=\{4\}$.

Paso 8. Como $L > (n - m)=5$ y $L < n$ ($6 > 5$ y $6 < 7$), ir al Paso 9.

Paso 9. Como $VM=\{1, 2\}$ clacular β^1 y β^2.

$\beta^{k*} = \max_{k \in VM}\{ap_{0X^k}^k + WL^k\} = \max_{k \in VM}\{24 + 44;\ 37 + 42\} = \max_{k \in VM}\{68; 79\} = 79$, entonces $k^*=2$.

Hacer $VM=VM-\{2\}=\{1\}$ y $s=2-1=1$. Ir al Paso 4.

Paso 4. $T=J=\{4\}$, $M=VM=\{1\}$ y $b= s= 1$. Como $b= 1$, ir al Paso 7.

Paso 7. $L= 6+1=7$.

Dado que la máquina remanente es h= 1, X^l= 1, y Y^l= 3. Para W^l= 4 y Z^l= 4, se cumple que:

$P_1^1 = ap_{01}^1 + ap_{34}^1 = 24 + 18 = 42$ y

$P_2^1 = ap_{04}^1 + ap_{41}^1 = 25 + 23 = 48$ son mínimos respectivamente.

Como $P_1^1 < P_2^1$, asignar el trabajo 4 después del trabajo 3 en la máquina 1. $WL^1 = WL^1 + ap_{34}^1 = 44 + 18 = 62$. Eliminar fila 3, columna 4 y elemento (4,1) de $[AP^1]$, y fila y columna 4 de $[AP^2]$. Y^l= 4 y J= J–{4}={ϕ}.

Paso 8. Como $L = n$ (7 = 7), ir al Paso 10.

Paso 10. $WL^1 = WL^1 + ap_{01}^1 = 62 + 24 = 86$; y $WL^2 = WL^2 + ap_{05}^2 = 42 + 37 = 79$.

Por lo tanto $C_{\max} = \underset{k=1,\ldots,m}{max}\{WL^k\} = max\{86;79\} = 86$, y los trabajos son asignados a cada máquina según las siguientes secuencias, para la máquina 1: 0-1-7-3-4 y para la máquina 2: 0-5-6-2.

4.2 Comparación de LACH v/s SAP-SL.

La heurística constructiva LACH fue comparada con una heurística constructiva glotona (greedy) llamada SAP-SL (Moraga 2004, Rabadi *et al.* 2006) utilizando la librería de problemas propuesta por Rabadi (2005). Los problemas fueron generados para combinaciones de 20, 40, 60, 80, 100, 120 trabajos, y m = 2, 4, 6, 8, 10, 12 máquinas. Considerando tres escenarios:

1. Tiempos de procesamiento y preparación balanceados (tiempos de procesamiento ~*Uniforme*[50,100] y tiempos de preparación ~*Uniforme* [50,100])

2. Tiempos de procesamiento dominante (tiempos de procesamiento ~*Uniforme*[125,175] y tiempos de preparación ~*Uniforme*[50,100])
3. Tiempos de preparación dominante (tiempos de procesamiento ~*Uniforme*[50,100] y tiempos de preparación ~*Uniforme*[125,175])

La librería considera 15 problemas para cada combinación y para cada escenario. En total 1620 problemas.

Los resultados fueron presentados por Muñoz *et al.* (2005). En la Tabla Nº4.5 se muestra un resumen de los resultados obtenidos agrupados por tipo de escenario de tiempos de procesamiento y prepearación (tiempo de proceso y preparación balanceados, tiempos de proceso dominante y tiempo de preparación dominante). El análisis excluye los problemas para combinaciones de m = 8, 10, 12; y n=20 debido a que no cumplen la condición $n \geq 3m$ exigida por LACH (se excluyeron un total de 135 problemas).

Las desviaciones fueron calculadas según la expresión (4.5), de tal manera que desviaciones negativas indican que SAP-SL reporta mejores resultados, y en caso contrario LACH es quien reporta mejores resultados:

$$\delta = \left(\frac{Existente - Propuesto}{Existente} \right) \times 100\% \qquad (4.5)$$

Donde:

Existente: resultado generado por la aplicación de SAP-SL.

Propuesto: resultado generado por la aplicación de LACH.

Tabla Nº4.5: Comparación de LACH v/s SAP-SL.

	Balanceados	***Proceso dominante***	***Preparación dominante***	***Total***
Desviación Media (δ)	0,58%	0,60%	0,52%	0,57%
Desviación Mínima (min δ)	-9,90%	-5,43%	-4,82%	-9,90%
Desviación Máxima (max δ)	9,48%	8,30%	7,23%	9,48%
Mejores LACH	291	315	313	919
Proporción	*58,79%*	*63,64%*	*63,23%*	*61,89%*
Mejores SAP-SL	197	174	170	541
Proporción	*39,80%*	*35,15%*	*34,34%*	*36,43%*
Iguales	7	6	12	25
Proporción iguales	*1,41%*	*1,21%*	*2,42%*	*1,68%*

En el 61.89% de los problemas, LACH encontró mejores resultados que SAP-SL. Para los tres escenarios: tiempos de procesamiento y preparación balanceados, tiempos de procesamiento dominante, y tiempos de preparación dominantes, las soluciones encontradas por LACH son mejores que las encontradas por SAP-SL en 0.58%, 0.60%, 0.52% respectivamente en promedio. En general, LACH entrega soluciones que son 0.57% mejores que SAP-SL en promedio. Se puede observar que las desviaciones son pequeñas, eso se debe exclusivamente a la forma en que se calcularon (propuesta por Rabadi *et al.* (2006)).

En la Figura Nº4.7 se muestra un histograma de las desviaciones obtenidas para todos los problemas del estudio, donde se puede observar que la mayor cantidad de observaciones tiene desviaciones ubicadas en la parte positiva del eje horizontal, lo que indica que LACH reporta mejores resultados para la mayoría de los problemas. También se observa que las desviaciones son bajas (cercanas a cero) en los problemas donde LACH no logra encontrar mejores soluciones que SAP-SL.

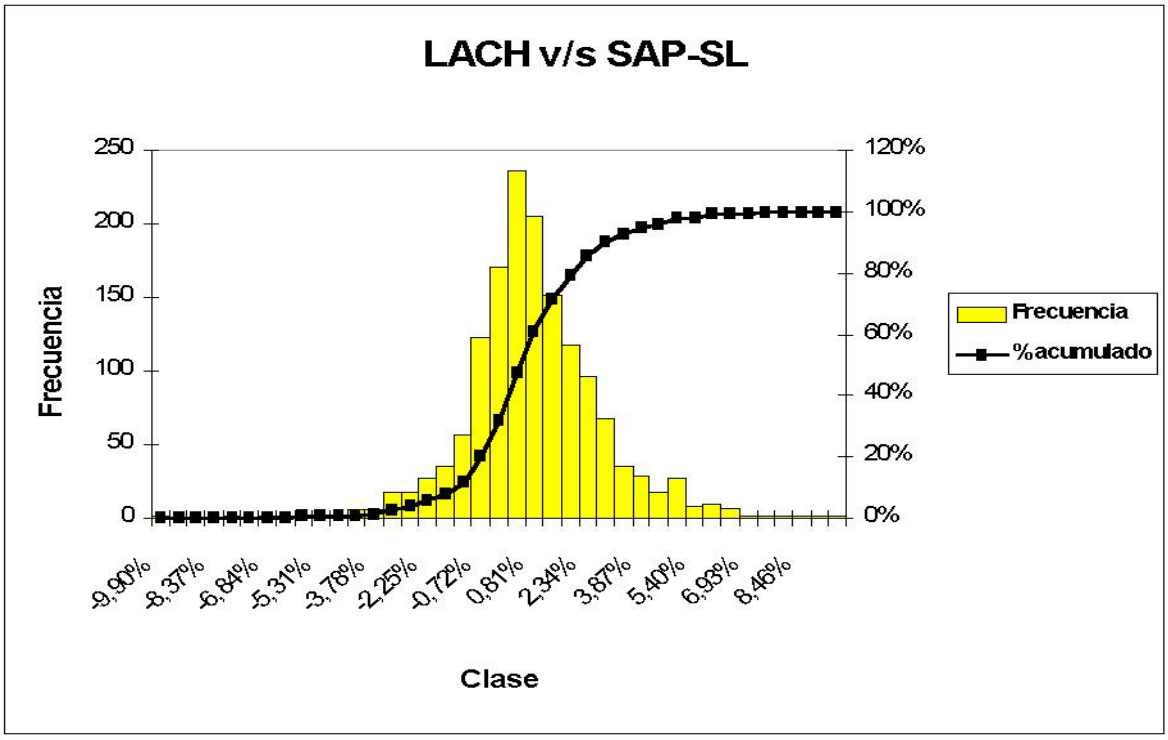


Figura Nº4.7: Histograma de desviaciones LACH v/s SAP-SL.

En las Figuras Nº4.8, Nº4.9 y Nº4.10, se muestran gráficos con las desviaciones promedio obtenidas en todos los problemas, agrupadas por cantidad de máquinas y trabajos, para los escenarios de tiempos de proceso y setup balanceados, tiempos de proceso dominante y tiempos de setup dominante respectivamente. En ellas se puede observar al igual que en la Figura N°4.7, que la mayoría de las desviaciones son positivas , por lo tanto se puede apreciar que LACH deja fuera de desempeño a SAP-SL.

Con los resultados obtenidos a partir de esta comparación, se puede observar que los resultados generados por LACH son en promedio mejor que los que entrega SAP-SL.

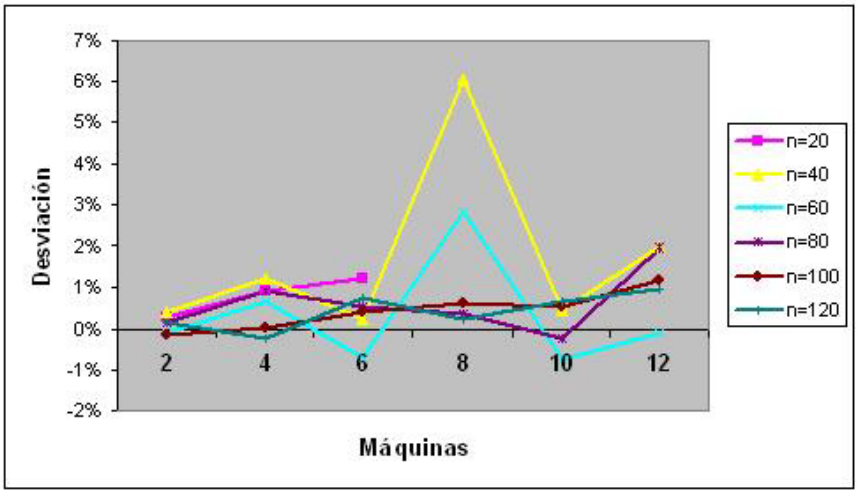


Figura Nº4.8: Desviaciones LACH v/s SAP-SL, tiempos balanceados.

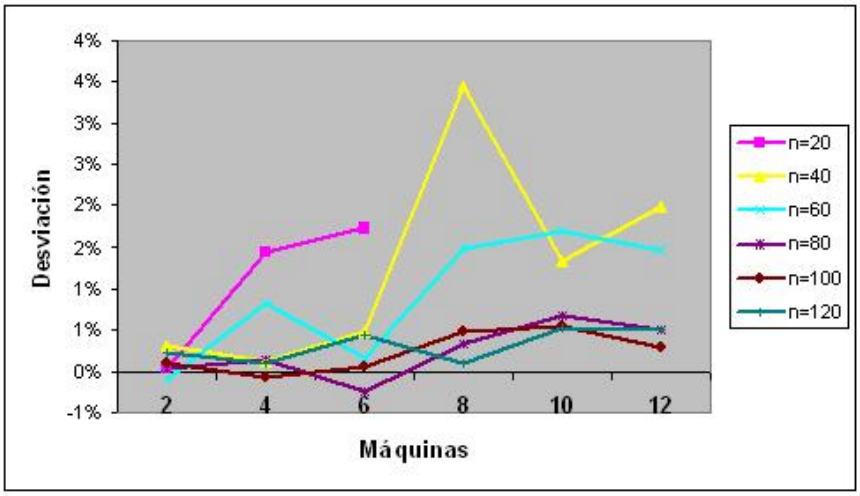


Figura Nº4.9: Desviaciones LACH v/s SAP-SL, tiempos de proceso dominante.

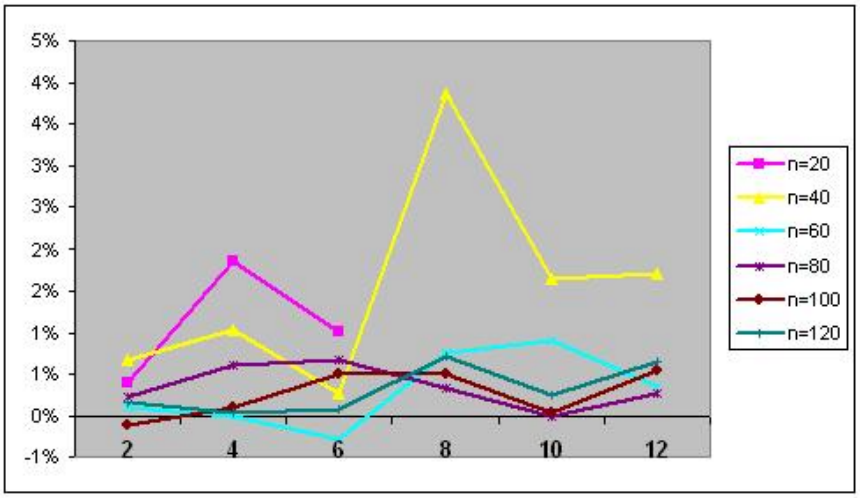


Figura Nº4.10: Desviaciones LACH v/s SAP-SL, tiempos de setup dominante.

4.3 Adición de Aleatoriedad, modificación de LACH utilizando %p y %r.

La adición de aleatoriedad basada en los parámetros de Meta-RaPS, %*prioridad* y %*restricción*, se realiza sobre la heurística constructiva LACH. Esta adición de aleatoriedad se realiza en dos partes de la heurística constructiva, la primera se realizó en la parte correspondiente a la primera asignación de cada máquina, y la segunda en el proceso de reserva y asignación de trabajos. Los beneficios de incluir aleatoriedad controlada en LACH se presentan en la siguiente sección.

A continuación se muestra la modificación de LACH (Figura N°3.1) la cual se realiza sobre los Pasos 2, 5 y 7 del Algoritmo de LACH.

Paso 2. Generar un número aleatorio *rnd*

Si $rnd \leq \%p/100$

Hacer: $\alpha_{i^*j^*}^{k^*} = \min\limits_{\substack{k\in M \\ i\in J; j\in J \\ i\neq j}} \left\{ ap_{ij}^k + mc_i^k + mf_j^k \right\}$

En otro caso, calcular:

$$\alpha = \min_{\substack{k\in VM \\ i\in J \\ j\in J \\ i\neq j}} \left\{ ap_{ij}^k + mc_i^k + mf_j^k \right\} \text{ y } \beta = \max_{\substack{k\in VM \\ i\in J \\ j\in J \\ i\neq j}} \left\{ ap_{ij}^k + mc_i^k + mf_j^k \right\}$$

Crear una lista restringida de candidatos, que cumplan:

$\left\{ ap_{ij}^k + mc_i^k + mf_j^k \right\} \leq \alpha + (\beta - \alpha)\%r/100$.

Elegir aleatoriamente un elemento de la lista restringida de candidatos y tomar i^*, j^* y k^* del elemento seleccionado.

$L = L+2$, $WL^{k^*} = ap^{k^*}{}_{i^*j^*}$, $X^{k^*} = i^*$, $Y^{k^*} = j^*$, $J = J-\{i^*\}-\{j^*\}$, $M = M-\{k^*\}$.

Asignar la secuencia (i^*, j^*) a la máquina k^*. Eliminar fila i^*, columna j^* y elemento (j^*, i^*) de $[AP^{k^*}]$. Eliminar filas y columnas i^* y j^* de $[AP^k]$ para todo $k \neq k^*$.

Paso 5. Encontrar $k^* \in M$ tal que: $\lambda^{k^*} = \max_{k \in M}(WL^k + ap^k_{0X^k})$

Generar un número aleatorio *rnd*.

Si $rnd \leq \%p/100$

Para k^* encontrar los trabajos W^{k^*} y Z^{k^*} tal que $P_1{}^{k^*}$ y $P_2{}^{k^*}$ (costos de oportunidad) sean mínimos:

$$P_1^{k^*} = ap^{k^*}_{0X^{k^*}} + ap^{k^*}_{Y^{k^*}W^{k^*}} \qquad W^{k^*} \in T$$
$$P_2^{k^*} = ap^{k^*}_{0Z^{k^*}} + ap^{k^*}_{Z^{k^*}X^{k^*}} \qquad Z^{k^*} \in T$$

En otro caso, para k^* calcular:

$$\alpha_1 = \min_{j \in T}\left\{ap^{k^*}_{Y^{k^*}j}\right\} \qquad \beta_1 = \max_{j \in T}\left\{ap^{k^*}_{Y^{k^*}j}\right\}$$
$$\alpha_2 = \min_{\substack{i \in T \\ i \neq X^{k^*}}}\left\{ap^{k^*}_{iX^{k^*}}\right\} \quad \text{y} \quad \beta_2 = \max_{\substack{i \in T \\ i \neq X^{k^*}}}\left\{ap^{k^*}_{iX^{k^*}}\right\}$$

Crear dos listas restringidas de tiempos de procesamiento ajustados, que cumplan:

$$ap^{k^*}_{Y^{k^*}W^{k^*}} \leq \alpha_1 + (\beta_1 - \alpha_1)\%r/100 \qquad W^{k^*} \in T$$
$$ap^{k^*}_{Z^{k^*}X^{k^*}} \leq \alpha_2 + (\beta_2 - \alpha_2)\%r/100 \qquad Z^{k^*} \in T$$

Seleccionar aleatoriamente un elemento de cada lista restringida de candidatos y hacer W^{k*}= *candidato seleccionado de la lista 1*; y Z^{k*}= *candidato seleccionado de la lista 2*.

Calcular:

$$P_1^{k*} = ap_{0X^{k*}}^{k*} + ap_{Y^{k*}W^{k*}}^{k*}$$
$$P_2^{k*} = ap_{0Z^{k*}}^{k*} + ap_{Z^{k*}X^{k*}}^{k*}$$

Hacer $M = M-\{k*\}$

Si $P_1^{k*} < P_2^{k*}$, hacer $T= T-\{W^{k*}\}$ (empates se resuelven en forma aleatoria).

En otro caso, hacer $T= T-\{Z^{k*}\}$

Paso 7. Hacer $L= L+1$.

Generar un número aleatorio *rnd*.

Si $rnd \leq \%p/100$

Para la máquina remanente (h) en M, encontrar los trabajos W^h y Z^h tal que P_1^h y P_2^h sean mínimos:

$$P_1^h = ap_{0X^h}^h + ap_{Y^hW^h}^h \qquad W^h \in T$$
$$P_2^h = ap_{0Z^h}^h + ap_{Z^hX^h}^h \qquad Z^h \in T$$

En otro caso, calcular:

$$\alpha_1 = \min_{j \in T}\left\{ap_{Y^h j}^h\right\} \qquad \beta_1 = \max_{j \in T}\left\{ap_{Y^h j}^h\right\}$$

y

$$\alpha_2 = \min_{\substack{i \in T \\ i \neq X^h}}\left\{ap_{iX^h}^h\right\} \qquad \beta_2 = \max_{\substack{i \in T \\ i \neq X^h}}\left\{ap_{iX^h}^h\right\}$$

Crear dos listas restringidas de tiempos de procesamiento ajustados, que cumplan:

$$ap^h_{Y^hW^h} \le \alpha_1 + (\beta_1 - \alpha_1)\%r/100 \qquad W^h \in T$$
$$ap^h_{Z^hX^h} \le \alpha_2 + (\beta_2 - \alpha_2)\%r/100 \qquad Z^h \in T$$

Seleccionar aleatoriamente un elemento de cada lista restringida de candidatos y hacer W^{k*}= *candidato seleccionado de la lista 1*; y Z^{k*}= *candidato seleccionado de la lista 2*.

Calcular:

$$P_1^h = ap^h_{0X^h} + ap^h_{Y^hW^h} \qquad W^h \in T$$
$$P_2^h = ap^h_{0Z^h} + ap^h_{Z^hX^h} \qquad Z^h \in T$$

Si $P_1{}^h < P_2{}^h$ asignar el trabajo W^h después del trabajo Y^h en la máquina h.

Hacer $WL^h = WL^h + ap^h_{Y^hW^h}$. (empates se resuelven en forma aleatoria).

Eliminar fila Y^h, columna W^h y elemento (W^h, X^h) de [AP^h]. Adicionalmente, eliminar fila y columna W^h de [AP^h] para todo $k \neq h$. Hacer $Y^h = W^h$ y $J = J - \{W^h\}$.

En otro caso, asignar el trabajo Z^h antes de X^h en la máquina h. Hacer $WL^h = WL^h + ap^h_{Z^hX^h}$.

Eliminar fila Z^h, columna X^h y elemento (Y^h, Z^h) de [AP^h]. Adicionalmente, eliminar fila y columna Z^h de [AP^h] para todo $k \neq h$. Hacer $X^h = Z^h$ y $J = J - \{Z^h\}$.

4.3.1 Beneficio de incluir aleatoriedad controlada en LACH.

En esta sección se presentan los beneficios de introducir aleatoriedad controlada en la heurística constructiva LACH. Esta inclusión de aleatoriedad se logra mediante los parámetros *%p* y *%r* de Meta-RaPS, fijados mediante la metodología de fijación (sección 6.1). Adicionalmente, la cantidad de iteraciones se fijó en 5000 para todos los problemas propuestos por Rabadi (2005).

Para calcular el beneficio de la aleatoriedad, se utilizó la siguiente expresión:

$$\delta = \left(\frac{A - B}{A} \right) \times 100\% \tag{4.6}$$

Donde:

A: resultado generado por la aplicación de LACH.

B: resultado generado por la aplicación de LACH con la inclusión de aleatoriedad controlada mediante los parámetros *%prioridad* y *%restricción* (LACH(*%p,%r*)).

En la Figura Nº4.11 se muestra un histograma de las desviaciones (δ) entre LACH y LACH aleatorizado mediante *%p* y *%r*.

En promedio, las soluciones mejoran en un 2.9 %, encontrándose soluciones construidas que mejoran desde 0 % hasta 16.48 %.

En teoría, la versión aleatorizada de LACH ofrece mayor variabilidad que la versión aleatorizada de SAP-SL para construir soluciones. Esto se debe a que LACH, al momento de agregar una actividad a la solución, desarrolla los procesos de reserva y asignación, permitiendo agregar mayor aleatoriedad a las soluciones construidas que SAP-SL, ya que SAP-SL utiliza aleatoriedad solo para elegir la próxima actividad.

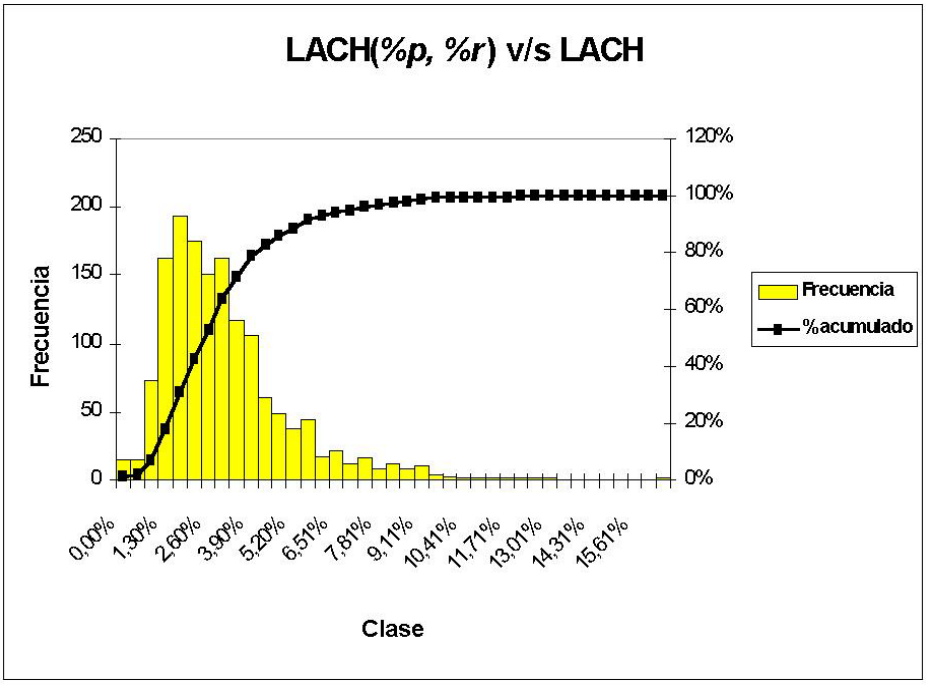


Figura Nº4.11: Beneficio de LACH incluyendo aleatoriedad controlada.

