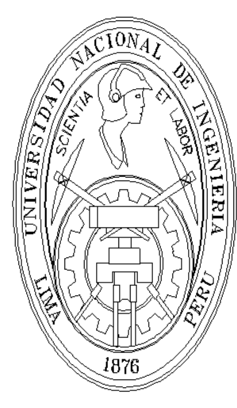
UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA MECANICA



ALGORITMOS GENÉTICOS PARA LA OPTIMIZACIÓN DE LA PROGRAMACIÓN DE LA PRODUCCIÓN DE UNA REFINERÍA DE ÓLEOS CON TIEMPOS DE PROCESAMIENTO Y FECHAS DE ENTREGA DIFUSAS

TESIS

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO MECATRÓNICO

PRESENTADO POR:

German Baca Espinoza

PROMOCIÓN
2005–II

LIMA – PERÚ
2007

A mis padres por su dedicación y comprensión

ALGORITMOS GENÉTICOS PARA LA OPTIMIZACIÓN DE LA PROGRAMACIÓN DE LA PRODUCCIÓN DE UNA REFINERÍA DE ÓLEOS CON TIEMPOS DE PROCESAMIENTO Y FECHAS DE ENTREGA DIFUSAS

SUMARIO

Al incrementar las compañías sus niveles de personalización de su oferta de productos, moverse hacia lotes mas pequeños de producción,y experimentar con contratos cliente/proveedor más flexibles tales como los que hace posible mediante intercambio electrónico EDI (Electronic data interchange), requieren cada vez más la habilidad de (1) responder rápida, precisa y competitivamente a los requerimientos del cliente para ofertas en nuevos productos y (2) encontrar eficientemente contratos proveedor/subcontratista para estos productos. Esto conlleva a requerir de la habilidad de (1) rápidamente convertir productos basados en especificaciones estándar en procesos de planta y (2) rápidamente integrar procesos de planta para nuevas órdenes en el programa de producción existente para acomodar de la mejor forma el estado actual de la compañía manufacturera. Con el propósito de asegurar el uso eficiente de la capacidad de una producción por lotes de gran escala, el control de la producción y su programación debe tomar en cuenta flexiblemente tanto las órdenes como el estado de los procesos de su producción. El plan de producción a largo plazo determina las cantidades para los diferentes productos a ser fabricados durante, por ejemplo, un mes o un año. El programa operacional a corto plazo debe tomar en cuenta el plan a largo plazo así como el estado prevalescente del procesos de producción. Los cambios en el plan de producción, nuevas campañas de marketing, intervalos de mantenimiento/servicio y otras perturbaciones hacen al ambiente dinámico y lleno de incertidumbres, lo que dificulta la tarea de la programación. El presente trabajo propone un método de computación evolucionaria, basado en los algoritmos genéticos, para la optimización de la programación de la producción de una refinería de óleos. La planta a tratar se caracteriza por ser multiproducto y de manufactura mixta, ya que tiene etapas con procesos por lotes (batch process) y etapas operadas continuamente (flowshop). El objetivo principal de esta tesis es demostrar que la solución de los algoritmos genéticos es capaz de cumplir con la optimización de los objetivos operacionales de la refinería. Para esto, la teoría de conjuntos fuzzy se aplicara para modelar la incertidumbre de los tiempos de procesamiento y demanda para el problema de la programación flowshop. Con esto se puede mostrar que el módulo de programación puede integrar planes de implementación de nuevos procesos de manufactura en el programa de producción, ya que cuando se hacen las pruebas con nuevos procesos, no se conoce de manera

certera los nuevos tiempos de determinadas tareas. Los resultados se mostrarán a partir de simulaciones basadas en data real de producción de una refinería de óleos, de una corporación local dedicada a la manufacturación de productos de consumo masivo. Dicha compañia es una de las más grandes a nivel nacional y mediana a nivel internacional. Cuenta con una variedad muy diversa de productos. Las evaluaciones mostrarán la efectividad del método propuesto.

ÍNDICE

TABLA DE ILUSTRACIONES 1

TABLA DE ILUSTRACIONES

A mis padres por su dedicación y comprensión

ALGORITMOS GENÉTICOS PARA LA OPTIMIZACIÓN DE LA PROGRAMACIÓN DE LA PRODUCCIÓN DE UNA REFINERÍA DE ÓLEOS CON TIEMPOS DE PROCESAMIENTO Y FECHAS DE ENTREGA DIFUSAS

SUMARIO

Al incrementar las compañías sus niveles de personalización de su oferta de productos, moverse hacia lotes mas pequeños de producción,y experimentar con contratos cliente/proveedor más flexibles tales como los que hace posible mediante intercambio electrónico EDI (Electronic data interchange), requieren cada vez más la habilidad de: (1) responder rápida, precisa y competitivamente a los requerimientos del cliente para ofertas en nuevos productos y (2) encontrar eficientemente contratos proveedor/subcontratista para estos productos. Esto conlleva a requerir de la habilidad de (1) rápidamente convertir productos basados en especificaciones estándar en procesos de planta y (2) rápidamente integrar procesos de planta para nuevas órdenes en el programa de producción existente para acomodar de la mejor forma el estado actual de la compañía manufacturera.

Con el propósito de asegurar el uso eficiente de la capacidad de una producción por lotes de gran escala, el control de la producción y su programación debe tomar en cuenta flexiblemente tanto las órdenes como el estado de los procesos de su producción. El plan de producción a largo plazo determina las cantidades para los diferentes productos a ser fabricados durante, por ejemplo, un mes o un año. El programa operacional a corto plazo debe tomar en cuenta el plan a largo plazo así como el estado prevalescente del procesos de producción. Los cambios en el plan de producción, nuevas campañas de marketing, intervalos de mantenimiento/servicio y otras perturbaciones hacen al ambiente dinámico y lleno de incertidumbres, lo que dificulta la tarea de la programación.

El presente trabajo propone un método de computación evolucionaria, basado en los algoritmos genéticos, para la optimización de la programación de la producción de una refinería de óleos. La planta a tratar se caracteriza por ser multiproducto y de manufactura mixta, ya que tiene etapas con procesos por lotes (batch process) y etapas operadas continuamente (flowshop). El objetivo principal de esta tesis es demostrar que la solución de los algoritmos genéticos es capaz de cumplir con la optimización de los objetivos operacionales de la refinería. Para esto, la teoría de conjuntos fuzzy se aplicara para modelar la incertidumbre de los tiempos de procesamiento y demanda para el problema de la programación *flowshop*. Con esto se puede mostrar que el módulo de programación pue-

de integrar planes de implementación de nuevos procesos de manufactura en el programa de producción, ya que cuando se hacen las pruebas con nuevos procesos, no se conoce de manera certera los nuevos tiempos de determinadas tareas.

Los resultados se mostrarán a partir de simulaciones basadas en data real de producción de una refinería de óleos, de una corporación local dedicada a la manufacturación de productos de consumo masivo. Dicha compañia es una de las más grandes a nivel nacional y mediana a nivel internacional. Cuenta con una variedad muy diversa de productos. Las evaluaciones mostrarán la efectividad del método propuesto.

ÍNDICE

TABLA DE ILUSTRACIONES 1

PRÓLOGO 3

CAPÍTULO I.

EL PROBLEMA DE LA PROGRAMACIÓN DE LA PRODUCCIÓN 6

1.1. El Problema en su Forma General 6
1.2. Características de la Forma Generalizada 8
1.2.1. Tareas (Actividades) 8
1.2.2. Recursos 9
1.2.3. Restricciones y Objetivos 9
1.2.4. Variaciones Dinámicas 11
1.3. Instancias del Problema Generalizado 11
1.3.1. El Problema del *flowshop* 13
1.3.2. Clasificación de Procesos de Programación por Lotes . . . 14
1.3.3. Aspectos del Modelado de la Programación por Lotes . . . 16
1.3.4. Problemas NP-hard 20
1.4. Lo que hace *hard* al Problema de Programación 20
1.4.1. Es un Asunto de Escalamiento 20
1.4.2. Incertidumbre y la Naturaleza Dinámica de los Problemas Reales 23
1.4.3. Infactibilidad del Espacio de Solución 24

CAPÍTULO II.

MODELAMIENTO DEL PROCESO ESTUDIADO 25

2.1. Planta: Refinería 26

2.1.1. Proceso de Neutralización y Blanqueo: 26
2.1.2. Hidrogenación: 28
2.1.3. Fraccionamiento 28
2.1.4. Desodorizado 28
2.1.5. Descerado 29
2.2. Planta: Envasado 29
2.2.1. Envasado de Aceite. 29
2.3. La Programación de la Producción en la Práctica 31
2.4. Formulación del Modelo de Optimización 31
2.4.1. Puntos de Tiempo Global: Formulación contínua RTN . . 32
2.4.2. Precedencia General 35

CAPÍTULO III.

MÉTODOS DE SOLUCIÓN 39

3.1. Métodos de Solución Exacta 40
3.1.1. Programación Lineal y Entera 41
3.1.2. Técnicas de Satisfacción de Restricciones 43
3.2. Métodos Heurísticos 44
3.2.1. Programación Heurística 45
3.2.2. Secuenciamiento Heurístico 46
3.2.3. Enfoques Jerárquicos 47
3.2.4. Enfoques de Inteligencia Artificial 47
3.3. Métodos Metaheurísticos 48
3.3.1. Búsqueda Local Iterada 50
3.3.2. Simulated Annealing 50
3.3.3. Tabu Search 52
3.3.4. Algoritmos Evolucionarios 54

CAPÍTULO IV.

APLICACIÓN DE LA SOLUCIÓN PROPUESTA 62

4.1. Modelo del Problema 63
4.1.1. Consideraciones sobre Tareas 65

4.1.2. Consideraciones de los Recursos 66
4.1.3. Consideraciones de los Objetivos 67
4.2. Método de Búsqueda 67
4.3. Funcionamiento del Algoritmo 73
4.3.1. Operación de Selección 74
4.3.2. Procedimiento de Búsqueda Local 74
4.3.3. Estrategia Elitista 75
4.3.4. Algoritmo Multiobjetivo Genético de Búsqueda Local . . . 76
4.4. Representación Genética 77
4.4.1. Demanda y Tamaño de Batch 79
4.4.2. Tiempos de Inicio 81
4.5. Operadores Genéticos 82
4.5.1. Inicialización 82
4.5.2. Crossover 83
4.5.3. Mutación 84
4.6. Formulación de Funciones Objetivo y Manejo de la Incertidumbre . . 86
4.6.1. Tiempo de Procesamiento Difuso 86
4.6.2. Fecha de Entrega Difusa 88
4.6.3. Criterio de Programación Difuso 90
4.6.4. Evaluación: Fitness 90

CAPÍTULO V.

PRUEBAS Y RESULTADOS 92

5.1. Especificación de Parámetros 92
5.1.1. Generaciones, Tamaños de Población y su Efecto en el Tiempo Computacional 93
5.1.2. Búsqueda Local 94
5.2. Tendencias de los Objetivos 96
5.2.1. Soluciones No Dominadas 96
5.2.2. Escenarios 99

CONCLUSIONES Y RECOMENDACIONES 108

APÉNDICE A: GLOSARIO 111

BIBLIOGRAFÍA 119

TABLA DE ILUSTRACIONES

1.1. Tipos de instalaciones de producción por lotes 13

2.1. Diagrama de Proceso general de manufactura de óleos 27

2.2. Diagrama del proceso de desodorizado 30

2.3. Flujo de la programación y planeamiento de la producción 33

2.4. Diferentes conceptos usados para la representación del problema de programación 33

3.1. Métodos de solución usados en problemas *flowshop* y de *batch* o por lotes 41

3.2. Esquema general de un algoritmo evolucionario 55

3.3. Clasificación de los métodos de resolución 59

3.4. Resumen de varias formulaciones de algoritmos genéticos. Las referencias en la Figura son representativas del tipo de solución; este diagrama no contiene una lista exhaustiva de publicaciones de los trabajos. La mayoría de las representaciones son basadas en las órdenes, p.e. el orden en el cual los items aparecen en la lista es parte de la estructura del problema 61

4.1. Solución elegida 64

4.2. Cuadro de flujo del algoritmo genético. Son posibles muchas variaciones , desde varios métodos de selección a una amplia variedad de métodos reproductivos específicos para cada representación. 69

4.3. Soluciones no dominadas (círculos cerrados) y soluciones dominadas (círculos abiertos) 71

4.4. Flujo de trabajo del algoritmo 78

4.5. Representación del genoma, tambión llamado cromosoma. Cada gen representa un batch o lote de producción . Y esta compuesto por una lista de datos necesarios para responder al problema que se quiere responder. 80
4.6. Proceso de fabricación simplificado 81
4.7. Crossover de dos puntos 84
4.8. Por ejemplo, se muta el gen sombreado con naranja y se realiza los siguientes pasos. 85
4.9. Suma de dos números fuzzy a y b; y máximo de dos números fuzzy, a y b. 87
4.10. *Tardiness* difusa para fechas de entrega difusas 89

5.1. Función fitness y su dependencia al parámetro de N_{elite} 95
5.2. Para (a) se uso $N_{gen} = 50$, (b) $N_{gen} = 50$ con otro intervalo de inicialización. 97
5.3. (c) uso $N_{gen}n = 100$ y (d) uso $N_{gen} = 150$. 98
5.4. Soluciones no dominadas para (a) escenario 1 y (b) escenario 2. 100
5.5. Función fitness para (a) escenario 1 y (b) escenario 2. 101
5.6. Función Makespan para (a) escenario 1 y (b) escenario 2. 102
5.7. Función tardiness para (a) escenario 1 y (b) escenario 2. 103
5.8. Criterio para la demanda no satisfecha 105
5.9. Volumen de demanda perdida o no satisfecha en toneladas para (a) escenario 1 y (b) escenario 2. 106
5.10. Diagrama de Gantt para la producción de 6 productos 107

PRÓLOGO

Con el motivo de asegurar el uso eficiente en la producción a gran escala, el control de la producción debe tomar flexiblemente en consideración tanto las ordenes como el estado del proceso de producción. El plan de producción a largo plazo determina las cantidades de diferentes productos a ser manufacturados durante, por ejemplo, un mes o un año. El programa operacional de corto plazo debe tomar en cuenta tanto el plan a largo plazo como el estado prevalescente del proceso de producción. Los cambios en el plan de producción, cambios en las campañas, intervalos de servicio, y perturbaciones pueden llegar a hacer la tarea muy exigente. El operador debe predecir los efectos de las acciones de control de varias variables del proceso con horas o incluso días de anticipación. Un programa operacional detallado incluye los tiempos para las operaciones, set points, conexiones entre unidades de procesamiento, etc. En la industria de proceso, el planeamiento de programas operacionales sigue siendo mayormente basada en la experiencia del operador, ya que las herramientas comerciales de control de la producción no han probado ser útiles en la práctica.

El punto de partida para el presente trabajo fue la planta de refinación de óleos y grasas de la empresa manufacturera de productos de consumo masivo de mayor presencia en el Perú. El ambiente se caracteriza por combinar aspectos de manufactura completamente continua y por lotes (batch), por lo que nos veremos obligados a combinar el modelado *flowshop* con algunas consideraciones del tipo batch. La meta fue desarrollar una solución basada en algoritmos genéticos que pueda producir programas operacionales de producción de manera eficiente.

El trabajo esta organizado en 5 capítulos más conclusiones. En el capítulo 1, se presenta el problema de la programación de la producción en su forma general. Se listan sus características y se describen los diversos modelos para los diferentes tipos de producción existentes. Se termina remarcando algunos factores que hacen del problema de la programación de la producción difícil.

El segundo capítulo presenta el caso real de la refinería como una producción contínua mixta por lotes. Se explica cómo es que funciona en la práctica, asi como las fases de la producción del proceso completo de producción del caso estudiado.

El tercer capítulo expone el marco teórico de los métodos de solución dispo-

nibles, que pueden lidiar con el problema presentado en el capítulo 1. Se presentan métodos de acuerdo a su enfoque, ya sea del campo de la investigación de operaciones como de la inteligencia artificial.

El capítulo 4 detalla la propuesta del presente trabajo, los algoritmos genéticos como solución al problema de la programación de la producción. Se explica su funcionamiento, consideraciones, operación de dichos algoritmos. La sección final de este caítulo se introduce la representación mediante variables difusas como alternativa para el manejo de la incertidumbre del entorno.

El último capítulo se presentan las tablas y gráficos de los resultados, luego de las pruebas de la simulación del algoritmo propuesto. Se especifican los parámetros utilizados para dichas simulaciones.

Finalmente se culmina el trabajo con las conclusiones, recomendaciones y propuestas para trabajos futuros en campos similares.

CAPÍTULO I

El Problema de la Programación de la Producción

En este capítulo representaremos el modelo de los problemas de programación para la producción. Aunque relacionados y estrechamente acoplados, la planificación y la programación son muy diferentes actividades. Planificación es la construcción del modelo de proceso y/o proyecto y la definición de restricciones/objetivos. Programación se refiere a la asignación de recursos a actividades(o actividades a recursos) en puntos especifícos de, o a lo largo de, el tiempo. La definición del problema es a partir de esto primordialmente un asunto de planificación, en tanto que la ejecución del plan es asunto de la programación. Aunque planificación y programación están ligados, la performance del algoritmo de la programación depende de la formulación del problema, y dicha formulación se puede beneficiar de la información obtenida durante la programación. La sección 1.1 contiene una descripción de la formulación general de la programación con recursos restringidos, la sección 1.2 detalla características especificas del problema, la sección 1.3 describe instancias del problema en general, el problema flow-shop y el batch (o por lotes), y la sección 1.4 remarca algunos factores que hacen el problema de programación difícil. El glosario con los términos se incluye en el apéndice.

1.1. El Problema en su Forma General

En su forma más general, el problema de la programación de recursos restringidos se define como sigue:

- un conjunto de actividades que deben ser ejecutadas,
- un conjunto de recursos con el cual se ejecutan las actividades,
- un conjunto de restricciones que debe ser satisfecha, y

- un conjunto de objetivos que debe juzgar la performance del programa

Cual es la mejor manera de asignar los recursos a las actividades en los tiempos específicos tales que todas las restricciones sean satisfechas y se produzca las mejores medidas objetivas.

La forma general incluye las siguientes características:

- Cada tarea podría ser ejecutada en más de una manera, dependiendo de cual(es) recurso(s) sea(n) asignado(s) a éste.
- las relaciones de precedencia de las tareas podrían incluir traslapes tal que para una tarea dada podría comenzar cuando su predescesor sea parcialmente completada
- cada tarea podría ser interrumpida de acuerdo a un conjunto pre-definido de modos de interrupción (específicos de cada tarea), o no se permite ninguna interrupción.
- cada tarea podría requerir más de un recurso de varios tipos
- los requerimientos de un recurso de tarea podría variar durante la duración de la tarea
- los recursos podrían ser renovables (p.e. mano de obra, maquinas) o no renovables (p.e. materia prima).
- la disponibilidad de los recursos podría variar durante la duración del programa o la tarea
- los recursos podrían tener restricciones temporales.

Para poder modelar con precisión la incertidumbre común en los problemas reales, la formulación general incluye las siguientes características dinámicas.

- disponibilidad de los recursos podrían variar
- requerimientos de los recursos podrían variar
- objetivos podrían cambiar

Las siguientes secciones clarifican estas características viendo el problema desde las perspectivas: tareas, recursos, y objetivos.

1.2. Características de la Forma Generalizada

Muchas de las características fueron inicialmente definidas a partir de la manufactura discreta. Más adelante explicaremos las particularidades que se tienen en cuenta para el caso de la manufactura continua.

1.2.1. Tareas (Actividades)

Las tareas tienen estimaciones medibles del criterio de performance tales como duración, costo, y consumo recurso. Cualquier tarea requiriría un solo recurso o varios, y el uso del recurso podría variar sobre el intervalo de tiempo de la duración de la tarea. Los estimados de duración y costo pueden depender de los recursos aplicados a (o usados por) la tarea. Las medidas de performance pueden ser probabilísticas o determinísticas.

Una tarea podría tener múltiples modos de ejecución. Cualquier tarea puede ser ejecutada en más de una manera dependiendo a partir de cuales recursos sean usados. Por ejemplo, si dos personas se asignan a un trabajo, este podría ser completado en la mitad del tiempo requerido por solo una persona. Alternativamente, un producto puede ser manufacturado por uno de los 3 diferentes procesos, dependiendo de cuál de las máquinas estén disponibles.

Una tarea simple puede ser compuesta por múltiples partes. La definición de parte incluye la especificación para la cual la tarea puede o no ser interrumpida durante las partes o solo entre las partes. Por ejemplo, una operación de fresado que se realiza en una fresadora CNC podría requerir un tiempo de setup diferente que el que se realiza en una fresadora de múltiples ejes. El setup puede ser abortado en cualquier tiempo, pero una vez que la máquina ha comenzado, la tarea no puede ser interrumpida hasta que el proceso de fresado se complete.

Frecuentemente el modo incluye información adicional que lleva a mayores restricciones. Algunas tareas, una vez iniciadas, podrían no se paradas ni cambiar su modo hasta que la tarea sea completada. Alternativamente, algunas tareas pueden ser abortadas en cualquier momento , posiblemente con algún tipo de costo asociado.

Los modos de interrupción dependen de los recursos aplicados a la tarea. Algunas tareas podrían ser interrumpidas, pero los recursos que ellas usan no pueden ser usados en ningún otro sitio hasta que la tarea sea terminada. Otras

tareas podrían ser interrumpidas, reasignando recursos, y luego cualquier recurso reaplicado cuando la tarea se retome. Seria posible mover un recurso desde una tarea a otra una vez que la primera tarea se inicie. Para otras tareas, un recurso una vez asignado a una tarea debe permaneces con esa tarea hasta terminada. Algunas tareas podrían ser interrumpidas y luego reiniciadas mas tarde, pero con algún costo de degradación en performance o incremento en el tiempo de terminación.

1.2.2. Recursos

Los recursos pueden ser renovables o no renovables. Los recursos renovables están disponibles en cada periodo sin ser agotados. Ejemplos de recursos renovables pueden ser la mano de obra y varios tipos de equipos. Los recursos no-renovables incluyen el capital y la materia prima. Notar que la distinción entre renovables y no-renovables puede ser extensa. En algunos casos, los recursos renovables pueden convertirse en no-renovables, en otros, los no-renovables pueden ser considerados como renovables.

Los recursos varían en capacidad, costo, y otras medidas de performance. Por ejemplo, cada vendedor tiene un grado asociado de confiabilidad en sus tiempos de entrega. Un trabajo de equipo podría ser más eficiente que otro. La disponibilidad de recurso puede variar también. Los recursos podrían convertirse en indisponibles debido a interrupciones imprevistas, fallas o accidentes.

Los recursos podrían tener restricciones adicionales. Muchos recursos incluyen restricciones temporales que limitan el período de tiempo cuando estos pueden ser usados. Por ejemplo, un equipo de maquinistas podría estar disponible solo durante el primer turno. Así mismo, las restricciones pueden ser más complicadas: Otro equipo de maquinistas puede estar disponible en cualquier turno con un mejor indicador de desempeño, siempre y cuando se cumpla la condición de darles un turno de descanso por cada dos trabajados.

1.2.3. Restricciones y Objetivos

Las restricciones y los objetivos se definen durante la formulacion del problema. Las restricciones definen la *factibilidad* de un programa. Los objetivos

definen el *optimizmo* de un programa. Mientras que los objetivos *deberían* ser satisfechos, las restricciones *deben* ser satisfechas. Ambas pueden basarse en la tareas, en los recursos, medidas de performance, o alguna combinación de éstos.

Un programa *factible* satisface todas las restricciones. Un programa *óptimo* no solo satisface todas las restricciones, sino también es por lo menos tan bueno como cualquier otro programa factible. Esto es definido por las mediciones objetivas. Cuando se modela el problema es frecuentemente conveniente pensar en los objetivos y restricciones como equivalentes, pero al momento de solucionarlo deben ser tratados de diferente manera. La programación de proyectos típicamente especifica la minimización de la duración del proyecto como objetivo primario. Sin embargo, la mayoría de problemas reales están sujetos a objetivos múltiples, mayormente conflictivos. Otros objetivos incluyen la minimización del costo, maximización del valor actual neto del proyecto, utilización de los recursos, eficiencia de los recursos, número de fechas de entrega que son cumplidas o no cumplidas, y minimización del trabajo en el proceso. Frecuentemente los objetivos son conflictivos. Por ejemplo, podría ser fácil acortar la duración de un proyecto mediante la asignación de recursos más costosos al trabajo en las asignaciones/tareas, pero entonces el costo del proyecto se incrementaría. Cuando más objetivos sean tomados en cuenta, la posibilidad de los conflictos crece. La consideración de múltiples objetivos requiere de la definición de un mecanismo para definir la relación entre los objetivos conflictivos para tomar decisiones acerca de cuales objetivos son importantes. Las restricciones aparecen de diversas formas. Las restricciones de precedencia definen el orden en el cual se pueden realizar las tareas. Por ejemplo, la producción de margarina requiere antes del desodorizado un proceso de hidrogenado, cuyo objetivo es estabilizar los aceites y proveer de la consistencia adecuada (convertir los aceites en grasas sólidas a temperatura ambiente) para su aplicación final. Las restricciones temporales limitan los tiempos de duración en los que los recursos son utilizados y/o las tareas ejecutadas. Por ejemplo, los backup de los software SCADA de una planta se realizan solo en las paradas de planta anuales.