Con los resultados obtenidos a partir de esta comparación, se puede observar que los resultados generados por la aleatorización de LACH ofrece mejores resultados que la versión de LACH sin aleatoriedad.

En las Tablas Nº4.6, Nº4.7 y Nº4.8 se presenta un resumen de las desviaciones (δ) promedio agrupadas por cantidad de máquinas y trabajos, obtenidas para los escenarios de tiempos de procesamiento y preparación balanceados, tiempos de procesamiento dominante y tiempos de setup dominante respectivamente. En las Figuras Nº4.12, Nº4.13 y Nº4.14, se muestran los gráficos de desviaciones para los escenarios. Para calcular el promedio de las desviaciones, las desviaciones de los problemas fueron agrupadas por escenario (tiempos

de proceso y preparación balanceados, tiempos de proceso dominante y tiempos de preparación dominante), y luego se agruparon por cantidad de trabajos y máquinas.

Tabla Nº4.6: Beneficio LACH(*%p,%r)* problemas balanceados.

Máquinas	***2***	***4***	***6***	***8***	***10***	***12***
n=20	*4,702%*	*6,006%*	*7,921%*			
n=40	*3,094%*	*3,442%*	*5,946%*	*6,051%*	*7,785%*	*6,507%*
n=60	*2,110%*	*3,223%*	*4,881%*	*3,803%*	*5,717%*	*7,236%*
n=80	*1,571%*	*2,463%*	*3,760%*	*3,811%*	*4,297%*	*3,333%*
n=100	*1,469%*	*2,366%*	*2,571%*	*2,856%*	*3,596%*	*3,533%*
n=120	*1,083%*	*2,183%*	*2,330%*	*2,615%*	*3,369%*	*3,323%*
Promedio	2,338%	3,281%	4,568%	3,827%	4,953%	4,786%

Tabla Nº4.7: Beneficio LACH(*%p,%r)* problemas proceso dominante.

Máquinas	***2***	***4***	***6***	***8***	***10***	***12***
n=20	*2,845%*	*3,800%*	*4,411%*			
n=40	*1,959%*	*2,835%*	*3,421%*	*3,451%*	*4,717%*	*3,823%*
n=60	*1,363%*	*1,935%*	*2,890%*	*2,305%*	*2,931%*	*3,407%*
n=80	*1,066%*	*1,704%*	*2,361%*	*2,569%*	*2,587%*	*2,662%*
n=100	*0,961%*	*1,444%*	*1,623%*	*1,734%*	*2,201%*	*2,568%*
n=120	*0,684%*	*1,254%*	*1,525%*	*1,714%*	*2,018%*	*2,191%*
Promedio	1,480%	2,162%	2,705%	2,355%	2,891%	2,930%

Tabla Nº4.8: Beneficio LACH(*%p,%r)* problemas setup dominante.

Máquinas	***2***	***4***	***6***	***8***	***10***	***12***
n=20	*2,635%*	*3,818%*	*5,197%*			
n=40	*1,604%*	*2,532%*	*3,413%*	*3,861%*	*5,091%*	*4,421%*
n=60	*1,521%*	*2,237%*	*2,747%*	*2,565%*	*3,378%*	*4,167%*
n=80	*1,100%*	*1,563%*	*1,945%*	*2,782%*	*2,692%*	*2,468%*
n=100	*1,056%*	*1,541%*	*1,504%*	*1,662%*	*2,371%*	*2,193%*
n=120	*0,752%*	*1,517%*	*1,733%*	*1,618%*	*1,775%*	*2,317%*
Promedio	1,445%	2,202%	2,757%	2,497%	3,061%	3,113%

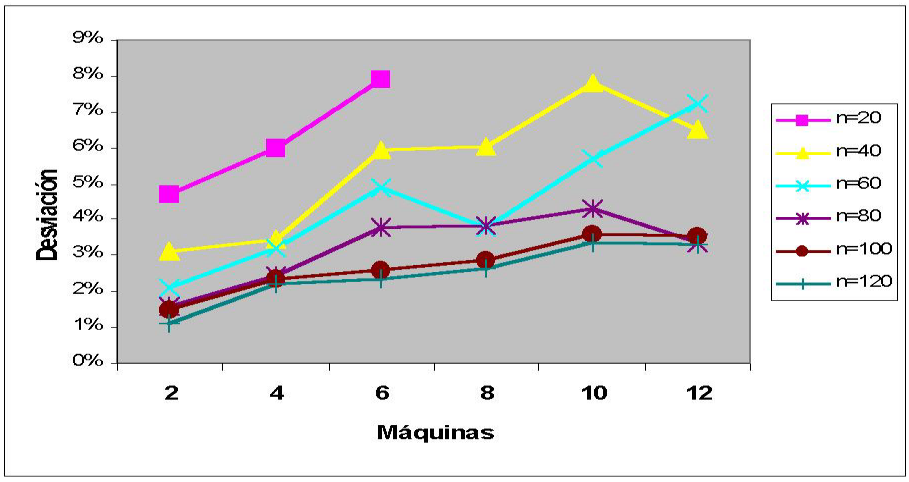


Figura Nº4.12: Beneficio LACH(*%p,%r)* problemas balanceados.

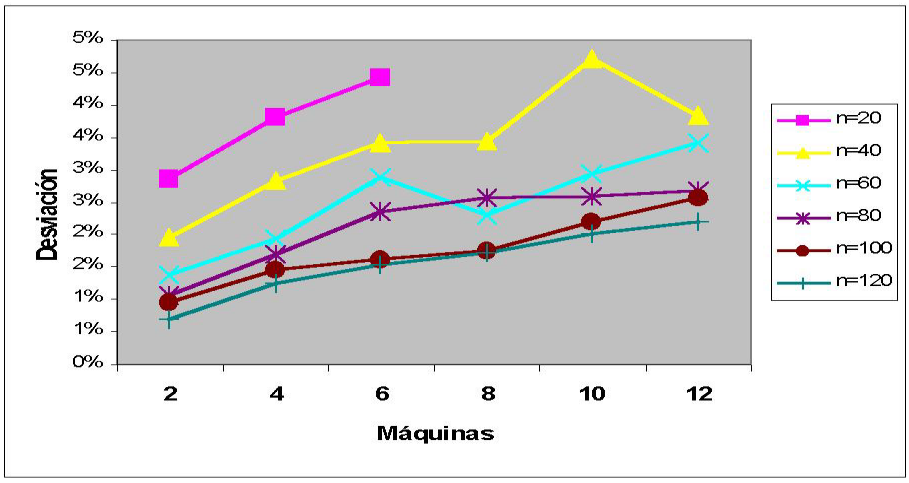


Figura Nº4.13: Beneficio LACH(*%p,%r)* problemas proceso dominante.

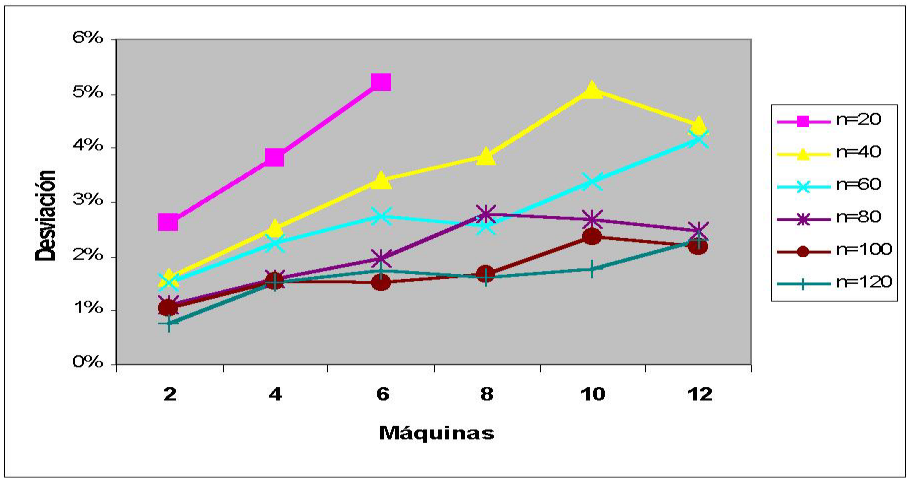


Figura Nº4.14: Beneficio LACH(*%p,%r)* problemas setup dominante.

CAPITULO 5: META-RAPS, FASE MEJORAMIENTO.

Meta-RaPS utiliza una fase de mejoramiento, para la cual es necesario diseñar una heurística de mejoramiento local. Para mejorar soluciones al problema $R_m|S_{ijk}|C_{max}$, y lograr un mejor desempeño en tiempo es necesario estudiar la estructura del problema y las propiedades de las buenas soluciones. El algoritmo de mejoramiento descrito en este capítulo, intenta explotar algunas características de las buenas soluciones para lograr un mejor desempeño.

5.1 Heurística de mejoramiento para $R_m|S_{ijk}|C_{max}$

Una vez que un programa de producción Ω es construido, Meta-RaPS puede proceder a mejorarlo con técnicas de búsqueda en la vecindad (mejoramiento local). El parámetro *%i* es usado para decidir si el programa Ω es aceptado o rechazado para pasar por una etapa de mejoramiento. Mientras Meta-RaPS es ejecutado, conserva el mejor y peor makespan de los programas construidos. De tal forma que una determinada función Ψ(*%i*), determinará la calidad exigida para que una solución pase por la rutina de mejoramiento. La forma de calcular Ψ(*%i*) es la siguiente:

$\Psi(\%i) = Mejor + (Peor - Mejor)\%i/100$, donde *Mejor* y *Peor* se refieren al menor y mayor C_{max} obtenidos en las construcciones en lo que va corrido de las iteraciones.

De tal forma que si la solución construida Ω cumple con la condición:

$C_{max}(\Omega) \leq \Psi(\%i)$, la solución construida Ω es aceptada para pasar a la fase de mejoramiento, de lo contrario es rechazada.

La heurística de mejoramiento diseñada para la fase de mejoramiento de Meta-RaPS se basa en la heurística de mejoramiento propuesta por Rabadi *et al*. (2006). Siendo ésta una heurística de mejoramiento de descenso simple, que a partir de la solución construida en la fase de construcción, conduce al óptimo local más cercano mediante una serie de cambios de mejora (movidas).

Para problemas combinatorios se pueden diseñar diferentes búsqueda locales. El éxito de éstas depende de diferentes factores, tales como la estructura de entorno, la estrategia empleada para examinar éste, la eficiencia en la evaluación de las soluciones y la solución inicial. La fase de construcción juega un papel fundamental respecto a proporcionar un punto inicial de calidad en la búsqueda. Entornos relativamente sencillos basados en intercambios o inserciones son los más utilizados. Las dos estrategias típicas de selección de la solución en el entorno (movimiento) son la *mejor-mejorada* (best-improving) y la *primera-mejora* (first-improving). En la primera, se examina todo el entorno para determinar la mejor solución a la cual moverse (cambiar la solución actual); en la segunda, en el momento en que se encuentra una solución que mejore a la actual, se analiza la exploración y se realiza el movimiento sin explorar el resto del entorno (Marti & Moreno, 2003).

Para esta heurística de mejoramiento se considera la estrategia de *mejor-mejorada*, y se consideran tres técnicas de búsqueda local: inserción de trabajos intra-máquina, intercambio de trabajos inter-máquinas, e inserción de trabajos inter-máquina, que son realizados sobre una base cíclica, similar a lo propuesto por Rabadi *et al*. (2006).

Mientras las alteraciones inter-máquinas intercambian o insertan trabajos entre diferentes máquinas, la alteración intra-máquina inserta trabajos sobre la misma máquina (Pinedo, 2002). Después de que la rutina sea aplicada durante diez ciclos, se mantiene el mejor programa de producción encontrado. La cantidad de ciclos fue propuesta por Rabadi *et al.* (2006), considerando el mejor compromiso, o balance, entre calidad de la solución y costo computacional de la etapa de mejoramiento.

En la Figura N°5.1 se muestra un ejemplo de la movida de inserción de trabajos intra-máquina. En (a) se muestra el programa de producción inicial en una máquina en particular; luego una movida de inserción es probada, donde el trabajo 1 es ubicado entre los trabajos 4 y 5 (b). Luego al programa obtenido se le realiza otra movida de inserción, que consiste en ubicar el trabajo 3 entre los trabajos 7 y 8, dando lugar al programa (c).

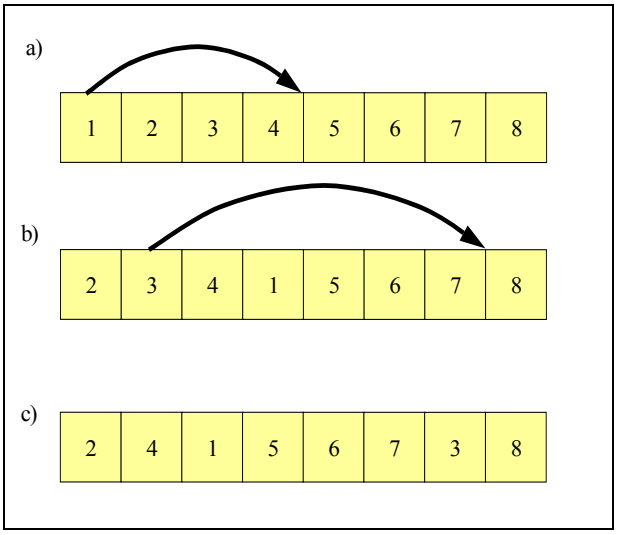


Figura N°5.1: Inserción intra-máquina.

En las Figuras N°5.2 y N°5.3 se muestra un ejemplo de la movida de inserción e intercambio de trabajos inter-máquina respectivamente.

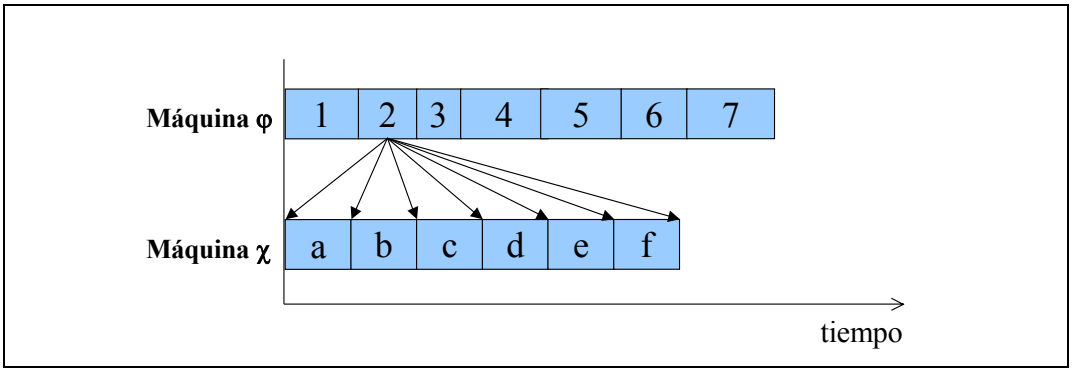


Figura N°5.2: Inserción entre máquinas.

En la Figura N°5.2, se muestra que por medio de movidas de inserción se pueden obtener varias soluciones. El programa de la máquina φ que se obtiene mediante esta movida es: [1-3-4-5-6-7], y los programas que se pueden generar para la máquina χ son: [**2**-a-b-c-d-e-f], [a-**2**-b-c-d-e-f], [a-b-**2**-c-d-e-f], [a-b-c-**2**-d-e-f], [a-b-c-d-**2**-e-f], [a-b-c-d-e-**2**-f] o [a-b-c-d-e-f-**2**]

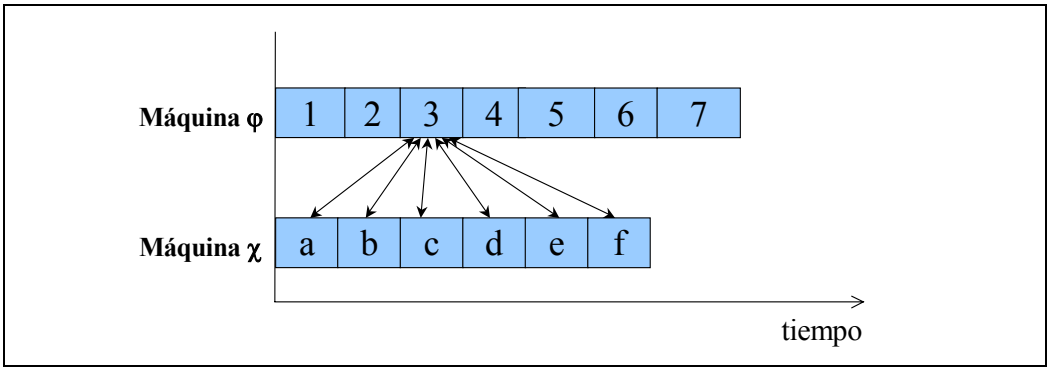


Figura N°5.3: Intercambio de trabajos entre máquinas.

En la Figura N°5.3, se muestra que por medio de movidas de intercambio se pueden obtener varias soluciones. En la Tabla N°5.1 se muestran los programas que se pueden generar.

Tabla N°5.1: Programas generados por movida.

Máquina φ	***Máquina χ***
1-2-**a**-4-5-6-7	**3**-b-c-d-e-f
1-2-**b**-4-5-6-7	a-**3**-c-d-e-f
1-2-**c**-4-5-6-7	a-b-**3**-d-e-f
1-2-**d**-4-5-6-7	a-b-c-**3**-e-f
1-2-**e**-4-5-6-7	a-b-c-d-**3**-f
1-2-**f**-4-5-6-7	a-b-c-d-f-**3**

5.1.1 Algoritmo de la Heurística de Mejoramiento.

Definiciones:

- INS(*X*): procedimiento que analiza y realiza movidas de inserción de trabajos en la máquina *X*, disminuyendo la carga de trabajo en la máquina *X*.
- INSERCIÓN(*X*, *Y*): procedimiento que analiza y realiza movidas de inserción de un trabajo perteneciente a la máquina *X*, ubicándolo en la máquina *Y*, disminuyendo el C_{max}.
- INT_MAQ(*X*, *Y*): procedimiento que analiza y realiza movidas de intercambio de 2 trabajos entre las máquinas *X* e *Y*, disminuyendo el C_{max}.
- *m*: cantidad de máquinas.
- X_S: es la secuencia de trabajos programados en la máquina *S*. *S*= 1,..., *M*.
- *VM*: es el conjunto que contienen todas las máquinas.
- $WL(X_S)$: carga de trabajo asociada a la máquina *S*.

- *Ciclos*: cantidad de máxima de movidas que se realizan intra-máquina e inter-máquina.

El algoritmo de la rutina de mejoramiento es el siguiente:

Paso 0.

Hacer S= 1 y *cont=1*

Paso 1.

cont=1

Mientras (*cont* ≤ *Ciclos*), hacer: INS(X_S) ; y *cont=cont+1*

Si $S < m$, hacer $S = S+1$, ir a Paso 1.

De lo contrario, hacer $S = 0$, *cont=1*, continuar en Paso 2.

Paso 2.

Si (*cont* ≤ *Ciclos*), hacer: $A_{X_S*} = \max_{S \in VM} \{WL(X_S)\}$, ir a Paso 3.

De lo contrario, reportar la solución mejorada.

Paso 3.

Hacer:

$S = S+1$

Si ($S \neq S^*$), hacer $\alpha_{X_S} = \min_{\forall i,j} \left\{ ap_{ij}^{X_S} \right\}$,

Si $(A_{X_S*} - WL(X_S)) \geq \alpha_{X_S}$, probar: INSERCIÓN($X_{S*}$, X_S),

Probar: INT_MAQ(X_{S*}, X_S),

Si $S < m$, ir a Paso 3.

De lo contrario, hacer *cont=cont+1*, $S = 0$ ir a Paso 2.

A continuación se describe cada parte del algoritmo detalladamente.

Paso 1

En este paso se realizan movidas de inserción de trabajos intra-máquina durante la cantidad prefijada de ciclos en cada una de las máquinas, para minimizar su carga de trabajo.

Paso 2

Aquí se busca la máquina que tiene mayor carga de trabajo (máquina *S**), y se evalúa el criterio de detención de la rutina.

Paso 3

En este paso se realizan las movidas de alteración inter-máquina. Para ello se calcula un valor α, que corresponde al menor tiempo de procesamiento ajustado en la máquina en evaluación *S*. Si el valor α indica que existe la posibilidad de que una movida de inserción entre la máquina *S** y la máquina *S*, reduzca el máximo tiempo de completación (makespan) determinado por la máquina *S** se evalúa esa opción. Luego se evalúa el análisis de intercambio de trabajos entre las máquinas en análisis.

El valor α actúa como una controlador de movidas para lograr un mejor aprovechamiento de tiempo y con ello un ahorro en tiempo de corrida.

Si la relación $(A_{X_{S^*}} - WL(X_S)) \geq \alpha_{X_S}$ no se cumple, indica que no existe ningún trabajo que al ser insertado en la máquina *S*, logre reducir el máximo tiempo de completación (makespan). Por lo tanto sólo las movidas de intercambio podrían reducir el máximo tiempo de completación (makespan).

Por otro lado, si la relación se cumple, indica que puede existir al menos un trabajo que al ser insertado en la máquina *S* logre reducir el máximo tiempo de completación (makespan).

5.2 Evaluación de Heurística de Mejoramiento.

En esta sección se evaluará el desempeño de la heurística de mejoramiento propuesta, para ello, se compararán los resultados obtenidos mediante ésta con los resultados obtenidos por la heurística de mejoramiento propuesta por Rabadi *et al.* (2006). Para construir las soluciones se utilizará la heurística constructiva LACH aleatorizada, los parámetros utilizados para *%p* y *%r* fueron fijados mediante la metodología de fijación mostrada posteriormente en la sección 6.1. Adicionalmente, para comparar en igualdad de condiciones ambas heurísticas, la cantidad de iteraciones se fijó en 5000 y el porcentaje de mejoramiento (*%i*) en 60% (valores utilizados por Rabadi *et al.,* 2006). Los problemas utilizados para esta evaluación fueron obtenidos aleatoriamente desde el grupo de problemas propuesto por Rabadi (2005), de tal forma que se consideran dos problemas para cada combinación de cantidad de máquinas (2, 4, 6, 8, 10 y 12), trabajos (20, 40, 60, 80, 100 y 120) y escenario (tiempos de procesamiento y setup balanceados, tiempos de procesamiento dominante y tiempos de procesamiento dominante). En este estudio, al igual que el realizado en el capítulo 4, se excluyen los problemas más pequeños de combinaciones de 20 trabajos, y 8, 10 y 12 máquinas, ya que el diseño de LACH no permite su resolución debido a la condición $n \geq 3m$, donde n y m indican la cantidad de trabajos y máquinas del problema respectivamente. Considerándose un total de 198 problemas.

Las desviaciones se calcularon mediante la siguiente expresión:

$$\delta = \left(\frac{A - B}{A} \right) \times 100\% \qquad (5.1)$$

Donde:

A: resultado generado por la aplicación de Meta-RaPS, utilizando LACH aleatorizado y el mejoramiento propuesto por Rabadi *et al.* (2006).

B: resultado generado por la aplicación de Meta-RaPS, utilizando LACH aleatorizado y el mejoramiento propuesto en este estudio.

Los resultados obtenidos se resumen en la Tabla N°5.2.

Tabla N°5.2: Comparaciones de Heurísticas de Mejoramiento.

	Balanceados	***Proc. Dominante***	***Setup dominante***	***Total***
Promedio (δ)	0,765%	0,104%	0,083%	0,317%
Mínimo (min δ)	-1,190%	-0,637%	-0,657%	-1,190%
Máximo (max δ)	13,318%	0,843%	0,732%	13,318%
Cantidad Mejores	51 (77,3%)	39 (59,1%)	43 (65,2%)	133 (67,2%)
Cantidad Peores	8 (12,1%)	12 (18,2%)	11 (16,7%)	31 (15,7%)
Cantidad Iguales	7 (10,6%)	15 (22,7%)	12 (18,2%)	34 (17,2%)

En promedio, la heurística de mejoramiento propuesta permite encontrar resultados mejores en un 0.317% para los problemas de la muestra. También mediante ella, se obtienen mejores o iguales resultados que la heurística propuesta por Rabadi *et al.* (2006), siendo esta cantidad una proporción de 84.4% (67.2% + 17.2%) de los problemas evaluados. Además, al analizar las desviaciones máximas se puede observar que la heurística propuesta permite alcanzar un máximo de 13,318%, lo que indica que mediante la heurística propuesta se obtuvo un resultado mejor que el encontrado por Rabadi *et al.* (2006) en un 13.318%; la desviación mínima es de –1.19%, lo que indica que el peor resultado observado es un 1.19% más malo que el encontrado mediante la aplicación de la heurística propuesta por Rabadi *et al.* (2006).

En la Figura Nº5.4 se muestran histogramas de las desviaciones (δ) observadas en los problemas de muestra, para los escenarios de "tiempos de procesamiento y setup balanceados" (a), "tiempos de procesamiento dominante" (b) y "tiempos de setup dominante" (c). Adicionalmente, se incluye un histograma con las desviaciones agrupadas, donde se puede observar la mejora lograda mediante la heurística de mejoramiento propuesta, por sobre la propuesta por Rabadi *et al.* (2006). En la Figura Nº5.5 se muestran las desviaciones promedio agrupadas por cantidad de trabajos y máquinas en conjunto para todos los escenarios.

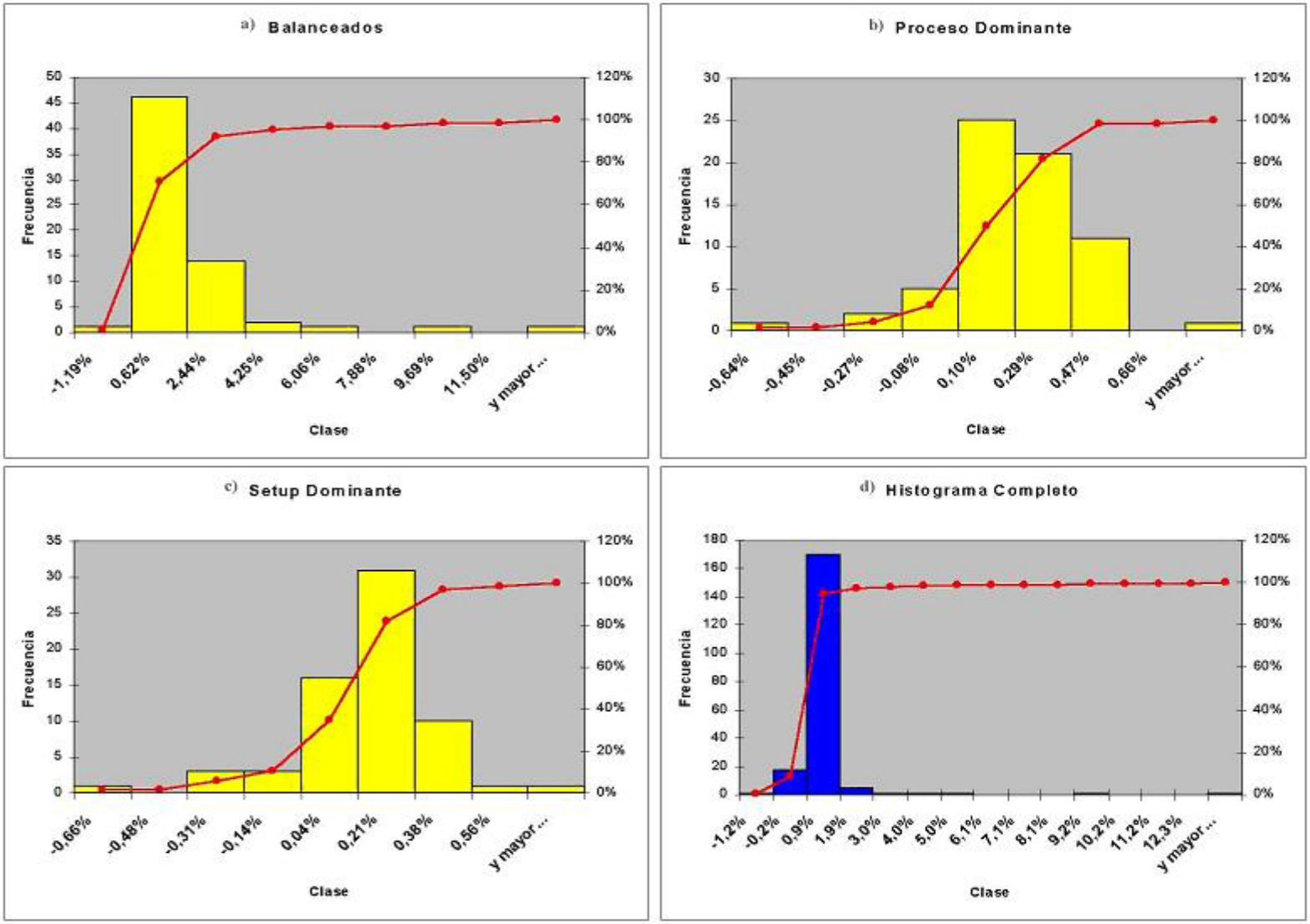


Figura N°5.4: Comparación de Heurísticas de Mejoramiento.

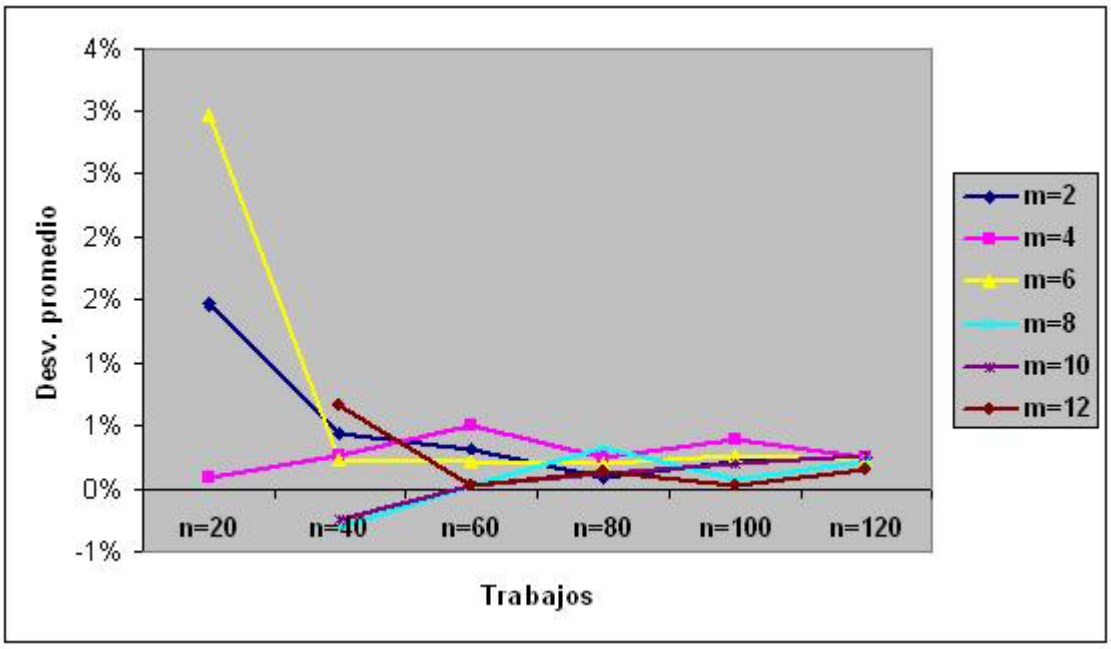


Figura N°5.5: Desviaciones promedio agrupadas por cantidad de máquinas y trabajos.

En las Figura Nº5.5 se puede apreciar que las desviaciones promedio están, la mayoría, por sobre el eje horizontal, lo que indica claramente la mejora en promedio lograda mediante la aplicación de la heurística de mejoramiento propuesta, por sobre la de Rabadi *et al.*(2006). En la Tabla Nº5.3 se presentan los datos de la Figura Nº5.5. Por lo tanto, se comprueba la hipótesis "*Considerar la estructura del problema en el diseño de heurísticas de mejoramiento local, permite obtener mejores resultados*".

Tabla Nº5.3: Desviaciones promedio agrupadas por cantidad de máquinas y trabajos.

Máquinas	***2***	***4***	***6***	***8***	***10***	***12***
n=20	*1,461%*	*0,092%*	*2,976%*			
n=40	*0,441%*	*0,270%*	*0,244%*	*-0,316%*	*-0,239%*	*0,666%*
n=60	*0,323%*	*0,510%*	*0,220%*	*0,023%*	*0,026%*	*0,034%*
n=80	*0,098%*	*0,250%*	*0,211%*	*0,336%*	*0,129%*	*0,137%*
n=100	*0,213%*	*0,394%*	*0,260%*	*0,070%*	*0,201%*	*0,030%*
n=120	*0,257%*	*0,259%*	*0,242%*	*0,228%*	*0,264%*	*0,163%*
Promedio	*0,465%*	*0,296%*	*0,692%*	*0,068%*	*0,076%*	*0,206%*

CAPITULO 6: AJUSTE DE PARÁMETROS Y RESULTADOS PRELIMINARES.

En este capítulo, se presentan los resultados obtenidos mediante la aplicación de Meta-RaPS usando la heurística constructiva LACH discutida en Capítulo 4, y la heurística de mejoramiento presentada en el Capítulo 5.

En la sección 6.1 se describe la metodología utilizada para fijar los parámetros *%prioridad* y *%restricción* de la fase constructiva de Meta-RaPS, considerando la relación existente entre los parámetros. También se extiende el criterio de fijación clásico de calidad de la mejor solución construida, a un criterio de dos objetivo que considera adicionalmente la calidad de todas las soluciones construidas. En la sección 6.2 se presenta una comparación de la aplicación propuesta de Meta-RaPS (Meta-RaPS LACH) con un enfoque (Meta-RaPS SAPSL) que considera una heurística constructiva glotona propuesta por Moraga (2004) y Rabadi *et al*. (2006). Donde se muestra la ventaja de usar Meta-RaPS LACH por sobre Meta-RaPS SAPSL.

6.1 Ajuste de parámetros.

Todas las metaheurísticas requieren de fijación de parámetros, Michalewicz *et al.* (2002) distinguen dos formas generales para abordar el problema de fijación de parámetros: *parameter tuning* y *parameter control*. Por *parameter tuning*, los autores se refieren a la metodología comúnmente usada para encontrar buenos valores para los parámetros antes de correr el algoritmo (ajuste off-line), y luego correr el algoritmo con esos valores.

Alternativamente, *parameter control* (ajuste on-line) se refiere a comenzar la corrida del algoritmo con valores iniciales de los parámetros, que son modificados durante la corrida. Michalewicz *et al.* (2002) establece que el valor de los parámetros determina la calidad de la solución en términos de cercanía a la solución óptima, por lo tanto una buena fijación de los parámetros determina la efectividad del algoritmo. Sin embargo, elegir los valores correctos para los parámetros puede ser una tarea consumidora de tiempo.

En esta sección se fijarán los parámetros *%p* y *%r* de la fase constructiva de Meta-RaPS utilizando la metodología de *parameter tuning*, presentando el criterio de fijación de parámetros como una extensión del criterio comúnmente usado. También se presenta la metodología utilizada para fijar los parámetros y los resultados obtenidos. Posteriormente en el Capítulo 7 se abordará el problema de fijación de parámetros basándose en técnicas de *parameter control*.

Como algo previo al ajuste de parámetros, es necesario determinar el criterio que se utilizará para dicho ajuste, para poder determinar que valor de los parámetros es recomendable. Entre esos criterios, se puede mencionar: mejor función objetivo de las soluciones construidas, mejor media de soluciones construidas, maximización de variabilidad, etc.

También es necesario determinar si existe alguna relación entre los parámetros en estudio, para poder determinar si los parámetros pueden ser fijados aisladamente o en conjunto. Según Michalewicz *et al.* (2002) los parámetros en la mayoría de las metaheurísticas no son independientes, y eso genera la complejidad e imposibilidad de probar todas las combinaciones. En el caso particular de Meta-RaPS, los parámetros *%p* y

%r tienen una alta relación, en el sentido que ambos permiten la adición de aleatoriedad, por lo que es necesario ajustarlos conjuntamente.

6.1.1 Criterio de fijación.

Hasta el momento, en todas las aplicaciones de Meta-RaPS el criterio utilizado para fijar los parámetros *%p* y *%r* se basa en buscar aquella combinación (*%p, %r*) que reporte la solución de mejor calidad (mejor valor de la función objetivo). Sin embargo en este estudio se propone una extensión a aquel criterio, considerando adicionalmente la calidad de todas las soluciones construidas. Lo que se propone es fijar el par (*%p, %r*) utilizando un criterio de dos objetivos, de tal forma que permita optimizar la calidad de la mejor solución construida, y la calidad de todas las soluciones construidas (consideradas como una población de soluciones).

La premisa básica de esta metodología es que cada par de parámetros (*%p, %r*) permite generar una población muy grande de soluciones construidas, y que mientras mejor sea esa población, mejores resultados podrán ser obtenidos por Meta-RaPS. Para considerar la calidad de la población de soluciones es necesario comparar las soluciones construidas con respecto a algún valor de referencia (valor de función objetivo). El makespan del programa de producción que genera la heurística constructiva sin la adición de aleatoriedad (LACH) es un buen valor de referencia, ya que siempre es posible obtenerlo. La forma de considerar la calidad de la población es mediante la proporción de mejores soluciones construidas que la generada por LACH. Por ejemplo, para el problema N°1 de 100 trabajos y 2 máquinas del escenario de tiempos de procesamiento dominante de la librería de problemas propuesta por Rabadi (2005), se corrieron 1000 iteraciones y se probaron

combinaciones de *%p= {80%, 90%}* y *%r={10%, 20%}*, obteniendo los resultados que se muestran en la Tabla N°6.1, donde se puede apreciar que la combinación de *%p=90* y *%r=10* reporta la mejor solución construida y la mayor cantidad de soluciones mejores que el resultado obtenido por LACH sin aleatoriedad (9814). Es posible que ambos criterios (mejor solución y cantidad de mejores) difieran en la combinación de *%p, %r*. Por lo que es necesario crear un factor que permita decidir con cierto grado de inteligencia la mejor combinación de (*%p, %r*). Para evaluar el par (*%p, %r*) de mejor compromiso se propone el siguiente factor (llamado *factor de afinidad*):

$$factor\ de\ afinidad = \left(\frac{Mejor\ construida}{cont\ +1} \right) \tag{6.1}$$

Donde: *Mejor construida* se refiere a la mejor solución construida en las iteraciones de prueba, y "*cont"* es la cantidad de soluciones construidas mejores que la generada por LACH sin aleatoriedad. En la ecuación se agrego "+1" para evitar problemas en el cálculo del factor de afinidad cuando *cont =0*.

Tabla N°6.1: Ejemplo criterio de fijación.

%p	*%r*	Mejor solución	Cantidad de mejores	Factor de afinidad
80	10	9725	556	17,46
80	20	9736	302	32,13
90	10	***9724***	***627***	15,48
90	20	9732	526	18,47

Minimizando el *factor de afinidad* se puede encontrar una combinación de (*%p, %r*) de buen compromiso entre los objetivos de minimizar la función objetivo y generar una población de soluciones construidas de buena calidad.

A pesar de que con esta metodología no se garantiza generar una combinación óptima de *%p* y *%r*, el factor de afinidad tiene varias ventajas; una de ellas es que disminuye las posibilidades de fijar los parámetros erróneamente cuando se genera una solución atípica (cuya función objetivo es muy buena). También provee de un mecanismo que permite diferenciar las combinaciones de parámetros con respecto a los dos criterios comentados anteriormente.

A continuación se muestra como se deduce la expresión del *factor de afinidad*. Desde un punto de vista multi-objetivo, el problema de fijación abordado en este estudio es el siguiente:

$$minimizar \left(\frac{Mejor\ construida}{LACH} \right) \tag{6.2}$$

$$maximizar \left(\frac{cont+1}{Q} \right) \tag{6.3}$$

La ecuación (6.2) indica la minimización del porcentaje de reducción del makespan obtenido por LACH. Además, el valor de *LACH* es fijo y conocido, por lo tanto mientras menor sea el valor de la *Mejor construida*, menor será el valor de la expresión (6.2). La ecuación (6.3) indica la proporción de soluciones construidas que tienen un makespan mejor que el encontrado por LACH. Además, Q representa el tamaño de la muestra, es decir, cuantas soluciones son construidas para evaluar el par *(%p, %r)*. Por lo tanto mientras más soluciones construidas son mejores que *LACH*, mayor será el valor de la expresión (6.3).

Como por un lado se desea minimizar la expresión (6.2), y por el otro se desea maximizar la expresión (6.3). Si agrupamos ambos criterios en la siguiente expresión (6.4) y minimizamos, podríamos encontrar una combinación de *%p* y *%r* de buen compromiso.

$$minimizar \quad \frac{\left(\frac{Mejor\ construida}{LACH}\right)}{\left(\frac{cont+1}{Q}\right)} = \left(\frac{Mejor\ construida}{LACH}\right)\left(\frac{Q}{cont+1}\right) \qquad (6.4)$$

Ordenando se obtiene:

$$minimizar \quad \left(\frac{Q}{LACH}\right)\left(\frac{Mejor\ construida}{cont+1}\right) \qquad (6.5)$$

Como *Q* y *LACH* son valores constantes y conocidos, la expresión (6.5) es equivalente a:

$$minimizar \quad \left(\frac{Mejor\ construida}{cont+1}\right) \qquad (6.6)$$

Luego de la expresión (6.6) se obtiene el *factor de afinidad.* Por lo tanto al minimizar la expresión (6.6) se logra obtener una combinación de parámetros *%p* y *%r* de buen compromiso.

6.1.2 Metodología de fijación.

Los parámetros de Meta-RaPS pueden ser fijados arbitrariamente o utilizando un método sistemático. En esta sección, se utiliza una metodología formal para fijar los parámetros *%p* y *%r* similar a la propuesta por Rabadi *et al*. (2006), la cual se describe a continuación:

Paso 1: Seleccionar una muestra representativa de problemas analizar. En este estudio, se utilizan los problemas propuestos por Rabadi (2005), de tal forma que se seleccionó un par

de problemas para cada combinación de cantidad de máquinas, trabajos y escenario de tiempos de procesamiento y preparación (los problemas utilizados son los mismos que se utilizaron en la sección 5.2)

Paso 2: Seleccionar el dominio de parámetro sobre el cual cada parámetro puede variar. El dominio de parámetro puede variar en el rango [0%-100%] para *%p* y *%r*. En este estudio, se fijarán los parámetros *%p* y *%r* utilizando el rango discretamente con incrementos de 10%.

Paso 3: Para cada problema en la muestra, correr Meta-RaPS usando una técnica apropiada para fijar los parámetros. Luego de correr cada par de *%p* y *%r* durante 1000 iteraciones, calcular para cada uno el *factor de afinidad* descrito en la sección 6.1.1, y seleccionar aquella combinación de *%p* y *%r* que reporte el menor *factor de afinidad*. En este estudio, la metodología consistió en correr cada par de problemas de la muestra durante 1000 iteraciones para cada combinación de *%p* y *%r* en el dominio, utilizando incrementos de 10%.

Paso 4: Usar los parámetros obtenidos en el Paso 3, agruparlos según escenario, cantidad de trabajos y máquinas, y promediarlos. Luego, correr todos los problemas usando esos promedios. Se considerará adicionalmente que los parámetros *I* y *%i* se fijarán en 5000 iteraciones y 60% respectivamente, similar a los utilizados por Rabadi *et al*. (2006).

Es simple notar que la metodología propuesta en esta sección no produce una combinación óptima de *%p* y *%r*. Pero la intención al igual que Rabadi *et al*. (2006) y Moraga (2004) es proveer un método sistemático para seleccionar buenos parámetros.

Los resultados obtenidos mediante la aplicación de la metodología propuesta se muestran en el Anexo A.

6.2 Resultados de Meta-RaPS LACH, comparación con Meta-RaPS SAP-SL.

En esta sección se realiza una comparación entre los resultados obtenidos por Meta-RaPS SAPSL (Rabadi *et al.,* 2006) y los resultados obtenidos por la aplicación propuesta (Meta-RaPS LACH) usando los parámetros *%p* y *%r* obtenidos en la sección 6.1.2. Adicionalmente, para comparar en igualdad de condiciones ambas aplicaciones, la cantidad de iteraciones se fijó en 5000 y el porcentaje de mejoramiento (*%i*) en 60% (valores utilizados por Rabadi *et al.,* 2006). Para realizar la comparación, se utilizó la muestra de problemas generada en la sección 5.2. Las desviaciones fueron calculadas mediante la siguiente expresión:

$$\delta = \left(\frac{Existente - Propuesto}{Existente} \right) \times 100\% \qquad (6.7)$$

Donde:
Existente: resultado generado por la aplicación de Meta-RaPS SAPSL (Rabadi *et al.,* 2006).
Propuesto: resultado generado por la aplicación de Meta-RaPS LACH.

Los resultados obtenidos, agrupados por ambientes de tiempos de procesamiento, se resumen en la Tabla N°6.2.

Tabla N°6.2: Comparación Meta-RaPS SAPSL vs. Meta-RaPS LACH.

	Balanceados	*Proc. Dominante*	*Setup dominante*	*Total*
Promedio (δ)	1,086%	0,233%	0,195%	0,505%
Mínimo (min δ)	-1,695%	-0,552%	-0,430%	-1,695%
Máximo (max δ)	13,318%	1,798%	1,836%	13,318%
Cantidad Mejores	51 (77,3%)	39 (59,1%)	38 (57,6%)	128 (64,6%)
Cantidad Peores	12 (18,2%)	18 (27,3%)	21 (31,8%)	51 (25,8%)
Cantidad Iguales	3 (4,5%)	9 (13,6%)	7 (10,6%)	19 (9,6%)

En promedio, para los problemas de la muestra, Meta-RaPS LACH permite obtener resultados 0.505% mejores en comparación con los obtenidos por Meta-RaPS SAPSL. También permite obtener mejores o iguales resultados que Meta-RaPS SAPSL, siendo esta cantidad una proporción de 74.2% (64.6% + 9.6%) de los problemas evaluados. Además, al analizar las desviaciones máximas se puede observar que el algoritmo de Meta-RaPS propuesto permite alcanzar un máximo de 13,318%, lo que indica que mediante la aplicación propuesta se obtuvo un resultado mejor que el encontrado por Meta-RaPS SAPSL en un 13.318%; la desviación mínima tiene un valor de -1.695%, lo que indica que el peor resultado observado es un 1.695% más malo que el encontrado por Meta-RaPS SAPSL. Es simple apreciar la calidad de Meta-RaPS LACH por sobre la de Meta-RaPS SAPSL, con lo cual se muestra que utilizar una heurística constructiva visionaria en la fase constructiva de Meta-RaPS permite obtener mejores resultados que si se utilizará una heurística glotona.

En la Figura Nº6.1 se muestran los histogramas de las desviaciones observadas en los problemas de muestra, para los tres escenarios propuestos por Rabadi *et al*. (2006). Adicionalmente, en la Figura Nº6.1 se incluye un histograma con las desviaciones agrupadas, donde se puede observar la mejora lograda mediante la aplicación de Meta-RaPS LACH, por sobre Meta-RaPS SAPSL.

En la Figura Nº6.2 se muestran gráficos para las desviaciones, diferenciados para cada tipo de escenario. Donde se puede observar una pequeña disminución en el desempeño de Meta-RaPS LACH para los problemas de la muestra que consideran 12 máquinas, pero a pesar de esa disminución en el desempeño, las desviaciones desfavorables a Meta-RaPS LACH son en promedio pequeñas.