1.2.4. Variaciones Dinámicas

Desde el momento en que son definidos, la mayoría de planes y programas están destinados al cambio. La secuenciación para una estación de trabajo se cambia frecuentemente en el transcurso de la primera hora de trabajo en un turno. Muchos programas de manufactura continua se cambian debido a las incertidumbres en el abastecimiento de proveedores de material o fallas en los equipos que ocasionen parad as de planta no esperadas. Muchos planes de proyectos requieren modificaciones cuando las estimaciones iniciales resultan imprecisas o cuando retrasos inesperados confunden a la disponibilidad de los recursos. Tanto los planes y los programas deben ser capaces de adaptarse a los cambios. Una parte importante de la estabilidad de un plan es la consistencia de la optimización dado un proyecto parcialmente completado. Asumiendo que la herramienta de planificación esta provista de una interface para mantener la integridad de la información, un optimizador debe ser capaz de utilizar el trabajo actual como restricción (como parte de una medida objetiva) tal que determina un nuevo programa optimizado. Otra característica deseable es que pueda çongelar"parte del proyecto tal como esta para optimizar solo el resto.

1.3. Instancias del Problema Generalizado

Como veremos en los términos definidos a continuación, los problemas de flow-shop, job-shop, open-shop y mixed-shop son todas instancias particulares del caso general del problema de recursos restringidos [78]. Además, muchos de los problemas de programación de la producción como programación de proyectos con recursos restringidos y modo simple, modo múltiple o multiproyecto, también son variaciones de la forma generalizada. Para el presente trabajo se comenzará planteando el problema de programación o secuenciación. El objetivo tradicional del problema flowshop, en el caso mono-objetivo, es minimizar la función objetivo siendo el tiempo total de finalización de todos los trabajos - el *makespan*. Desde el punto de vista notacional, hay dos formas de especificar un problema de programación/secuenciación en un contexto formal; siendo el primero propuesto por Conway [68], y la segunda, mayormente usada en la literatura, fue desarrollada por Graham [69], consistiendo de 3 distintos campos - $\alpha/\beta/\gamma$ - para describir el

problema como sigue:

- α: este campo indica la estructura del problema
- β: este campo acumula en conjunto explícito de restricciones (no implicadas en la estructura interna semántica - por ejemplo, para el flowshop, un trabajo o tarea no puede comenzar su ejecución en una máquina si ésta aun esta en procesamiento de la anterior.
- γ: el campo indica el objetivo a ser optimizado

Una revisión de varios problemas de secuenciación estudiados frecuentemente también se proponen en la literatura por Lee[12] y Pinedo[65]. Por ejemplo, aquí muestro una lista agrupando los problemas clásicos de programación en 6 clases:

- **workshop con una máquina -** $\alpha_1 =$: solo hay un máquina la cual debe ser usada para secuenciación de los trabajos dados, bajo especificaciones de las restricciones.
- **flowshop -**$\alpha_1 = F$: hay más de una máquina y cada trabajo debe ser procesado en cada máquina- el numero de operaciones para cada trabajo es igual al número de máquinas, siendo la j^{th} operación de cada trabajo procesada en la máquina j.
- **Compound flowshop:** cada unidad en la serie podría ser reemplazada por un conjunto de equipos (items) paralelos los cuales pueden ser idénticos o muy diferentes. Cada trabajo va a una unidad en la primera etapa, luego es transferida a una de la segunda etapa, y asi sucesivamente.
- **jobshop -** $\alpha_1 = J$: el problema es formulado bajo los mismos términos que para el problema flowshop, teniendo como específica diferencia el hecho que cada trabajo tiene un orden de procesamiento asociado asignada para sus operaciones.
- **openshop -** $\alpha_1 = O$: la misma similitud con el problema del flowshop, siendo el orden de procesamiento para las operaciones completamente arbitrario - el orden para procesar una operacion de un trabajo no es relevante, ya que cualquier orden lo podra realizar.

- **mixed workshop -** $\alpha_1 = X$: hay un subconjunto de trabajos para los cuales se específica un camino fijo, el resto de trabajos son secuenciados con el fin de minimizar la función objetivo.

En la Figura 1.1 se grafica de forma general a los tipos más importantes dentro de la producción por lotes.

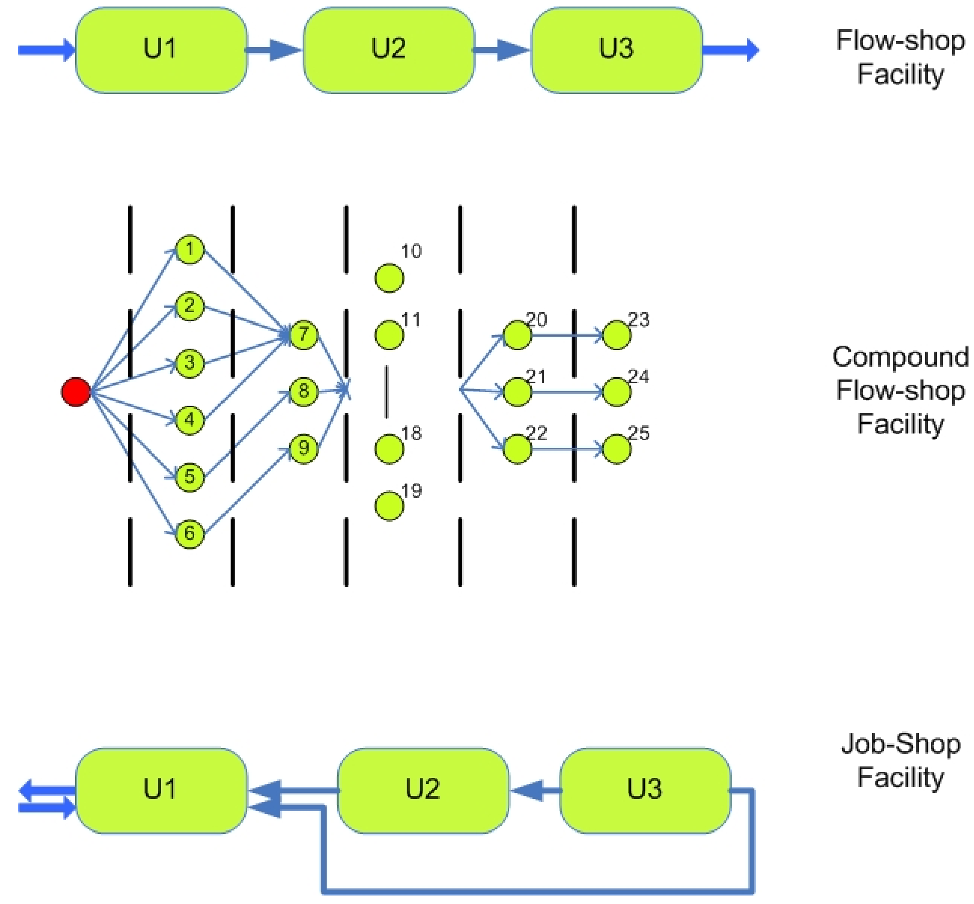


Figura 1.1: Tipos de instalaciones de producción por lotes

1.3.1. El Problema del *flowshop*

El problema de permutación flowshop representa un caso particular del problema de programación flowshop, teniendo como meta el despliegue de un programa óptimo para N trabajos en M máquinas. Resolver el problema flowshop consiste en secuenciar n trabajos $(i = 1, ..., n)$ en m máquinas $(j = 1, ..., m)$. Un trabajo consiste en m operaciones y la operación j^{th} de cada trabajo debe

ser procesada en la máquina j. Entonces, un trabajo puede comenzar en una máquina j si es terminada antes en la máquina $j-1$ y la máquina j esta libre. Cada operación tiene un tiempo de procesamiento conocido $p_{i,j}$. Para la permutación flowshop, las secuencias de operación de los trabajos son las mismas en cada máquina. Si uno de los trabajos esta en la posición i^{th} de la máquina 1, entonces este trabajo en la posición i^{th} en todas las máquinas. Como consecuencia, para el problema de permutación flowshop, considerando el *makespan* como función objetivo a ser minimizada, resolver el problema significa determinar la permutación para la cual se da el menor valor de *makespan*. En el contexto arriba especificado, un trabajo J_i, puede ser visto como un conjunto de operaciones, teniendo una operación para cada trabajo de las M máquinas.

- $J_i = \{O_{i1}, O_{i2}, O_{i3}, ..., O_{iM}\}$, donde O_{ij} representa la operación j^{th} de J_i;
- la operación O_{ij} debe ser procesada en la máquina M_j;
- para cada operación O_{ij} hay un tiempo de procesamiento asociado p_{ij}.

Notacionalmente el problema se referencia como $F/permu/C_{max}$ considerando como objetivo a ser minimizado el tiempo de procesamiento total - el *makespan*.

Sea $\prod_1, \prod_2, \prod_3, ..., \prod_N$ una permutación. Calcular el tiempo final $C(\prod_\tau, j)$ para el i^{th} trabajo de la permutación dada $\prod$ y la máquina j, puede ser como sigue:

$$
\begin{array}{ll}
C(\prod_1, 1) = p_{\prod_1,1} & \\
C(\prod_\tau, 1) = C(\prod_{\tau-1}, 1) + p_{\prod_1,1} & i = 2, ..., N \\
C(\prod_1, j) = C(\prod_1, j-1) + p_{\prod 1,j} & j = 2, ..., M \\
C(\prod_\tau, j) = max\{C(\prod_{\tau-1}, 1), C(\prod_1, j-1)\} + p_{\prod \tau,j} & i = 2, ..., N; j = 2, ..., M
\end{array}
$$

bajo estas especificaciones, el valor de la función objetivo, el *makespan*, C_{max}, es dado por $C(\prod_N, M)$-tiempo de terminación o *completion time*, para la última operación en la última máquina.

1.3.2. Clasificación de Procesos de Programación por Lotes

En la práctica, muchos procesos contínuos son por batchs o lotes. Y muchos de estos a su vez, son secuenciales, con etapas múltiples de procesamiento o

etapa única, donde una o varias unidades podrían estar trabajando en paralelo en cada etapa. Cada lote necesita ser procesado siguiendo una secuencia de etapas definidas a través de la receta del lote/producto. En nuestro caso vamos a lidiar con una planta que combina procesos netamente continuos con fabricación por lotes. Tenemos que considerar los siguientes aspectos al momento de abordar el modelamiento de una planta.

Topología del proceso Cuando las aplicaciones se vuelven mas complejas, se deben manejar redes con topologías arbitrarias. Las recetas complejas de productos incluyendo operaciones mezclados, separaciones y recirculación de materiales necesitan ser considerados en estos casos, lo que va adhiriendo restricciones/requerimientos.

Asignación del equipo Estas restricciones se expresan en términos de la asignación de los equipos involucrados, llendo desde arreglos fijos hasta los flexibles.

Conectividad de los equipos Interconexiones limitadas entre los equipos imponen fuertes restricciones al momento de tomar decisiones de ubicación de unidades.

Políticas de inventario Frecuentemente involucra almacenamiento dedicado y de capacidad finita, aunque en casos frecuentes incluye tanques compartidos así como políticas de cero-espera, almacenaje no-intermedio e ilimitado.

Transferencia de Material Se asume instantáneo, pero en algunos casos es necesario considerar en los casos de plantas sin tuberías.

Tamaño del lote En el envasado de alimentos por ejemplo se manejan tamaños de lotes fijos cuya integridad debe mantenerse para asegurar, sin mezclarse o separarse, mientras que en un proceso de refinado se manejan tamaños de batch variable, de acuerdo a la demanda de la diversidad de productos.

Tiempo de procesamiento de lote Similarmente a los tamaños de lotes, los tiempos dependen de la aplicación. En la industria alimentaria, los tiempos son fijos y los determinan por la receta del producto.

Patrones de Demanda Varían desde el caso en que las fechas de entrega deben ser obviadas hasta el caso en el que los objetivos de producción deben ser alcanzados sobre un horizonte de tiempo.

Reformas de proceso Son críticos en casos de transiciones que son dependientes de la secuencia de los productos, a diferencia de los que sólo son dependientes de configuraciones de las unidades.

Restricciones de Recurso a parte de los equipos, p.e. la mano de obra, las instalaciones son frecuentemente de gran importancia y pueden ser desde discretas hasta continuas.

Restricciones de Tiempo Consideraciones de operación prácticas como paradas en fines de semana o períodos de mantenimiento determinan restricciones temporales.

Costo Los costos asociados al uso del equipo, inventarios, reformas e instalaciones pueden tener un impacto considerable. Generalmente, en industrias de producción de bienes de consumo masivo como los alimentos, el costo fijo de los equipos es alto.

Grado de incertidumbre Finalmente, debe tomarse en cuenta qué tanta incertidumbre hay en la data recolectada, que se incrementa particularmente para la demanda calculada en horizontes de tiempo mayores.

1.3.3. Aspectos del Modelado de la Programación por Lotes

En esta sub sección describiremos los aspectos que se deben tener en cuenta al formular un mismo problema de programación. Estos aspectos tienen normalmente impacto en aspectos de performance computacional, capacidades y limitaciones del modelo de optimización.

Representación del Tiempo

Dependiendo si los eventos del programa pueden tomar lugar a sólo puntos de tiempo predefinidos, o pueden ocurrir en cualquier momento durante el horizonte de interés, los enfoques de optimización pueden ser clasificados como

discretos o continuos. Los modelos discretos se basan en: (i) dividir el horizonte de programación en un número finito de intervalos con duración predefinida y, (ii) permitir que los eventos tales como el inicio o termino de una tarea sucedan solo en los límites de estos períodos de tiempo. Asi se reduce la complejidad del modelo, ya que las restricciones solo se monitorean en tiempos específicos de tiempo ya conocidos. Por otro lado, el tamaño del modelo matemático asi como la eficiencia computacional depende fuertemente del número de intervalos. Además, se podrían generar soluciones sub-optimales o incluso infactibles, debido a la reducción del espacio de toma de decisiones. A pesar de esto, las formulaciones discretas han demostrado ser muy eficientes sobre todo en aquellos casos donde un número razonable de intervalos es suficiente para obtener una representación del modelo. Con motivo de lidiar con estas limitaciones y generar modelos independientes de la data, se emplea el tiempo continuo. En estas formulaciones, las decisiones de tiempo son representadas explícitamente por un conjunto de variables continuas representando los momentos exactos en que los eventos ocurren. En el caso general, un manejo de variable de tiempo permite obtener una significativa reducción del numero de variables del modelo y al mismo tiempo, soluciones más flexibles en términos de tiempo que puede ser generado. Sin embargo, el modelamiento de tiempos de procesamiento variables, recursos y limitaciones de inventario usualmente necesitan la definición de restricciones más complicadas, que tienden a incrementar la complejidad del modelo y la brecha de integración, pudiendo tener impacto negativo en la capacidad del método.

Balance de Materiales

El manejo de lotes y tamaños de lotes lleva a dos tipos de categorías de modelos de optimización. La primera categoría se refiere a un enfoque monolítico, el cual trata simultáneamente con el conjunto optimal de lotes (número y tamaño), la colocación y el secuenciamiento de los recursos de planta y la temporización de tareas de procesamiento. Estos métodos son capaces de manejarse con procesos de redes arbitrarios involucrando recetas de producto complejas. Su generalidad usualmente implica grandes tamaños de formulación del modelo y consecuentemente su aplicación esta actualmente restringida a procesos que involucren un número pequeño de tareas de procesamiento y horizontes estrechos

de programación. Estos modelos emplean redes STN (State-Task Network), o redes RTN (Resource-Task Network) para representar el modelo. Los modelos STN representan el problema asumiendo que las tareas de procesamiento producen y consumen estados (materiales). Un tratamiento especial se le da los recursos de manufactura aparte de los equipos. En contraste, la formulación RTN emplea un tratamiento uniforme para todos los recursos disponibles a través de la idea que las tareas de procesamiento consumen y liberan recursos en su comienzo y en su inicio, respectivamente. La segunda categoría comprende modelos que asumen que el número de lotes de cada tamaño se conoce con anterioridad. Estos algoritmos pueden, de hecho, ser considerados como uno de los módulos de un enfoque detallado de la programación de la producción, ampliamente usado en la industria, que descompone todo el problema en 2 etapas, batching y programación batch. El problema de batching convierte los requerimientos de productos primarios en batches o lotes individuales apuntando a optimizar algún criterio como la carga de planta. Luego, los recursos de planta disponibles se colocan en batches a lo largo del tiempo. Esta aproximación de dos etapas permite tratar con problemas de mayor tamaño que los monolíticos, especialmente aquellos que involucran un gran número de tareas batch relacionadas con diferentes productos intermedios o finales. Sin embargo, aun se encuentran restringidos a procesos comprometidos con recetas secuenciales de producto.

Representación de Eventos

Para formulaciones de tiempo discreto, la definición de intervalos de tiempo globales es la única opción para redes generales y procesos secuenciales. Para este caso, se predefine una malla fija y común valida para todos los recursos compartidos y las tareas por lotes se fuerzan a comenzar y terminar exactamente en un punto de la malla. Consecuentemente, se reduce el problema original de programación a un problema de colocación donde el modelo principal de toma de decisiones define la asignación de los intervalos de tiempo en el cual cada tarea por lote inicia. Por otra parte, para formulaciones en tiempo continuo y redes generales se pueden usar puntos globales de tiempo y eventos en puntos de unidad de tiempo especifica, mientras que en el caso de procesos secuenciales las alternativas incluyen el uso de casillas de tiempo y enfoques de relaciones de precedencia

para lotes. La representación por puntos en tiempo global corresponde a una generalización de los intervalos de tiempo globales donde la temporización de los intervalos es tratada como una nueva variable del modelo. A diferencia de esto, la idea de eventos en unidad especifica de tiempo define una malla variable diferente para cada recurso compartido, permitiendo iniciar diferentes tareas en diferentes momentos para el mismo punto de evento. Sin embargo, la falta de puntos de referencia para monitorear la disponibilidad limitada de recursos compartidos hace de la formulación algo más complicado. Se necesitan por tanto restricciones especiales y variables adicionales para lidiar con problemas de restricción en los recursos.

Las representaciones basadas en casillas tiene como idea proponer un número de éstas para cada unidad de procesamiento con el fin de colocarlas en la tarea por lotes a ser realizada. Pueden ser clasificadas como síncronas y asíncronas.

Para procesos secuenciales se tiene otro enfoque adicional: relaciones de precedencia. Se puede tener tres tipos, precedencia inmediata, con una variable única del modelo $X_{ii'j}$, precedencia general donde se define una variable de secuenciamiento para cada par de tareas por lote que puede ser colocado en el mismo recurso compartido; y, precedencia inmediata por unidad especifica.

Funciones Objetivo

Se pueden usar diferentes medidas de la calidad de solución para problemas de programación. Sin embargo, el criterio seleccionado para la optimización usualmente tiene un efecto directo en la performance del modelo computacional. Además, algunas funciones objetivo pueden ser muy difíciles de implementar para algunas representaciones de eventos, requiriendo variables adicionales y restricciones complejas. En la tabla 1.1 se resume los modelos más importantes para la optimización de la producción.

1.3.4. Problemas NP-hard

En teoría de la complejidad computacional, la clase de complejidad NP-hard es el conjunto de los problemas de decisión que contiene los problemas H tales que todo problema L en NP puede ser transformado polinomialmente en H. Esta clase puede ser descrita como conteniendo los problemas de decisión que son al menos tan difíciles como un problema de NP. Esta afirmación se justifica porque si podemos encontrar un algoritmo A que resuelve uno de los problemas H de NP-hard en tiempo polinómico, entonces es posible construir un algoritmo que trabaje en tiempo polinómico para cualquier problema de NP ejecutando primero la reducción de este problema en H y luego ejecutando el algoritmo A. Asumiendo que el lenguaje L es NP-completo,

1. L está en NP
2. $\forall L'$ en NP, $L' \leq L$

En el conjunto NP-Hard se asume que el lenguaje L satisface la propiedad 2, pero no la la propiedad 1. La clase NP-completo puede definirse alternativamente como la intersección entre NP y NP-hard.

1.4. Lo que hace *hard* al Problema de Programación

A parte de la cantidad de data esparcida y la gestión de la información requerida para programar un proyecto o una planta, hay algunas dificultades inherentes para resolver aun en los problemas simplificados.

1.4.1. Es un Asunto de Escalamiento

El tamaño de un problema de programación puede ser aproximado por una matriz *que-donde-cuando*. Usando la nomenglatura de Van Dyke Parunak, programando los problemas consiste en preguntar *qué* debe fabricarse, *dónde* y *cuándo*. Los recursos (dónde) operan en las tareas (qué) durante periodos específicos de tiempo (cuándo). Usando esta simple clasificación, y negando la precedencia y otras restricciones, se puede dar una aproximación burda del tamaño

[1]Las formulaciones orientadas al lote asumen que el problema total es descompuesto entre programación del tamaño de lote y, por otra parte, factores de secuenciamiento. El problema del tamaño del lote se resuelve primero para determinar el número y tamaño de los lotes a ser programados.

Tabla 1.1: Características Generales de modelos de optimización

Representación de tiempo	DISCRETO	CONTINUO					
Tipo de eventos	Intervalos de tiempo	Puntos de de tiempo global	Eventos de unidad específica	Casillas de tiempo [1]	Precedencia Inmediata de unidad específica[1]	Precedencia Inmediata[1]	Precedencia General[1]
Decisiones principales	–Tamaño de lote,colocación,secuenciamiento,temporizado–				–Colocación,secuenciamiento,temporizado–		
Variables discretas clave	W_{ijt} define si tarea I comienza en unidad j al comienzo del intervalo t.	Ws_{in}/Wf_{in} define si tarea i comienza/ termina en el punto n. $W_{inn'}$ define si tarea i inica en el punto n y finaliza en punto n	$Ws_{in}/W_{in}/$ Wf_{in} define si tarea i comienza/es activa/ finalizada en punto n	W_{ijk} define si unidad j comienza tarea i al comienzo de la casilla de tiempo k.	$X_{ii'j}$ define si lote i es procesada justo antes del lote i' en unidad j. Xf_{ij} define si lote i inicia la secuencia de procesamiento de unidad j.	$X_{ii'}$ define si lote i es procesada justo antes del lote i'. Xf_{ij}/W_{ij} define si lote i inicia/es asignado a unidad j	$X'_{ii'}$ define si lote i es procesada antes o despues del lote i'. W_{ij} define si lote i es asignado a unidad j unidad j.

Tabla 1.2: Continuación Tabla 1.1

Tipos de proceso	——————-Red General——————-				——————Secuencial——————		
Balance de materiales	Ecuaciones de Flujo de (STN o RTN)	Ecuaciones de Flujo de (STN o RTN)	–Ecuaciones de flujo de red– (STN)		——————Orientado al lote——————		
Factores de molelado críticos	Duración de de intervalos, ,período de programación (dependiente de data)	Número de puntos (estimados iterando)	Número de eventos (estimados iterando)	Número de casillas (estimados)	Número de tareas/lote de unidades compartidas (tamaño de lote) y y unidades	Número de tareas/lote de unidades compartidas (tamaño de lote)	Número de tareas/ lote de recursos compatidos (tamaño de lote)
Problemas críticos	Tiempo de procesamiento variable, recambios dependiente de secuencia	Fechas de entrega intermedias y suministro de material	Fechas de entrega intermedias y suministro de material	Limitaciones de recusos de recurso	Inventario, limitaciones de recurso	Inventario, limitaciones	Inventario

de un problema por el producto de qué, dónde y cuándo. Cuántas tareas deben ser completadas, por cuantos recursos y sobre qué intervalo de tiempo.

Hay muchas formas de delimitar el tamaño del espacio de búsqueda de solución. Muchos métodos han sido diseñados para determinar si las partes de un programa pueden ser factibles dado un conocimiento parcial del programa, o si una parte del árbol de decisión puede ser algo mejor que otra parte. Estos métodos intentan reducir el tamaño de búsqueda tomando ventaja de la información especifica del problema. Sin embargo, la heurística de la delimitación no es siempre disponible, y raramente es obvia.

La elección de la representación también controla el tamaño del espacio de búsqueda. Si uno escoge una representación muy generalizada, más tipos de problemas podrían ser resueltos a costa del espacio de búsqueda más extenso. Contrariamente, uno podría elegir una representación muy específica que significativamente reduzca la búsqueda, pero solo trabajaría para una instancia del problema.

1.4.2. Incertidumbre y la Naturaleza Dinámica de los Problemas Reales

Hablando prácticamente, encontrar un programa óptimo es frecuentemente menos importante que lidiar con incertidumbres durante el planeamiento y perturbaciones impredecibles durante la ejecución del programa. En algunos casos, los planes se basan a partir de procesos bien conocidos en los cuales el comportamiento de los recursos y requerimientos de tareas son todos bien conocidos y pueden ser predichos con precisión. En muchos otros casos, sin embargo, las predicciones son menos precisas debido a la falta de datos o modelos predictivos. En estos casos el programa podría estar sujeto a mayores cambios en tanto que el plan en el que esta basado cambia. En cualquiera de los dos casos, las perturbaciones no anticipadas podrían ocurrir. Si es un error mecánico o humano, o clima inclemente, las perturbaciones son inevitables. Tales perturbaciones podrían requerir sólo de la reasignación de un único recurso o de la reformulación completa del plan. Cualquier técnica de optimización debe ser capaz de adaptarse a los cambios en la formulación del problema manteniendo el contexto del trabajo ya completado.

1.4.3. Infactibilidad del Espacio de Solución

Dependiendo de la representación y de las suposiciones del modelado, podrían no haber soluciones factibles para un problema dado. Por ejemplo, si todos los recursos están disponibles en cantidades de periodo constante y no hay restricciones temporales en las tareas o recursos, se garantiza que un proyecto tiene un programa factible. En el peor de los casos, uno sólo tendría que extender el proyecto hasta que todas las tareas se completen. Si, por otro lado, los recursos, una vez usados, se terminan para siempre y las tareas pueden ser ejecutadas una determinada cantidad de veces o con ciertos límites de tiempo, no se podría garantizar una solución factible. Algunos algoritmos son capaces de determinar si existe tal situación factible. La mayoría de métodos heurísticos no puede.

Las restricciones hacen la búsqueda de una solución óptima aun más difícil, rompiendo el espacio continuo y convirtiéndolo en discontinuo. Cuando se adicionan muchas restricciones, se confunde el espacio de búsqueda. Además, la suma de restricciones reduce típicamente el número de soluciones factibles para una representación dada.

CAPÍTULO II

Modelamiento del Proceso Estudiado

El complejo industrial modelado se encuentra dividido en varias plantas que producen sus propios productos. En algunos casos los productos finales de algunas plantas son materias primas para otras. Dentro de este ”site”, el alcance de la Etapa 1 está centrado en procesos de las Plantas de Refinería (aceites y grasas) y Envasado de aceites. En la Figura 2.1 se puede apreciar el proceso completo de la transformación de materias primas en productos finales de esta industria.

2.1. Planta: Refinería

La refinación tiene como objetivo retirar del aceite todos aquellos compuestos no deseables y adecuar su estructura química con la finalidad de lograr un producto terminado óptimo para el consumo humano. En la Figura 2.1 se presenta el diagrama de flujo de las principales etapas del proceso general de aceites en refinería.

2.1.1. Proceso de Neutralización y Blanqueo:

Este proceso se realiza en 2 plantas que funcionan de forma independiente.

El aceite crudo debe purificarse para mejorar algunos atributos y permitir procesarlo exitosamente para obtener finalmente productos terminados de calidad adecuada. Se dispone, a la entrada, de tanques de alimentación, desde donde se provee el aceite crudo a un intercambiador de calor. A continuación el aceite es tratado con ácido fosfórico y luego con soda cáustica para su neutralización. La neutralización, se lleva a cabo en mezcladores contínuos especialmente diseñados para tener tiempos de residencia adecuados. El producto de reacción de estas etapas - el jabón - junto al aceite, es conducido a una centrífuga, para su separación. Después de esta etapa, el aceite aun lleva jabón en suspensión, el que es removido por lavados con agua blanda caliente (sólo en Planta 1), la que a su vez, con los residuos de jabón, es separada del aceite por medio de otra centrífuga. Con todos estos procesos se han reducido fuertemente del aceíte, los ácidos grasos libres, la humedad, jabones, compuestos azufrados, de oxidación, metales, etc. El aceite neutro se lleva a la etapa de "blanqueo", en la cual, el aceite, es tratado con sílice para la remoción de jabones y metales. Finalmente, el aceite es tratado con tierras ácido-activadas, las que tienen la propiedad de retirar componentes de oxidación, pigmentos (clorofila, carotenos) y residuos de jabón, por adsorción. Si bien esta etapa se conoce como "Blanqueo", en los procesos posteriores como son la Hidrogenación y la Desodorización del aceite, también se produce una reducción del color.

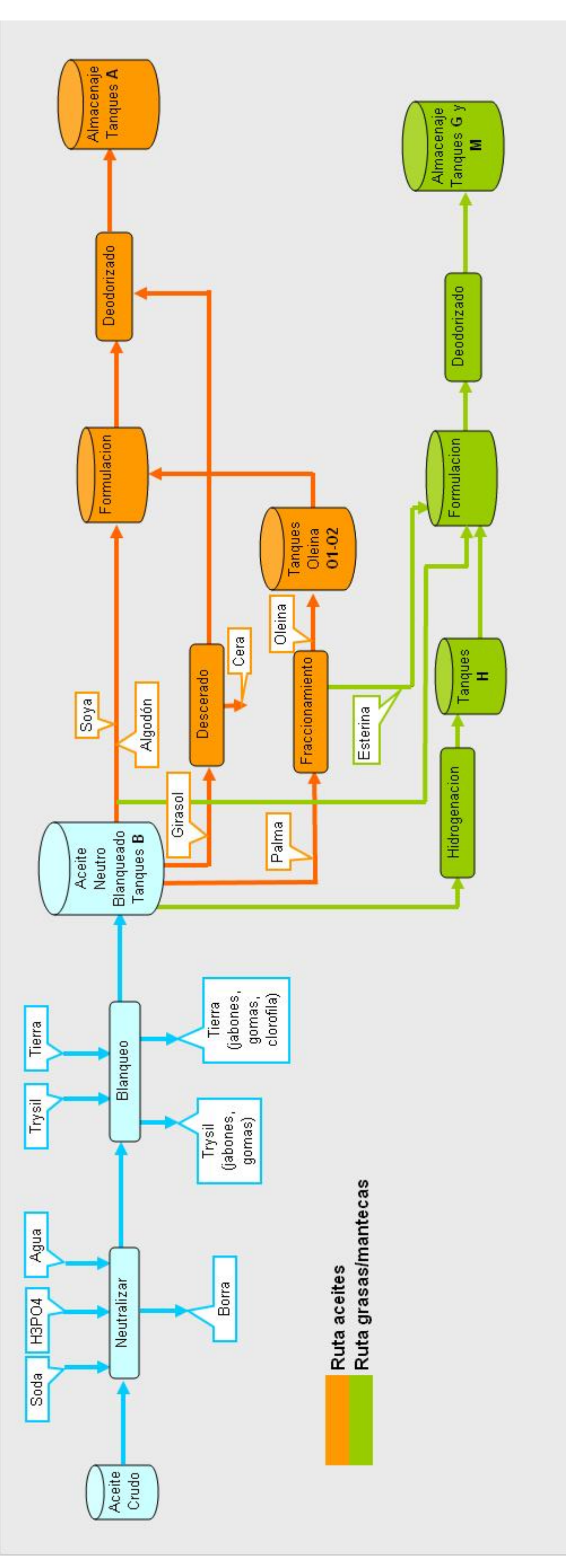


Figura 2.1: Diagrama de Proceso general de manufactura de óleos

2.1.2. Hidrogenación:

La Hidrogenación es un cambio de estructura química del aceite, que se realiza a nivel molecular, para producir las modificaciones, que tienen como objetivo estabilizar los aceites y proveer de la consistencia adecuada (convertir los aceites en grasas sólidas a temperatura ambiente) para su aplicación final. El proceso de Hidrogenación se realiza sobre el aceite neutro-blanqueado. La Hidrogenación se lleva a cabo en reactores, en los cuales se realiza una reacción en fase heterogénea, es decir, interactúa el gas hidrógeno con aceite líquido en presencia de un catalizador sólido. Son reactores cerrados, diseñados para operar bajo presiones moderadas, provistos de agitación, de intercambiadores de calor, dispersores de gas, etc. Una vez que el gas llegue al cabezal del reactor, con las paletas superiores del agitador se debe lograr una mezcla uniforme. La reacción prosigue, en las curvas de temperatura que corresponda, hasta que el operador verifique que se hayan obtenido las características finales del producto.

2.1.3. Fraccionamiento

El fraccionamiento no modifica la estructura molecular de los aceites, es un cambio físico que consiste en el enfriamiento controlado del aceite neutro-blanqueado de palma y pescado y una posterior filtración para separar la parte líquida (oleína) que será utilizada en la formulación de aceites, de la parte sólida (estearina) que será utilizada en la formulación de mantecas y margarinas.

2.1.4. Desodorizado

Este proceso se lleva a cabo después de la neutralización, el blanqueo y la hidrogenación, una vez mezclados y estandarizados todos los componentes de la base aceite o grasa de los productos finales. La desodorización es un proceso de destilación al vacío con arrastre a vapor, cuya finalidad es retirar de los aceites las trazas de sustancias que les confieren olor y sabor. Esto es factible debido a la gran diferencia de volatilidad que existe entre los triglicéridos y aquellas sustancias odoríferas que les imparten olor y sabor. La desodorización se realiza bajo vacío y a alta temperatura, para facilitar la remoción de las sustancias volátiles, para evitar la hidrólisis de las grasas y aceites y para hacer más eficiente el uso del vapor. La desodorización no tiene ningún efecto significativo en la composición de

los ácidos grasos de los aceites. El aceite desodorizado es llevado a las plantas de envasado donde, como ingrediente base, se utiliza para producir las líneas de los productos finales según sus propias características que son: aceite, margarinas, mantecas y mayonesas. El diagrama de equipos del desodorizador se muestra en la Figura 2.2.

2.1.5. Descerado

Es un caso particular de fraccionamiento, ya que en aquel proceso se separan por cristalización, pequeñas cantidades de los componentes sólidos, ya sean éstos triglicéridos o bien ceras. En todos los casos, el proceso se realiza enfriando el aceite, cristalizando cuidadosamente los sólidos que existan a esa temperatura, y separando por un método que normalmente es la filtración.

2.2. Planta: Envasado

Los aceites y grasas desodorizadas son llevados a las plantas de envasado donde, se utilizan para producir las líneas de los productos finales según sus propias características: aceites, margarinas, mantecas y salsas.

2.2.1. Envasado de Aceite.

Existen actualmente 6 Líneas de envasado de Aceite a saber:

- Línea 1 - Aceite botella 1 litro.
- Línea 2 - Aceite botella 1 litro / $\frac{1}{2}$ litro.
- Línea 3 - Aceite botella 200 cc..
- Línea 4 - Aceite baldes y latas 18 litros/bidones 5 litros
- Línea 5 - Aceite bidones 5 litros.
- Línea 6 - Aceite bidones 5 litros (Extrusión soplado y llenado)

Las línea de envasado de Aceite 1 y 2 se encuentran automatizadas por máquina individual (ordenadora de botellas, llenadora y tapadora, etiquetadora, envasadora, etc.), sin contar con una red donde los PLCs puedan intercambiar información o enviar información a un sistema SCADA.

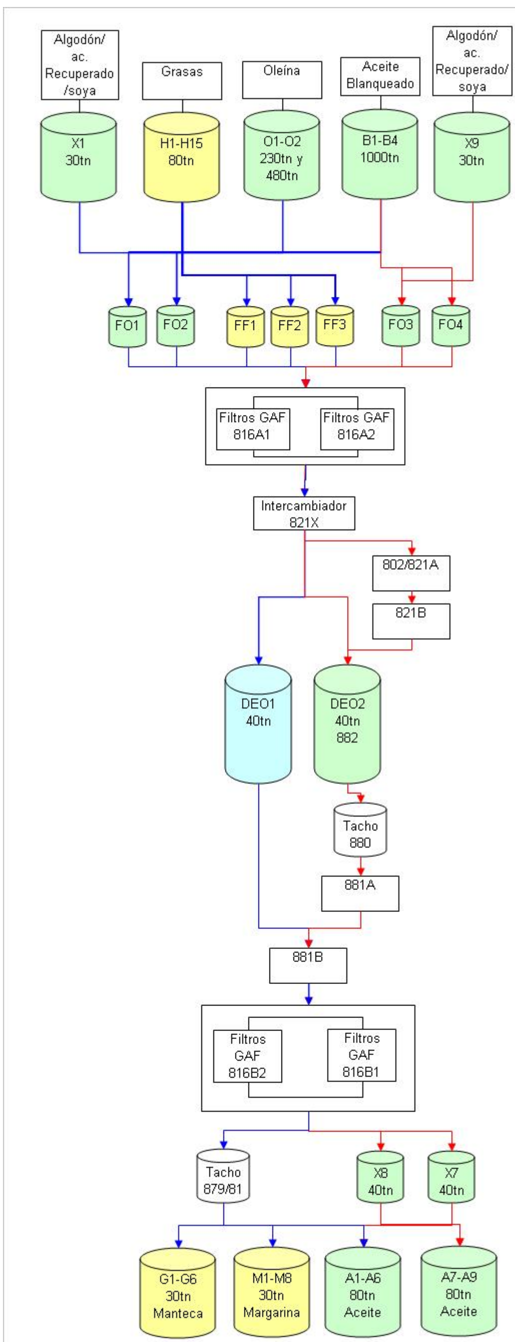


Figura 2.2: Diagrama del proceso de desodorizado

2.3. La Programación de la Producción en la Práctica

El proceso de planear y programar la producción en la planta de aceites y grasas estudiada se diagrama en la Figura 2.3. Se caracteriza por ser un proceso cíclio, que se inicia con el pronóstico de la demanda mensual. Para esto, las areas de producción, departamento de ventas y departamento de marketing proponen un plan de demanda, cada uno, basado en sus propias proyecciones (flechas amarillas en Figura 2.3). Luego de cotejar estas propuestas, se llega a un plan de demanda único, el cual se traduce al programa de producción industrial para envasado semanal. Según el plan de producción de envasamiento, es que se realiza el programa de producción del desodorizado.

Una vez que se tienen estos programas de producción semanales, se trabaja bajo la demanda del cliente. Todos los días los programas se revisan para verificar si necesitan modificaciones para cumplir con la demanda real del cliente (flecha azul en la Figura 2.3). Ya entrando al detalle de la operación de planta, el operador de refinería, específicamente, del desodorizador, recibe semanalmente el programa de producción, el cual debe revisar al inicio de su jornada, para según eso programar su producción de turno. La elección de los tanques iniciales, es decir, de los tanques de donde extrae la materia a procesar así como de los finales, a donde descarga el producto procesado, es manejada por los operadores de tanques o balanceros. Los balanceros están en constante monitoreo de los consumos de sus clientes para poder determinar la mejor opción en cuanto a disponibilidad de sus tanques. Ellos revisan también el programa diario de producción del desodorizador y de envasamiento.

2.4. Formulación del Modelo de Optimización

Como hemos descrito en este capítulo, la refinería estudiada mezcla las características de un problema *flowshop*, donde la producción es continua y nunca, teóricamente, debe parar; con aspectos muy peculiares de la programación por lotes, donde las operaciones se rigen por recetas así como las cargas a producir. Por esta razón, en el presente trabajo se analizan dos formulaciones del problema de optimización; la primera es una formulación continua con puntos de tiempo global [62] y la segunda, orientada al lote, de precedencia general. Estas se pueden ver de manera general en la Figura 2.4. En la tabla 2.1 se muestra la nomenglatura

utilizada.

2.4.1. Puntos de Tiempo Global: Formulación contínua RTN

El concepto lo introdujo Pantelides [62]. El trabajo desarrollado por Castro [15] que fue luego probado por Castro [14] cae también en esta categoría y se describe abajo. Se hizo las siguientes suposiciones: (i) las unidades de procesamiento se consideran individualmente, p.e., un recurso se define para cada unidad disponible, y (ii) solo una tarea se puede realizar en cualquier equipo dado en cualquier tiempo (recurso unitario). Estas suposiciones crecen con el incremento de tareas y recursos a definir, pero al mismo tiempo ayuda a reducir la complejidad del modelo.

Restricciones de Tiempo: Un conjunto de puntos globales de tiempo N se predefinen, donde el primero de ellos toma lugar en el comienzo $T_l = 0$ y el último, en el horizonte de tiempo dado, es $T_n = H$. Sin embargo, la mayor diferencia en comparación con el modelo STN viene con la definición de la variable de colocación $W_{inn'}$, la cual es 1 cuando la tarea i comienza en el punto de tiempo n y termina en o antes de el punto $n' > n$. De esta manera, los puntos de inicio y termino de una tarea dada i están definidos mediante un solo conjunto de variables binarias. Se debe notar que esta definición, por una parte, hace el modelo más simple y compacto, pero por otra, se incrementa significativamente el número de restricciones y variables a ser definidas. Las restricciones (2.4.1) y (2.4.2) imponen que la diferencia entre los tiempos absolutos de cualquier par de puntos de tiempo (nyn') debe ser mayor o igual (para tareas de *espera cero*) que el tiempo de procesamiento de todas las tareas comenzando y terminando en esos mismos puntos de tiempo. Como puede verse en estas ecuaciones, el tiempo de procesamiento de una tarea dependerá tanto de la activación de la tarea como del tamaño del lote.

$$T_{n'} - T_n \geq \sum_{i \epsilon I_r} (\alpha_i W_{inn'} + \beta_i B_{inn'}) \qquad \forall r \epsilon R^J, n, n', (n < n') \tag{2.4.1}$$

$$T_{n'} - T_n \leq H \left(1 - \sum_{i \epsilon I_r^{ZW}} W_{inn'} \right) + \sum_{i \epsilon I_r^{ZW}} (\alpha_i W_{inn'} + \beta_i B_{inn'}) \quad \forall r \epsilon R^J, n, n', (n < n') \tag{2.4.2}$$

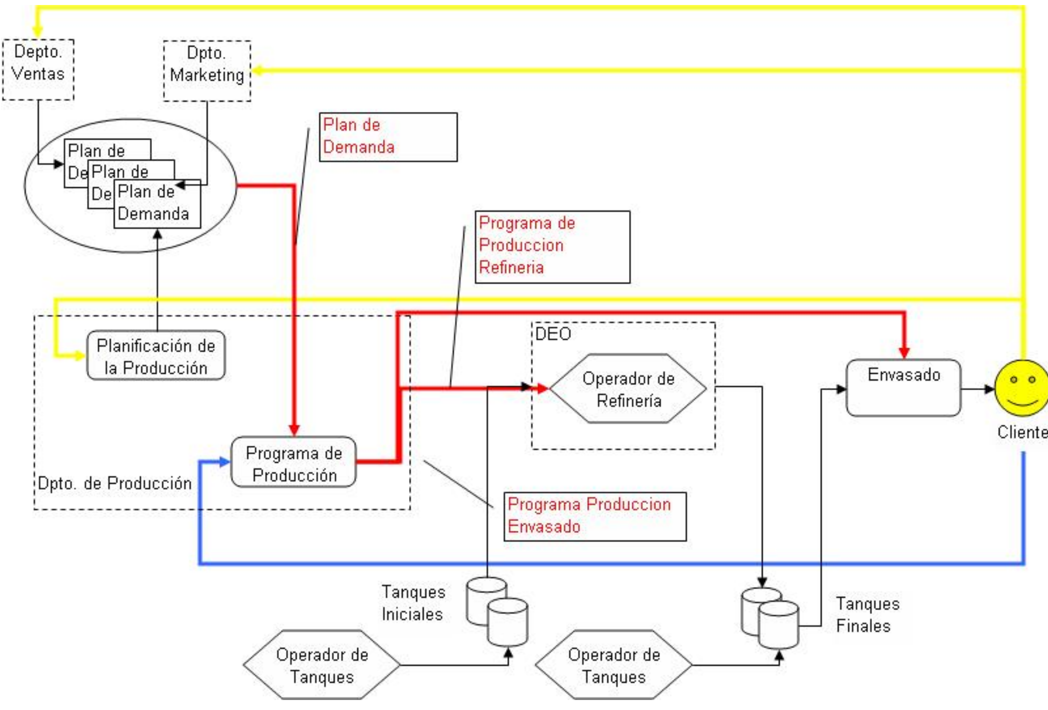


Figura 2.3: Flujo de la programación y planeamiento de la producción

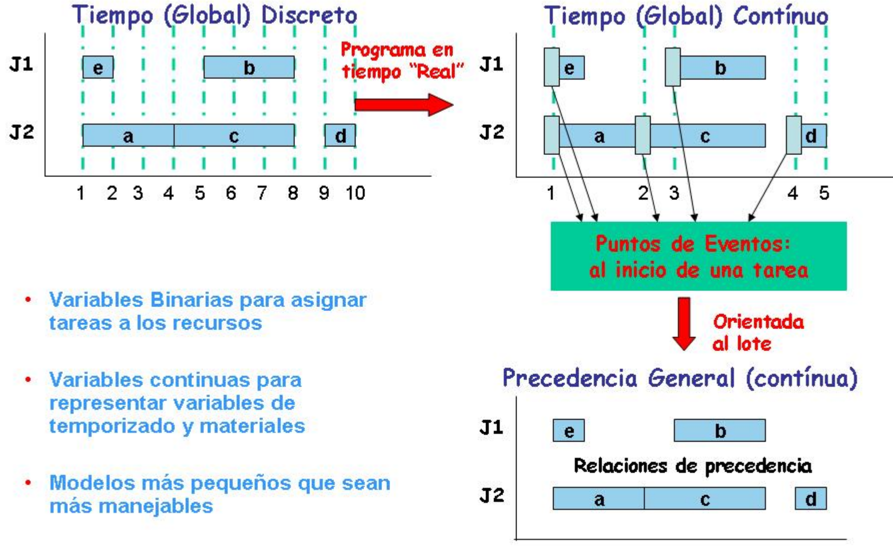


Figura 2.4: Diferentes conceptos usados para la representación del problema de programación

Restricción del Tamaño de lote: Considerando la suposición que una tarea solo puede ser procesada en una unidad de procesamiento, se toma en consideración la capacidad limitada del equipo mediante la restricción (2.4.3).

$$V_i^{min} W_{inn'} \leq B_{inn'} \leq V_i^{max} W_{inn'} \forall i, n, n', (n < n') \qquad (2.4.3)$$

Balance de Recursos: La disponibilidad de recursos es una expresión típica de balance multiperíodo, en la que el exceso de un recurso en un punto de tiempo n es igual a la cantidad en exceso en el evento previo, punto $(n-1)$, ajustado por la cantidad de recurso consumido/producido por todas las tareas iniciadas/terminadas en el punto n, como lo expresa la restricción (2.4.4). Un termino especial que toma en cuenta el consumo/entrega de los recursos de almacenamiento se incluye para cualquier tarea de almacenaje I^{ST}. Aquí, los valores negativos se usan para representar consumo, mientras que los términos positivos definen la producción de un recurso. La cantidad de recursos disponibles también es limitada por la restricción (2.4.5).

$$R_{rn} = R_{r(n-1)} + \sum_{i \epsilon I_r} \left[\sum_{n'<n} (\mu_{ir}^p W_{in'n} + \nu_{ir}^p B_{in'n}) + \sum_{n'>n} (\mu_{ir}^c W_{inn'} + \nu_{ir}^c B_{inn'}) \right] +$$
$$\sum_{i \epsilon I^{ST}} \left(\mu_{ir}^p W_{i(n-1)n} + \mu_{ir}^c W_{in(n+1)} \right) \ \forall r, n > 1 \qquad (2.4.4)$$

$$R_r^{min} \leq R_{rn} \leq R_r^{max} \ \forall r, n \qquad (2.4.5)$$

Restricciones de almacenaje: Asumiendo una tarea de almacenaje por cada recurso de material, la definición de la restricción (2.4.5) en combinación con las ecuaciones (2.4.6) y (2.4.7) garantizan que si hay un exceso en cantidad de recurso en el tiempo n. entonces la tarea correspondiente será activada para ambos intervalos $n-1$ y n.

$$V_i^{min} W_{in(n+1)} \leq \sum_{r \epsilon R_i^{ST}} R_{rn} \leq V_i^{max} W_{in(n+1)} \qquad \forall i \epsilon I^{ST}, n, (n \neq \mid N \mid) \qquad (2.4.6)$$

$$V_i^{min} W_{i(n-1)n} \leq \sum_{r \epsilon R_i^{ST}} R_{rn} \leq V_i^{max} W_{i(n-1)n} \qquad \forall i \epsilon I^{ST}, n, (n \neq 1) \qquad (2.4.7)$$

Se puede ver que el modelo contínuo de RTN basados en la definición de puntos de tiempo globales son bastante generales. Es capaz de fácilmente incluir una variedad de funciones objetivo tales como maximización de las ganancias o minimización del tiempo total de producción. Sin embargo, eventos que se den durante el horizonte de tiempo tales como las fechas de entrega múltiples y recepciones de material son mas complejas de implementar dado que la posición exacta de los puntos de tiempo es desconocida.