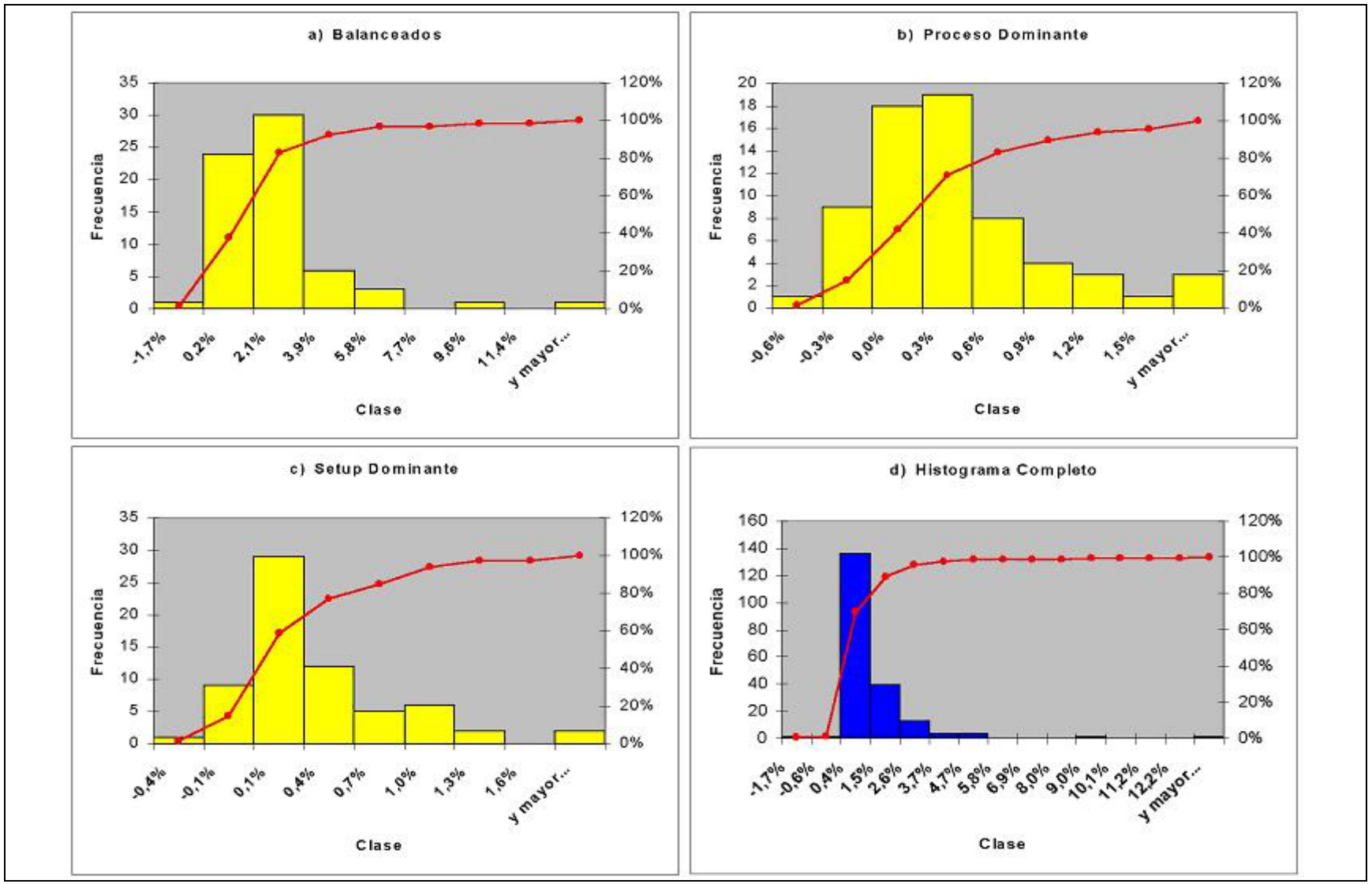
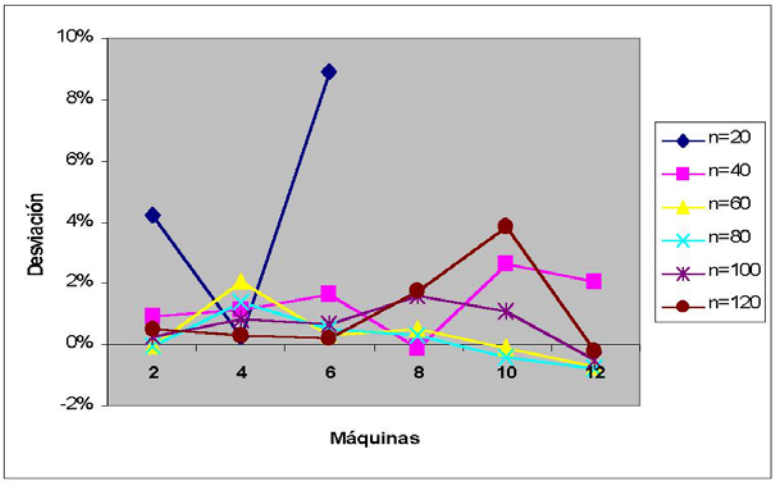
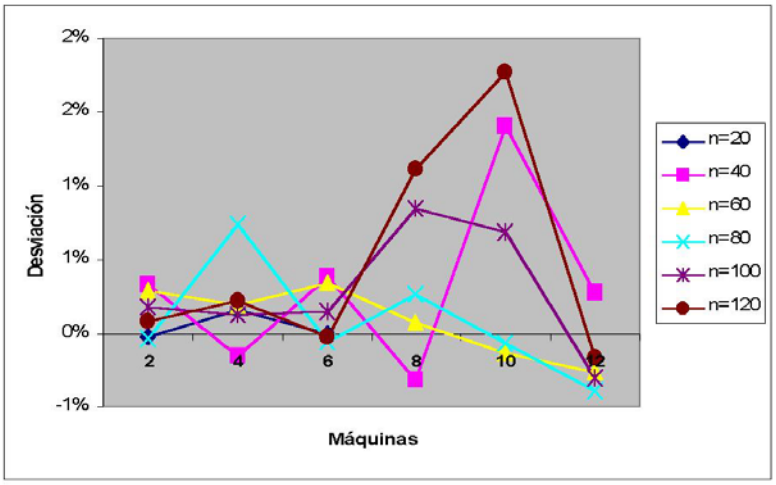


Figura N°6.1: Comparación Meta-RaPS SAPSL vs. Meta-RaPS LACH.

a) Tiempos de proceso y setup balanceados.



b) Tiempos de proceso dominante.



c) Tiempos de setup dominante.

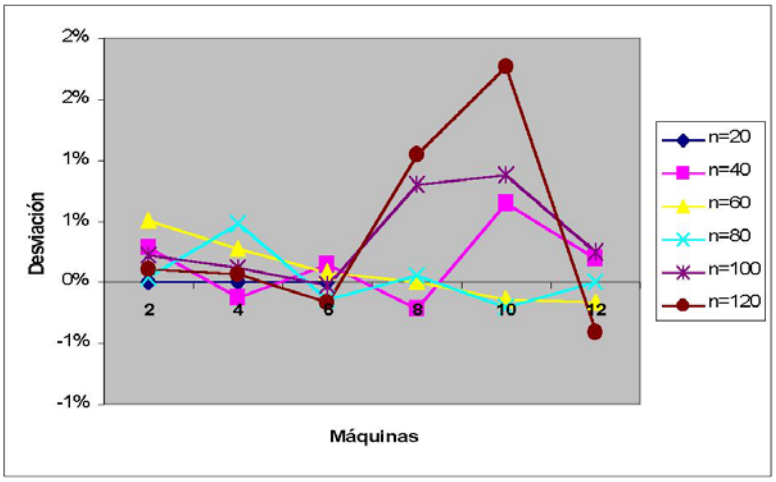


Figura Nº6.2: Comparación de Meta-RaPS SAPSL vs. Meta-RaPS LACH.

CAPITULO 7: METODOLOGÍAS DE AUTO-FIJACIÓN DE PARÁMETROS.

En este capítulo, se presentan técnicas que facilitan al usuario la difícil tarea de fijar los parámetros involucrados en el algoritmo de Meta-RaPS LACH.

En la sección 7.1 se aborda el problema de fijación de los parámetros *%p* y *%r*, y se presenta el diseño de una metodología basada en técnicas de control de parámetros (parameter control) y métodos de gradiente para optimización global, en la cual se parte con una fijación inicial arbitraria de los parámetros *%p* y *%r*. Luego a medida que se construyen soluciones los parámetros *%p* y *%r* se adaptan, para finalmente fijarse automáticamente. Se comparan los resultados obtenidos mediante este enfoque con respecto a los resultados obtenidos por el ajuste off-line realizado en la sección 6.1.

En la sección 7.2 se estudia el problema de fijación del parámetro *%i*, y se presenta una metodología que permite evitar la fijación del parámetro *%i*, la cual modifica la forma general del algoritmo de Meta-RaPS. Se compara este método con la metodología clásica implementada por Meta-RaPS.

7.1 Auto-Ajuste de %prioridad y %restricción.

Como se presentó en la sección 6.1, el problema de fijación de los parámetros *%p* y *%r* es una tarea difícil, que consume mucho tiempo y memoria computacional. Por lo que es necesario desarrollar metodologías o herramientas que permitan evitar o automatizar esta tarea al usuario.

Para desarrollar esta metodología se considera el *factor de afinidad* definido en la sección 6.1, y se considera la misma premisa básica de que cada par (*%p, %r*) permite generar una población semi-infinita de soluciones construidas, y que mientras mejor sea esa población, mejores resultados podrán ser obtenidos por Meta-RaPS. Por lo tanto cuando se construyen *Q* soluciones con un determinado par (*%p, %r*), lo que se está haciendo indirectamente es extraer una muestra de tamaño *Q* de la población de (*%p, %r*). Como en el proceso de construcción de soluciones influye el factor aleatorio, la muestra en cuestión es una muestra aleatoria.

Con lo anterior se puede pensar que existe una combinación óptima de *%p* y *%r*, o bien, existe una zona donde *%p* y *%r* ofrecen un buen *factor de afinidad*.

La metodología propuesta de auto-ajuste de *%p* y *%r* ofrece diversidad, a medida que se avanza en las iteraciones se pasa a mejores zonas de *%p* y *%r*, en ese proceso se exploran otras zonas, que a pesar de no generar buenas soluciones construidas ofrecen diversidad en el proceso de construcción de soluciones. Así, a medida que las iteraciones avanzan, los valores de *%p* y *%r* se acercan a la mejor combinación.

Para explicar el funcionamiento de la metodología propuesta, se hará un símil con los métodos de optimización de problemas no-lineales no-restringidos, que se basan en el uso del gradiente, comúnmente llamados Métodos de Gradiente, entre ellos: método de ascenso y descenso acelerado, método de Newton, etc

Todos los métodos de gradiente, utilizan comúnmente tres conceptos: punto conocido de partida, dirección de búsqueda y longitud de cambio. Estos métodos permiten en particular obtener soluciones óptimas locales a partir de un punto de partida. Para ello, partiendo desde un punto arbitrario, se determina la mejor dirección y la mejor longitud de

recorrido. Este es un proceso iterativo, y en cada iteración se determina la dirección y la longitud para poder alcanzar el punto óptimo local en el menor tiempo posible (menor cantidad de iteraciones).

La metodología propuesta para el auto-ajuste de los parámetros *%p* y *%r* es una emulación de los métodos de gradiente. Para ello, se considera el problema desde un punto de vista de optimización no-lineal, no-restringido, de dos dimensiones (*%p, %r*). (ver Figura N°7.1).

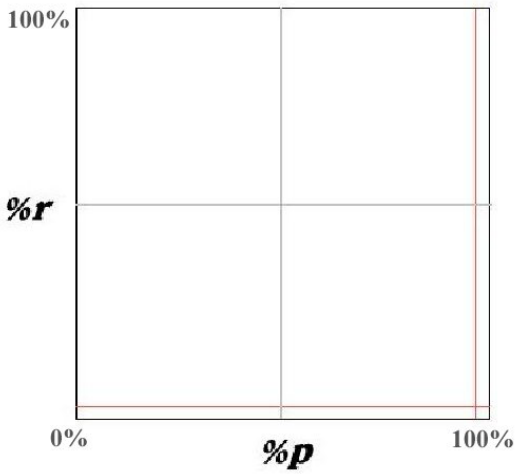


Figura N°7.1: Esquema bidimensional de *%p* y *%r*.

Supongamos que se parte el proceso de búsqueda de la combinación óptima (*%p, %r*)* en un punto arbitrario (*%p, %r*)^0. Luego, a partir de ese punto arbitrario, se determina la dirección en que debe cambiar y la longitud del cambio. Para ello, se deben extraer muestras de las poblaciones de soluciones construidas en la vecindad de (*%p, %r*)^0, para realizar esto es necesario determinar previamente la longitud del cambio. Si *S* es la longitud, para evaluar la vecindad de (*%p, %r*)^0, es necesario evaluar cuatro combinaciones: (*%p*+*S, %r*), (*%p*–*S, %r*), (*%p, %r*–*S*) y (*%p, %r*+*S*).

Luego de eso es necesario calcular el *factor de afinidad* de cada par, y cambiar el punto $(\%p, \%r)^0$ por $(\%p, \%r)^1$, correspondiente a la mejor combinación (aquella que reporte el menor *factor de afinidad*). El proceso es iterativo, y si en algún momento ocurre que la mejor combinación elegida es idéntica a la que se tenia es necesario reducir *S* (esto ocurre cuando: $(\%p, \%r)^i = (\%p, \%r)^{i+1}$). Esto se debe a que en la vecindad de longitud *S* ya no se encuentran mejores combinaciones de (*%p, %r*) que la combinación $(\%p, \%r)^i$. La reducción mencionada se realiza a la mitad de *S* a mediada que el algoritmo se acerca a los parámetros pseudo-óptimos (*%p, %r*)*.

A continuación se presenta el algoritmo asociado a la metodología de auto-ajuste de parámetros *%p* y *%r*.

7.1.1 Algoritmo de Auto-Ajuste de *%p* y *%r*.

Paso 0.

%p y *%r* iniciales: arbitrarios (recomendado *%p*=*%r*=50%)

S inicial: distancia o salto de cambio (recomendado *S*=40%)

Q: Tamaño de la muestra (recomendado *Q*=50)

Paso 1.

Generar muestra de tamaño *Q* para la combinación (*%p, %r)*, y calcular *factor de afinidad_0*.

Generar muestra de tamaño *Q* para la combinación (*%p+S, %r)*, y calcular *factor de afinidad_1*.

Generar muestra de tamaño *Q* para la combinación (*%p–S, %r)*, y calcular *factor de afinidad_2*.

Generar muestra de tamaño *Q* para la combinación (*%p, %r–S)*, y calcular *factor de afinidad_3*.

Generar muestra de tamaño *Q* para la combinación (*%p, %r+S)*, y calcular *factor de afinidad_4*.

Paso 2.

Seleccionar el menor *factor de afinidad*.

Si es *factor de afinidad*_0, hacer *S=S/2*

Si es *factor de afinidad*_1, hacer *%p=%p+S*

Si es *factor de afinidad*_2, hacer *%p=%p–S*

Si es *factor de afinidad*_3, hacer *%r=%r–S*

Si es *factor de afinidad*_4, hacer *%r=%r+S*

Paso 3.

Si $S < 1$, ir a Paso 4.

De lo contrario ir a Paso 1.

Paso 4.

Correr las restantes iteraciones hasta completar *I* iteraciones.

El Paso 0, corresponde a la inicialización de las variables involucradas, entre ellas *%prioridad* y *%restricción*, los cuales deben partir con una valor inicial, el cual puede ser elegido arbitrariamente. También es necesario determinar el valor de la distancia inicial de cambio *S*, cuyo valor será reducido a medida que el algoritmo se acerque a la zona (*%p, %r*)*. Adicionalmente, es necesario determinar el tamaño de las muestras *Q*. Por experiencia se propone usar valores iniciales de *%p=%r*=50%, y *S*=40%, ya que permiten

llegar rápidamente a las zonas extremas en el dominio de *%p* y *%r*. También se recomienda un valor de *Q*=50, ya que se realizaron estudios de *Q={50, 70, 100}*, y no se observaron diferencias en los resultados. Un valor demasiado grande de *Q* hace que el algoritmo use una mayor cantidad de iteraciones en acercarse a la zona (*%p, %r*)*, y con ello una mayor cantidad de tiempo.

El Paso 1 consiste en la extracción de muestras en la vecindad de distancia *S* del punto (*%p, %r)*, y la caracterización de cada muestra con su respectivo *factor de afinidad*. En la Figura N°7.2, se muestra lo que se realiza en este paso en forma algorítmica como un procedimiento. Es importante mencionar que en esta parte del proceso, toda solución construida es evaluada para decidir si es aceptada para pasar por la fase de mejoramiento. También en esta parte existe un criterio de detención, el cual se detalla en la Figura Nº7.2.

En el Paso 2, se realiza el cambio de (*%p, %r)* a una mejor combinación (aquella que genera un menor *factor de afinidad*). Si ocurre que en la vecindad de distancia *S* no existe ninguna combinación mejor, se reduce la distancia *S* para continuar la búsqueda en una vecindad más cercana, esto ocurre cuando el menor *factor de afinidad* es el correspondiente a la combinación central de *%p* y *%r* (*factor de afinidad*_0). Es importante considerar que pasa con los puntos extremos, ya que cuando *%p* es muy cercano a 100%, o cuando *%r* es muy cercano a 0%, no existe mucha adición de aleatoriedad, y es posible desperdiciar iteraciones construyendo la misma solución varias veces. Para solucionar este problema, siempre que se prueben cambios del tipo *%p*+*S*, se evalúa si superan a 100%, si es así se corrige a 95%; de igual forma cuando se prueben cambios del tipo o *%r*–*S*, se evalúa si el valor es inferior a 0%, si es así se corrige a 5% (ver los márgenes en la Figura N°7.1).

El Paso 3, es el que determina si se continúa con el proceso de búsqueda de (*%p, %r)**, o si ya fue fijada esa mejor combinación. Esto se realiza determinando la distancia *S* más pequeña, valor 1 en el algoritmo, de tal forma que si *S* es menor que 1% no se continúa con el proceso de búsqueda.

```
Generar muestra (P, R)
        cont = 0
        Mejor_construida= infinito
         i=1
        mientras (i<Q), hacer:
                i=i+1
                Iter=Iter+1                          % se crea una iteración
                Si Iter>I, terminar                  % criterio de detención
                x = Construcción(P, R)               % construye solución
                si función_obj(x) ≤ LACH
                        cont = cont +1               % conteo de soluciones buenas
                si función_obj(x) < Mejor_construida
                        Mejor_construida = función_obj(x)  % actualización de mejor construida
                si función_obj(x) ≤ Ψ(%i),           % regla filtro de mejoramiento
                        x = mejoramiento(x);         % búsqueda mejor solución
                si función_obj(x) ≤ mejor_objetivo,  % actualización
                        mejor_objetivo ← función_obj(x); x* ←x;
        Retornar  factor de afinidad = ( Mejor construida / cont +1 )
fin
```

Figura N°7.2: Procedimiento de generación de muestras y cálculo de factor de afinidad.

El Paso 4 consiste en continuar la corrida del algoritmo hasta completar las *I* iteraciones establecidas por el usuario con los parámetros (*%p, %r)** encontrados. Por ejemplo, si el usuario fija una cantidad de 5000 iteraciones (*I*), y se utilizan 2000 iteraciones en el proceso de auto ajuste, se deberán correr las restantes 3000 iteraciones con los parámetros fijados.

Es posible que mediante el procedimiento propuesto, no se logre encontrar la combinación (*%p, %r)**. Pero si es posible llegar a las cercanías, es decir a la zona (*%p, %r)**, y continuar indefinidamente la búsqueda con un *S* muy pequeño, hasta que se cumpla el criterio de detención.

En el Algoritmo de Auto-Ajuste de *%p* y *%r*, se asumió que existe convergencia a una zona de (*%p, %r*). Pero es necesario responder a la pregunta: ¿Qué pasa si no existe convergencia a la zona?. La respuesta a esta pregunta es simple, y sólo basta recordar el símil que se realizó con los métodos de gradiente, donde para solucionar este problema, basta con correr reiteradas veces el algoritmo partiendo desde puntos aleatorios o distanciados. Para el problema $R_m|S_{ijk}|C_{max}$, se observa convergencia a una zona de (*%p, %r*), por lo que no es necesario correr reiteradas veces el algoritmo partiendo desde puntos aleatorios o distanciados.

Entre las ventajas del algoritmo propuesto, se destacan:

1. Economía de tiempo y esfuerzo con respecto al ajuste off-line.
2. No es discreto, genera precisión en la fijación y tiene posibilidad de corrección si se genera un cambio erróneo en (*%p, %r)*.
3. Es una metodología general y puede ser aplicada en otros desarrollos de Meta-RaPS.
4. Los resultados son satisfactorios.

En la siguiente sección se presentan resultados computacionales.

7.1.2 Comparación Meta-RaPS LACH off-line vs. on-line.

En esta sección se realiza una comparación entre los resultados obtenidos por Meta-RaPS LACH con ajuste off-line y los resultados obtenidos por la aplicación propuesta (Meta-RaPS LACH on-line). Para ello se utilizan los resultados obtenidos para la muestra de problemas de la sección 6.2. Adicionalmente se considera una cantidad de iteraciones I=5000, y un porcentaje de mejoramiento de $\%i$=60%, para igualar las condiciones de los resultados obtenidos en la sección 6.2.

Las desviaciones fueron calculadas mediante la siguiente expresión:

$$\delta = \left(\frac{A - B}{A} \right) \times 100\% \qquad (7.1)$$

Donde:

A: resultado generado por la aplicación de Meta-RaPS LACH utilizando ajuste off-line.

B: resultado generado por la aplicación de Meta-RaPS LACH utilizando ajuste on-line.

Los resultados obtenidos, agrupados por ambientes de tiempos de procesamiento y setup, se resumen en la Tabla N°7.1.

Tabla N°7.1: Comparación Meta-RaPS LACH off-line vs. on-line.

	Balanceados	***Proc. Dominante***	***Setup dominante***	***Total***
Promedio (δ)	0,139%	-0,017%	0,087%	0,070%
Mínimo (min δ)	-4,751%	-0,850%	-0,420%	-4,751%
Máximo (max δ)	4,563%	1,227%	1,578%	4,563%
Cantidad Mejores	36 (54,5%)	25 (37,9%)	28 (42,4%)	89 (44,9%)
Cantidad Peores	25 (37,9%)	28 (42,4%)	26 (39,4%)	79 (39,4%)
Cantidad Iguales	5 (7,6%)	13 (19,7%)	12 (18,2%)	30 (15,2%)

En promedio, el ajuste on-line permite encontrar resultados mejores en un 0.07% para los problemas de la muestra. También permite obtener resultados mejores o iguales

que el ajuste off-line, siendo esta cantidad una proporción de 60.1% (44.9% + 15.2%) de los problemas evaluados. Además, al analizar las desviaciones máximas se puede observar que el algoritmo de Meta-RaPS on-line propuesto permite alcanzar un máximo de 4,563%, lo que indica que mediante el ajuste on-line se obtuvo un resultado mejor que el encontrado mediante ajuste off-line en un 4.563%; la desviación mínima es de –4.751%, lo que indica que el peor resultado observado es un 4.751% más malo que el encontrado mediante el ajuste off-line. A pesar de que los indicadores no son extremadamente favorables, son mejores en promedio que los obtenidos con el ajuste off-line. Si se corrieran más iteraciones con el ajuste on-line, la calidad de las soluciones sería mejor. Es importante destacar que esto se debe en gran medida a que el ajuste on-line usa una cantidad de las iteraciones del total fijado por el usuario para encontrar los mejores parámetros *%p* y *%r*, o en su defecto llegar a la vecindad más cercana.

Con los resultados obtenidos, se puede apreciar que el ajuste on-line permite obtener mejores resultados que mediante un ajuste off-line. Además las ventajas desde el punto de vista de complejidad son favorables al ajuste on-line debido al ahorro de tiempo y esfuerzo que le otorga al usuario.

En la Figura Nº7.3 se muestran histogramas de las desviaciones observadas en los problemas de muestra, para los escenarios de “tiempos de procesamiento y setup balanceados” (a), “tiempos de procesamiento dominante” (b) y “tiempos de setup dominante” (c). Adicionalmente, se incluye un histograma con las desviaciones agrupadas, donde se puede observar la mejora lograda mediante la aplicación de la técnica de auto-ajuste de parámetros basada en control de parámetros, con respecto al ajuste off-line.

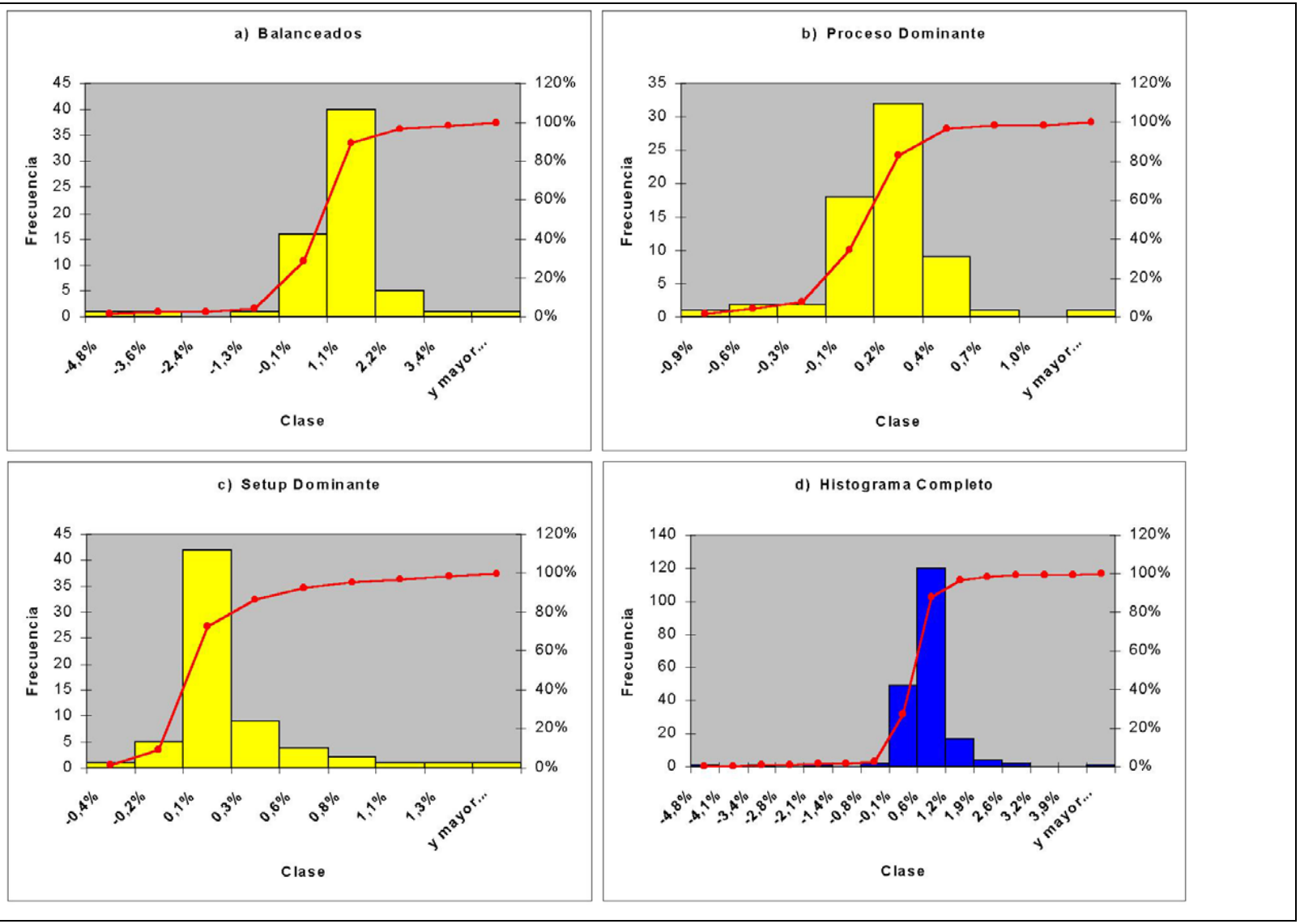


Figura N°7.3: Comparación Meta-RaPS LACH off-line vs. on-line

7.2 Metodología de filtro de mejoramiento.

En esta sección se describe una metodología que permite evitar la fijación del parámetro *%i*, mediante el uso de una regla que permite diferenciar la calidad de las soluciones, de tal forma que las buenas soluciones construidas tienen alta posibilidad de pasar por la heurística de mejoramiento, y las malas soluciones construidas tienen una baja posibilidad de pasar por la heurística de mejoramiento.

Uno de los fundamentos básicos de Meta-RaPS, relacionado a la fase de mejoramiento, es: *"buenas soluciones construidas generan buenas soluciones mejoradas"*. En otras palabras, las soluciones construidas se dividen en dos grupos: buenas y malas. Aceptando solo buenas soluciones construidas para pasar por la fase de mejoramiento. Usando la lógica preposicional y la teoría de conjuntos, definamos:

A: conjunto de buenas soluciones construidas.

B: conjunto de buenas soluciones mejoradas.

M(•): operación de mejoramiento, sólo aplicable a soluciones construidas.

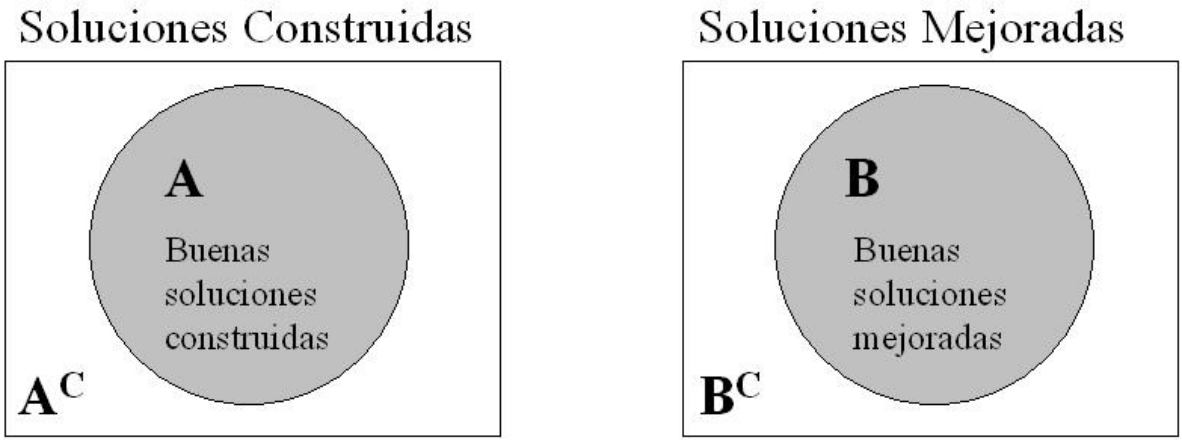


Figura Nº7.4: Conjuntos de soluciones construidas y mejoradas.

Entonces, según el fundamento básico de Meta-RaPS:

$$\forall x \in \mathbf{A} \mid \mathrm{M}(x) \in \mathbf{B}$$

"*todas las buenas soluciones construidas generan buenas soluciones mejoradas*"

Lo que se propone en este estudio es que además de las buenas soluciones construidas, existen algunas malas soluciones construidas con potencial para generar buenas soluciones mejoradas. Con lógica preposicional esto sería:

$$\forall x \in \mathbf{A} \mid \mathrm{M}(x) \in \mathbf{B} \quad \wedge \quad \exists y \in \mathbf{A^C} \mid \mathrm{M}(y) \in \mathbf{B}$$

"*todas las buenas soluciones construidas generan buenas soluciones mejoradas, pero además existen algunas malas soluciones construidas que también generan buenas soluciones mejoradas*"

Mediante el Filtro de Mejoramiento clásico de Meta-RaPS basado en la fijación del parámetro *%i*, al aceptar solamente buenas soluciones construidas, se puede caer en el error de repetir las soluciones construidas aceptadas para pasar por mejoramiento y con ello repetir varias iteraciones con las mismas soluciones. Esto genera un problema de diversidad, y con el concepto propuesto lo que se busca es generar soluciones construidas más diversas para pasar por la fase de mejoramiento local.

Para abordar este concepto, se generó una regla aleatorizada de aceptación/rechazo de soluciones construidas, donde las buenas soluciones construidas tienen altas posibilidades de ser aceptadas para pasar por la fase de mejoramiento, y las malas soluciones construidas tienen una posibilidad menor de ser aceptadas. Este mecanismo propuesto, que no requiere de una previa fijación del parámetro *%i*, se denomina *Filtro de Mejoramiento por Prioridad*, y se detalla en la siguiente sección.

7.2.1 Filtro de Mejoramiento por Prioridad.

La idea fundamental de esta regla de filtro, es añadir diversidad al proceso de búsqueda de Meta-RaPS, y tiene la gran ventaja de no requerir del parámetro *%i*. Pero en su lugar hay que diseñar funciones de filtro. En este estudio, se describen funciones de tipo: trigonométrica, lineal, cuadrática, cúbica y raíz cuadrada.

El *Filtro de Mejoramiento por Prioridad*, mide en cierta forma la calidad de la solución construida, y con ello permite decidir que soluciones deben tener mayor posibilidad de pasar por la fase de mejoramiento. A continuación se realiza una descripción formal.

F(x): es una función que mide la calidad de la solución construida *x*, en base al valor de la función objetivo asociada a la solución construida *x*. Mientras mayor es *F(x)*, la solución *x* tiene mayor probabilidad de ser aceptada para pasar por la rutina de mejoramiento. A medida que *F(x)* disminuye, las probabilidades de que la solución *x* sea aceptada son menores. Definamos F: [menor, mayor] $\rightarrow$ [0, 1], donde *menor* y *mayor* se refieren al valor de la función objetivo de la menor y mayor solución construida encontrada. Para problemas de maximización la función *F* debe ser creciente, y los valores extremos deben ser: *F(menor)*=0, y *F(mayor)*=1. Por otro lado, si el problema es de minimización (como el problema $R_m|S_{ijk}|C_{max}$) la función *F* debe ser decreciente, y los valores extremos deben ser: *F(menor)*=1, y *F(mayor)*=0.

Además, la función puede ser convexa o cóncava. En general una función cóncava, por su geometría, es más permisiva que una convexa, en términos de aceptación de las soluciones. En la Figura Nº7.5 se muestra un gráfico con las funciones vinculadas a un

problema de minimización, en esta figura también se puede apreciar el funcionamiento del parámetro *%i* en Meta-RaPS.

La aceptación para mejoramiento de una solución construida, está determinada aleatoriamente. Para ello se crea un número aleatorio entre 0 y 1, si ese número aleatorio es menor o igual al valor de prioridad de la solución construida, esa solución es aceptada para pasar por la fase de mejoramiento. De tal forma que si x es una solución construida, $F(x)$ determina la probabilidad de que x sea aceptada para pasar por mejoramiento. Por lo tanto se crea un número aleatorio (*rnd*) entre 0 y 1, y si se cumple que $rnd \leq F(x)$, la solución x es aceptada para pasar por mejoramiento, de lo contrario la solución se rechaza y no pasa por mejoramiento.

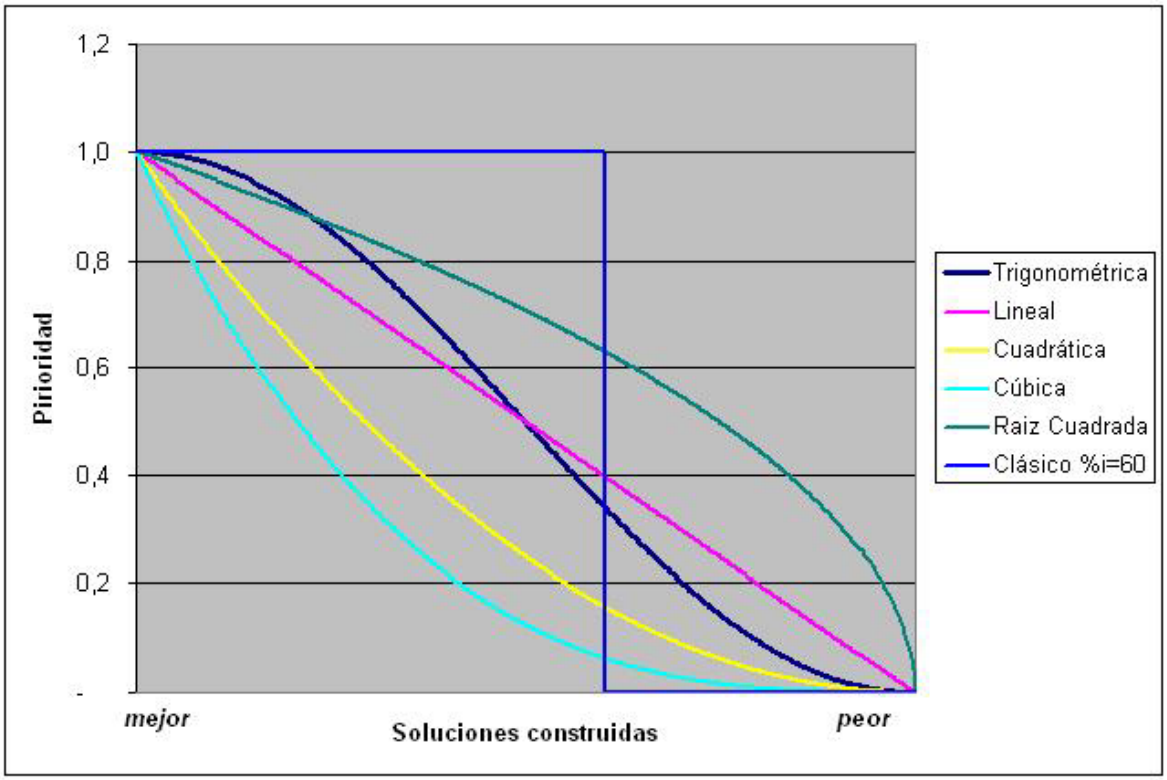


Figura Nº7.5: Funciones de Filtro de Mejoramiento por Prioridad.

7.2.2 Algoritmo del Filtro de Mejoramiento por Prioridad.

Sea:

Mejoramiento(x): procedimiento de mejora para la solución x.

Las variables *mejor* y *peor* se refieren al valor del makespan de la mejor y peor solución construida respectivamente obtenidas en lo que va corrido de las iteraciones en la fase de construcción.

Inicialmente *mejor* y *peor* toman el valor de la heurística constructiva LACH sin la adición de aleatoriedad. A continuación se presenta el algoritmo genérico de Meta-RaPS con el Filtro de Mejoramiento propuesto (Figura Nº7.6).

```
mejor_objetivo = infinito;                          % caso de minimización
mejor = LACH; peor = LACH                           % inicialización
repetir I veces
        x = Construcción(%p, %r);                   % construye solución
        si función_obj(x) < mejor
            mejor ← función_obj(x);                 % actualización mejor construcción
        si función_obj(x) > peor
            peor ← función_obj(x);                  % actualización peor construcción
        Generar un número aleatorio rnd(.)
        si rnd(.) ≤ F(x)                            % regla filtro de mejoramiento
            Mejoramiento(x);                        % búsqueda mejor solución
            Si función_obj(x) ≤ mejor_objetivo,     % actualización
                mejor_objetivo ← función_obj(x); x* ←x;
        fin
```

Figura Nº7.6: Algoritmo de Meta-RaPS con Filtro de Mejoramiento por Prioridad.

7.2.3 Funciones para Filtro de Mejoramiento por Prioridad.

7.2.3.1 Función F(x) Lineal.

Supongamos una función del tipo: $F(x) = a \cdot x + b$, con condiciones extremas:

Si *x=peor*, entonces: F*(x)* = 0, por lo tanto:

$$a \cdot peor + b = 0 \Rightarrow b = -a \cdot peor \tag{7.2}$$

Si *x=mejor*, entonces: F*(x)* = 1, por lo tanto:

$$a \cdot mejor + b = 1 \tag{7.3}$$

Reemplazando (7.2) en (7.3) se obtiene:

$$a = \frac{1}{(mejor - peor)} \tag{7.4}$$

Reemplazando (7.4) en (7.3) se obtiene:

$$b = \frac{-peor}{(mejor - peor)} \tag{7.5}$$

Finalmente:

$$\boxed{F(x) = \left[\frac{x - peor}{mejor - peor} \right]} \tag{7.6}$$

En la Figura N°7.7 se muestra un ejemplo del funcionamiento de esta Función de *Filtro de Mejoramiento por Prioridad.* En donde se puede observar como son aceptadas o rechazadas las soluciones. Las soluciones cuyo correspondiente número aleatorio (puntos rojos) esta por sobre la función lineal son rechazadas, por el contrario, las soluciones cuyo número aleatorio esta por debajo de la función lineal son aceptadas.

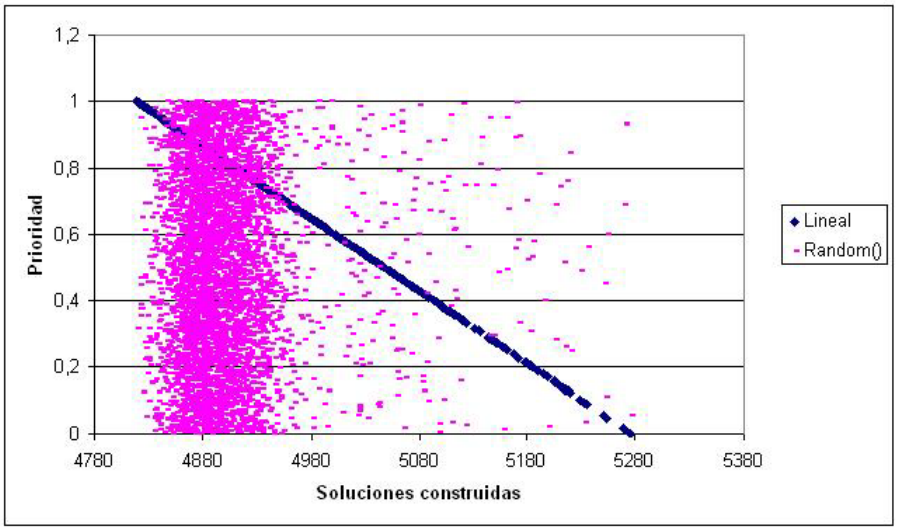


Figura Nº7.7: Filtro de Mejoramiento por Prioridad Lineal.

7.2.3.2 Función F(x) Polinomial.

Supongamos una función polinomial de grado k de la forma: $F(x) = (a \cdot x + b)^k$, con condiciones extremas:

Si *x=peor*, entonces: F*(x)* = 0, por lo tanto:

$$F(peor) = (a \cdot peor - b)^k = 0 \Rightarrow b = -a \cdot peor \quad (7.7)$$

Si *x=mejor*, entonces: F*(x)* = 1, por lo tanto:

$$F(mejor) = (a \cdot mejor - b)^k = 1 \Rightarrow a \cdot mejor - b = 1 \quad (7.8)$$

Reemplazando (7.7) en (7.8) se obtiene:

$$a = \frac{1}{(mejor - peor)} \quad (7.9)$$

Reemplazando (7.9) en (7.7) se obtiene:

$$b = \frac{-peor}{(mejor - peor)} \quad (7.10)$$

Finalmente:

$$F(x) = \left[\frac{x - peor}{mejor - peor} \right]^{k} ; \ k \in]0,+\infty[\qquad (7.11)$$

Al factor k se le denomina grado de la función, y mediante él se pueden crear una gran cantidad de *Funciones para Filtro de Mejoramiento por Prioridad*, entre ellas: *k*=1, función lineal; *k*=2, cuadrática; *k*=3, cúbica; *k*=0.5, raíz cuadrada, etc. En las Figuras N°7.8, N°7.9 y N°7.10 se muestra un ejemplo del funcionamiento de estas funciones.

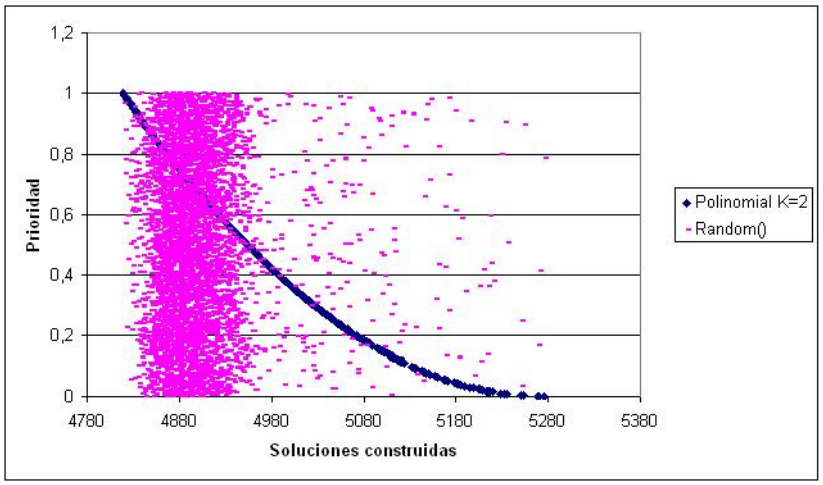


Figura Nº7.8: Filtro de Mejoramiento por Prioridad Cuadrático.

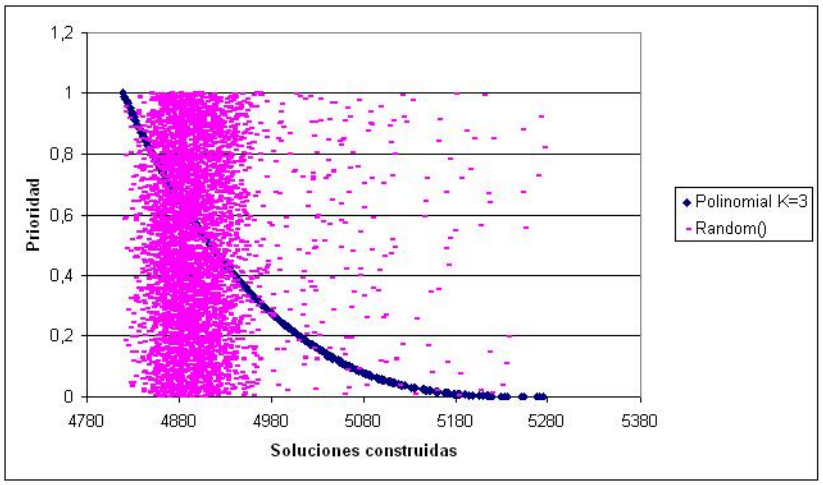


Figura Nº7.9: Filtro de Mejoramiento por Prioridad Cúbico.

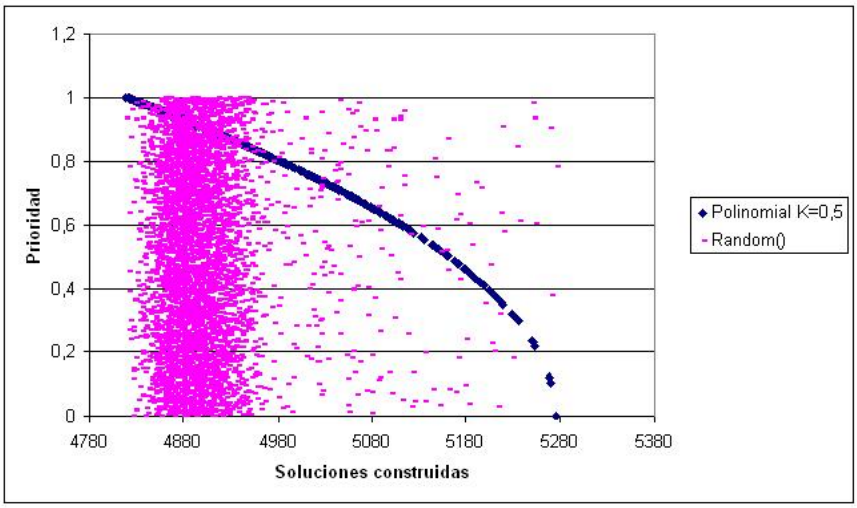


Figura Nº7.10: Filtro de Mejoramiento por Prioridad Raíz Cuadrada.

7.2.3.3 Función F(x) Trigonométrica.

Supongamos una función del tipo: $F(x) = \frac{\cos(ax+b)}{2} + \frac{1}{2}$, con condiciones extremas:

Si *x=peor*, entonces: F*(x)* = 0, por lo tanto:

$$F(peor) = \frac{\cos(a \cdot peor + b)}{2} + \frac{1}{2} = 0 \qquad (7.12)$$

Si *x=mejor*, entonces: F*(x)* = 1, por lo tanto:

$$F(mejor) = \frac{\cos(a \cdot mejor + b)}{2} + \frac{1}{2} = 1 \qquad (7.13)$$

De (7.12) y (7.13) se obtiene que:

$$\cos(a \cdot peor + b) = -1 \Rightarrow a \cdot peor + b = \pi \qquad (7.14)$$

$$\cos(a \cdot mejor + b) = 1 \Rightarrow a \cdot mejor + b = 0 \Rightarrow b = -a \cdot mejor \qquad (7.15)$$

Reemplazando (7.15) en (7.14) se obtiene:

$$a = \frac{\pi}{(peor - mejor)} \qquad (7.16)$$

Reemplazando (7.16) en (7.15) se obtiene:

$$b = \frac{-\pi \cdot mejor}{(peor - mejor)} \tag{7.17}$$

Finalmente:

$$\boxed{F(x) = \frac{1}{2}\cos\left[\frac{\pi(x - mejor)}{peor - mejor}\right] + \frac{1}{2}} \tag{7.18}$$

En la Figura N°7.11 se muestra un ejemplo del funcionamiento de esta Función de *Filtro de Mejoramiento por Prioridad.*

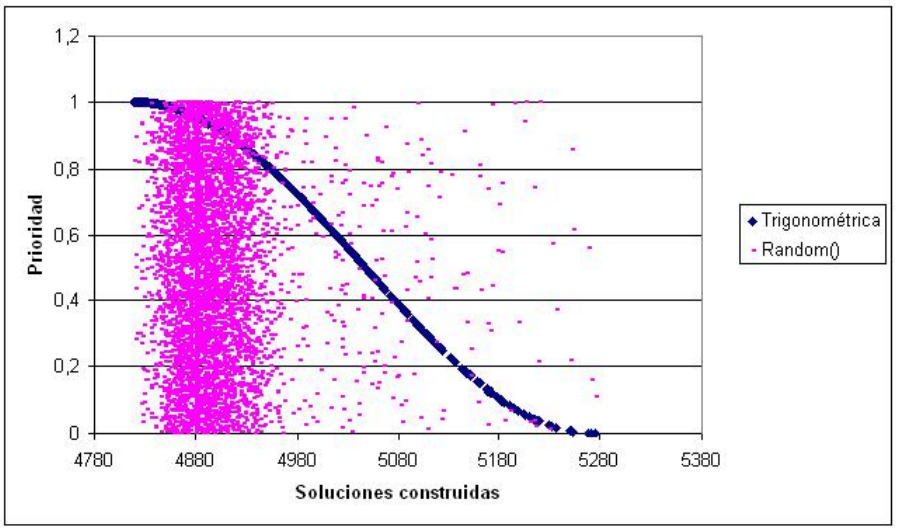


Figura Nº7.11: Filtro de Mejoramiento por Prioridad Trigonométrico.