2.4.2. Precedencia General

La noción de precedencia generalizada extiende el concepto de precedencia inmediata no solo para considerar el predecesor inmediato, sino también todos los lotes procesados anteriormente en la misma secuencia de procesamiento. De esta manera, la idea básica es completamente generalizada, lo que simplifica el modelo y reduce el número de variables de secuencia. Esta reducción se obtiene definiendo solo una variable de secuencia para cada par de tareas de lotes que puede ser colocada en el mismo recurso. Adicionalmente, este enfoque enfatiza que la utilización de los diferentes tipos de recursos renovables compartidos tales como unidades de procesamiento, tanques de almacenaje, servicios y mano de obra pueden ser manejados eficientemente a través del mismo conjunto de variables de secuencia sin comprometer la optimización de la solución. Parte del trabajo realizado en esta categoria esta representado por los enfoques desarrollados por Méndez [51] y Méndez y Cerdá [52], [53], [54], [55]. *Restricciones de Colocación:* Una unica unidad de procesamiento j debe ser asignada a cada etapa requerida l para manufacturar el lote i, llamado aquí tarea (i, l).

$$\sum_{j \epsilon J_{il}} W_{ijl} = 1 \qquad \forall i, l \epsilon L_i \tag{2.4.8}$$

*Restricciones de Tiempo:*Para definir el tiempo exacto para cada tarea de lote (i, l), la restricción (2.4.9) determina el tiempo final de la tarea desde el tiempo de inicio y procesamiento en la unidad asignada j. Las restricciones de precedencia entre las etapas consecutivas $l-1$ y l del lote i se imponen mediante la restricción 2.4.10.

$$Tf_{il} = Ts_{il} + \sum_{j \epsilon J_{il}} pt_{ilj} W_{ilj} \qquad \forall i, l \epsilon L_i \qquad (2.4.9)$$

$$Ts_{il} \geq Tf_{i(l-1)} \qquad \forall i, l \epsilon L_i, l > 1 \qquad (2.4.10)$$

*Restricciones de Secuencia:*Las restricciones de secuencia (2.4.11)y (2.4.12), que son expresadas en términos de restricciones M-big, se definen para cada par de tareas (i, l) y (i', l') que puedan ser colocados en la misma unidad j. Si ambos son colocados en la unidad j, p.e. $W_{ilj} = W_{i'l'j} = 1$, ninguna de las restricciones, ni (2.4.11) ni (2.4.12), estarán activas. Si la tarea (i, l) se procesa antes que (i', l'), entonces $X'_{il,i'l'}$ es igual a uno y la restricción (2.4.11) se fuerza a garantizar que la tarea (i', l') comenzara después de completar ambas tareas, la (i, l) y la subsecuente operación de cambio de producto en la unidad j. Además, la restricción 2.4.12 se convierte en redundante. En caso que la tarea (i', l') corra antes en la misma unidad, la restricción 2.4.12 se aplicara y la restricción 2.4.11 se relaja. De otro modo, tal par de tareas no se lleva a cabo en la misma unidad y, consecuentemente, ambas restricciones, 2.4.11 y 2.4.12, serán redundantes y el valor de la variable de secuencia $X'_{il,i'l'}$ no tendra sentido para la unidad j. Se debe notar que el concepto de precedencia usado en la variable de secuencia incluye no solo al predecesor inmediato sino que a todos los lotes procesados anteriormente en el mismo equipo compartido.

$$Ts_{i'l'} \geq Tf_{il} + cl_{il,i'l'} + su_{i'l'} - M(1 - X'_{il,i'l'}) - M(2 - W_{ilj} - W_{i'l'j})$$
$$\forall i, i', l \epsilon L_i, l' \epsilon L'_i, j \epsilon J_{il,i'l'} : (i, l) < (i', l') \qquad (2.4.11)$$

$$Ts_{il} \geq Tf_{i'l'} + cl_{i'l',il} + su_{il} - MX'_{il,i'l'} - M(2 - W_{ilj} - W_{i'l'j})$$
$$\forall i, i', l \epsilon L_i, l' \epsilon L'_i, j \epsilon J_{il,i'l'} : (i, l) < (i', l') \qquad (2.4.12)$$

Limitaciones de Recursos: Tomando ventaja del concepto de la precedencia general, esta formulación es capaz de manejar las limitaciones de los recursos aparte de las unidades de procesamiento sin predefinir puntos de referencia para chequear disponibilidad de recurso. La idea general es utilizar un tratamiento uniforme de las limitaciones de los recursos, donde el uso de unidades de procesamiento y otros recursos como mano de obra, herramientas y servicios, se manejen mediante la colocación común y decisiones de secuencia. Para hacer esto, la formulación

define diferentes tipos de recursos r (mano de obra, herramientas, vapor, energía, etc.) incluidas en el problema de programación asi como los items individuales o piezas de recursos z disponible para cada tipo r. Por ejemplo, 3 operarios de un centro de supervisión, $z1, z2$ y $z3$ puede ser definido por el recurso tipo "mano de obra", llamado aquí $r1$. Por lo tanto, la restricción (2.4.13) asegura que suficiente recurso r será colocado para cumplir con el requerimiento del lote i, donde q_{rz} es la cantidad de recurso r disponible en el item de recurso z del tipo r y ν_{ilrj} define la cantidad de recurso r requerido cuando la tarea (i,l) se coloca en la unidad j, p.e. demandas de recursos dependientes de la unidad pueden ser fácilmente tomados en cuenta. Además, el par de restricciones (2.4.14) y (2.4.15) enfatizan el uso secuencial de cada item de recurso mediante la misma idea presentada arriba para el secuenciamiento de unidades de procesamiento. Debe notarse que la misma variable de secuenciamiento $X'_{il,i'l'}$ se utiliza para unidades de procesamiento y otros recursos, las restricciones (2.4.11), (2.4.12), (2.4.14) y (2.4.15), permiten generar una formulación simple del problema con un reducido número de variables binarias.

$$\sum_{z \epsilon RR_{ir}} q_{rz} Y_{iz} = \sum_{j \epsilon J_{il}} \nu_{ilrj} W_{ilj} \qquad \forall r \epsilon R_i, i, l \epsilon L_i \tag{2.4.13}$$

$$Ts_{i'l'} \geq Tf_{il} - M(1 - X'_{il,i'l'}) - M(2 - Y_{ilz} - Y_{i'l'z})$$
$$\forall i, i', l \epsilon L_i, l' \epsilon L_{i'}, r \epsilon Z_r : (i,l) < (i',l') \tag{2.4.14}$$

$$Ts_{il} \geq Tf_{i'l'} - MX'_{il,i'l'} - M(2 - Y_{ilz} - Y_{i'l'z})$$
$$\forall i, i', l \epsilon L_i, l' \epsilon L_{i'}, r \epsilon Z_r : (i,l) < (i',l') \tag{2.4.15}$$

(2.4.16)

Tabla 2.1: Nomenglatura

Índices	
i, i'	Tarea del Lote
n, n'	tiempo o punto de evento (tiempo continuo)
r, r'	tipo de recurso
Conjuntos	
I_r	Tareas que requieren el recurso r
I_s^{ST}	Tareas de almacenaje para el estado s
I_r^{ZW}	Tareas que producen el recurso r el cual requiere de la politica *cero espera*
$J_{ii'}$	Unidades de procesamiento que pueden realizar tanto tareas i como i'
R^J	Recursos correspondientes al equipo de procesamiento
Z_r	Items del recurso de tipo r
Parámetros	
α_i	Tiempo fijo de procesamiento de la tarea i
β_i	Tiempo variable de procesamiento de la tarea i
μ_{irt}	Coeficiente para la produccion/consumo fijo del recurso r en el tiempo t relativo al inicio de la tarea i
ν_{irt}	Coeficiente para la produccion/consumo variable del recurso r en el tiempo t relativo al inicio de la tarea i
su_{ij}	Tiempo de setup para la tarea de procesamiento i en la unidad j
$cl_{il,i'l'}$	Tiempo de recambio requerido entre la etapa l de la tarea i y la etapa l' de la tarea i'
H	Horizonte de tiempo de interés
pt_{ij}	Tiempo de procesamiento de la tarea i en la unidad j
V_i^{min}	Tamaño de lote mínimo de la tarea i
V_i^{max}	Tamaño de lote máximo de la tarea i
Variables Binarias	
$W_{inn'}$	Define si la tarea i comienza en el punto n y termina en el punto n'
W_{ijt}	Define si la tarea i inicia en la unidad j al inicio del intervalo de tiempo t
$X'_{il,i'l'}$	Defina si la etapa l de la tarea i es procesada antes/despues de la etapa l' de la tarea i' en alguna unidad
Y_{ilz}	Define si el item z de recurso es colocado en la etapa l de la tarea i
Variables Contínuas	
$B_{inn'}$	Tamaño del lote de la tarea i inició en el tiempo n y terminó en el tiempo n'
R_{rn}	Cantidad de recurso r que es consumido en el punto temporal n
T_n	Tiempo que corresponde al punto temporal n
Ts_{in}	Tiempo de inicio de la tarea i que comienza en el punto temporal n
Ts_{il}	Tiempo de inicio de la etapa l de la tarea i
Tf_{il}	Tiempo de fin de la etapa l de la tarea i

CAPÍTULO III

Métodos de Solución

Usando métodos de planeamiento automatizados, el objetivo es obtener mejores planes, para hacer trabajar más fácilmente al planeamiento y evitar que emerjan situaciones problemáticas debido a una pobre planificiación. Mejores planificaciones se alcanzan debido a que los métodos basados en optimizaciones son capaces de manejar una gran cantidad de factores restrictivos simultaneamente. Estos métodos se basan bien en métodos matemáticos de programación (por ejemplo programación lineal (LP), programación no lineal (NLP), Mixed Integer Linear Programming (MILP) o Mixed Integer Non-Linear Programing (MINLP)) o bien en técnicas de búsqueda estocástica (p.e. Simulated Annealing o Algoritmos Genéticos (GA)).

Cuando se modelan situaciones de si/no, como lo hacen los métodos MILP y MINLP, emergen problemas combinatorios. Para algunos problemas combinatorios, se pueden diseñar algoritmos polinómicos, resultando en tiempos computacionales justos incluso para problemas de gran tamaño. Desafortunadamente, hay problemas para lo cual los algoritmos polinomiales no se pueden obtener, resultando en un incremento exponencial de los tiempos computacionales al incrementar el tamaño del problema. Estos problemas se llaman NP-hard. Considerando los problemas de programación, se puede notar que la mayoría de ellos pertenecen a esta categoria y se puede ver que obteniendo la primera solución factible es muchas veces NP-hard también.

Debido a los problemas computacionales para los métodos determinísticos, se justifica la evaluación de los métodos alternos. Una alternativa que va ganando interés durante los últimos años son los basados en búsquedas estocásticas. Aunque no se pueda garantizar soluciones globales optimales, se pueden obtener buenas soluciones en tiempos cortos. Un uso efectivo de estos métodos también

asegura que todas las soluciones evaluadas son posibles para el problema actual. Las discontinuidades y simulaciones complejas se pueden modelar fácilmente con estos métodos. Diferentes formas de resolver de manera exacta el problema de programación flowshop han sido propuestas a lo largo del tiempo, considerando también definiciones en el contexto de multi criterio. Aproximaciones más complejas consideran capas estructuradas para funciones inferiores a nivel de máquina, cada capa siendo responsable del procesamiento de una sola operación; de acuerdo con el tipo de problema, pudiendo las máquinas estar disponibles en una sola capa o multiples capas. También, otra clase general de problemas complejos consideran la programación bajo términos generales de asignación, siendo cada operación dependiente de un conjunto de máquinas para su terminación. Diferentes criterios pueden ser usados para evaluar un programa, el criterio clásico más usado es el tiempo de terminación acumulado de todos los trabajos, dado un conjunto de máquinas, conocido dicho tiempo como *makespan* y notacionalmente especificado por C_{max}. Los objetivos del problema son usualmente especificados como funciones objetivo a ser minimizadas o como restricciones que tienen que ser satisfechas para una instancia a ser considerada como solución válida. El reto lo determinan los problemas -$F/\beta/\gamma$- que apuntan a la optimización de la permutación del *makespan* en problemas *flowshop* , donde todos los trabajos deben ser secuenciados en el mismo orden para todas las máquinas, siendo esto una restricción intrínseca. Como una breve introducción para problemas de *flowshop*, una separación puede ser realizada considerando el número de máquinas designadas: hay *flowshops* limitados sólo a una o dos máquinas - $F1/\beta/\gamma$- asi como problemas con un número variable de máquinas de acuerdo con cada especificación de la instancia -$F2/\beta/\gamma$. Todos los problemas de *flowshop* que pertenecen a la segunda clase mencionada son conocidos como altamente *NP-hard*. Una revisión incluyendo un gran número de diferentes problemas de ́flowshop puede ser encontrado en [23] y [82].

3.1. Métodos de Solución Exacta

Diferentes maneras de resolver el problema de programación *flowshop* de forma exacta han sido propuestos a lo largo de los años, considerando también las definiciones en el contexto de multicriteria. Se desarrollo uno, llamado *cercar y limitar* o Branch & Bound B&B, para problemas de *flowshop* con secuencia

dependiente de los tiempos de setup con Rios-Mercado, Bard como autores. Kohler [72] así como un enfoque B&C (Branch & Cut) [71], resolviendo Steiglitz problemas *flowshop* de dos máquinas usando diferentes métodos, exactos y metaheurísticos [44]. También, Carlier & Neron propuso un método de resolución para problemas híbridos de *flowshop* [13]. Otros enfoques, incluyendo los multi-objetivo, reutilizan técnicas de mono-objetivo, tal método ha sido desarrollado por Sayin y Karabati para un FSP (Flowshop Scheduling Problem) bi-objetivo [75]. Considerando el hecho que el problema *flowshop* es altamente NP-hard, los métodos exactos están restringidos a pequeñas instancias debido a su tiempo de ejecución; siendo la importancia de tales métodos esenciales para investigaciones futuras, aproximaciones distribuidas están siendo desarrolladas para resolver problemas de mayor escala. En la Figura 3.1 se muestran los métodos agrupados por disciplinas más usados.

Figura 3.1: Métodos de solución usados en problemas *flowshop* y de *batch* o por lotes

3.1.1. Programación Lineal y Entera

En general, los métodos exactos dependen de las características de la función objetivo (p.e. estrictamente valores enteros) y formulaciones de restricciones especificas (p.e. sólo tareas de modo único). Como lo nota Lawrence Davis, muchas de las restricciones comúnmente encontradas en los problemas reales de

programación no se adaptan bien a la investigación de operaciones tradicional o técnicas matemáticas de programación [20]. Además, las formulaciones de programación lineal típicamente no están bien escalados, así que pueden ser usadas sólo para instancias específicas o pequeños problemas. El enfoque de la programación dinámica fue descrita por Held y Karp en donde un programa optimal se desarrolla de manera incremental construyendo primero un programa optimal para cualquiera de las 2 tareas y luego extendiendo ese programa adicionando tareas hasta que todas las tareas hallan sido programadas [30]. Dada un horizonte de producción con fechas de entrega para las demandas de varios productos, las decisiones claves a tomar son las asignaciones de los productos a las unidades así como el secuenciamiento de los productos que son asignados a la misma unidad. Cada producto se procesa sólo una vez por exactamente una unidad de cada etapa por la que circula. El objetivo usualmente es minimizar el *makespan* o la demora total de las órdenes (*tardiness*). Mucho trabajo ha sido realizado en versiones determinísticas de este problema de programación en plantas *flowshop* con unidades paralelas. Pinto y Grossman [66] presentaron una formulación de tiempo continuo MILP para minimizar el total prorrateado del *earliness* (prontitud) y la demora (*tardiness*) de las órdenes. Esta formulación fue un modelo basado en *slots* o casillas que colocaba unidades y órdenes en dos coordenadas de tiempo paralelas y las encajaba con variables tetra-indexadas. Los mismos autores después [67] incorporaron restricciones en el modelo para reemplazar las variables tetra-indexadas por variables tri-indexadas (orden-etapa-unidad) y de ahi que redujeron el esfuerzo computacional requerido para resolver el modelo. McDonnald y Karimi [50] desarrollaron una formulación MILP para la programación a corto plazo de plantas con etapa única, multi-producto del tipo lote (*batch*) con unidades semi-continuas en paralelo. Más recientemente, Hui [36] presentó una formulación MILP la cual usa variables tri-indexadas (orden-orden-etapa) para minimizar el *earliness* y *tardiness* total de las órdenes. Con la eliminación de la unidad indexada, la formulación requería menos variables binarias y tiempo computacional reducido. Como el enfoque MILP se basa en relaciones lineales en su formulación, se requieren de suposiciones para la simplificación, ya que los problemas de programación son frecuentemente no lineales en la práctica. A pesar que han sido desarrollados modelos MILP para sistemas más realísticos [10], [47],

estos requieren formulaciones matemáticas complicadas y de ahí que demandan de un gran esfuerzo computacional en tiempo. La mayor debilidad del MILP es que, cuando la complejidad de la planta crece, el problema de programación se vuelve muy difícil de formular propiamente. Muchos casos de estudio han indicado que el enfoque MILP es sólo para problemas de pequeña escala.

3.1.2. Técnicas de Satisfacción de Restricciones

Constraint Satisfaction Techniques CST, o técnicas de satisfacción de restricciones, son métodos que expresan y resuelven problemas de satisfacción de restricciones CSPm (Constraint Satisfaction Problem). Un CSP esta representado por restricciones y variables restringidas. Cada variable tiene asociado un dominio, el cual incluye un conjunto de sus valores potenciales. Una solución al CSP es una asignación de valores a las variables restringidas del problema tal que todas las restricciones sean satisfechas simultáneamente. Un conjunto de todas las posibles asignaciones de valores a las variables se conoce como espacio de búsqueda. De ahí que, resolver un CSP es buscar una asignación de valor en dicho espacio.

CST esta ampliamente usado recientemente para resolver varios problemas que van desde colocación de recursos hasta programación. Estas técnicas no requieren formulaciones matemâticas complejas, pero a cambio requieren que el problema sea planteado en términos de restricciones. De ahi que, un ventaja inmediata ventaja del CST es su facilidad de implementación. Otra ventaja del CST es que, más que buscar ciegamente en el espacio completo una solución, explota a las mismas restricciones para reducir su esfuerzo de búsqueda. Las restricciones son explotadas en una forma constructiva para deducir otras restricciones y detectar inconsistencias entre las posibles soluciones, las cuales son conocidas como propagación de restricciones y chequeo de consistencia. Valores no adecuados en los dominios de las variables debido a inconsistencias serán removidos. Una búsqueda libre de *backtrack* es posible usando estas formas constructivas. El siguiente ejemplo sencillo ilustra como un CST trabaja. Suponga que hay 2 variables "n" y "m", y sus dominios son $\{0, 1, 2, ..., 10\}$ y $\{5, 6, 7\}$, respectivamente. Se imponen dos restricciones "$n < 10$" y "$n + m = 15$". Se remueven los valores inadecuados de los dominios cuando las restricciones se establecen. Por ejemplo,

cuando $n < 10$ se establece, 10 se remueve de el dominio de n para convertirse en $\{0, 1, 2, ..., 9\}$. Cuando la restricción $n + m = 15$ se establece, la remoción de los valores inadecuados actualizara ambos dominios a $\{8, 9\}$ y $\{6, 7\}$, respectivamente. De ahi que, el espacio de búsqueda pueda ser enormemente reducido antes que ocurra la búsqueda. Para buscar la solución, un punto escogido (también conocido como "nodo") se establece asignando 6 o 7 a m. Si se asigna 6 a m, entonces la propagación de la restricción removerá el valor 8 del dominio de n y unirá 9 a n, y la solución se encontrara. Después de eso, el sistema puede volver atrás de la solución $m = 7$ y $n = 8$. Si el criterio es minimizar m entonces la última solulción con el mayor valor de m se descarta. Con este ejemplo simple, se puede ver que encontrando una solución con CST involucra dos etapas: 1) la etapa de procesamiento, donde las técnicas tales como chequeo de consistencia, se aplican para reducir el espacio de búsqueda, y 2) la etapa de búsqueda, donde se explora el espacio de búsqueda, usando técnicas tales como *backtracking* o "vuelta atrás"para encontrar diferentes soluciones. A diferencia del MILP, las variables en CSPs no estan limitadas sólo a tomar valores numéricos. También pueden ser del tipo enumerativo. No hay limitación para el tipo de restricción con la que CST pueda lidiar, mientras que en el MILP, las restricciones deben ser ecuaciones o inecuaciones lineales. Estas ventajas hace muy flexible al CST. A diferencia de métodos de búsqueda local como simulated annealing y tabú search, el CST no va tras la solución optimal, sino la solución factible que satisfaga todas las restricciones. De ahi que CST es bastante adecuado para resolver problemas altamente restringidos. Por otra parte, CST, con respecto a tiempo de resolulción computacional y calidad de solución, tiende a ocasionar problemas.

3.2. Métodos Heurísticos

Mientras que los métodos de solución exacta garantizan encontrar la solución óptima (si existe una), los métodos heurísticos algunas veces encuentran la solución óptima, pero mayormente encuentran simplemente la una "buena"solución. Los métodos heurísticos típicamente requieren mucho menos tiempo y/o espacio que los métodos exactos. La heurística especifica cómo tomar decisiones dada una situación particular; la heurística comprende reglas para decidir que acción tomar. La heurística en programación es referenciada comúnmente como reglas

de programación o reglas de despacho. La definición de estas reglas es muy compleja con frecuencia, y la mayoría es asignada a un tipo especifico de problema con un conjunto muy especifico de restricciones y suposiciones. La heurística puede ser determinística - terminan con el mismo resultado todas las veces - o estocástica - cada vez que se corran podrían resultar de diferente manera. Podrían ejecutar una regla a la vez, o ser capaz de decidir en paralelo. Los algoritmos híbridos pueden combinar múltiples heurísticas. Tradicionalmente los métodos siguen estos pasos: planeamiento, secuenciamiento, luego programación. Algunos métodos usan heurística para buscar el espacio combinatorio de permutaciones en secuencias de tareas, otros usan heurística para determinar asignaciones de tiempos/tareas/recursos factibles durante la generación del programa, y otros usan heurística para combinar secuenciamiento y programación. Unos pocos incluyen la planificación en la generación de programas permitiendo más de un plan y la búsqueda para elegir entre los planes al programar. En el secuenciador típicamente dominan las restricciones de precedencia, mientras que las restricciones de los recursos dominan en el programador. Las soluciones híbridas tratan de mantener más de una representación o combinar técnicas de búsqueda múltiple o algoritmos de satisfacción de restricciones.

3.2.1. Programación Heurística

La programación heurística opera sobre un conjunto de tareas y determina cuando cada tarea debe ser ejecutada. Si una tarea debe ser ejecutada en más de un modo de ejecución o en cualquiera de un conjunto de recursos, la heurística también debe determinar cual recurso y/o modo de ejecución debe usarse. Las heurísticas comunes se listan en la tabla 3.1. El programador refuerza la satisfacción de restricciones asignando una tarea a un recurso (o recurso a una tarea) a la vez cuando el recurso se encuentra disponible y la tarea puede ejecutarse. Panwalker y Iskander encuestaron un rango de heurísticas desde reglas de prioridad simple hasta reglas de despacho muy complejas [63]. Davis y Patterson compararon ocho heurísticas estándares de un conjunto de problemas de proyectos de modo único y recursos restringido [18]. Ellos compararon la performance de la heurística con soluciones optimales encontradas por una método de enumeración limitada de Davis y Heidorn [19]. Los resultados mostraron que el MIN

Tabla 3.1: Algunas heurísitcas comúnmente usadas. Las reglas de despacho (una forma de programación heurísitca) deciden cual de los recursos deben recibir tareas cuando llegan a la planta.

Heurística	Qué es lo que hace
MIN SLK	escoge la tarea con el retraso total menor
MIN LFT	escoge la tarea con el tiempo final aproximado aproximadamente mayor
SFM	escoge el modo de ejecución con la duración factible más corta
LRP	escoge el modo de ejecución con la proporción de recursos menor

SLK se desenvolvió mejor. También mostraron que la heurística no se desenvuelve tan bien cuando los recursos están estrechamente restringidos. Lawrence y Morton describieron un enfoque que intentaba minimizar las demoras prorrateadas usando una combinación de proyectos-, actividades-, y mediciones relacionadas con recursos [46]. Los resultados de su enfoque fueron comparados con un gran conjunto de problemas con cientos de tareas distribuidas entre 5 proyectos con varias penalidades por tardanza, duración de actividades, y requerimientos de recursos. Su heurística produjo programas con menor costo de tardanza promedio que las heurísticas normales. En su revisión de técnicas heurísticas, Hildum hace distinción entre enfoques de heurística única y múltiple [32]. Mientras enfatiza la importancia de mantener las perspectivas de programación múltiple, Hildum noto que un programador con heurística múltiple es más capaz de reaccionar a las topologías multi dimensionales en desarrollo de los espacios de búsqueda. Los experimentos de Boctor con heurísticas múltiples claramente muestran los beneficios de combinar el mejor de los métodos de heurística única [9].

3.2.2. Secuenciamiento Heurístico

Mientras que la programación heurística opera en tareas para decidir cuando éstas deben ser ejecutadas, el secuenciamiento heurístico determina el orden en el cual las tareas deben ser programadas. Estas heurísticas son frecuentemente usadas ne combinación con árboles de decisión para determinar cual parte del árbol evitar. Por ejemplo, búsqueda de discrepancia limitada con *backtracking* ha sido usada por William Harvey y Mathew Ginsberg para resolver muy efectivamente algunas clases de problemas de programación cuando la secuencia para tareas de programación es estructurada como árboles de decisión [29]. Sampson

y Weiss diseñaron un procedimiento de búsqueda local para resolver problemas de programación de proyectos de modo único [74]. Ellos describieron una representación específica para el problema, una estructura de vecindad, y un método para buscar dentro de esta vecindad. Su método determinístico realiza muy bien los problemas presentados por Patterson [64].

3.2.3. Enfoques Jerárquicos

Goal Programming o programación basada en la meta ha sido usada par a resolver problemas de programación con objetivos múltiples. Norbis y Smith [59] describen un método para encontrar programas casi-optimales usando niveles de órdenes consecutivas de sub-problemas. Al subir una colección de ordenes através de los niveles, las tareas son reordenadas para acomodar la prioridad de los objetivos en cada nivel. Además de los cambio dinámicos tales como órdenes sorpresa y cambios en la disponibilidad de los recursos, la implementación de Norbis y Smith también permite ingresar datos al usuario durante el proceso de solución.

3.2.4. Enfoques de Inteligencia Artificial

Hildum agrupo los enfoques de inteligencia artificial para programación como sistemas expertos o como los sistemas basados en el conocimiento [32]. Ambos son métodos heurísticos estructurados que difieren en la manera que éstos controlan la aplicación de sus heurísticas. Los sistemas expertos consisten en reglas como base, una muestra de la solución actual, y un mecanismo de inferencia. El mecanismo de inferencia determina como las reglas if-then en la base serán aplicadas a la solución actual con el propósito de ejecutar la búsqueda. La base de reglas puede ser expandida cuando la solución progresa. La base de reglas se agrega explícitamente al problema específico, de ahí que los sistemas expertos son altamente especializados. Los sistemas basados en el conocimiento típicamente dividen el problema en sub-problemas o diferentes vistas. Se definen *agentes*, cada uno de los cuales concierne con un aspecto particular de la solución. Cada agente atrae la solución en la dirección más concerniente a dicho agente. Las variaciones de los algoritmos incluyen combinaciones micro y macro de las modificaciones a las soluciones así como los tipos de atributos con los cuales los

agentes han sido configurados para responder. Hildum distingue entre 3 soluciones comúnmente conocidas de inteligencia artificial, ISIS (Intelligent Scheduling and Information System) [25], OPIS (Opportunistic Intelligent Scheduler) [77], y MicroBOSS (Micro-Bottleneck Scheduling System) [73], basados a partir de reglas que usan para guiar sus búsquedas. Hildum nota que su propio método, DSS (Dynamic Scheduling System) o sistema dinámico de programación, es básicamente un enfoque de heurística dinámica y múltiples atributos que se centra en el problema no resuelto más urgente en cualquier momento dado.

3.3. Métodos Metaheurísticos

Hay varias maneras diferentes de clasificar y describir a los algoritmos metaheurísticos. Dependiendo de las características seleccionadas para diferenciarlas, es posible diversas clasificaciones, cada una de ellas como resultado de un punto de vista específico. Resumiré brevemente la manera más importante de clasificar a la metaheurística.

Inspirados por la naturaleza vs. No inspirados por la naturaleza Quizá la forma más intuitiva de clasificar la metaheurística es basada en los orígenes del algoritmo. Hay algoritmos inspirados en la naturaleza, tales como los Algoritmos Genéticos y Algoritmos de Hormigas, y los no inspirados tales como Búsqueda Tabú y Búsqueda Local Iterada. Esta clasificación no es muy significativa por las siguientes dos razones. Primero, muchos recientes algoritmos híbridos no caben en ninguna de las dos clases (o, caben en las dos a la vez). Segundo, es difícil a veces atribuir claramente un algoritmo a la metaheurística.

Basada en la población vs. búsqueda de punto único Otra característica que puede ser usada para la clasificación de la metaheurística es el número de soluciones usadas al mismo tiempo: El algoritmo trabaja en una población o en un sólo punto a la vez?. Los algoritmos que trabajan en soluciones únicas se llaman *métodos de trayectoria* y comprenden metaheurística basada en búsqueda local, tal como Búsqueda Tabú, Búsqueda Local Iterada y Búsqueda de Vecindad Variable (VNS Variable Neighborhood). Todos comparten la propiedad de describir una trayectoria en el espacio de búsqueda

durante el procesos de búsqueda. La metaheurística basada en la población, por el contrario, realiza procesos de búsqueda que describen la evolución de un conjunto de puntos en el espacio de búsqueda.

Dinámica vs. Función Objetivo Estática La metaheurística también puede ser clasificada de acuerdo a la manera en que hace uso de la función objetivo. Mientras algunos algoritmos mantienen la función objetivo dada en el problema de representación "tal y como es", algunas otros, tales como la Búsqueda Local Guiada (GLS Guided Lecal Search), la modifican durante su búsqueda. La idea detrás de este enfoque es escapar del punto óptimo local modificando el paisaje de búsqueda. De acuerdo con esto, durante la búsqueda, la función objetivo se altera intentando incorporar información recolectada durante el proceso de búsqueda.

Una vs. Varias Estructuras de Búsqueda La mayoría de algoritmos metaheurísticos trabaja sobre una sola estructura de vecindad. En otras palabras, la aptitud ("*fitness*") del paisaje que es buscado no cambia en el transcurso del algoritmo. Otras metaheurísticas, como VNS, usa un conjunto de estructuras de vecindades que dan la posibilidad de diversificar la búsqueda y abordar el problema saltando entre varios paisajes de *fitness* diferentes.

Uso de Memoria vs. Sin Memoria Una característica muy importante para clasificar la metaheurística es el uso que se hace de la historia de la búsqueda, esto es si ésta usa memoria o no. Los algoritmos sin memoria realizan el proceso de Markov, como información exclusiva es el estado actual del proceso de búsqueda para determinar la siguiente acción. Hay varias maneras diferentes de hacer el uso de memoria. Usualmente se puede diferenciar entre estructuras de memoria a corto y largo plazo. La primera usualmente mantiene la pista de los movimientos recientemente realizados, soluciones visitadas, o en general, decisiones tomadas. La segunda es usualmente una acumulación de parámetros sintéticos e índices acerca de la búsqueda. El uso de memoria es hoy en día reconocido como un elemento fundamental para una metaheurística poderosa.

Se enfocará en la metaheurística basada en el paradigma de la búsqueda local. Esto es, dada una solución factible inicial, estas son sucesivamente cambiadas realizando movimientos que alteran las soluciones "localmente". En la subsección siguiente describiremos diferentes maneras de tales definiciones de movimientos que conducen a correspondientes vecindades, Los movimientos deben ser evaluados por alguna medida para guiar la búsqueda. En nuestro caso, la búsqueda local se aplica a cada solución generada por las operaciones genéticas, que se da mediante un algoritmo de búsqueda local genético multiobjetivo [34].

3.3.1. Búsqueda Local Iterada

Iterated Local Search o Búsqueda Local Iterada (ILS). A pesar de su simplicidad, es una metaheurística poderosa que aplica algoritmos de búsqueda local iterativamente para modificar la solución actual. Una descripción detallada de ILS puede ser encontrada en [27]. A grandes rasgos, el ILS funciona de la siguiente manera. Primero se construye una solución inicial localmente optimal, con respecto a la búsqueda local dada. Un buen punto de inicio puede ser importante, si se quiere soluciones de alta calidad. Más importante es entonces definir una perturbación, que es una manera de modificar la solución actual hacia un estado intermedio al cual se le puede aplicar después una búsqueda local. Finalmente, se usa un criterio de aceptación para decidir a partir de cual solución continuará en el proceso.

3.3.2. Simulated Annealing

El Recocido o Templado Simulado, más conocido como Simulated Annealing, es un proceso computacional que simula el proceso físico de tratamiento térmico de materiales. En el recocido, por ejemplo, un metal es llevado a elevados niveles energéticos, hasta que alcanza su punto de fusión. Luego, gradualmente es enfriado hasta alcanzar un estado sólido, de mínima energía, previamente definido.

Simulated Annealing extiende la búsqueda local básica permitiendo el movimiento hacia soluciones inferiores; [43] y [16]. El algoritmo básico de Simulated Annealing (SA) podría ser descrito como sigue: Un movimiento de un candidato es sucesivamente y arbitrariamente seleccionado; este movimiento es aceptado

si conduce a la solución con un mejor valor de la función objetivo que la solución actual. De otro modo, el movimiento es aceptado con una probabilidad que depende del deterioro δ del valor de la función objetivo. La probabilidad de aceptación es usualmente calculada como $\epsilon^{-\delta/T}$, usando una temperatura T como parámetro de control. Esta temperatura T es gradualmente reducida de acuerdo a un programa de enfriamiento, tal que la probabilidad de aceptación de movimientos corrosivos decremente en el transcurso del proceso de recocido.

Desde el punto de vista teórico, el proceso de SA podría proveer convergencia a una solución optimal si algunas condiciones se cumplen (p.e., con respecto a un programa de enfriamiento apropiado y una vecindad que conduzca a un espacio de solución conectado); como en Van Laarhoven y Arts, [83] y [1],los que dan encuestas en SA con resultados teóricos como tema principal. Como los ratios de convergencia son usualmente muy lentos, en la práctica uno aplica típicamente programas de enfriamiento más rápidos (renunciando a la propiedad teorica de convergencia).

Si seguimos la parametrización robusta del proceso general de SA como lo describe Johnson [41], T es inicialmente alto, que permiten muchos movimientos inferiores a ser aceptados, y es gradualmente reducido mediante multiplicación por un pa̍rametro $\alpha < 1$ de acuerdo con un programa geométrico de enfriamiento. En cada temperatura los candidatos de movimiento size Factor $\times$ $|N|$ se prueban ($|N|$ denota el tamaño de vecindad actual), antes que T se reduce a $\alpha \times T$. La temperatura de inicio se determina como sigue: dado un parámetro initial Acceptance Fraction y basado en una corrida de prueba inicial, la temperatura de inicio se configura de tal forma que la fracción de movimientos aceptados es aproximadamente initial Acceptance Fraction. Otro parámetro, frozen Acceptance Fraction se usa para decidir si el proceso de recocido se congela (fronzen) y debe ser terminado. Cada vez que una temperatura se complete con menos del frozen Acceptance Fraction de los movimientos candidatos aceptados, un contador se incrementa por uno. Este contador se resetea cada vez que una mejor nueva solución se encuentra. El procedimiento se termina cuando el contador alcanza 5. Entonces, es posible recalentar la temperatura para continuar la

búsqueda realizando otro procesos de recocido.

3.3.3. Tabu Search

El paradigma básico de la tabú search [26] es usar la información concerniente al historial de búsqueda para guiar la búsqueda local hacia la superación de la optimizacion local. En su forma básica, se hace dinámicamente prohibiendo ciertos movimientos durante la selección de vecindad para diversificar la búsqueda. Las diferentes estrategias de la búsqueda Tabú difieren especialmente en la manera que se define el criterio Tabú, tomando en consideración la información acerca del historial de búsqueda (realizar movimientos, soluciones atravesadas)

En general, la búsqueda tabú procede seleccionando en cada iteración el mejor movimiento admisible. Una vecindad, con su respectivo movimiento, se llama admisible, si no es tabú o si un criterio de aspiración se cumple. Dicho criterio puede conllevar a un posible estado de tabú. El criterio de aspiración usado puede ser, por ejemplo, permitir todos los movimientos que permiten conducir a una vecindad con un mejor valor de la función objetivo que la encontrada anteriormente. La búsqueda tabú estricta encierra la idea de prevenir entrar en un ciclo de soluciones atravesadas anteriormente. Esto es, la meta básica de la búsqueda tabú estricta es proveer necesidad y suficiencia con respecto a la idea de no volver a visitar una solución visitada anteriormente. De acuerdo con esto, un movimiento es clasificado como tabú si y sólo si éste conduce hacia un vecindario que ya ha sido visitado durante una parte previa de la búsqueda. Para lograr este criterio, se almacena (aproximadamente) información acerca de todas las soluciones visitadas usando una función "hash"que define una transformación no inyectiva desde el conjunto de soluciones a los numeros enteros; [84]. Esto es, se chequea para cada vecindad si un código *hash* respectivo esta incluido en la data de la trayectoria. Dado un vector $(r_1, ..., r_n)$ de enteros pseudo-aleatorios, usamos códigos *hash* $\Sigma_{i-1}^{n} r_i \pi_i$. Cuando el código hash de dos soluciones diferentes puede ser el mismo al colisionar, los movimientos podrían ser innecesariamente considerados tabú. Sin embargo, como se muestra en [4], este efecto arbitrario usualmente no afecta la búsqueda negativamente. Mientras que la búsqueda estricta tabú es fácil de aplicar, ya que no necesita calibración de parámetros, su criterio tabú es frecuentemente muy débil de proveer una diversificación suficiente

en el procesos de búsqueda.

La búsqueda tabú más comúnmente usada es aplicar una memoria que almacena los movimientos en base a su grado de cuan reciente ocurran, más exactamente, mueve atributos del pasado reciente. La idea básica de tal enfoque de búsqueda estática tabú es prohibir una adecuada inversión definida de movimientos realizados para un periodo dado. Considerando un movimiento realizado (p_1, p_2), se almacena, dependiendo de la vecindad usada, atributos del movimiento que representen las esquinas insertadas en la lista estática tabú de longitud l. Para obtener el status de tabú de un movimiento, se debe chequear si las esquinas a ser borradas son restringidas en la lista tabú. Cuando se aplica movimientos multi-atributos, hay diferentes maneras de definir el criterio tabú: un movimiento podría ser clasificado como tabú cuando al menos uno, dos, o, dependiendo de la vecindad usada, 3 o 4 atributos de este movimiento estén contenidos en la lista tabú. Para este propósito, toda este criterio se puede habilitar usando un parámetro *umbral tabú (#)*. Este define el numero de atributos de un movimiento que tienen que ser contenidos en la lista tabú para que de acuerdo a esto el movimiento sea considerado tabú.

La aplicación de búsqueda estática tabú se complica con la necesidad de configurar el tamaño de la lista con un valor que sea apropiado para el problema a ser resuelto. El tabú reactivo apunta a una adaptación automática de su parámetro durante el proceso de búsqueda [8]. La idea básica es incrementar el largo de la lista tabú cuando la memoria tabú indica que la búsqueda esta volviendo a visitar soluciones anteriormente atravesadas. El algoritmo aplicado en [4] se puede describir como sigue: comienza con una lista tabú de longitud l de 1 e incrementa hacia $min\{max\{l-2, l{,}2\}, u\}$ cada vez que una solución halla sido repetida, tomando en cuenta un limite superior adecuado u. Si no ha habido repetición para algunas iteraciones, se decrementa adecuadamente hacia $max\{min\{l-2/1{,}2\}, 1\}$. Para lograr la detección de una repetición de la solución, se aplica una memoria basada en la trayectoria usando código *hash* como se describió antes. Como noto Battiti [8], puede ser apropiado diversificar explícictamente la búsqueda hacia nuevas regiones del espacio de búsqueda cuando la memoria tabú indique que se este atrapado en cierta región de dicho espacio. La información de la trayectoria provee de los medios para detectar tales situaciones. Como mecanismo de

disparo correspondiente, se usa la combinación de por lo menos 3 soluciones, cada una siendo atravesada 3 veces. La estrategia de diversificación simple usada en [4] consiste en realizar arbitrariamente algunos movimientos de la vecindad correspondiente.

3.3.4. Algoritmos Evolucionarios

Los Algoritmos de Computación Evolucionaria (EC), o EA (Evolutionary Algorithms) son inspirados por la capacidad de la naturaleza de evolucionar seres vivientes bien adaptados a su ambiente. Los algoritmos EC pueden ser brevemente caracterizados como modelos computacionales de procesos evolutivos. En cada iteración, un numero de operadores se aplica a los individuos de poblaciones actuales para generar los individuos de la población de la siguiente generación (iteración). Usualmente los algoritmos EC utilizan operadores para recombinar 2 o más individuos para producir un nuevo individuo llamados operadores de *recombinación* o *cruce (crossover)*, y también operadores que causan una autoadaptación de individuos llamados operadores de *mutación* o *modificación* (dependiendo de su estructura). La fuerza de los EA radica en la *selección* de los individuos basada en su aptitudes, *fitness*, (esto puede ser el valor de una función objetivo o el resultado de un experimento de simulación, o algún tipo de medida de calidad). Los individuos más aptos tiene una mayor probabilidad de ser escogidos como miembros de las siguientes iteraciones de población (o como padres para la generación nueva de individuos). Esto corresponde al principio de *supervivencia del más fuerte* en lo que concierne a evolución natural. Esta es la capacidad de la naturaleza de adaptarse a ambientes cambiantes, en lo que se inspiran los EA. Hay una variedad de ligeramente diferentes EA propuestos a lo largo de los años. Básicamente caen en 3 diferentes categorías que han sido desarrolladas independientemente una de la otra. Éstas son Programación Evolucionaria (EP) desarrollada por Fogel [48] y [24], Estrategias Evolucionarias (ES) propuestas por Rechenberg en 1973 [70] y Algoritmos Genéticos iniciados por Holland en 1975 [35]. EP viene del deseo de generar inteligencia en las máquinas. Mientras que originalmente EP se propuso para operar en representaciones discretas de máquinas de estados finitos, la mayoría de las variantes presentes son usadas para problemas de optimización continua. Más tarde también se presen-

taron las variantes de ES, mientras que los GA son usados frecuentemente para abordar problemas de optimización discretos. Sobre los años hubo muchas investigaciones sobre los métodos EC. Entre éstos están los presentados por Bäck [5], Fogel [21] y Michalewicz [57]. Calégary en [60] trata de dar taxonomia a los EA. Una introducción con orientación a una *optimización combinatoria* en el campo de los EA fue dado por Hertz [31], quien da una buena revisión de los diferentes componentes de los EA y de las posibilidades de definición. El algoritmo de la Figura 3.2 muestra la estructura basica de cada EA.

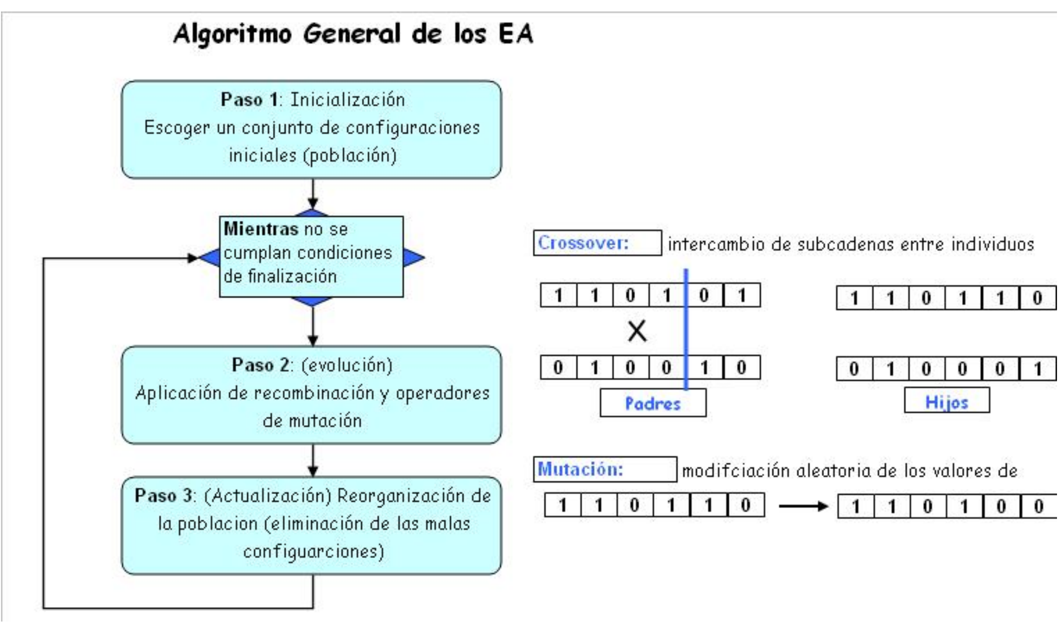


Figura 3.2: Esquema general de un algoritmo evolucionario

En este algoritmo, P denota la población de individuos. Una población de nuevos individuos (*offspring*) se genera mediante los operadores de *recombinación* y *mutación* y los pobladores de la nueva generación se *seleccionan* a partir de la unión de la antigua población y la población *offspring*.

Lawrence y Davis fueron uno de los primeros en sugerir el uso de algoritmos genéticos para la programación. En su trabajo [20], Davis nota las bondades del uso de un método de búsqueda estocástica debido al tamaño del espacio de búsqueda y sugiere una representación indirecta en la que el algoritmo genético opera en una lista que fue decodificada para formar el programa real. En particular, él noto la importancia de mantener la factibilidad de representación.

Davis observó que muchos problemas reales de programación incorporan capas de restricciones críticas que son frecuentemente difíciles, sino imposibles, de representar usando técnicas de programación matemática tradicionales. Nótese que las soluciones basadas en el conocimiento son típicamente determinísticas, por lo que son susceptibles a bloquearse dentro de regiones sub-optimales del espacio de búsqueda; por lo que Davis sugirió que los algoritmos genéticos, debido a su naturaleza estocástica, podrían evitar soluciones sub-optimales.

A partir del trabajo de Davis,numerosas implementaciones han sido sugeridas no sólo para el problema del *flowshop* sino que también para otras variaciones generales del problema de programación con recursos restringidos. En algunos casos, la representación para una clase de problemas puede ser aplicada para otros también. Pero en la mayoría, la modificación en las definiciones de las restricciones requiere de una representación diferente.

Ralf Bruns resume los enfoques de la programación de la producción en 4 categorias: directa, indirecta, independiente del dominio, y especifica del problema [11]. La mayoría de enfoques de algoritmos genéticos utilizan representaciones indirectas. Estos métodos son caracterizados por representaciones tradicionales binarias [58] y [17] o representaciones basadas en las ordenes [80]. Información especifica del problema fue usada en algunos aspectos de los métodos indirectos para mejorar la performance, pero estos seguían siendo basadas en listas o en ordenes, requiriendo la transformación del genoma al programa, y en algunos casos requería de un constructor de programas como en [6]. Una representación directa propuesta por Kanet usa una lista de trios órden-maquina-tiempo, pero, como lo noto Bruns, no era extensible [42]. Notando la relación inversa entre la generalidad de un algoritmo y su performance, la representación de Bruns fue afinada para "desenvolverse tan eficiente como sea posible en el problema de programación de la producción". Esta representación directa, específica del problema, usaba una lista de asignaciones de órdenes en las que la secuencia de órdenes no era importante.

Bagchi comparó tres representaciones diferentes y concluyó que cuanto más específica sea la información del problema, incluida en la representación, mejor se desenvolvería el algoritmo [6].

Más recientemente, Philip Husbands subrayo el estado del arte en algorit-

mos genéticos para programación [37]. Husbands notó la similitud entre la programación y los problemas basados en el secuenciamiento tales como el problema del vendedor viajante. El también referenció otros problemas NP-hard tales como problemas de layout (diseño de esquemas) y empacamiento que son similares a la formulación de *jobshop*.

Es necesario notar también el trabajo de Cleveland y Smith en el que se comparan 3 modelos: una versión puramente secuencial, un modelo con tiempos de despliegue, y un modelos con costos de trabajo en progreso [17]. Cuando se usaron las funciones objetivo más realistas, el modelo de puro secuenciamiento se desenvolvió pobremente mientras que el modelo con información del programa encontró significativamente mejores soluciones.

En el trabajo presentado aquí, ensayaremos un enfoque secuencia-luego-programa, del tipo híbrido. Son dos los tipos de formulación, una continua basada en Resource-Task-Network, y la otra orientada a la programación por lotes, basada en reglas de precendencia general.Se utiliza un algoritmo híbrido, que podria ser clasificado como Genetic Memetics, ya que aplica una búsqueda local a cada solución generada por las operaciones genéticas.

La Figura 3.3 se puede observar la agrupación que he realizado de los diferentes métodos de los que se habla arriba.

Varios autores han implementado diversas soluciones híbridas. Syswerza y Palmucci combinaron un secuenciador GA con un programador determinístico con operadores especiales basados en las órdenes [79].

Los algoritmos genéticos han sido usados para evolucionar la heurística en programación. Hilliard implemento un sistema clasificador para mejorar la heurística para determinar la secuencia de actividades que deben enviarse a un programador (analogo a decidir cual camino escoger en un árbol de decisiones) [33]. El sistema de Hilliard descubrió la heurística con el "tipo de trabajo incrementando su duración"para el problema de programación de planta con una sola maquina, sin restricciones en la ordenes, y de una operación por trabajo.

Dorndorf y Pesch usaron un algoritmo genetico para encontrar la secuencia optimal de reglas de decisiones locales para ser usadas con los algoritmos tradicionales de búsqueda para un rango de problemas de programación *jobshop* estáticos y determinísticos. Su método encontró *makespans* más cortos y más rápidamente que el procedimiento de cambio de *cuello de botella* de Adams, Balas y Zwack [7] o el metodo de *simulated annealing* de Lenstra [2].

Husbands anotó el tema de acoplar la programación con la planificación implementando un sistema integrado en el cual los planes de proceso evolucionaban y luego eran combinados mediante un programador para construir programas que puedan de ahi ser evaluados [38]. Este trabajo fue importante por la integración entre la planificación y la programación; en muchos casos un cambio en el programa necesita un cambio en el plan o resultado y en la posibilidad de optimización modificando las restricciones, entonces un sistema adaptativo e integrado encaja bien para aplicaciones reales de programación.

Varios trabajos han sido escritos describiendo las implementaciones de algoritmos genéticos en paralelo, pero la mayoría de estos son extensiones directas de algoritmos genéticos seriales y ofrecen poca mejora en su performance algorítmica. La "paralelización"distribuyendo el computo acelerara la ejecución, pero se requieren operaciones evolutivas adicionales tales como la migración para mejora de calidad de la solución.