7.2.3.4 Enfoque clásico de Meta-RaPS.

El enfoque clásico utilizado por Meta-RaPS requiere de la fijación del parámetro *%i*, y la filosofía detrás de él es similar a la propuesta. Para considerar esto, es necesario calcular un *Umbral*, el cual puede ser obtenido mediante la expresión (7.19) o (7.20).

$$Umbral = \left(mejor \cdot (1 - i) + peor \cdot i\right)/100 \tag{7.19}$$

$$Umbral = mejor + (peor - mejor) \cdot i / 100 \tag{7.20}$$

Luego, la función *Filtro de Mejoramiento por Prioridad* para el enfoque clásico se define de la siguiente manera:

$$F(x) = \begin{cases} 1, \ si \ x \le Umbral \\ 0, \ si \ x > Umbral \end{cases} \tag{7.21}$$

Como se puede observar el criterio de filtro clásico no da lugar a la aleatoriedad en la decisión de aceptación/rechazo. De tal manera que todas las soluciones que cumplan la condición *F(x)*=1, pasan por la rutina de mejoramiento local.

En la Figura N°7.12 se muestra un ejemplo del funcionamiento del filtro clásico como una Función de *Filtro de Mejoramiento por Prioridad.*

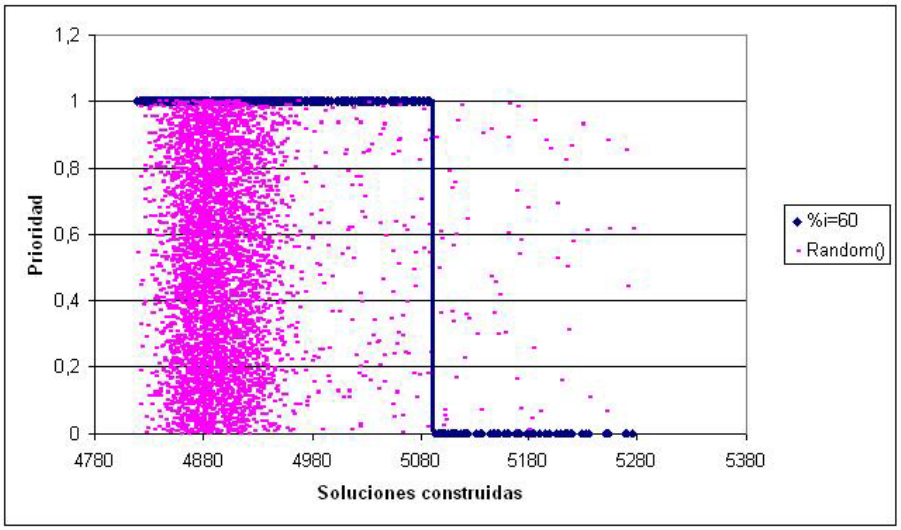


Figura Nº7.12: Filtro de Mejoramiento por Prioridad Clásico.

7.2.4 Comparación Meta-RaPS LACH on-line con Filtro Lineal vs. Trigonométrico.

En esta sección se realiza una comparación de Meta-RaPS LACH con ajuste on-line para las Funciones de *Filtro de Mejoramiento por Prioridad* Lineal y Trigonométrica. Para

ello se utiliza la muestra de problemas de la sección 6.2. Adicionalmente se considera una cantidad de iteraciones I=5000.

Las desviaciones fueron calculadas mediante la siguiente expresión:

$$\delta = \left(\frac{A - B}{A} \right) \times 100\% \qquad (7.22)$$

Donde:

A: resultado generado por la aplicación de Meta-RaPS LACH utilizando ajuste on-line y función de filtro Lineal.

B: resultado generado por la aplicación de Meta-RaPS LACH utilizando ajuste on-line y función de filtro Trigonométrico.

Los resultados obtenidos, agrupados por ambientes de tiempos de procesamiento, se resumen en la Tabla N°7.2.

Tabla N°7.2: Comparación de Funciones de Filtro Lineal vs. Trigonométrica.

	Balanceados	***Proc. dominante***	***Setup dominante***	***Total***
Promedio (δ)	0,269%	0,009%	0,055%	0,111%
Mínimo (min δ)	-4,873%	-0,863%	-0,931%	-4,873%
Máximo (max δ)	11,321%	0,860%	2,557%	11,321%
Cantidad F. Trigonométrico	36 (54,5%)	28 (42,4%)	24 (36,4%)	88 (44,4%)
Cantidad F. Lineal	20 (30,3%)	28 (42,4%)	28 (42,4%)	76 (38,4%)
Cantidad Iguales	10 (15,2%)	10 (15,2%)	14 (21,2%)	34 (17,2%)

En promedio para los problemas de la muestra, el filtro trigonométrico permite encontrar resultados mejores en un 0.111% que el filtro lineal. También el filtro trigonométrico permite obtener resultados mejores o iguales que el filtro lineal, siendo esta cantidad una proporción de 61.6% (44.4% + 17.2%) de los problemas evaluados. Además, al analizar las desviaciones máximas se puede observar que el filtro trigonométrico permite

alcanzar una desviación máxima de 11,321%, lo que indica que el mejor resultado encontrado en comparación con el filtro lineal es un 11,321% mejor; la desviación mínima es de –4.873%, lo que indica que el peor resultado observado es un 4.751% más malo que el encontrado mediante el filtro lineal. Por lo tanto, en promedio, el filtro trigonométrico es mejor que el filtro lineal.

En la Figura Nº7.13 se muestran histogramas de las desviaciones observadas en los problemas de muestra, para los escenarios de "tiempos de procesamiento y setup balanceados" (a), "tiempos de procesamiento dominante" (b) y "tiempos de setup dominante" (c). Adicionalmente, se incluye un histograma con las desviaciones agrupadas, donde se puede observar la mejora lograda mediante la aplicación del filtro trigonométrico por sobre el filtro lineal.

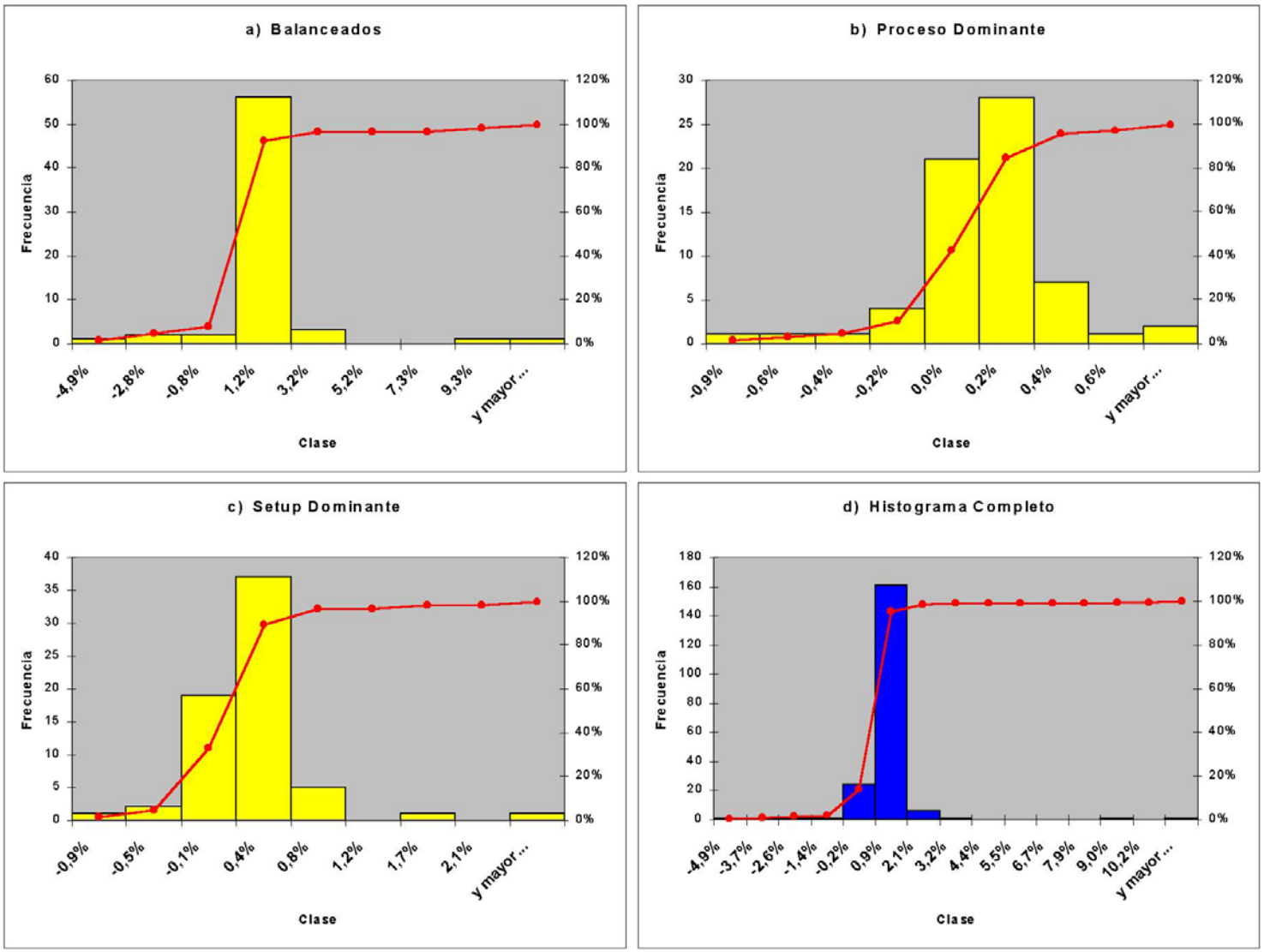


Figura N°7.13: Comparación de Funciones de Filtro Lineal vs. Trigonométrica.

7.2.5 Comparación Meta-RaPS LACH on-line con Filtro Clásico vs. Trigonométrico.

En esta sección se realiza una comparación de Meta-RaPS LACH con ajuste on-line para la función de filtro Trigonométrica versus el enfoque clásico de Meta-RaPS. Para ello se utilizan los resultados obtenidos para la muestra de problemas de la sección 6.2. Adicionalmente se considera una cantidad de iteraciones I=5000, y un porcentaje de mejoramiento de $\%i$=60% para el enfoque clásico.

Las desviaciones fueron calculadas mediante la siguiente expresión:

$$\delta = \left(\frac{A - B}{A} \right) \times 100\% \qquad (7.23)$$

Donde:

A: resultado generado por la aplicación de Meta-RaPS LACH utilizando ajuste on-line y función de filtro Clásico.

B: resultado generado por la aplicación de Meta-RaPS LACH utilizando ajuste on-line y función de filtro Trigonométrico.

Los resultados obtenidos, agrupados por ambientes de tiempos de procesamiento, se resumen en la Tabla N°7.3.

Tabla N°7.3: Comparación Filtro Trigonométrico vs. Filtro Clásico.

	Balanceados	***Proc. dominante***	***Setup dominante***	***Total***
Promedio (δ)	-0,031%	-0,004%	0,074%	0,013%
Mínimo (min δ)	-4,781%	-1,347%	-0,916%	-4,781%
Máximo (max δ)	8,612%	0,941%	2,557%	8,612%
Cantidad Mejores	29 (43,9%)	25 (37,9%)	29 (43,9%)	81 (41,9%)
Cantidad Peores	30 (45,5%)	29 (43,9%)	24 (36,4%)	81 (41,9%)
Cantidad Iguales	7 (10,6%)	12 (18,2%)	13 (19,7%)	32 (16,2%)

En promedio, el filtro trigonométrico permite encontrar resultados mejores en un 0.013% para los problemas de la muestra. También permite obtener la misma cantidad de

resultados mejores que el filtro clásico con *%i*=60%, siendo esta cantidad una proporción de 41,9% de los problemas evaluados, encontrando ambos el mismo resultado en el 16.2% de los problemas de la muestra. Además, al analizar las desviaciones máximas se puede observar que el filtro trigonométrico permite alcanzar un máximo de 8,612%, lo que indica que mediante el filtro trigonométrico se obtuvo un resultado mejor que el encontrado mediante el filtro clásico (*i%*=60%) en un 8,612%; la desviación mínima es de –4.781%, lo que indica que el peor resultado observado es un 4.781% más malo que el encontrado mediante el filtro clásico. A pesar de que los indicadores no son extremadamente favorables, son mejores en promedio que los obtenidos mediante el filtro clásico. Además, en términos de complejidad, el filtro trigonométrico no requiere del parámetro *%i*, por lo tanto, no es necesario que el usuario dedique tiempo a elegir el mejor valor para el parámetro *%i*.

En la Figura Nº7.14 se muestran histogramas de las desviaciones observadas en los problemas de muestra, para los escenarios de "tiempos de procesamiento y setup balanceados" (a), "tiempos de procesamiento dominante" (b) y "tiempos de setup dominante" (c). Adicionalmente, se incluye un histograma con las desviaciones agrupadas, donde se puede observar la mejora lograda mediante la aplicación de la Función de *Filtro de Mejoramiento por Prioridad Trigonométrica.*

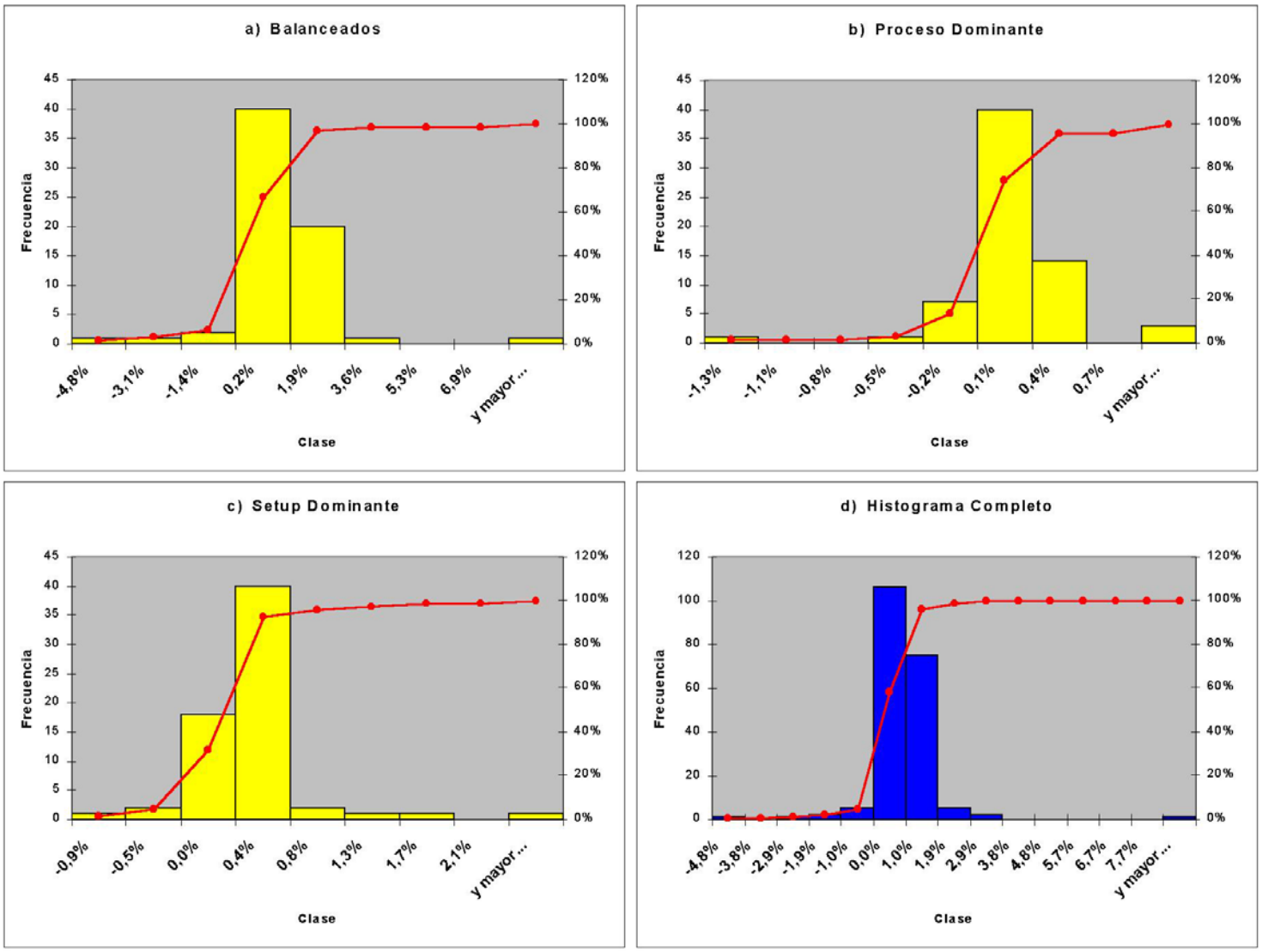


Figura N°7.14: Comparación Filtro Trigonométrico vs. Filtro Clásico.

CAPITULO 8: CONCLUSIONES.

En este capítulo se presentan los resultados y las conclusiones obtenidas de la aplicación de Meta-RaPS LACH.

La aplicación consideró la heurística constructiva LACH descrita en el capítulo 4, la heurística de mejoramiento descrita en el capítulo 5, la técnica de auto-ajuste de parámetros *%p* y *%r*, y el filtro de mejoramiento por prioridad trigonométrico descrito en el capítulo 7. Además se utilizaron todos los problemas propuestos por Rabadi (2005).

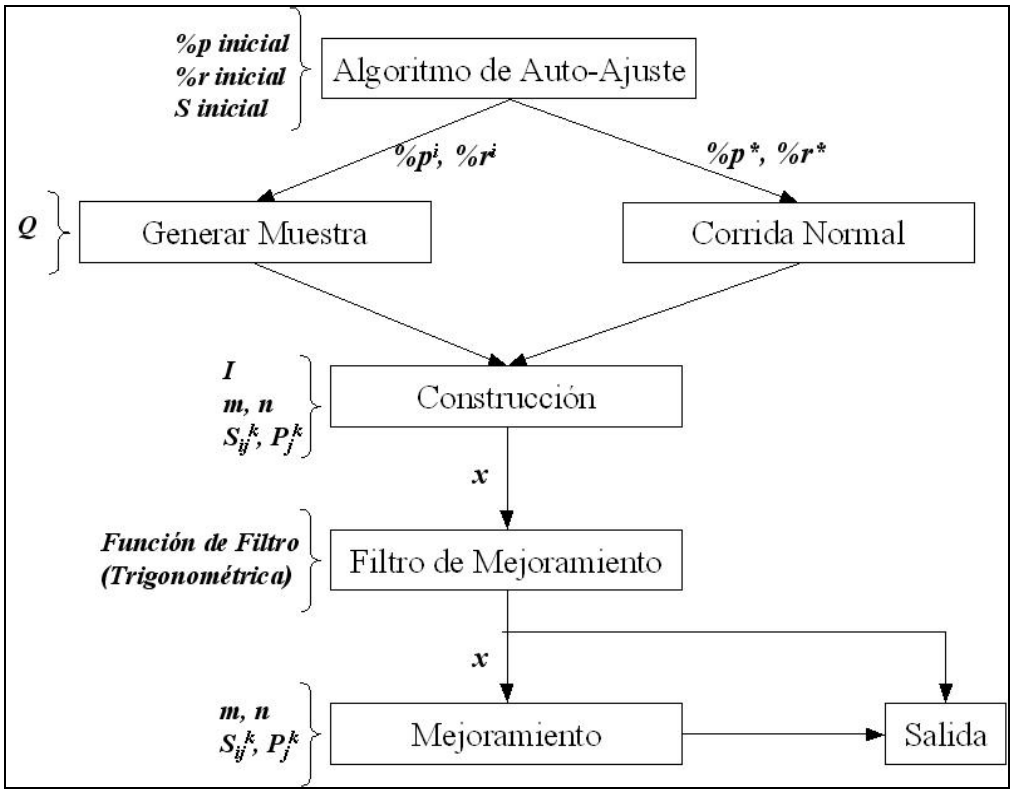


Figura Nº8.1: Esquema de procedimientos de Meta-RaPS LACH.

Continuando con la notación utilizada en los capítulos anteriores, se propone el esquema de la Figura N°8.1 para mostrar y definir el funcionamiento del algoritmo propuesto.

La aplicación de Meta-RaPS LACH es controlada con el Algoritmo de Auto-Ajuste descrito en la sección 7.1, el cual realiza cambios (genera muestras) en los parámetros *%prioridad* y *%restricción* hasta fijar los parámetros en sus valores pseudo óptimos. En esta etapa, el algoritmo utiliza una cantidad de iteraciones de las fijadas por el usuario. Cuando esos valores pseudo óptimos son alcanzados se termina con el proceso de generación de muestras y se utilizan esos parámetros para terminar la cantidad de iteraciones fijadas por el usuario (corrida normal). Siempre que una solución es construida, ya sea en el proceso de Generación de Muestras o en la Corrida Normal, es evaluada por el Filtro de Mejoramiento para decidir si debe pasar por la fase de Mejoramiento. Luego de eso se termina con esa iteración y se continúa con la siguiente.

El Filtro de Mejoramiento utilizado fue el Filtro Trigonométrico, debido a que fue el que reportó los mejores resultados.

La heurística constructiva utilizada en esta aplicación es LACH, específicamente la versión aleatorizada de LACH presentada en el Capítulo 4, donde se mostró que LACH permite obtener en promedio mejores resultados que la heurística constructiva glotona (SAPSL).

La heurística de mejoramiento utilizada fue la presentada en el Capítulo 5, debido a que entregó mejores resultados que la existente en la literatura. Por un lado, genera mejores resultados y además permite ahorrar tiempos de corrida. Por lo que se hace presente que

considerar la estructura del problema en el diseño de heurísticas de mejoramiento permite obtener un mejor diseño.

Recordemos, de los Capítulos 6 y 7, que los resultados obtenidos por la aplicación propuesta (Meta-RaPS LACH) son mejores en promedio que los propuestos por Rabadi *et al.* (2006), considerando una ajuste de parámetros off-line, por lo tanto podemos concluir que el diseño de Meta-RaPS LACH es mejor que el de Meta-RaPS SAPSL.

Si se considera un ajuste on-line de los parámetros *%p* y *%r* los resultados son mejorados, esto se debe a que mediante el ajuste on-line, se produce un ajuste más exacto. Esto comprueba la premisa de que existe una relación directa entre precisión en ajuste de parámetros y calidad de los resultados generados.

De los resultados anteriores, se puede apreciar que utilizar una heurística constructiva visionaria en la fase constructiva de Meta-RaPS permite obtener mejores resultados que si se utiliza una heurística glotona.

Como ya sabemos, la comparación entre Meta-RaPS LACH y Meta-RaPS SAPSL (Rabadi *et al., 2006*), reportó resultados satisfactorios. Las desviaciones para hacer la comparación se calcularon utilizando la siguiente medición:

$$\delta = \left(\frac{Existente - Propuesto}{Existente} \right) \times 100\% \qquad (8.1)$$

Donde:

Existente: resultado de literatura, generado por la aplicación de Meta-RaPS SAPSL (Rabadi *et al., 2006*).

Propuesto: resultado generado por la aplicación propuesta de Meta-RaPS LACH.

De tal manera que desviaciones negativas indican que Meta-RaPS SAP-SL reporta mejores resultados, y en caso contrario Meta-RaPS LACH es quien reporta mejores resultados.