Las aplicaciones de algoritmos genéticos para problemas de programación se resumen en la Figura 3.4. Los diagramas de la primera columna ilustran la representación basica usada en cada caso. Nótese que estas representaciones no

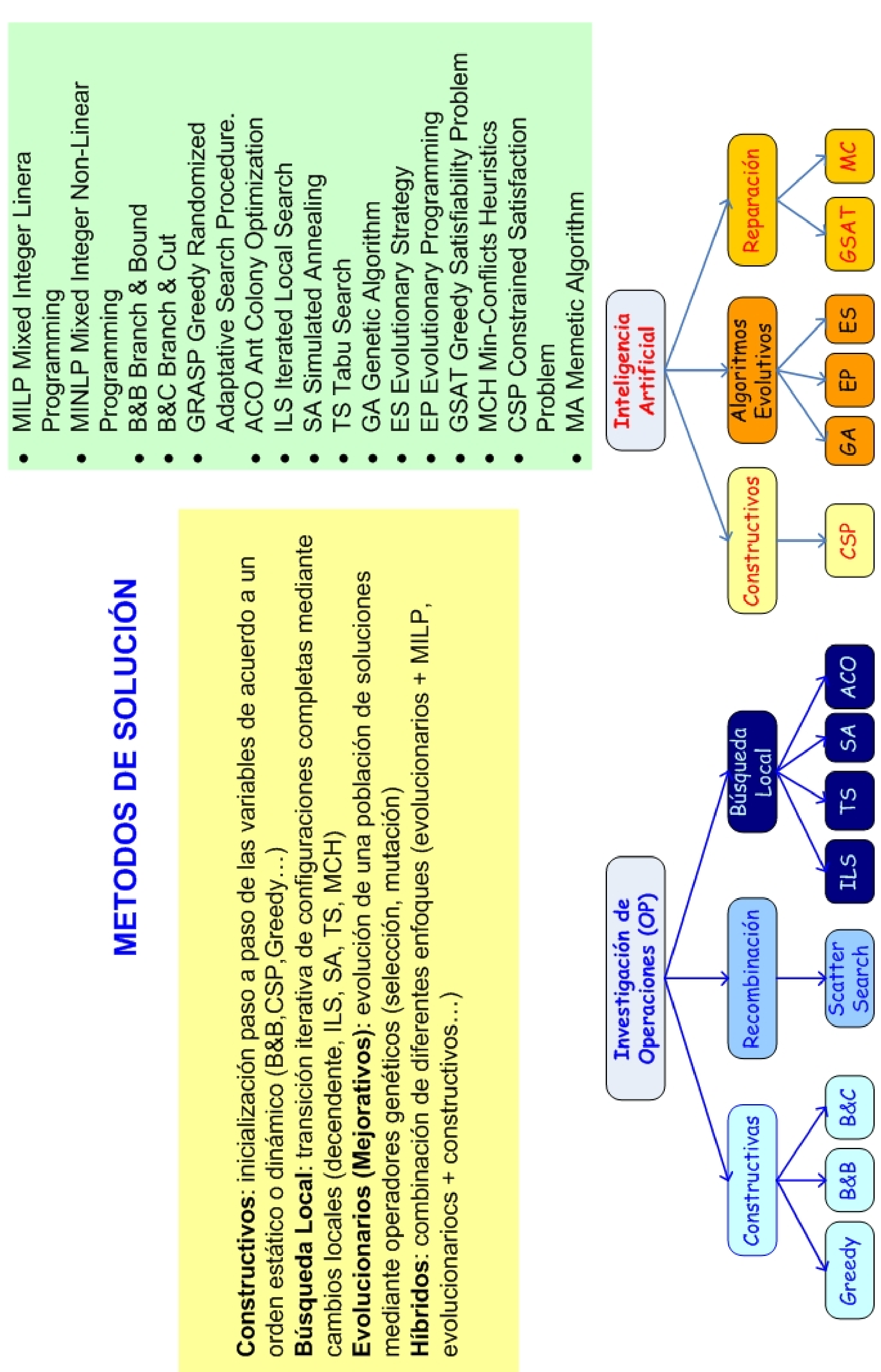


Figura 3.3: Clasificación de los métodos de resolución

fueron completamente diseñadas para resolver el mismo problema. Muchas de estas representaciones requieren de operadores específicos de cada una de éstas.

A pesar que alguno de estos métodos puede ser extendido para hacerlo, sólo la representación de Tseng y Mori considera múltiples modos de ejecución. Ninguna considera disponibilidad de recursos no uniformes. La mayoría requiere modificaciones significativas para acomodar la partición de los trabajos.

Ninguno de los algoritmos evolucionarios puede determinar si un programa es factible. Los problemas para los cuales éstos métodos se diseñaron, tenían todos soluciones factibles, pero en problemas reales, la factibilidad no es garantizada. Por ejemplo, en un problema de programación de proyectos, si todos los recursos están disponibles en cantidades constantes, un programa factible siempre puede lograrse simplemente extendiendo la duración del proyecto hasta que todos los recursos queden libres de restricciones. Cuando los recursos están disponibles sólo en intervalos específicos, no existe tal garantia de factibilidad.

A pesar que Husbands comparo el problema de programación con otros métodos basados en la secuencia, su analogía no es del todo apropiada. El problema de programación es sólo un problema de secuenciamiento cuando es visto sólo del punto de vista tradicional de la investigación de operaciones, p.e. qué debe ser programado después?. Mientras que esta perspectiva es completamente apropiada para determinado tipo de problemas de programación, hay otras que la mayoría no han sido explotadas con algoritmos genéticos.

Los operadores específicos del problema frecuentemente mejoran la performance del algoritmo genético, pero también lo hace la definición de la representación específica del problema. La clave es definir una representación que es generalmente suficiente para acomodar todas las instancias del problema que uno desea resolver, aunque no sea lo suficiente especifico para realmente funcionar.

Representación	Característica	Referencia
Orden 6 · Orden 1 · Orden 4 ...	Lista de ordenes a ser programadas en un job-shop.	[Syswerda 91]
101100111100...	Representación binaria en la cual cada bit determina el orden del par a ser ejecutado primero en una máuina dada.	[Nakano 91] [Cleveland 89]
Orden 2 Plan X · Orden 1 Plan Q · Orden 4 Plan B ...	Lista de pares (orden, plan)	[Bagchi 91]
Orden 5 Plan A Op1 : m3 Op2 : m5 Op3 : m3 · Orden 2 Plan X Op1 : m1 Op2 : m7 · Orden 1 Plan A Op1 : m4 Op2 : m6 Op3 : m7 ...	Lista de (orden, plan, recurso)	[Bagchi 91]
Estación 1 0 : o1 espera o4 inactvio 60 o2 espera o1 espera o4 ... · Estación 2 0 : o1 espera o4 inactivo 60 o5 espera o6 inactivo	Lista de preferencias de orden/tiempo para cada estación de trabajo	[Davis 85]
Orden 5 Maquina 3 00:00 · Orden 2 Maquina 6 01:05 · Orden 1 Maquina 2 00:12 ...	Lista orden/máquina/tiempo	[Kanet 91]
Orden 7 Op7B1:m9:[10,15] Op7B2:m3:[16,17] Op7B3:m6:[18,22] · Orden 9 Op9A1:m9:[10,15] Op9A2:m3:[16,17] ...	Lista completa de información de orden	[Bruns 93]
Tarea 2, modo 5 Orden 5 01:00-04:30 · Tarea 5, modo 1 Orden 1 00:30-00:560 ...	Lista de (modo-actividad,orden,tiempo de incio-fin)	[Mori y Tseng 96]
Proyecto 1 Actividad 2 Modo 2 · Proyecto 1 Actividad 3 Modo 4 · Proyecto 2 Actividad 2 Modo 1 ...	Lista de (poryecto,actividad,modo)	[Tseng y Mori 96]

Figura 3.4: Resumen de varias formulaciones de algoritmos genéticos. Las referencias en la Figura son representativas del tipo de solución; este diagrama no contiene una lista exhaustiva de publicaciones de los trabajos. La mayoría de las representaciones son basadas en las órdenes, p.e. el orden en el cual los items aparecen en la lista es parte de la estructura del problema

CAPÍTULO IV

Aplicación de la Solución Propuesta

En este capítulo se detalla el método a emplear para abordar todos los campos de nuestro problema. Ver Figura 4.1, donde se resume los métodos escogidos, previamente descritos.

La Figura 4.1 resalta 4 bloques. Para la resolución del problema, se parte, como ya he mencionado, del esquema del *flowshop* en el modelado del nivel de control de planta. Se usa una formulación de tiempo continuo para el problema de optimización. Usando las reglas de precedencia general, orientadas al lote o batch. En el modelo presentado, se puede ver que el inicio de las diferentes tareas a programar se pueden dar en cualquier momento del tiempo dado, independientemente de intervalos de tiempo. Esto es resultado directo de una formulación contínua y orientada al batch. Dado la orientación al lote que se le ha dado a la problemática, he dividido la resolución en dos etapas, en donde la primera se halla el tamaño del lote a producir, y en la segunda se hace la programación de dicho trabajo o procesamiento del lote.

El trabajo luego pasa a tratar el bloque del programador. Se puede distinguir dos clases de programación: una la predictiva y otra la reactiva. El problema de programación incluye crear un programa de procesos de producción (*programación predictiva*) y adaptar un programa existente debido a los eventos del entorno existente (*programación reactiva*). En la Figura 4.1 estoy mostrando el uso de la metaheurística para dedicarse a la programación predictiva, asignaciones de recursos y tiempos, mientras que el manejo de la incertidumbre para la programación reactiva lo manejo con modelos difusos o *fuzzy*. La programación reactiva maneja la incertidumbre ajustando el programa a partir de la realización de parámetros de incertidumbre para eventos inesperados. En la metaheurística usada, he escogido un método híbrido, combinando los algoritmos genéticos junto con un método de búsqueda local, iterativo (Iterative Local Search). La mezcla convierte al método en un algoritmo memético.

El resultado del bloque programador es un programa de producción óptimo, a corto plazo, el cual puede ser variado tomando en cuenta las incertidumbres del entorno a través de la variación de los representaciones *fuzzy*. La explicación de la resolución se detalla en las siguientes secciones del presente capítulo.

4.1. Modelo del Problema

Las consideraciones hechas cuando se modela un problema determinan las variaciones de dicho problema que el modelo podrá soportar. Las siguientes sub secciones se listará las consideraciones sobre las tareas, recursos y objetivos hechas

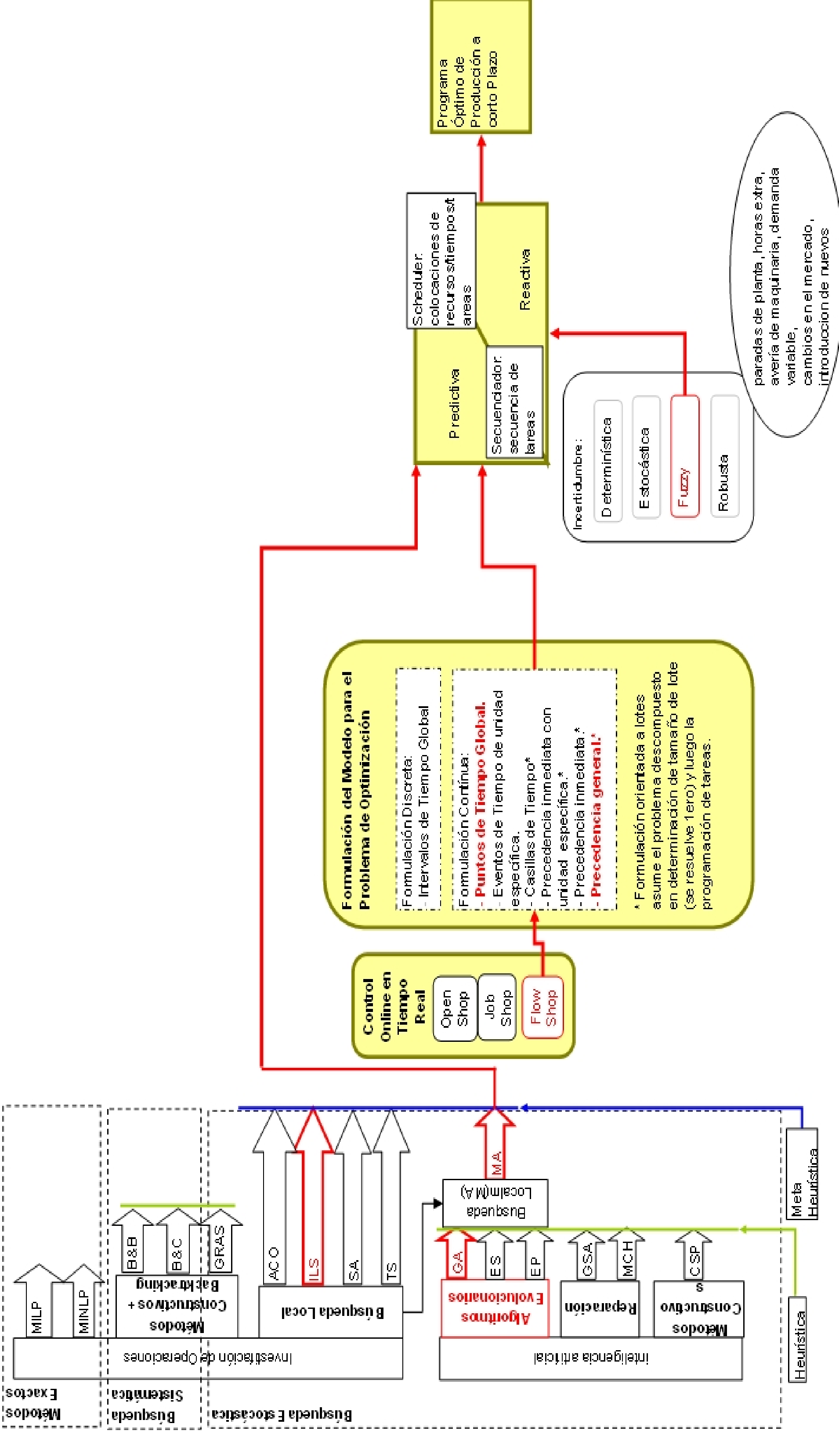


Figura 4.1: Solución elegida

en el presente trabajo. La satisfacción de las restricciones determina la factibilidad de una solución, mientras que la satisfacción de los objetivos determina la *optimización* de una solución.

4.1.1. Consideraciones sobre Tareas

Modos de Ejecución En cuanto a los llamados modos de ejecución, para el caso de la refinería no se considera múltiple. En otras palabras, solo se considera un solo modo. Mayormente en la industria del mecanizado y fabricación de productos más elaborados, como por ejemplo, celulares, se debe considerar más de un modo de ejecución para las maquinas como tornos y fresadoras. En nuestro caso, el arranque de planta, durante el cual varían los tiempos de procesamiento y residencia, podría ser un modo adicional a considerar. Sin embargo, para simplificar el modelado, decidi pasarlo por alto.

Precedencia y Traslape Dado el tipo de producción, no se permite el traslape entre lote y lote. Sin embargo, el inicio de cada procesamiento de lote esta representado por un número difuso, lo que permite cierta flexibilidad para el inicio de tareas simultáneas.

Partición de la Tarea Si entendemos por tarea a cada lote, podemos entender como partición a dividir el tamaño del lote. En el presente trabajo se considera un tamaño de lote fijo una vez calculado. Implementar la capacidad de tamaño de lote variable es cuestión de adicionar unas líneas de código, que permita al programa subdividir el lote del trabajo que no se halla podido colocar. Sin embargo, dado la representación genetica escogida, esto resultaría en un tamaño de cromosoma variable, lo que acompleja la resolución.

Derecho Preferente Entendamos este criterio con este ejemplo: si Juan para de hacer su dibujo en el tablero después de completar su diseño de la pantalla de un celular, luego trabaja en en la electrónica de la misma. Pero si alguien retoma la actividad de diseño en ese punto, tendra que aprender los estándares de diseño, asi que la tarea tomara más tiempo que si Juan hubiese continuado en ella. Las tareas pueden ser definidas de tal forma que sus recursos podrían ser usados en una tarea diferente. Aquí, dada la naturaleza *flowshop*, ningún recurso o tanque puede ser usado, para otra

tarea a la que no esta asignada. Es decir, una vez que un tanque se asigno a procesar un producto, tiene que esperar a que ese producto sea vaciado. Lo que si se considera es usar el mismo tanque para procesar un producto diferente, mas dentro de la clase del mismo anterior. La refinería separa los productos de acuerdo a grasas y aceites. Por ejemplo, los tanques asignados a grasas, pueden variar su contenido dentro de ese tipo de producto. Lo mismo para los aceites.

Restricciones Temporales Estas restricciones se definen por las fechas de entrega. Si una tarea se ejecuta muy por adelantado o demasiado tarde con respecto a su fecha de entrega, se considera como solución infactible.

Restricciones de Asignación En el caso de la refinería, esto se entiende directamente como la capacidad de los tanques. Es decir, cuantos tanques disponibles se tiene para el lote pendiente a producir.

4.1.2. Consideraciones de los Recursos

Tipo de Recurso Los recursos pueden ser renovables (no consumibles) o no-renovables. En algunos casos los recursos no renovables se convierten en renovables. El recurso más importante en el presente caso es el desodorizador y los tanques de almacenamiento.

Restricciones Temporales Los recursos tienen restricciones temporales. En el caso presentado, es la disponibilidad de cada tanque, que es el momento en que están vacíos y listos para recibir producto. También es el tiempo que demoran en vaciarse o llenarse.

Performance El recurso desodorizador, dado los circuitos auxiliares, intercambiadores de calor, bombas y demás maquinaria, puede variar su performance. Por ejemplo, la performance y tiempos que toma en procesar producto varían si acaba de realizarse un mantenimiento total al sistema o se ha hecho cambio de filtros. Estas variaciones de performance no son consideradas en nuestro trabajo. Además, en el presente trabajo se considera el balance de materia lineal. Es decir, en el proceso de desodorizado, si entra A toneladas, se tiene A toneladas al final. En la practica, hay un 4 a 5 % de pérdida en la

fabricación del producto. Para lidiar con esto, si la demanda pide 34tn de producto, se redondea a mayor, p.e. 40tn, lo cual cubre las no linealidades del procesos. El equipo del desodorizador viene a ser un tipo de columna de destilación, la cual tiene ecuaciones específicas de balance de masa. Esto no se ve en el presente trabajo.

Restricciones de Asignación Esto se refiere directamente, en nuestro caso, a la capacidad de cada tanque.

Dependencia Tarea-Recurso Los recursos podrían estar restringidos para ciertas asignaciones, o pueden ser móviles. En la refinería, se usan tanques sólo para determinadas clases de producto, como ya lo hemos explicado. El algoritmo presentado no resuelve la asignación de tanques.

Atributos de Performance Los recursos tienen costos asociados, asi como calidad, o eficiencia. Para el presente trabajo, los costos de mantenimiento, personal que opera los recursos, sea los tanques o el desodorizador, no entran en la representación del algoritmo.

4.1.3. Consideraciones de los Objetivos

Cualquier medición de los objetivos puede usarse en tanto sea determinada desde un programa completo. Los objetivos para estos casos suelen incluir, minimización del *makespan*, minimización del *tardiness* promedio, maximización del grado de satisfacción de una solución (objetivo), minimización del trabajo-en-proceso, etc. Más adelante abordaremos los objetivos que hemos considerado en el presente trabajo.

4.2. Método de Búsqueda

Los algoritmos genéticos son métodos heurísticos de búsqueda estocástica, cuyos mecanismos se basan de simplificaciones de procesos evolutivos observados en la naturaleza. Dado que operan en más de una solución a la vez, los algoritmos genéticos son buenos tanto en *exploracion* como en *explotación* del espacio de búsqueda. Goldberg [22] provee una descripción comprensiva de los principios básicos de funcionamiento en los algoritmos genéticos, y Michalewicz [56] describe muchos de los detalles de implementación para usar los algoritmos genéticos

con los diferentes tipos de variables. La mayoría de los algoritmos genéticos operan sobre una población de soluciones en vez de una sola solución. La búsqueda genética comienza inicializando la población de individuos. Soluciones individuales, o genomas, son seleccionadas de la población, luego se *reproducen* para formar nuevas soluciones. El proceso de reproducción, implementado tipicamente por la combinación, o *crossover*, material genetico de dos padres para formar material genetico nuevo para una o dos soluciones, confiriendo la data de una generación a la siguiente. Se aplica mutación aleatoria periodicamente para promover la diversidad. Si las nuevas soluciones son mejores que las demás de la población, los individuos de la población se reemplazaran por las nuevas soluciones. Esta descripción se puede ver en la Figura 4.2.

Los algoritmos genéticos operan independientemente de los problemas a los que sean aplicados. Los operadores genéticos son heurísticos, pero más que operar en el espacio definido por el problema en sí (el *espacio de solución*, o *espacio de fenotipos*), los operadores genéticos tipicamente operan en el espacio definido por la representación real de la solución (*espacio de representación* o *espacio de genes*). Además, los algoritmos genéticos incluyen otra heurística para determinar cuáles individuos se reproducirán (*selección*), cuales sobrevivirán a la siguiente generación (*reemplazo*), y como la evolución progresará.

Se ve que los algoritmos genéticos, o GA, han sido aplicados con éxito a varios problemas de optimización [22]. La extensión de los GA a los problemas de optimización multiobjetivo ha sido propuesta en diversas maneras (por ejemplo, en Schaffer [76], Kursawe [45], Horn [40], Fonseca y Flemming [86], [87], Murata y Ishibuchi [85], y Tamaki [28]. En el presente trabajo se usa el algoritmo propuesto por Ishibuchi y Murata [34], que es un algoritmo híbrido de un GA multiobjetivo [85] y un procedimiento de búsqueda local modificado. Muchos algoritmos híbridos de GA y sus algoritmos de búsqueda de vecindad (p.e. búsqueda local, recocido simulado o SA, y búsqueda tabú) se propusieron para problemas de optimización de objetivo unico para mejorar la habilidad de búsqueda de los GA. En estudios previos [61]-[81] se demuestra claramente que la performance de los GAs para los problemas del vendedor viajero y problemas de programación se mejora significativamente combinando algoritmos de búsqueda de vecindades. Mientras se puede esperar una mejora significativa de la performance de los GA multiobje-

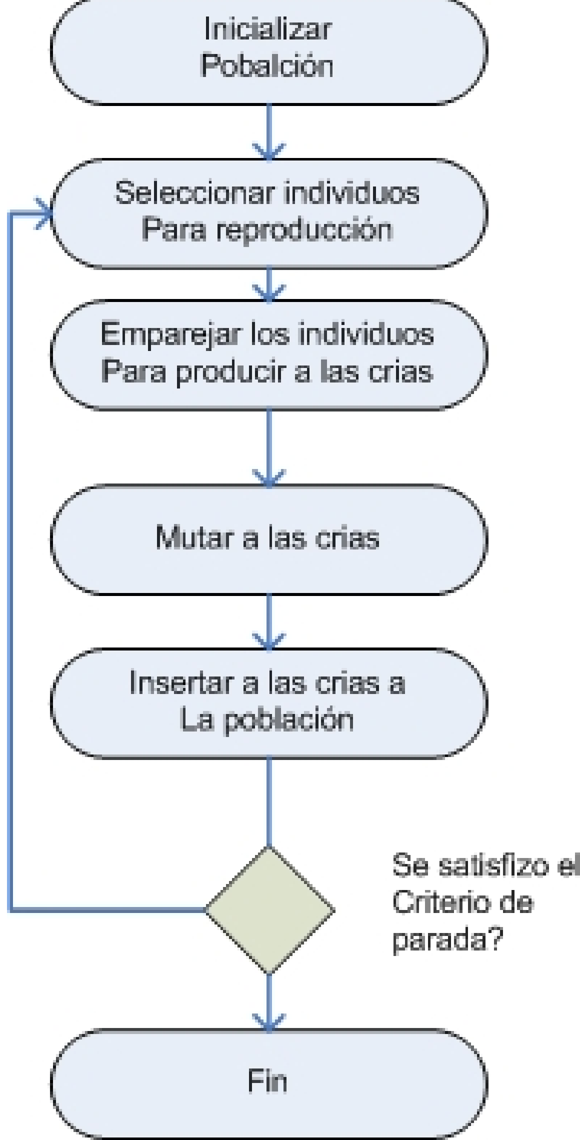


Figura 4.2: Cuadro de flujo del algoritmo genético. Son posibles muchas variaciones , desde varios métodos de selección a una amplia variedad de métodos reproductivos específicos para cada representación.

tivo, Murata y Ishibuchi demuestran que los GA multiobjetivos híbridos también poseen esta cualidad. Cuando se trata de implementar un GA multiobjetivo híbrido, la dificultad radica en determinar la dirección de búsqueda apropiada para los algoritmos de búsqueda de vecindades. El algoritmo usado aquí en el presente trabajo maneja dicha dificultad eficientemente.

El algoritmo genético multiobjetivo de búsqueda local intenta encontrar todas las soluciones no-dominadas del problema de optimización con multiples objetivos. Para explicarlo de manera simple, considere siguiente problema de optimización:

$$\texttt{Minimizar} \quad f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_n(\mathbf{x}) \tag{4.2.1}$$

donde $f_1(), f_2(), ..., f_n()$ son n objetivos a ser minimizados. Se dice que una solución $\mathbf{x}$ es una solución dominada si existe una solución $\mathbf{y}$ que satisface la siguiente relación:

$$f(\mathbf{y}) \leq_{LU} f(\mathbf{x}) \quad \text{y} \quad f(\mathbf{y}) \neq f(\mathbf{x}) \tag{4.2.2}$$

Si una solución no es dominada por ninguna otra de las soluciones del problema multiobjetivo, esa solución se le denomina solución no dominada. Es usual que los problemas de optimización tengan varias soluciones no dominadas.

La meta del algoritmo híbrido es no determinar no sólo una sino todas las soluciones no dominadas del problema de optimización multiobjetivo. Cuando se aplica GA a dicho problema, se tiene que evaluar un valor objetivo o *fitness* para cada solución. Definimos una función fitness de la solución $\mathbf{x}$ mediante la siguiente suma de n objetivos:

$$f(\mathbf{x}) = w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x}) ... w_n f_n(\mathbf{x}) \tag{4.2.3}$$

donde $w_1, ..., w_n$ son pesos no negativos para n objetivos, que satisfacen la siguiente relación:

$$w_i \geq 0 \quad \texttt{para} \quad i = 1, 2, ..., n \tag{4.2.4}$$

$$w_1 + w_2 + ... + w_n = 1 \tag{4.2.5}$$

Si se usan valores constantes para los pesos, la dirección de la búsqueda por GA son fijos. Cuando la dirección de búsqueda es fija, no es fácil obtener una variedad de soluciones no dominadas.