Meta-RaPS LACH fue codificado en C++ y corrido en un Pentium IV CPU 3.00GHz de 512 MB de RAM. Además se utilizaron los siguientes parámetros: *Iteraciones*=5000, *Filtro*=Trigonométrico, *%p inicial*=50%, *%r inicial*=50%, *S inicial*= 40, *Q*=50 y 10 *ciclos*.

Tabla Nº8.1: Desviaciones promedio, escenario de tiempos balanceados.

Máquinas	***2***	***4***	***6***	***8***	***10***	***12***
n=20	*3,562%*	*1,749%*	*8,209%*			
n=40	*1,906%*	*1,407%*	*0,968%*	*-0,107%*	*2,511%*	*0,031%*
n=60	*0,905%*	*1,825%*	*0,741%*	*0,416%*	*-0,108%*	*-0,258%*
n=80	*0,438%*	*1,671%*	*0,757%*	*0,638%*	*-0,296%*	*-0,384%*
n=100	*1,068%*	*0,658%*	*0,235%*	*1,817%*	*1,323%*	*-0,076%*
n=120	*0,548%*	*0,547%*	*0,216%*	*1,667%*	*3,456%*	*-0,204%*
Promedio	1,405%	1,310%	1,854%	0,886%	1,377%	-0,178%

En la Tabla Nº8.1 y Figura Nº8.5 se presentan las desviaciones (δ) promedio agrupadas por cantidad de trabajos y máquinas para el escenario de tiempos de procesamiento y preparación balanceados. En la Figura Nº8.6 se presenta un histograma con las desviaciones (δ) observadas en todos los problemas del escenario de tiempos de procesamiento y preparación balanceados.

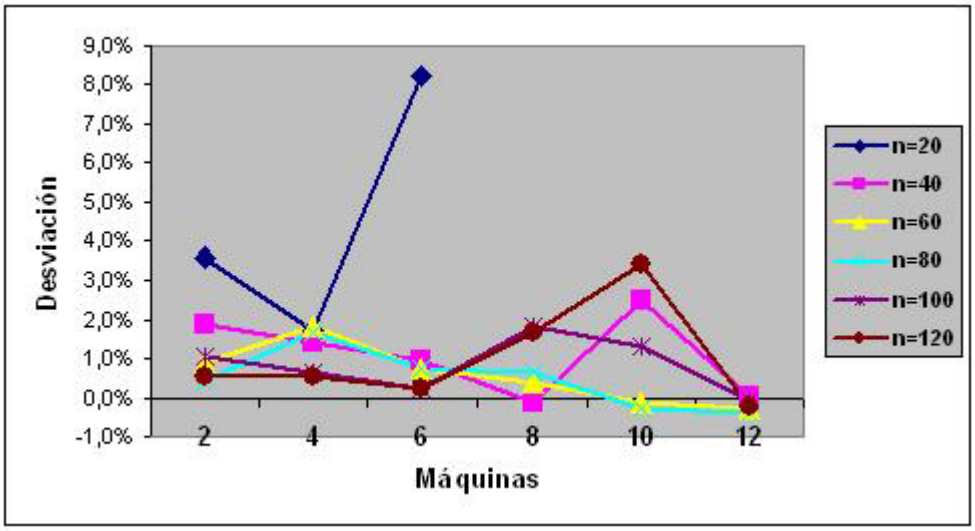


Figura Nº8.5: Desviaciones promedio, escenario de tiempos balanceados.

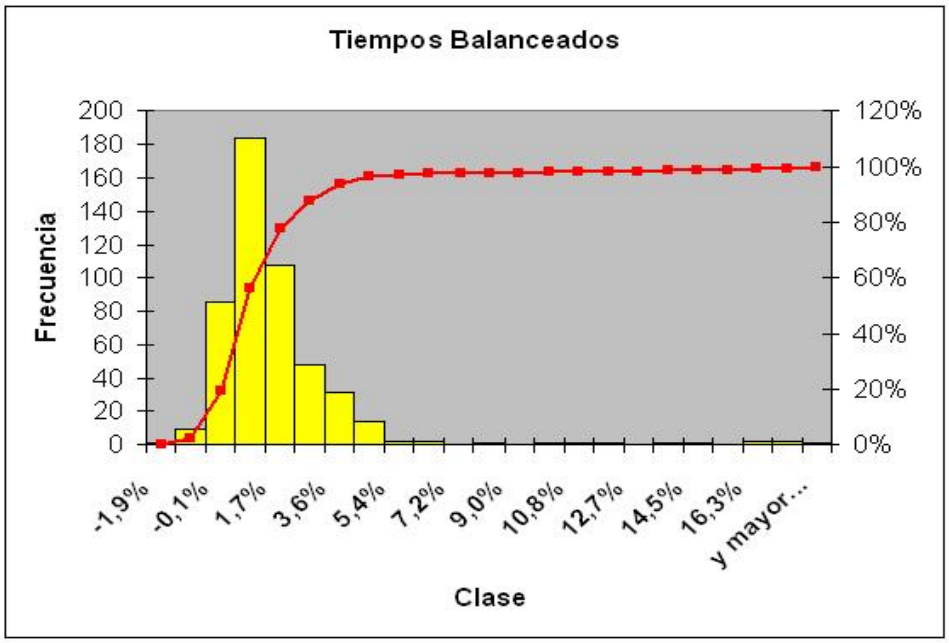


Figura Nº8.6: Histograma para las desviaciones, escenario de tiempos balanceados.

En la Tabla Nº8.2 y Figura Nº8.7 se presentan las desviaciones promedio agrupadas por cantidad de trabajos y máquinas para el escenario de tiempos de procesamiento dominante. En la Figura Nº8.8 se presenta un histograma con las desviaciones observadas en todos los problemas del escenario de tiempos de procesamiento dominante.

Tabla Nº8.2: Desviaciones promedio, escenario de tiempos de proceso dominante.

Máquinas	2	4	6	8	10	12
n=20	1,353%	-0,095%	1,858%			
n=40	1,017%	0,414%	0,106%	0,006%	1,469%	-0,010%
n=60	0,570%	0,626%	0,168%	0,090%	-0,024%	-0,042%
n=80	0,406%	0,860%	-0,055%	0,159%	-0,138%	-0,227%
n=100	0,218%	0,137%	0,021%	0,852%	0,707%	-0,074%
n=120	0,181%	0,186%	-0,043%	1,022%	1,822%	-0,108%
Promedio	0,624%	0,355%	0,343%	0,426%	0,767%	-0,092%

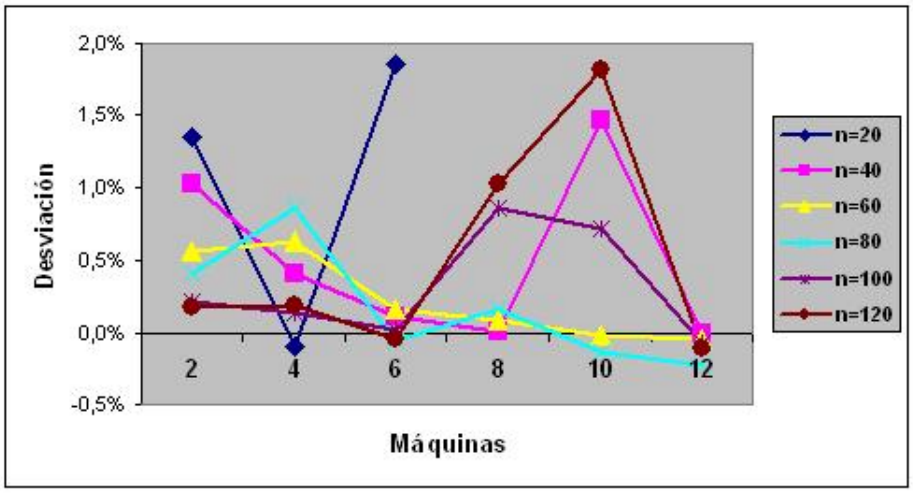


Figura Nº8.7: Desviaciones promedio, escenario de tiempos de proceso dominante.

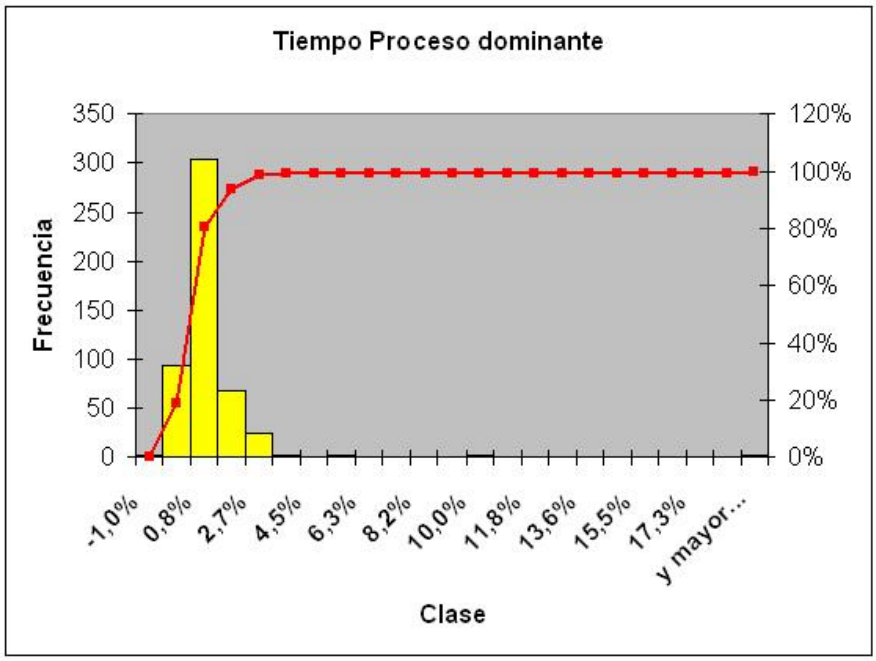


Figura Nº8.8: Histograma de desviaciones, escenario de tiempos de proceso dominante.

En la Tabla Nº8.3 y Figura Nº8.9 se presentan las desviaciones promedio agrupadas por cantidad de trabajos y máquinas para el escenario de tiempos de preparación dominante. En la Figura Nº8.10 se presenta un histograma con las desviaciones observadas en todos los problemas del escenario de tiempos de preparación dominante.

Tabla Nº8.3: Desviaciones promedio, escenario de tiempos de setup dominantes.

Máquinas	***2***	***4***	***6***	***8***	***10***	***12***
n=20	*0,733%*	*0,326%*	*0,126%*			
n=40	*0,773%*	*0,201%*	*0,310%*	*-0,021%*	*1,512%*	*0,027%*
n=60	*0,551%*	*0,671%*	*0,253%*	*0,009%*	*0,088%*	*-0,106%*
n=80	*0,358%*	*0,794%*	*0,039%*	*0,322%*	*-0,222%*	*-0,083%*
n=100	*0,319%*	*0,130%*	*0,014%*	*0,822%*	*0,824%*	*-0,086%*
n=120	*0,235%*	*0,281%*	*-0,050%*	*1,101%*	*1,925%*	*-0,172%*
Promedio	0,495%	0,400%	0,115%	0,446%	0,825%	-0,084%

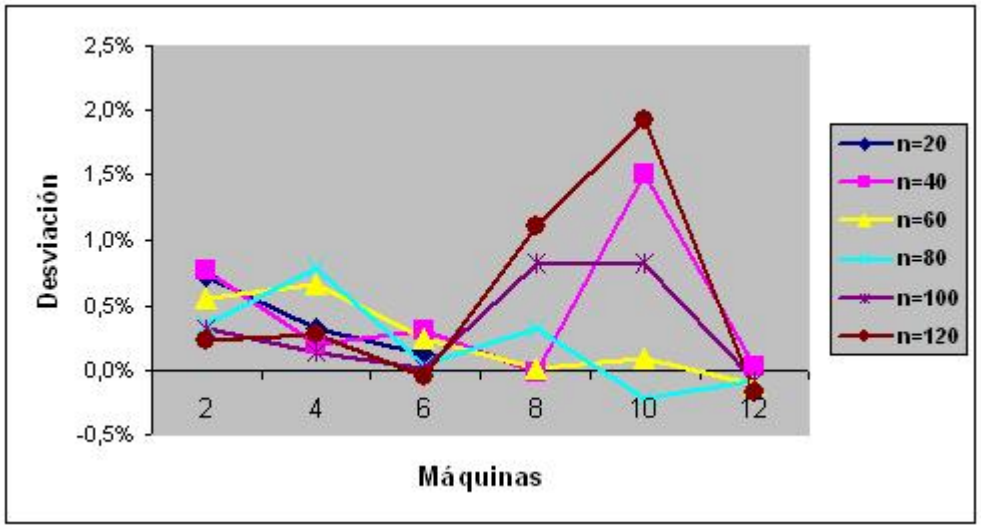


Figura Nº8.9: Desviaciones promedio, escenario de tiempos de setup dominantes.

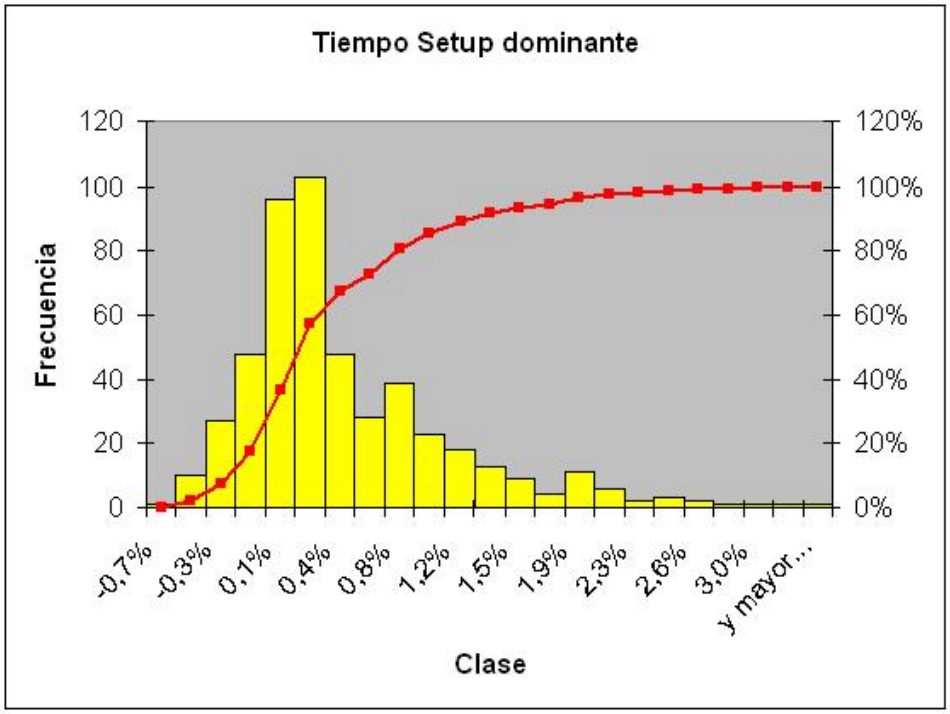


Figura Nº8.10: Histograma para las desviaciones, escenario de tiempos de setup dominante.

En la Tabla Nº8.4 y Figura Nº8.11 se presentan las desviaciones promedio de todos los escenarios agrupadas por cantidad de trabajos y máquinas. En la Figura Nº8.12 se presenta un histograma con las desviaciones observadas en todos los problemas propuestos por Rabadi (2005).

En las Figuras Nº8.5, Nº8.7, Nº8.9 y Nº8.11 se puede observar un comportamiento similar. Además se puede apreciar que para los problemas de 12 máquinas (en todos lo escenarios), Meta-RaPS LACH no logró superar en promedio los resultados obtenidos por Meta-RaPS SAPSL. Al observar las desviaciones promedio asociadas a los problemas de 12 máquinas se puede observar que en promedio son pequeñas en comparación con las demás. En la Figura Nº8.11 se agregó una desviación promedio agrupada solo por cantidad

de máquinas, y se puede apreciar que la calidad de los resultados obtenidos por Meta-RaPS LACH disminuye para los problemas de 12 máquinas. Este es un problema cuya causa se puede deber a la estructura de vecindario definida en la heurística de mejoramiento. Sin embargo, es necesario desarrollar un estudio formal para determinar la causa del problema. Por otro lado, a pesar de este problema, es posible observar que la calidad de los resultados obtenidos por Meta-RaPS LACH es superior a los resultados encontrados en la literatura.

Tabla Nº8.4: Desviaciones promedio, agrupadas por cantidad de trabajos y máquinas.

Máquinas	***2***	***4***	***6***	***8***	***10***	***12***
n=20	*1,412%*	*0,495%*	*2,548%*			
n=40	*0,924%*	*0,506%*	*0,346%*	*-0,031%*	*1,373%*	*0,012%*
n=60	*0,507%*	*0,781%*	*0,291%*	*0,129%*	*-0,011%*	*-0,101%*
n=80	*0,301%*	*0,831%*	*0,185%*	*0,280%*	*-0,164%*	*-0,173%*
n=100	*0,401%*	*0,232%*	*0,068%*	*0,873%*	*0,714%*	*-0,059%*
n=120	*0,241%*	*0,254%*	*0,031%*	*0,947%*	*1,801%*	*-0,121%*
Promedio	0,631%	0,516%	0,578%	0,440%	0,743%	-0,089%

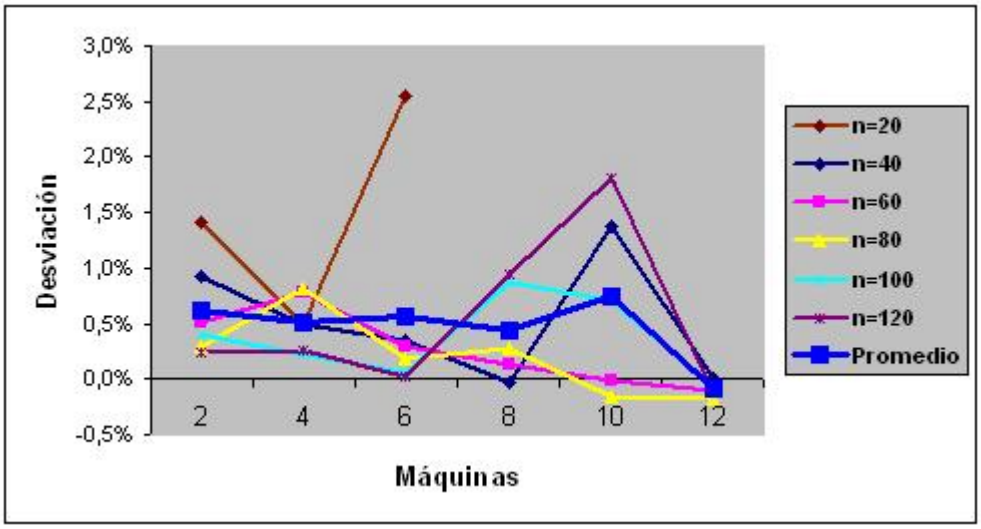


Figura Nº8.11: Desviaciones promedio, agrupadas por cantidad de trabajos y máquinas.

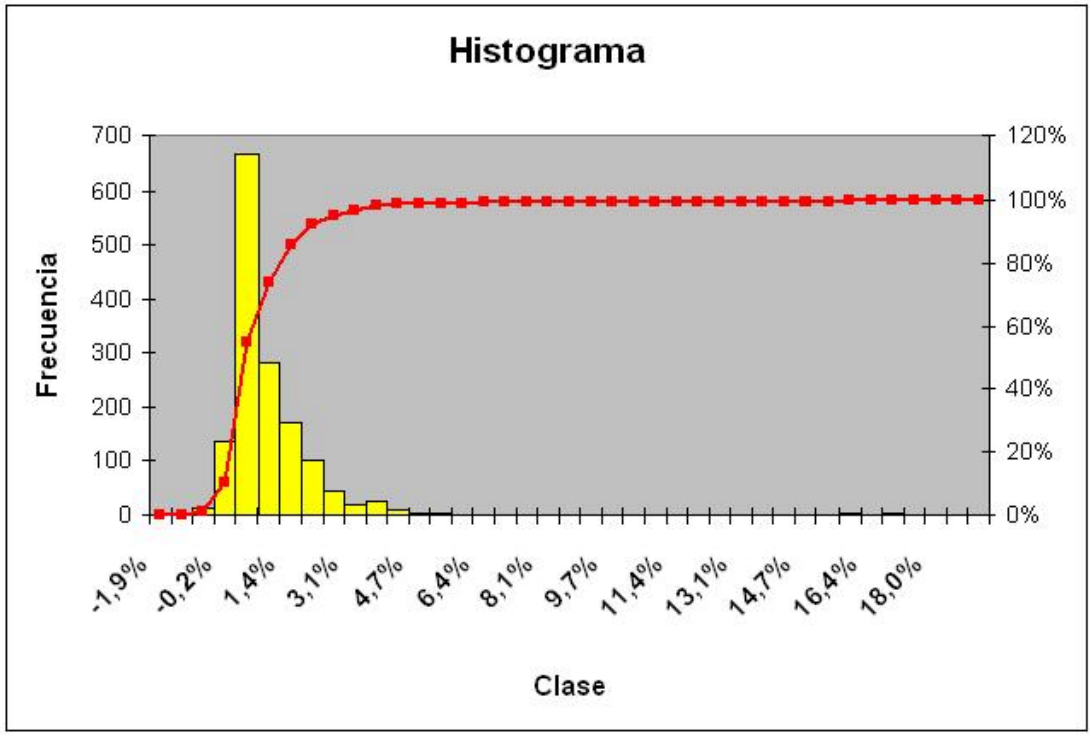


Figura Nº8.12: Histograma para las desviaciones, todos los escenarios.

En la Tabla Nº8.5 se presenta un resumen con las desviaciones promedio y la cantidad de problemas para los cuales cada aplicación encontró mejor solución. En dicha tabla se puede observar que en promedio Meta-RaPS LACH obtuvo mejores resultados que los que encontró Meta-RaPS SAPSL (en la literatura) por una magnitud de 0.639%. Se observa que las desviaciones mínimas y máximas alcanzan valores de –1,093% y 19,143% respectivamente, lo que indica que el peor resultado encontrado por Meta-RaPS LACH es un 1,093% más malo que el encontrado por Meta-RaPS SAPSL, por otro lado el mejor resultado encontrado por Meta-RaPS LACH es mejor que el encontrado por Meta-RaPS SAPSL en un 19,143%. Lo cual confirma que a pesar del problema que se genera en los problemas de 12 máquinas, la diferencia entre ambas metaheurísticas es muy pequeña.

En conclusión, el problema que se presenta en Meta-RaPS LACH para los problemas de 12 máquinas es producto de que las soluciones construidas por LACH aleatorizado son difíciles de mejorar con la estructura de vecindario utilizada.

También se puede observar que del total de problemas evaluados (1485), Meta-RaPS LACH encontró mejores resultados en 1012 instancias (proporción de 68.15% del total de problemas evaluados); por otro lado Meta-RaPS SAPSL encontró mejores resultados en 355 instancias (proporción de 23.9% del total de problemas evaluados); encontrando ambas aplicaciones el mismo resultado en 118 ocasiones (proporción de 7,95% del total de problemas evaluados).

Tabla Nº8.5: Desviaciones promedio, agrupadas por escenario.

	Balanceados	***Proc. dominante***	***Setup dominante***	***Total***
Promedio (δ)	1,147%	0,407%	0,364%	0,639%
Mínimo (min δ)	-1,903%	-1,006%	-0,669%	-1,903%
Máximo (max δ)	18,121%	19,143%	3,358%	19,143%
Mejores	359	323	330	1.012
Peores	105	128	122	355
Iguales	31	44	43	118

Para que el funcionamiento de Meta-RaPS LACH sea aceptable, debe ser capaz de resolver los problemas en tiempos de corrida factibles. Los tiempos de respuesta, o tiempos de corrida, son muy importantes para el funcionamiento de las metaheurísticas, ya que el concepto fundamental para el uso de una metaheurística es la necesidad de obtener respuestas rápidas. Razón por la cual el usuario esta dispuesto a obtener resultados subóptimos.

Meta-RaPS LACH fue codificado en C++ y corrido en un Pentium IV CPU 3.00GHz de 512 MB de RAM.

Para cuantificar los tiempos de corrida de Meta-RaPS LACH, se agruparon los problemas de prueba de acuerdo a su tamaño, es decir, por cantidad de trabajos y máquinas, y se promediaron sus tiempos de corrida. Los tiempos de corrida promedio se muestran en la Tabla Nº8.6. Donde se puede observar que los tiempos de corrida promedio para los problemas de mayor tamaño (120 trabajos en 12 máquinas) no superan los 3 minutos. Por otra parte, los problemas de tamaño más modesto (20 trabajos en 2 máquinas) no superan los 3 segundos.