Un enfoque alterno es escoger uno de los n objetivos como fúncion *fitness* para cada solución. Por ejemplo, Schaffer [76] dividió una población (un conjunto de soluciones) en n sub poblaciones , cada una de las cuales era gobernada por uno de los n objetivos. Kursawe [45] sugiere la idea de escoger uno de los n objetivos de acuerdo a una probabilidad asignada por el usuario a cada objetivo. De ahi que dichos GA tengan n direcciones de búsqueda en Schaffer [76] y Kursawe [45]. Como se ve en la Figura 4.3, las direcciones de búsqueda $\mathbf{w}^a$ y $\mathbf{w}^c$ pueden fácilmente encontrar las soluciones A y D, pero no es fácil encontrar las soluciones B y C.

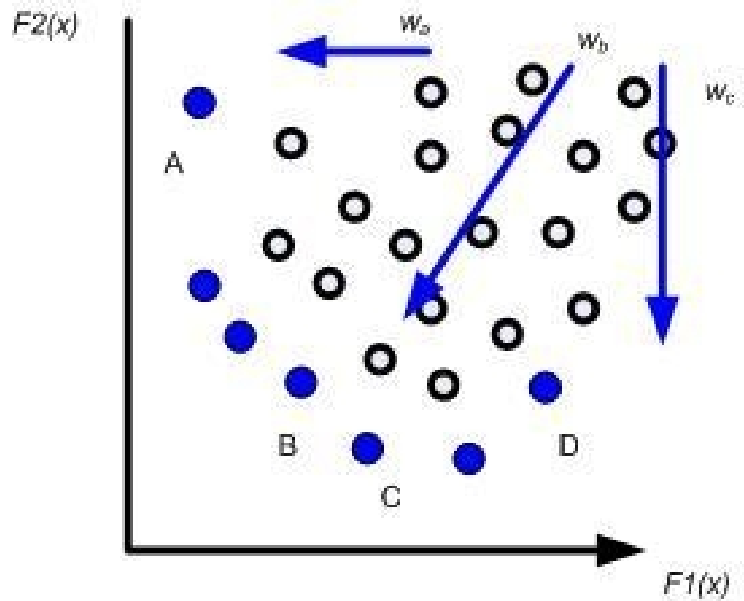


Figura 4.3: Soluciones no dominadas (círculos cerrados) y soluciones dominadas (círculos abiertos)

De las discusiones de arriba, se ve que ni el enfoque de valores de peso constantes ni la elección de un objetivo son apropiados para encontrar todas las soluciones no dominadas del problema multiobjectivo de optimización en 4.2.1. Esto es porque varias direcciones de búsqueda se requieren para encontrar una variedad de soluciones no dominadas. Para realizar varias direcciones de búsqueda,

el algoritmo usa la idea de especificar aleatoriamente los valores de los pesos:

$$\begin{aligned} w_i &= \frac{random_i}{random_1 + ... + random_n} \\ i &= 1, 2, ..., n \end{aligned} \tag{4.2.6}$$

donde $random_1, random_2, ..., random_n$ son números reales no negativos y enteros. Debe notarse que los valores de los pesos se especifican por 4.2.6 cada vez que un par de soluciones padre se seleccionan para generar una nueva solución en la operación crossover. Por ejemplo, cuando N pares de soluciones padre se seleccionan para generar una nueva solución, se especifican mediante 4.2.6 N diferentes vectores de pesos. Esto significa que N diferentes direcciones de búsqueda se utilizan se utilizan en una sola generación del GA. En otras palabras, cada selección (p.e., la selección de dos soluciones padres) es gobernada por su propia función de fitness.

En el algoritmo multiobjetivo genetico de búsqueda local utilizado, se especifica los valores de los pesos mediante 4.2.6 cada vez que un par de padres se selecciona. Estos pesos especificados aleatoriamente son también utilizados en el proceso de búsqueda local, ya que dicha búsqueda se realiza para minimizar la función fitness en 4.2.3. En el algoritmo híbrido, la búsqueda local se aplica a cada nueva solución generada por las operaciones genéticas (p.e., selección, *crossover*, y mutación). La función *fitness* de la nueva solución se define con los valores de los pesos que fueron usados para seleccionas a sus padres. De ahi que, la dirección de búsqueda local para cada solución se determina por la función fitness usada en la selección de los padres. De esta manera, cada solución tiene su propia dirección de la búsqueda local. De ahi que ambas, la operación de selección y la búsqueda local tengan varias direcciones de búsqueda en el espacio de objetivos n-dimensional del problema multiobjetivo de optimización 4.2.1.

Otro problema a tratar en el algoritmo híbrido es como dividir el tiempo computacional disponible entre la búsqueda local y las operaciones genéticas. Si simplemente se combina la búsqueda local con las operaciones genéticas, la mayoría de todo el tiempo computacional se gasta en la búsqueda local y solo una pequeña cantidad de poblaciones se generan por las operaciones genéticas. Esto es porque el procedimiento de de búsqueda local se itera para cara solución generada por las operaciones genéticas hasta que se encuentre una solución localmente opti-

ma. Para prevenir que la búsqueda local se gaste la mayor parte del tiempo, se usa la idea de restringir el número de soluciones examinadas para cada movimiento de la búsqueda local. En procesos de búsqueda local convencionales, la búsqueda local se termina cuando no se encuentra ninguna solución mejor a la actual luego de examinar todo el vecindario de soluciones. Por otra parte, en la búsqueda local utilizada en el presente trabajo, ésta se termina cuando no se encuentra ninguna solución mejor luego de un número pre-especificado (digamos, k) de soluciones seleccionadas aleatoriamente. Esto es, si no hay solución mejor entre entre las k soluciones seleccionadas del vecindario, la búsqueda local para. Cuando se asigna un valor muy pequeño de k (p.e. $k = 2$), la búsqueda local terminara rápido. Por eso la búsqueda local no gasta mucho tiempo y la actualización de las generaciones mediante las operaciones genéticas puede ser iterada muchas veces. Por el contrario, si asignamos un valor grande de k (p.e. $k = 100$), casi todo el tiempo computacional puede ser generado por las operaciones genéticas. De esta manera, uno puede ajustar el tiempo computacional gastado por la búsqueda local. El algoritmo híbrido propuesto se aplica al problema de programación de la producción *flowshop*. La programación en entornos *flowshop* es uno de los problemas más conocidos y hay muchas soluciones propuestas como resolverlos, tal como las hemos descrito brevemente en los capítulos anteriores. Mientras varios enfoques se han propuesto para problemas multiobjetivo en *flowshop*, son algoritmos específicos para cada propósito. Por ejemplo, algunos algoritmos son solo aplicables a problemas de dos máquinas . Otros algoritmos pueden manejar solo el *makespan* y el *total tardiness* como funciones objetivo. Una ventaja del algoritmo utilizado aquí y propuesto por Ishibuchi y Murata [34] por sobre los demás algoritmos es su generalidad. Esto es, es un algoritmo de próposito general aplicable a cualquier problema multiobjetivo de *flowshop* con muchas maquinas y muchos objetivos. De hecho, el algoritmo is aplicable no solo a problemas *flowshop*, sino a cualquier problema multiobjetivo de optimización ajustando las operaciones genéticas y la búsqueda local.

4.3. Funcionamiento del Algoritmo

En esta sección se analiza el algoritmo híbrido para encontrar todas las soluciones no dominadas del problema de optimización de n-objetivos en 4.2.1:

Minimizar $f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_n(\mathbf{x})$.

4.3.1. Operación de Selección

Cuando un par de soluciones padre se seleccionan de la población actual Ψ para generar hijos mediante una operación de crossover, primero los n valores de los pesos $(w_1, w_2, ..., w_n)$ se especifican aleatoriamente por 4.2.6 y luego se calcula el valor *fitness* para cada solución $\mathbf{x}$ en la población actual Ψ como la suma prorrateada de los n objetivos mediante 4.2.3. La probabilidad de selección $P(\mathbf{x})$ de cada solulción $\mathbf{x}$ se define por el metodo de la *rueda de la ruleta* o *roulette wheel method*, usando el escalamiento lineal [22] :

$$P(\mathbf{x}) = \frac{f(\mathbf{x}) - f_{max}(\Psi)}{\sum_{\mathbf{x}\epsilon\Psi}\{f(\mathbf{x}) - f_{max}(\Psi)\}} \tag{4.3.1}$$

donde $f_{max}(\Psi)$ es el valor *fitness* de la peor solución en la población actual Ψ. Esto es, $f_{max}(\Psi) = max\{f(\mathbf{x}) \mid \mathbf{x}\epsilon\Psi\}$. De acuerdo con esta probabilidad de selección, se seleccionan un par de soluciones padre de la población actual Ψ.

Se genera un hijo (p.e. una nueva solución) mediante la operación de crossover de la pareja de padres. Luego se aplica la operación de mutación a esta nueva solución. Se aplica el procedimiento de búsqueda local a la nueva solución después de la mútacion. La búsqueda local intenta minimizar el valor de *fitness* (p.e., la suma de los n-pesos definidos por 4.2.3) de la nueva solución. Esto significa que la dirección de búsqueda local de la nueva solución se define por los valores de los pesos de sus soluciones padres.

Cuando otra pareja de padres se selecciona, el algoritmo especifica los n valores de los pesos $(w_1, w_2, ..., w_n)$ de nuevo. Esto es, usa un vector de pesos diferente para la selección de cada pareja. Como la búsqueda local de una nueva solución usa los mismo valores de pesos que en la selección de sus padres, cada nueva solución tiene su propia dirección de búsqueda local.DE ahi que, la selección y la búsqueda local en el algoritmo híbrido tenga varias direcciones de búsqueda.

4.3.2. Procedimiento de Búsqueda Local

Como se explica arriba, el procedimiento de búsqueda local se aplica a cada nueva solución generada por las operaciones genéticas (p.e., selección, *crossover*,

y mutación) para minimizar el valor de su valor *fitness* en 4.2.3. La búsqueda local también se aplica a las soluciones elite heredadas de la anterior generación. Si se quiere utilizar eficientemente la habilidad de búsqueda global del GA en el algoritmo híbrido, se tiene que reducir el tiempo computacional invertido por la búsqueda local. Esto se realiza, como ya se mencionó, restringiendo el número de soluciones de vecindad examinadas por el proceso de búsqueda local. En el presente algoritmo, se usa el siguiente procedimiento: *Procedimiento de Búsqueda Local Modificado*

1. Paso 1: Especificar la solución inicial $\mathbf{x}$.
2. Paso 2: Examinar la solución vecina $\mathbf{y}$ de la actual solución $\mathbf{x}$.
3. Paso 3: Si $\mathbf{y}$ es una mejor solución que $\mathbf{x}$, entonces reemplazar la actual solución $\mathbf{x}$ con $\mathbf{y}$ y regresar al Paso 2.
4. Paso 4: Si k soluciones vecinas escogidas aleatoriamente de la población actual $\mathbf{x}$ ya han sido examinadas (p.e., si no hay ninguna mejor entre las k soluciones vecinas de $\mathbf{x}$), terminar este procedimiento. Sino, regresar al Paso 2.

Este algoritmo se termina si no se encuentra una mejor solución entre las k soluciones vecinas que son seleccionadas aleatoriamente de la vecindad de la solución actual. Por esto, si se usa un pequeño valor de k (p.e. $k = 2$), el procedimiento de búsqueda local puede terminar rápido. Por el contrario, si usa un valor grande para k (p.e. $k = 100$), el procedimiento de búsqueda local examina muchas soluciones. De esta manera, se puede ajustar el tiempo computacional invertido por el proceso de búsqueda local en el algoritmo híbrido.

4.3.3. Estrategia Elitista

El algoritmo híbrido guarda dos conjuntos de soluciones: una población actual y un conjunto tentativo de soluciones no dominadas. Después de la búsqueda local, la población actual es reemplaza con la población mejorada por la búsqueda local (p.e., la población actual se mejora mediante la búsqueda local) y luego el conjunto tentativo de soluciones no dominadas se actualiza por la nueva población actual. Esto es, si una solución en la población actual es no dominada

por ninguna otra de las soluciones en la población actual ni el conjunto tentativo de soluciones no dominadas, esta solución se adicional conjunto tentativo. Entonces todas las soluciones dominadas por la recién adicionada son eliminadas del conjunto tentativo. De esta manera, el conjunto tentativo de soluciones no dominadas se actualiza en cada generación del algoritmo híbrido.

Desde el conjunto tentativo de soluciones no dominadas, unas pocas soluciones se escogen aleatoriamente como soluciones iniciales para la búsqueda local. Esto es, la búsqueda local se aplica a las soluciones no dominadas seleccionadas asi como a las nuevas soluciones generadas por los operadores genéticos. La dirección de la búsqueda local para cada solución no dominada se determina por su función *fitness* (p.e., la suma prorrateada de los n objetivos) usada en la selección de sus padres. Si una solución no dominada no tiene padres (p.e. si ésta es una solución escogida aleatoriamente como solución inicial), se asignan valores aleatorios de los pesos a la solución no dominada para realizar la búsqueda local. Las soluciones no dominadas seleccionadas aleatoriamente podrían ser vistas como un tipo de soluciones *elite*, porque ellas se adicionan a la población actual sin una operación genética.

4.3.4. Algoritmo Multiobjetivo Genético de Búsqueda Local

Se denota el tamaño de población N_{pop}. Se denota también el número de soluciones no dominadas adicionadas a la población actual como N_{elite} (p.e., N_{elite} es el número de soluciones elite, ver la Figura 4.4). El algoritmo puede ser escrito como sigue:

1. Paso 1 : Inicialización: Generar aleatoriamente una población inicial de N_{pop} soluciones.

2. Paso 2: Evaluación: calcular los valores de los n objetivos para cada solución en la población actual, y luego actualizar el conjunto tentativo de soluciones no dominadas.

3. Paso 3: Selección: Repetir el siguiente procedimiento para seleccionar ($N_{pop}-N_{elite}$) parejas de padres.

    a) Especificar aleatoriamente el peso de los valores $w_1, w_2, ..., w_n$ en la función *fitness* 4.2.3 y 4.2.6.

b) De acuerdo con la probabilidad de selección en 4.3.1 seleccionar un par de padres.

4. Paso 4: Crossover y Mutación: Aplicar el operador crossover a cada pareja seleccionada ($N_{pop} - N_{elite}$) de padres. Se genera una nueva solución a partir de la pareja. Luego aplicar el operador de mutación a la nueva solución generada.

5. Paso 5: Estrategia Elitista: seleccionar aleatoriamente N_{elite} soluciones del conjunto tentativo de soluciones no dominadas y luego adicionarlas a las soluciones ($N_{pop} - N_{elite}$) generadas en el Paso 4 para construir la población de N_{pop} soluciones.

6. Paso 6: Búsqueda Local: aplicar el procedimiento de búsqueda local de la sección anterior a todas las N_{pop} soluciones de la población actual. Las direcciones de búsqueda, que la búsqueda local opera en cada solución, se especifican mediante los valores de pesos en la función *fitness* mediante la cual sus padres fueron seleccionados. La población actual es luego reemplazada con las N_{pop} soluciones mejoradas por la búsqueda local.

7. Paso 7: Prueba: si una condición de parada pre-especificada se satisface, detiene el algoritmo. Sino, regresar al paso 2.

4.4. Representación Genética

A pesar que muchos de las primeros algoritmos presentados en Estados Unidos inicialmente se enfocaban a representaciones de bits (p.e. soluciones codificadas en 1s y 0s), los algoritmos genéticos pueden operar con cualquier tipo de dato. De hecho, las implementaciones más recientes usan representaciones basadas en listas. Sin embargo, sea por listas de instrucciones o por cadenas de bits, cualquier representación debe tener operadores genéticos apropiados y definidos para sí. Mientras la representación determina la frontera del espacio de búsqueda, los operadores determinan como dicho espacio puede ser atravesado.

Para cualquier algoritmo genético, la representación debe ser la mínima y completa expresión de la solución del problema. Una representación minima contiene solo la información necesaria para representar cualquier solución del

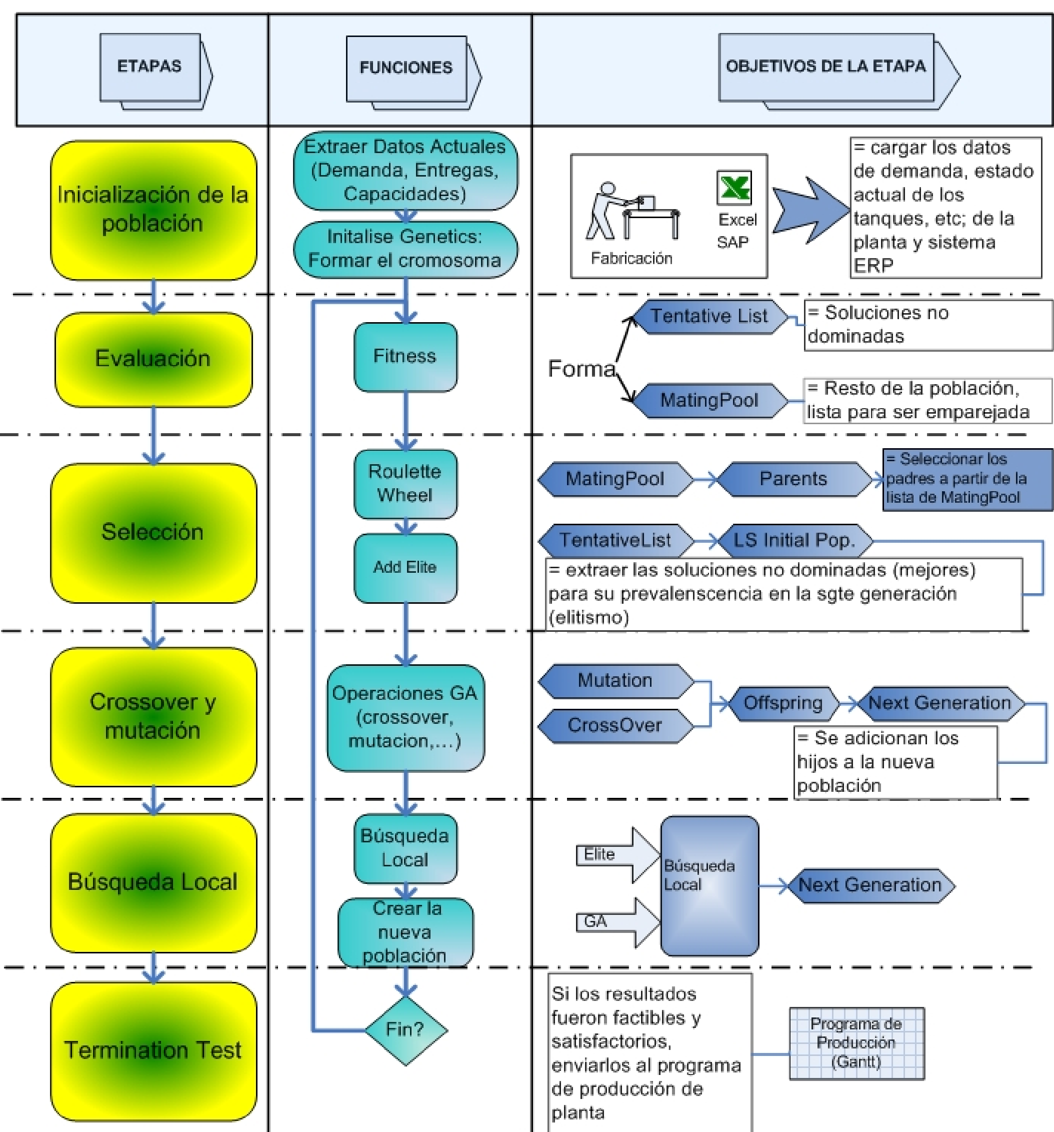


Figura 4.4: Flujo de trabajo del algoritmo

problema. Si una representación contiene más de la información necesaria, el espacio de búsqueda será más largo que el necesario.

En la medida de lo posible, la representación no debe ser capaz de representar soluciones infactibles. Si un genoma puede representar una solución infactible, se debe tener cuidado en la función objetivo para dar credito parcial a los genomas por su material genético "bueno"mientras penalizar suficientemente por ser infactible. En general, es mucho más deseable diseñar una representación que solo pueda representar soluciones factibles de tal manera que la función objetivo mida solo la optimalidad, no la factibilidad. Una representación que incluya infactibles incremente el espacio de búsqueda y solo hace la búsqueda más difícil. La siguiente representación para la programación es una representación mínima que puede representar soluciones con recursos infactibles. Como se ve en la Figura 4.5, el cromosoma (o genoma) consiste de un arreglo de genes. Usualmente en los problemas de programación de la producción continua se trata el problema como un problema de permutación, siendo el cromosoma un arreglo de números enteros que representan el orden en que se inician las producciones de los diferentes productos. Otros enfoques [49] plantean una representación directa de la demanda de cada producto en cada gen. En este caso, lo que se tiene es una producción del tipo batch, o por lotes. Escogí una representación indirecta, ya que requeriría que cada gen sea un objeto (lo que me ayuda a la programación orientada a objetos) con una lista de los datos relevantes para representar exactamente la solución deseada: el programa de producción.

4.4.1. Demanda y Tamaño de Batch

Recordar que el programa debe responder a las preguntas como qué producto, cuánto de éste y en qué tiempo iniciar la producción. Luego de varios intentos la representación quedó de la siguiente manera: el gen representa un batch o lote de producción de un producto dado. Para esto, se suma el total de producto que se requiere para ese turno en un dia dado. La suma se divide entre un número de batches a producir, teniendo en cuenta que solo se procesa cantidades prefijadas de batches: 10, 20, 30, 40, 50 y 60 toneladas (números enteros). La decisión de qué tamaño de batch se va a producir se basa en el criterio de cuanto más grande sea el tamaño, menos cambios y costos de setup se tiene. Para el horizonte total

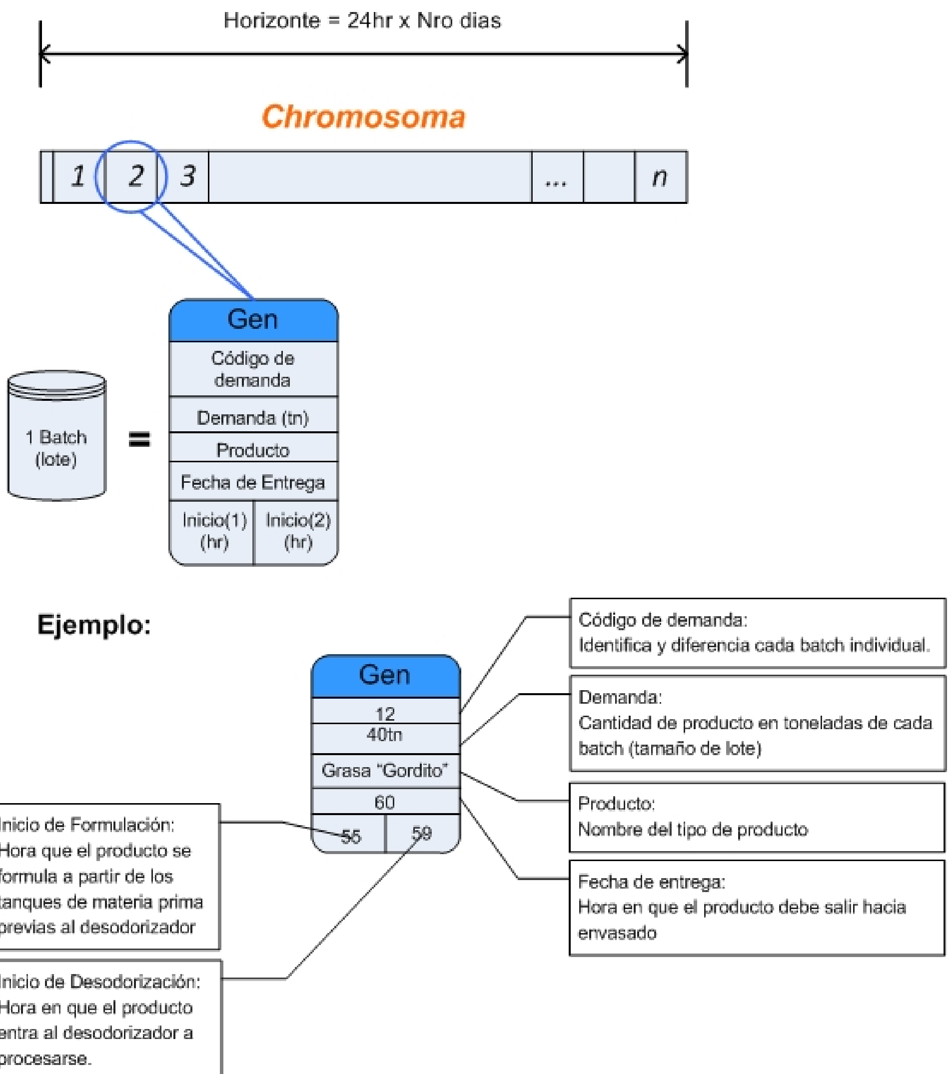


Figura 4.5: Representación del genoma, también llamado cromosoma. Cada gen representa un batch o lote de producción . Y esta compuesto por una lista de datos necesarios para responder al problema que se quiere responder.

de días considerado (7 días en nuestras pruebas), se tiene diferentes números de batches por dia a producir, cada uno con su fecha de entrega, tipo de producto y un código de demanda diferente. El final de turno de cada día pasa a ser la fecha de entrega del batch, de ahi que las fechas de entrega las tengamos como múltiplos de 8 (los turnos son de 8hrs).