Tabla Nº8.6: Tiempos de corrida promedio (seg.)

m|n	**20**	**40**	**60**	**80**	**100**	**120**
2	2,32	7,68	17,07	30,39	47,86	70,16
4	3,11	9,07	19,61	33,23	49,86	73,03
6	3,95	12,86	26,06	39,42	61,57	89,42
8		16,24	31,63	51,07	76,09	106,47
10		18,65	39,43	62,68	99,68	139,31
12		23,26	52,51	83,92	128,88	176,20

En la Figura Nº8.13 se muestran gráficamente los tiempos de corrida promedio.

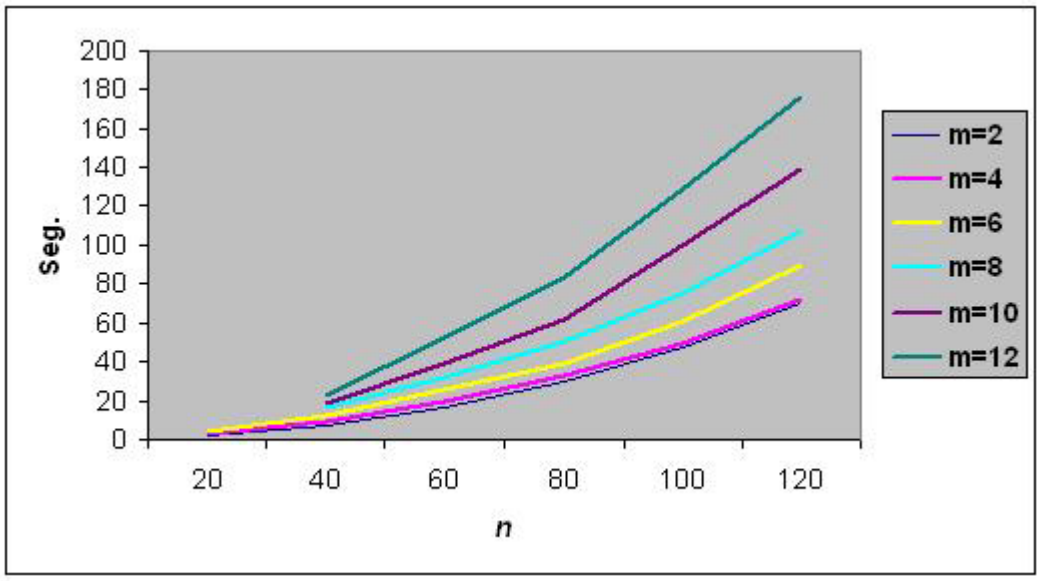


Figura Nº8.13: Tiempos de corrida promedio (seg.)

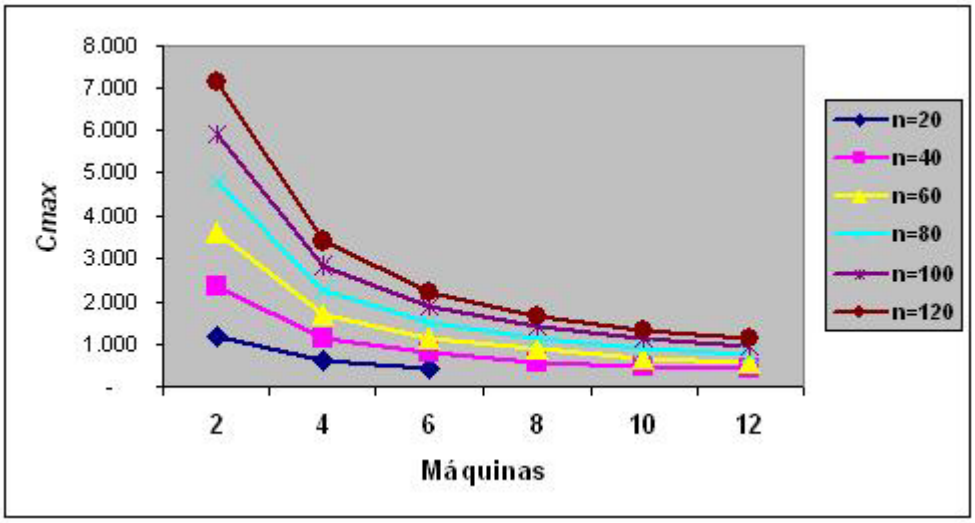
Los resultados obtenidos por Meta-RaPS LACH son en promedio mejores que los que entrega Meta-RaPS SAPSL. Sin embargo es importante resaltar la disminución de calidad que tiene para resolver los problemas de 12 máquinas de la librería propuesta por Rabadi (2005) en comparación con Meta-RaPS SAPSL, y la imposibilidad de resolver problemas de tamaño pequeño, es decir, aquellos problemas donde no se cumple la relación $n \geq 3m$ (restricción de diseño establecida por la heurística constructiva LACH).

Desde el punto de vista del ajuste de parámetros, Meta-RaPS LACH genera una ahorro de tiempo y esfuerzo en la tarea de ajuste de parámetros. Por lo que es una herramienta de mayor utilidad desde el punto de vista práctico.

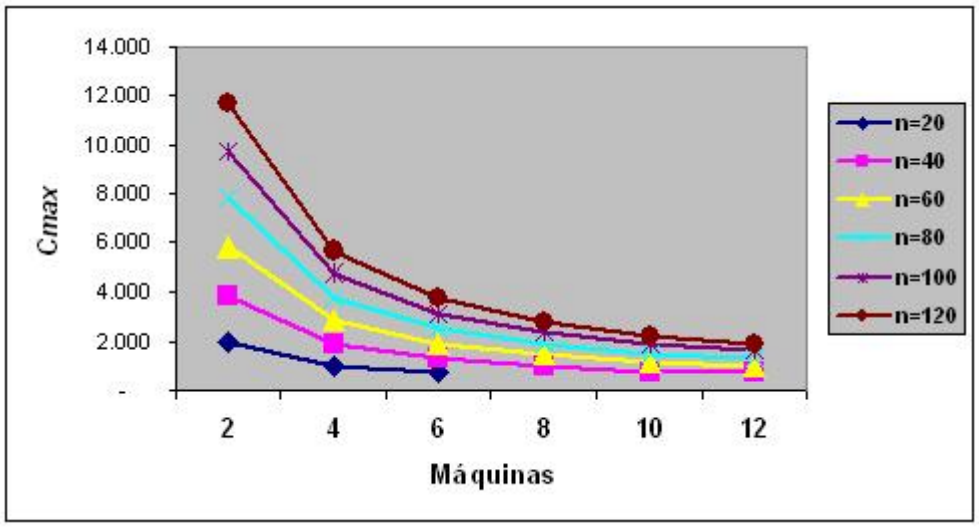
Además los tiempos de corrida son aceptables, y varían según el tamaño del problema. Desde tres segundos para los problemas más pequeños (programar 20 trabajos en 2 máquinas), hasta tres minutos para los problemas más grandes (programar 120 trabajos en 12 máquinas).

A raíz de lo anterior, fue posible cumplir con el objetivo principal de desarrollar una metodología eficiente y eficaz para resolver el problema $R_m|S_{ijk}|C_{max}$, que cumple con las características deseadas: facilidad de aplicación, tiempos de respuesta razonables y buena calidad de las soluciones generadas.

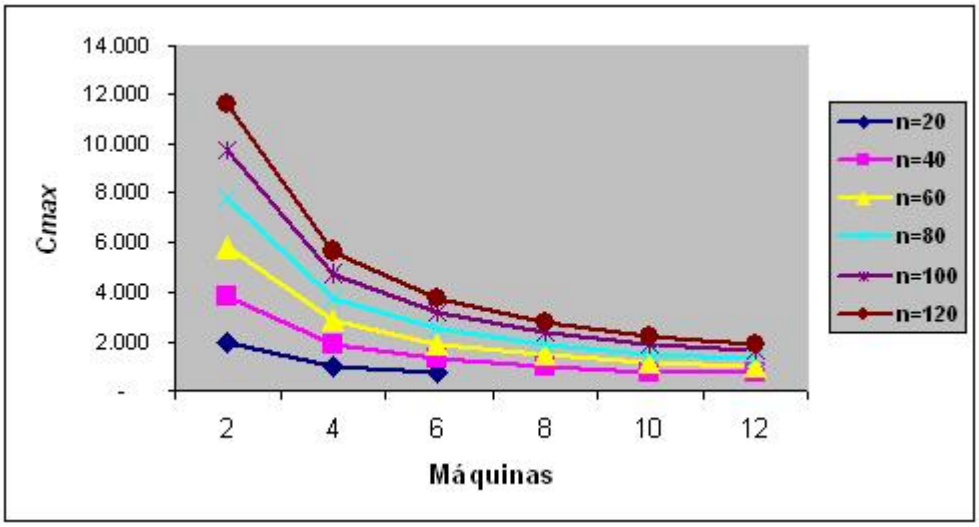
Según Rabadi *et al.*(2006), el makespan promedio disminuye a medida que aumenta la cantidad de máquinas del problema. En este estudio observamos el mismo comportamiento. En las Figuras Nº8.14, Nº8.15 y Nº8.16 se muestra este comportamiento, agrupando los problemas por escenarios. Este es un resultado obvio, ya que al considerar una mayor cantidad de máquinas, y una cantidad fija de trabajos, la cantidad de trabajos que realiza cada máquina disminuye a medida que se consideran más máquinas.



Figuras Nº8.14: C_{max} vs. cantidad de máquinas, escenario de tiempos balanceados.



Figuras Nº8.15: C_{max} vs. cantidad de máquinas, escenario de tiempos de proceso dominante.



Figuras Nº8.16: C_{max} vs. cantidad de máquinas, escenario de tiempos de setup dominante.

FUTURAS INVESTIGACIONES.

Dentro de las futuras investigaciones relacionadas con el estudio se consideran, en primer lugar, la creación de problemas de tamaño más grande, con respecto a la cantidad de máquinas (14, 16, etc.). Para poder determinar la sensibilidad del tamaño del problema con la estructura de vecindario utilizada en las heurísticas de mejoramiento. Esto sería de importancia para poder diseñar estructuras de vecindario que permitan mejorar en mayor cantidad las buenas soluciones construidas por LACH aleatorizado, y con ello acercarlas a la solución óptima.

Además de la creación de problemas de mayor tamaño, sería importante crear una nueva librería de problemas $R_m|S_{ijk}|C_{max}$, añadiendo algunas características a los problemas, como por ejemplo correlación entre los datos. Para poder evaluar el desempeño de Meta-RaPS LACH bajo esa condiciones.

Otra investigación factible de realizar, y muy importante para Meta-RaPS es aplicar las técnicas de ajuste de parámetros diseñadas en otros estudios y aplicaciones de Meta-RaPS.

Por último sería muy importante buscar una aplicación en la industria para poder evaluar las mejoras obtenidas en un sistema productivo por la aplicación de Meta-RaPS LACH.

REFERENCIAS

ADAMOPOULOS, G. y PAPPIS, C. (1998). *Scheduling under a common due-date on parallel unrelated machines*. European Journal of Operational Research, 105, 494-501.

AIEX, R.M., BINATO, S. y RESENDE, M.G.C. (2003). *Parallel GRASP with path-relinking for job shop scheduling*. Parallel Computing, 29,393-430.

AL-SALEM, A. (2004). *Scheduling to minimize máximo tiempo de completación (makespan) on unrelated parallel machines with sequence dependent setup times*. Engineering Journal of the University of Qatar, Vol. 17, pp 177-188.

ANAGNOSTOPOULOS, G. y RABADI, G. (2002). *A simulated annealing algorithm for the unrelated parallel machine scheduling problem*. Proceedings of the World Automation Congress 2002, Orlando, FL, June 09-13.

ARCUS, A.L. (1966). *COMSOAL: A computer method of sequencing operations for assembly lines*. International Journal of Production Research, 4, 259-277.

ARMACOST, R. y AL-SALEM, A. (1999). *Unrelated Parallel Machine Scheduling with Setup Times and Machine Eligibility Restrictions*. Proceedings of the eighth Industrial Engineering Research Conference, (CD-ROM), Phoenix, AZ, May 23-24, 1999.

ARMENTANO, V. y YAMASHITA, D. (2000). *Tabu search for scheduling on identical parallel machines to minimize mean tardiness*. Journal of Intelligent Manufacturing, 11, 453-460.

CHENG, T. y SIN, C. (1990). *A state of the art review of parallel machine scheduling research*. European Journal of Operation Research, 47, 271-292.

DEPUY, G.W., MORAGA, R.J. y WHITEHOUSE, G.E. (2005[1]). *"Meta-RaPS: A Simple And Effective Approach For Solving The Traveling Salesman Problem"*. Transportation Research Part E: Logistics and Transportation Review, Vol. 41, No. 2, pp. 115-130.

DEPUY G.W., WHITEHOUSE, G.E. y MORAGA, R.J. (2005[2]). *Un enfoque Meta-heurístico para el problema de programación de proyectos con recursos limitados,* Revista del Instituto Chileno de Investigación Operativa, Vol. 7, No. 2.

DEPUY, G.W., WHITEHOUSE, G.E. y MORAGA, R.J. (2003). *Using the Meta-Raps Approach to Solve Combinatorial Problems*, in review by Computers y Industrial Engineering, submitted 2003.

DEPUY, G.W., WHITEHOUSE, G.E. y MORAGA, R.J. (2002). *Using the Meta-Raps Approach to Solve Combinatorial Problems*, Proceedings of the 2002 Industrial Engineering Research Conference, May 19-21, Orlando, Florida.

DEPUY, G. y WHITEHOUSE, G. (2001). *A Simple and Effective Heuristic for the Multiple Resource Allocation Problem*. International Journal of Production Research, 32, 4, 24-31.

DEPUY G., WHITEHOUSE, G. y MORAGA, R. (2001). *"Meta-RaPS: A Simple and Efficient Approach for Solving Combinatorial Problems"*, Proceedings of 29º International Conference on Computers and Industrial Engineering, Montreal, Canada.

DEPUY, G. y WHITEHOUSE, G. (2000). *Applying the COMSOAL computer heuristic to the constrained resource allocation problem*. Computers and Industrial Engineering, 38, 413-422.

DEPUY G., WHITEHOUSE, G., SCHULMAN, N., ALMAZID, A. y MORAGA, R. (2000). *"Modification of the COMSOAL Approach to Solve Various Combinatorial Problems"*, Proceedings of 4º International Engineering Design and Automation Conference, USA.

DHAENENS-FLIPO, C. (2001). *A bicriterion approach to deal with a constrained single-objetive problem.* International Journal of Production Economics, Vol. 74, 93-101.

FEO, T.A., SARATHY, K. y MCGAHAN, J. (1996). *A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties*. Computers & Operations Research, 23, 881-895.

FEO, T. y RESENDE, M. (1995). *Greedy randomized adaptive search procedures*. Journal of Global Optimization, 6,109-133.

FEO, T.A., VENKATRAMAN, K. y BARD, J.F. (1991). *A GRASP for a difficult single machine scheduling problem*. Computers & Operations Research, 18, 635-643.

FEO, T. y RESENDE, M. (1989). *A probabilistic heuristic for a computationally difficult set covering problem*. OR letters 8, 67-71.

FRANCA, P., GENDREAU, M., LAPORTE, G. y MÜLLER, F. (1996). *A tabú search heuristic for the multiprocessor scheduling problem with sequence dependent setup times*. International Journal of Production Economics, Vol. 43, 79-89.

GAREY, M.R. y JOHNSON, D.S. 1979. *Computers and intractability: a guide to the theory of NP-Completeness*. W.H. Freeman and Company.

GRAHAM, R.L., LAWLER, E.L., LENSTRA, J.K. y RINNOOY KAN, A.H.G. (1979). *Optimization and approximation in deterministic sequencing and scheduling: a survey.* Annals of discrete mathematics.

GRAVES, S.C. (1981). *A review of Production Scheduling*, Operation Research, 29, 646-675.

GUINET, A. (1991). *Textile production systems: a succession of non-identical parallel processor shops*. Journal of the Operational Research Society, Vol. 42, Nº 8, 655-671.

GUINET, A. (1993). *Scheduling sequence-dependent jobs on identical parallel machines to minimize completion time criteria.* International Journal of Production Research, Vol. 31, Nº7, 1579-1594.

HAMAD, A., SANUGI, B. y SALLEH, S. (2003). *A Neural Network Model for the Common Due Date Job Scheduling on Unrelated Parallel Machines.* International Journal Computer Mathematics, Vol. 80, No. 7, pp. 845–851.

HELAL, M. y HOSNI, Y. (2003). *A Tabu Search Approach for the Non-identical Parallel-Machines Scheduling Problem With Sequence-Dependent Setup Times.* Proceedings of the 2003 Industrial Engineering Research Conference, Portland, OR, May 18-21.

HERRMANN, J., PROTH, J. y SAUER, N. (1997). *Heuristics for unrelated machine scheduling with precedence constraints.* European Journal of Operational Research, 102, 528-537.

HOROWITZ, E. y SAHNI, S. (1976). *Exact and Approximation Algorithms for Scheduling Nonidentical Processors,* Journal of the Association for Computing machinery, 23, 317-327.

HURINK, J. y KNUST, S. (2001). *List scheduling in a parallel machine environment with precedence constraints and setup times*. Operations Research Letters, 29, 231-239.

IBARRA, O. y KIM, C. (1977). *Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors*, Journal of the Association for Computing Machinery, 24, 280-289.

KIM, D., KIM, K., JANG, W. y CHEN, F. (2002). *Unrelated parallel machine scheduling with setup times using simulated annealing.* Robotics and Computer Integrated Manufacturing, 18, 223-231.

LAGUNA, M. y GONZÁLEZ, J.L. (1991*). A search heuristic for just-in-time scheduling in parallel machines.* Journal of Intelligent Manufacturing, 2, 253-260.

LAN, G., DEPUY, G. y WHITEHOUSE, G. (2005). *An Effective and Simple Heuristic for the Set Covering Problem*. Aceptada para publicación en European Journal of Operational Research.

LAWLER, E.L., LENSTRA, J.K., RINNOOY KAN, A.H.G. y SHMOYS, D.B. (1993). *Sequencing and Scheduling: Algorithms and complexity,* Handbooks in Operations Research and Management Science 4, Logistics of Production and Inventory, North Holland, Amsterdam, 445-524.

LEE, Y. y PINEDO, M. (1997*). Sheduling jobs on parallel machines with sequence-dependent setup times.* European Journal of Operational Research, 100, 464-474.

LIAW, CH., LIN, Y., CHENG, CH. y CHEN, M. (2003). Scheduling unrelated parallel machines to minimize total weighted tardiness. Computers & Operational Research, 30, 1777-1789.

LIN, C., WONG, C. y YEUNG, Y. (2002). *Heuristic approaches for a scheduling problem in the plastic molding department of an audio company*. Journal of heuristic, 8, 515-540.

MARTI, R. y MORENO, M. (2003). *MultiStart Methods. Inteligencia Artificial,* Revista Iberoamericana de Inteligencia Artificial, 19, pp. 49-60.

MICHALEWICZ, Z., EIBEN, A. y HINTERDING, R. (2002). *Parameter Setection, in Evolutionary Optimization.* R. Sarker, M. Mohammadian y X. Yao (eds.), Kluwer Academic Publisher, 279-306.

MIN, L. y CHENG, L. (1999). *A genetic algorithm for minimizing tha máximo tiempo de completación (makespan) in the case of scheduling identical parallel machines.* Artificial Intelligence in Engineering, 13, 399-403.

MOKOTOFF, E. y JIMENO, J. (2002). *Heuristics based on partial enumeration for the unrelated parallel processor scheduling problem.* Annals of Operations Research, 117, 133-150.

MOKOTOFF, E. (2001). *Parallel machine scheduling problems: A survey.* Asia-Pacific Journal of Operational Research, 18, 193-242.

MORAGA, R.J., DEPUY, G.W. y WHITEHOUSE, G.E. (2005). *Meta-RaPS Approach for the 0-1 Multidimensional Knapsack Problem,* Computers and Industrial Engineering, Vol. 48, No. 2, pp 83-96.

MORAGA, R.J., WHITEHOUSE, G.E., y DEPUY, G.W. (2004). *"Meta-RaPS: Un Enfoque de Solución Eficaz para Problemas Combinatorios"*, Revista de Ingeniería Industrial, Vol. 2, No. 1, pp. 5-18.

MORAGA, R.J. (2004). *Meta-RaPS Approach for the Unrelated Parallel Machine Problem with Sequence-Dependant Setup Times,* In Proceedings of the 2004 Industrial Engineering Research Conference, Houston, TX, May 15-19.

MORAGA, R., WHITEHOUSE, G., DEPUY, G., NEYVELI, B. y KUTTUVA, S. (2002), *"Solving the Vehicle Routing Problem Using the Meta-RaPS Approach,* 29º International Conference on Computers and Industrial Engineering, Nov. 1-3, Montreal, Canada, pp- 76-81.

MUÑOZ, F., MORAGA, R., y BAESLER, F. (2005). *Look-Ahead Constructive Heuristic for the Unrelated Parallel Machine Problem with Sequence Dependent Setup Times.* Proceedings of the 2005 Industrial Engineering Research Conference, Atlanta, GA, May 14-18.

PINEDO, M. 2002. *Scheduling: Theory, Algorithms, and Systems*, Second Edition, Prentice-Hall, New Jersey, USA.

RABADI, G., MORAGA, R. y AL-SALEM, A. (2006). *Heuristics for the Unrelated Parallel Machine Scheduling Problem with Setup Times*. Journal of Intelligent Manufacturing, 17, 85-97.

RABADI, G. (2005). Librería de problemas de prueba par el problema de máquinas paralelas no-relacionadas con tiempos preparación dependientes de la secuencia. Disponible en http://www.SchedulingResearch.com.

RABADI, G., MOLLAGHASEMI, M. y ANAGNOSTOPOULOS, G. (2004). *A new Branch-and-Bound for the Early/Tardy problem with a common due-date an sequence-dependent setup time.* Computers and Operations Research, Vol. 31, 1727-1751.

RESENDE, M. y GONZÁLEZ, J. (2003*). GRASP: Greedy Randomized Adaptive Search Procedures,* Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial. No.19, pp. 61-76.

ROJANASOONTHON, S. y BARD, J.F. (2005) *A GRASP for parallel machine scheduling with time windows*. To appear in, v17, n 1, INFORMS Journal on Computing.

SRIVASTAVA, B. (1998). *An effective heuristic for minimizing máximo tiempo de completación (makespan) on unrelated parallel machines.* Journal of the Operational Research Society, 49, 886-894.

VREDEVELD, T. y HURKENS, C. (2002). *Experimental comparison of approximation algorithms for scheduling unrelated parallel machines*. Journal on Computing, vol. 14, No 2, 175-189.

WENG, M., LU, J. y REN, H. (2001). *Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective*. International Journal of Production Economics, 70, 215-226.

YALAOUI, F. y CHU, CH. (2003*). An efficient heuristic approach for parallel machine scheduling with job splitting an sequence-dependent setup times*. IIE Transactions, 35, 183-190.

YU, L., SHIH, H., PFUND, M., CARLYLE, W. y FOWLER, J. (2002*). Scheduling of unrelated parallel machines an application to PWB manufacturing*. IIE Transactions, 34, 921-931.

ANEXO A: RESULTADOS DE FIJACIÓN OFF-LINE DE *%P* Y *%R*.

		Proceso dominante		*Setup dominante*		*Balanceados*	
Máquinas	***Trabajos***	***%p***	***%r***	***%p***	***%r***	***%p***	***%r***
2	20	25	15	25	10	30	15
	40	55	10	90	40	60	10
	60	85	15	85	10	75	10
	80	80	10	85	10	75	10
	100	85	10	90	10	90	10
	120	90	10	90	10	90	10
4	20	10	10	20	10	10	10
	40	55	10	50	10	85	15
	60	85	15	50	10	70	10
	80	80	10	85	10	90	10
	100	85	10	90	10	90	10
	120	90	10	90	10	90	10
6	20	25	10	35	10	15	10
	40	10	10	60	10	55	10
	60	60	10	60	10	35	10
	80	75	10	80	10	80	10
	100	90	10	85	10	80	10
	120	90	10	85	10	85	10
8	40	50	10	75	10	45	10
	60	75	10	80	10	50	10
	80	90	10	80	10	80	10
	100	90	10	90	10	80	10
	120	80	10	90	10	85	10
10	40	30	10	35	10	45	10
	60	55	10	80	10	70	10
	80	80	10	90	10	85	10
	100	85	10	55	10	55	10
	120	85	10	70	10	85	10
12	40	30	10	40	10	70	10
	60	55	10	50	10	45	10
	80	80	10	90	10	85	10
	100	85	10	90	10	80	10
	120	85	10	90	10	50	10