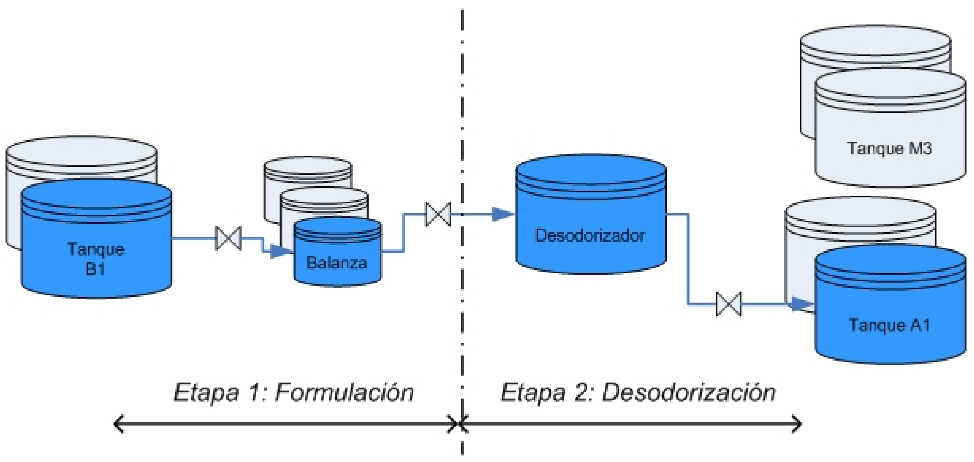


Figura 4.6: Proceso de fabricación simplificado

El código de demanda es un número entero que identifica cada batch. En otros casos, se puede identificar los batches de acuerdo al producto, pero dado que en la refinería se tiene varios batches del mismo producto, y más aun, con las mismas fechas de entrega, se consideró necesario asignar un número característico a cada batch.

4.4.2. Tiempos de Inicio

En la Figura 4.6 se simplificó el proceso a programar en 2 etapas: formulación y desodorización. El proceso considerado inicia cuando se pasan los ingredientes de los tanques de almacenaje (donde se almacenan la soya, palma, etc) a las balanzas de formulación. En éstas balanzas se mezclan de acuerdo a porcentajes de receta los diferentes productos de los tanques de almacenaje. Luego de finalizar la etapa de formulación se pasa a la etapa de desodorizado, donde el producto mezclado pasa a la columna de destilación o desodorizador, y según la cantidad de producto en el *batch* es sometido a diferentes temperaturas, tiempos y presiones. Finalmente pasa del desodorizador hacia los tanques de producto

listo para ser procesado en envasamiento. Teniendo un proceso multietapa, el gen tiene 2 tiempos de inicio, uno para cada etapa. Idealmente, el segundo ocurre justo después de haber procesado el primero, como en 4.4.1. Donde $start_{deso}$ se refiere al tiempo de inicio de la desodorización y $start_{for}$ al tiempo de inicio de la formulación.

$$start_{deso} = start_{for} + \text{Tiempo de Formulación} \tag{4.4.1}$$

Sin embargo, para conseguir el menor tiempo de *makespan* o duración total de producción, se procesan varios batch a la vez, usando todos los recursos (tanques). El potencial cuello de botella es entonces el desodorizador, que solo procesa un batch a la vez. De ahi que un *batch* tiene que esperar a que el desodorizador este desocupado para descargar, y no necesariamente esto ocurre cuando termina su formulación, sino que se le mantiene en espera en las balanzas de formulación listo para ser descargado. En nuestro algoritmo, al inicializar la población, modificamos la ecuación 4.4.1 con 4.4.2 para representar la diversidad en las posibles soluciones.

$$\begin{aligned} start_{for} &= random & ;rand\epsilon[0, fechadeentrega + 12] \\ start_{deso} &= start_{for} + random & ;rand\epsilon[0, 24] \end{aligned} \tag{4.4.2}$$

De esta manera vemos que nuestro gen contiene varios datos. De los cuales, los que nos van a llevar a encontrar la solución factible serán el código de demanda y los tiempos de inicio, por lo que esos son los valores que se modificarán con los operadores genéticos.

4.5. Operadores Genéticos

El uso de un algoritmo genético requiere de la definición de los operadores de inicialización, crossover, y mutación específicos al tipo de data utilizado en el cromosoma.

4.5.1. Inicialización

Como ya se explico en la sección anterior, los tiempos de inicio se definen con 4.4.2. Los códigos de demanda se asignan de acuerdo al orden inicial consecutivo y

son números enteros. Los demás datos, de demanda, fechas de entrega y producto se extraen del sistema ERP (Enterprise Resource Planning) de la refinería y se resumen en la tabla 4.1.

Tabla 4.1: Tabla de decodificación

Código de demanda	Producto	Cantidad (tn)	Fecha de entrega (hr)	Inicio formula -ción	Inicio desodoriza -ción
1	HDACEVEG3	30	8	4	7.5
2	HDACEVEG1	40	8	3	6
3	HDGORDITO	30	16	10	14
4	HDGALLETD	30	16	12	15.6
5	HDACEVEG1	40	24	23	26
:	:	:	:	:	:
N	HDPANISUAVES	30	112	120	125

4.5.2. Crossover

Para esta etapa se utiliza dos tipos de crossover. Como ya se mencionó antes, los valores que son modificables son 3: el código de demanda y los 2 tiempos de inicio de las diferentes etapas, formulación y desodorización.

Como operador de *crossover* para el código de demanda, se utiliza un *crossover* de doble punto ilustrado en la Figura4.7. Solo un hijo se genera a partir de dos soluciones padre.

El crossover de dos puntos se usa para modificar el orden de los batches (genes), más no modifica los tiempos de inicio. Dado que estos tiempos no son números reales, sino de precisión doble (double) y requieren un *crossover* del tipo usado en los algoritmos genéticos contínuos. El algoritmo presentado usa el $linearBLX - \alpha$.

El operador $linearBLX - \alpha$ genera uno o dos hijos, en este caso no más de uno, escogidos aleatoriamente en un segmento de línea que pasa a través de los dos padres, siendo α parámetro del algoritmo evolucionario. Este *crossover* es también conocido en varias denominaciones, de acuerdo a los autores que lo estudiaron, como *arithmetic crossover*, o el *intermediary recombination* para las .Estrategias de Evolución", que es equivalente a $BLX - 0$.

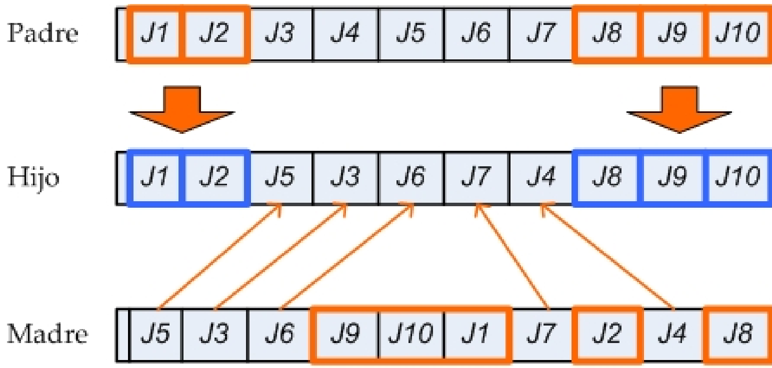


Figura 4.7: Crossover de dos puntos

x y y son puntos correspondientes a dos individuos en el espacio de búsqueda. Se escoge un individuo z resultante del crossover de x e y de acuerdo a una distribución normal en un segmento de linea que pasa a travez de x e y.

$$z = x - \alpha(y - x) + (1 + 2\alpha)(y - x) \cdot U(0, 1) \qquad (4.5.1)$$

donde $U(0, 1)$ indica un número aleatorio entre 0 y 1. Sea I la longitud de intervalo del segmento de linea $[x, y]$, z podría estar sobre el segmento de longitud $I \cdot (1 + 2\alpha)$ centrado en el segmento $[x, y]$.

El $BLX - \alpha$ lineal modifica la varianza de la siguiente forma:

$$[htp]V_c(X) = \frac{(1 + 2\alpha)^2 + 3}{6} \cdot V(X)$$

La varianza después del crossover se decrementa si:

$$\alpha = \frac{\sqrt{3} + 1}{2} \approx 0{,}366$$

En este caso, se dice que el crossover se esta *contrayendo*, y la aplicación del operador iterativa solo lleva a la población a colapsar en su centroide.

4.5.3. Mutación

Para la mutación se escoge un gen aleatoriamente y se le aplica lo que se llama *Neighborhood Mutation*. Este tipo de mutación se desarrollo con un simple objetivo: minimizar el número de modos operacionales, en este caso, lo

que representa cambio de productos. Los cambios de producto implican muchas veces, enjuagues, coberturas con otros productos si éstos son de alta calidad y costos adicionales. Para ilustrarlo, tomemos el ejemplo de la Figura 4.8, una programación con 16 modos de cambio o batches programados.

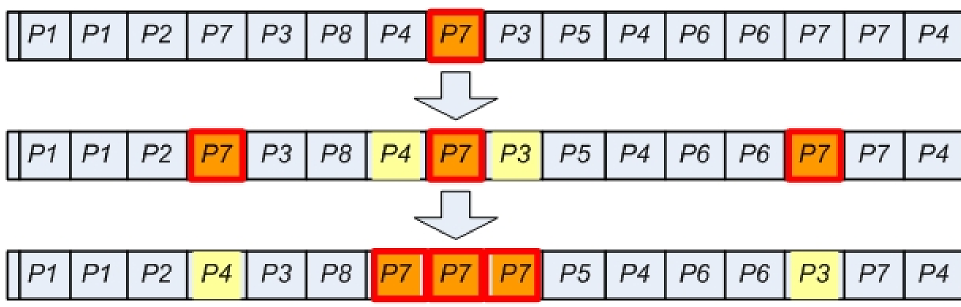


Figura 4.8: Por ejemplo, se muta el gen sombreado con naranja y se realiza los siguientes pasos.

Los pasos a seguir en la mutación son:

1. Escoger un gen aleatoriamente
2. Buscar un gen con igual producto, hacia la izquierda y hacia la derecha.
3. Reemplazar los genes vecinos (seguir buscando hasta encontrar un gen que sea diferente del que inicialmente se escogió) por los encontrados en el paso anterior.

El cromosoma termina teniendo 11 cambios de producto, en vez de los 12 iniciales. Aproximando los batches de productos iguales, la mutación de este tipo logra producir lotes más grandes de estos productos. De esta manera, se minimiza el número de cambios operacionales.

De nuevo, como en el caso de la operación de crossover, tenemos los datos de los tiempos de inicio intactos luego de la mutación por vecindario. Para variar los tiempos de inicio del batch seleccionado aleatoriamente, se aplica una modificación simple. Anteriormente habiá mencionado que el proceso ideal para un batch es iniciar su desodorización justo después de terminar con su formulación. Entonces los tiempos se varían de acuerdo a 4.4.1.

4.6. Formulación de Funciones Objetivo y Manejo de la Incertidumbre

En esta sección detallaremos todo lo que concierne a la evaluación del cromosoma, funciones objetivo y los criterio fuzzy aplicados para la implementación de estas funciones. Como hemos mencionado anteriormente, el manejo de la incertidumbre en una programación reactiva se puede hacer de varias maneras, como la estocástica, etc. Aquí aplicaré los conjuntos difusos para manejar variaciones del entorno y darle flexibilidad a las soluciones.

4.6.1. Tiempo de Procesamiento Difuso

Asumamos que el tiempo de procesamiento de cada batch en cada máquina es dado como un número difuso. Para representar el tiempo de procesamiento difuso, se puede usar varios tipos de funciones de membresía como triangular, trapezoidal, y con forma de campana. La presente tesis emplea las formas triangulares.

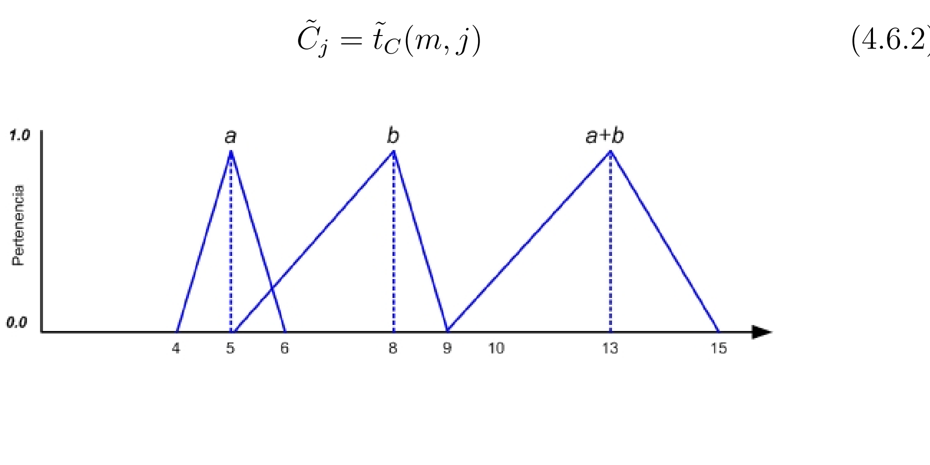
Sea $\tilde{t}_P(i,j)$ el tiempo de procesamiento difuso del trabajo (o batch) j en la máquina i. Cuando los n batches dados son procesados en el orden $\mathbf{x} = (x_1, x_2, ..., x_n)$, el tiempo de terminación difuso $\tilde{t}_C(i,j)$ de cada trabajo en cada máquina es calculado mediante aritmética difusa como:

$$
\begin{aligned}
\tilde{t}_C(1,x_1) &= \tilde{t}_P(1,x_1), \\
\tilde{t}_C(i,x_1) &= \tilde{t}_C(i-1,x_1) + \tilde{t}_P(i,x_1) \\
&\text{for } i = 2,3,...,m, \\
\tilde{t}_C(1,x_k) &= \tilde{t}_C(1,x_{k-1}) + \tilde{t}_P(1,x_k) \\
&\text{for } k = 2,3,...,n, \\
\tilde{t}_C(i,x_k) &= \max\{\tilde{t}_C(i-1,x_k), \tilde{t}_C(i,x_{k-1})\} + \tilde{t}_P(i,x_k) \\
&\text{for } i = 2,3,...,m; \\
&\text{for } k = 2,3,...,n.
\end{aligned}
\tag{4.6.1}
$$

De 4.6.1, se puede ver que el operador de adición y el operador máximo de dos números difusos se requieren para calcular el tiempo de terminación difuso $\tilde{t}_C(i,j)$. Estos operadores se ilustran en la Figura 4.9. El tiempo de terminación

difuso de cada trabajo se calcula como un número difuso por 4.6.1:

$$\tilde{C}_j = \tilde{t}_C(m, j) \tag{4.6.2}$$



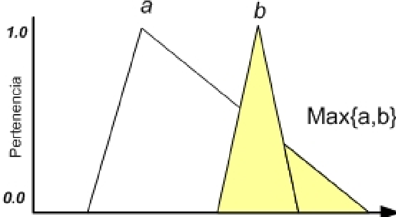


Figura 4.9: Suma de dos números fuzzy a y b; y máximo de dos números fuzzy, a y b.

En cálculos númericos del tiempo de terminación difuso, el tiempo de procesamiento de cada trabajo en cada máquina se maneja como una colección de α-cuts mediante varios valores (p.e. $\alpha = 0.01, 0.02,...,1.00$). El α-cut de un número difuso $\tilde{b}$ se define por $0 \leq \alpha \leq 1$ como

$$[\tilde{b}]_\alpha = \{x : \mu_{\tilde{b}}(x) \geq \alpha, x \epsilon \Re\} \tag{4.6.3}$$

Es sabido que el α-cut de un número difuso es un intervalo cerrado para $0 < \alpha < 1$. De ahi que la aritmetica difusa puede ser aproximadamente implementada aplicando la aritmetica de [3] intervalos a sus α-cuts para varios valores de α. En cálculos númericos, el *makespan* difuso $\tilde{C}$ se puede aproximar con varios niveles de α-cuts. Sea $\tilde{C}^\alpha$ el $\tilde{C}$ en el nivel-α, entones $\tilde{C}^\alpha$ puede ser representado por un intervalo, p.e., $\tilde{C}^\alpha = [\tilde{C}_l^\alpha, \tilde{C}_u^\alpha] = \{x|\mu_x \geq \alpha\}, 0 < \alpha < 1$, mientras que C_l^α y C_u^α representan respectivamente el limite inferior y superior del intervalo. El $\tilde{C}$ se calcula con las fórmulas en 4.6.1 aplicando niveles-α a los conjuntos difusos. Como el *makespan* difuso calculado $\tilde{C}$ no necesariamente tiene función de pertenencia

triangular, para simplificar, se usa el trio $(l, m, u$ para respectivamente representar la frontera inferior y superior de la función de pertenencia, y m representa su mayor punto (o punto más alto, p.e., en el caso triangular, representa el pico del triángulo). En el presente trabajo uso el *índice* propuesto por Yager [88] para evaluar el *makespan* difuso $\tilde{C}$ 4.6.4:

$$Y(\tilde{C}) = \frac{1}{2} \cdot \int_0^1 (C_l^\alpha + C_u^\alpha) d\lambda \tag{4.6.4}$$

4.6.2. Fecha de Entrega Difusa

En esta sección no sólo asumimos que el tiempo de procesamiento es difuso, sino que también las fechas de entrega. Precisamente asumimos que cada trabajo J_k tiene una fecha de entrega D_k y un tiempo de procesamiento P_k donde ambos, D_k y P_k, son intervalos difusos positivos.

Cada permutación π de trabajos determina un orden n:

$$(C_1^\pi, C_2^\pi, ..., C_n^\pi)$$

donde los tiempos de terminación son:

$$C_{\pi(k)}^\pi = P_{\pi(1)} \oplus P_{\pi(2)} \oplus ... \oplus P_{\pi(k)}, \quad 1 \leq k \leq n$$

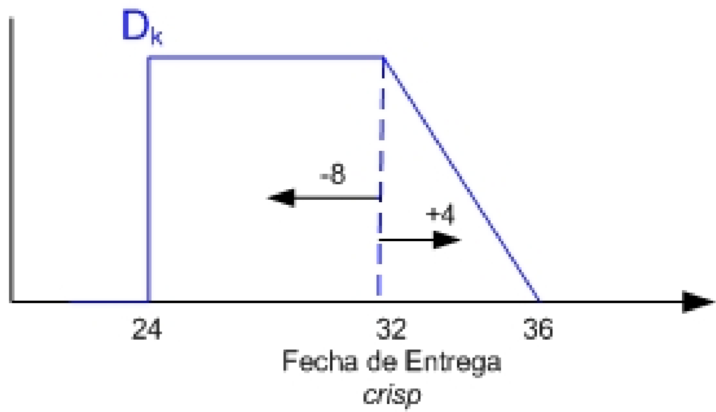
donde $\oplus$ se refiere a la adición de cantidades difusas. Consideré que en todos los papers se usa la version de convolución sup-min como $\oplus$. Si el tiempo de terminación C_k^π no es *crisp*, sino más bien como en el presente caso, difuso, no es muy claro entonces como evaluar el par (C_k^π, D_k). Se sabe que depende mucho del problema especíco. Itoh y Ishii [39] estudiaron el caso de tener fechas de entrega y tiempos de procesamiento difusos. En el caso presente se utiliza la siguiente función:

$$v_k(C_k^\pi, D_k) \doteq \begin{cases} 1 & \text{si sup-min}(C_k^\pi(t), D_k(t)) \leq \lambda \\ 0 & \text{cualquier otro caso} \end{cases} \tag{4.6.5}$$

donde λ es un número dado en el intervalo [0,1]; ver Figura 4.10. Lo que se quiere es entonces minimizar los trabajos con tardanza λ donde ahora el trabajo J_k tiene una tardanza λ en la permutación π si

$$\sup \min(C_k^\pi(t), D_k(t)) \leq \lambda$$

He considerado, luego de varias pruebas, un valor de $\lambda = 0{,}7$, valor con el cual se llevaron a cabo todas las pruebas finales.



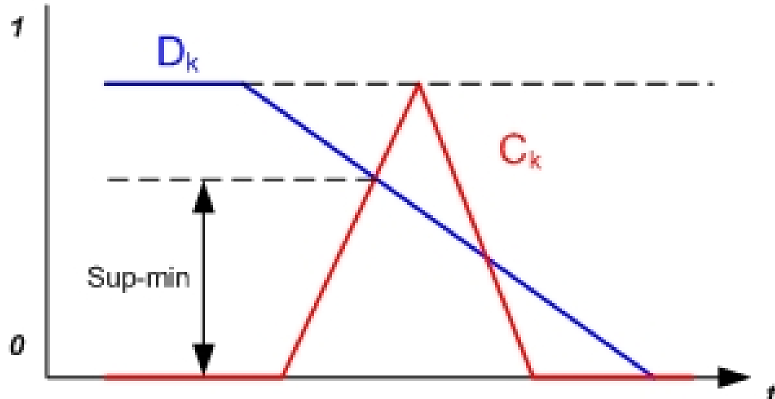


Figura 4.10: *Tardiness* difusa para fechas de entrega difusas

4.6.3. Criterio de Programación Difuso

Como se mencionó en la sección anterior, el tiempo de terminación de cada trabajo se calcula como un número difuso. De ahí que los criterios tales como el *makespan* y la mayor tardanza o *tardiness* se calculen también como números difusos. Si la fecha de entrega fuera *crisp*, la tardanza *tardiness* y el *earliness* se definen como:

$$\tilde{T}_j = \max\{0, \tilde{C}_j - d_j\} \equiv (\tilde{C}_j - d_j)^+ \tag{4.6.6}$$

$$\tilde{E}_j = \max\{0, d_j - \tilde{C}_j\} \equiv (d_j - \tilde{C}_j)^+ \tag{4.6.7}$$

Por lo tanto el problema de minimización del *tardiness* se escribe como:

$$\text{Minimizar } \tilde{T}_{max} = \max\{\tilde{T}_j | j = 1, 2, ..., n\}. \tag{4.6.8}$$

De la misma manera, el problema de minimización de la máxima tardanza y máxima prontitud *tardiness* y *earliness* se escribe:

$$\text{Minimizar } ET_{max} = \max\{\alpha_j \cdot \tilde{E}_j + \beta \cdot \tilde{T}_j | j = 1, 2, ..., n\} \tag{4.6.9}$$

El *makespan* y el flujo total de trabajo o *flowtime* pueden ser fuzzyficados de la misma manera:

$$\text{Minimizar } \tilde{C}_{max} = \max\{\tilde{C}_j | j = 1, 2, ..., n\} \tag{4.6.10}$$

$$\text{Minimizar } \tilde{C}_{sum} = \sum_{j=1}^{n} \tilde{C}_j \tag{4.6.11}$$

En el trabajo presente, dado que tenemos tanto el tiempo de procesamiento como la fecha de entrega como números difusos, usaremos la ecuación 4.6.5 para modificar 4.6.10 y 4.6.6.

Con esto nuestro problema se convierte en un problema de optimización de dos objetivos: el *makespan* y el *tardiness*.

4.6.4. Evaluación: Fitness

Tenemos nuestro problema como:

$$\text{Minimizar } f_1(\mathbf{x}) \text{y} f_2(\mathbf{x})$$

Donde f_1 representa el *makespan* y el f_2 el *tardiness*. Como estos dos objetivos deben ser minimizados, podemos definir la función fitness como:

$$f(\mathbf{x}) = w_1 \cdot f_1(\mathbf{x}) + w_2 \cdot f_2(\mathbf{x}) \tag{4.6.12}$$

Esta función de fitness se usa en la selección y la búsqueda local del algoritmo híbrido. Como la varianza del *makespan* es mucho mayor que la del *tardiness*, normalicé estos dos criterios de programación como sigue:

$$f_1(\mathbf{x}) = 1 \times (\text{makespan}) \tag{4.6.13}$$

$$f_2(\mathbf{x}) = 500 \times (\text{tardiness}) \tag{4.6.14}$$

donde $\mathbf{x}$ es una solución factible del problema de programación *flowshop*. Las constantes de multiplicación 4.6.13 se introducen para manejar dos criterios de programación equitativamente. Para la búsqueda local, se definen las soluciones vecinas mediante la mutación. Esto es, las soluciones vecinas de la actual solución $\mathbf{x}$ son generadas por la sola aplicación del operador de mutación. El número total de soluciones vecinas de $\mathbf{x}$ es $(n-1)^2$ para el problema de programación de n-trabajos.

CAPÍTULO V

Pruebas y Resultados

En el presente capítulo se presentan los resultados de las pruebas corridas con el algoritmo híbrido. Para analizar los resultados experimentales obtenidos por el GA, se uso data real de la compañia y su refinería. Se probo con varios escenarios de demanda, dos de los cuales se presentan como resultados. Uno de corto tiempo de resolución y baja demanda, para 6 productos; y otro de alta demanda, con 16 productos y 7000 horas de producción.

Como se puede ver, el objetivo de la presente tesis no fue la implementación del algoritmo, por lo que no se detalla. Debemos mencionar sin embargo, que las pruebas se realizaron en un procesador Intel Pentium 560 de 3.6GHz - 1Gb RAM. El código del algoritmo se implemento íntegramente en C♯, y se utilizo el paquete Visual Studio .NET 2003 en la plataforma Windows XP Profesional. Microsoft Excel sirvió como una base de datos de donde se sacan los datos de planta real y a su vez, a donde se publican los resultados.

5.1. Especificación de Parámetros

En esta sección analizaré los parametro más determinantes del algoritmo. En los GA convencionales, se tienen parámetros como tamaño de población, tamaño de cromosoma, numero de generaciones, probabilidad de mutación, probabilidad de crossover, parametro elitista (el número de soluciones que se preservan para la siguiente generación). Para el algoritmo usado aquí, adicionaré un parámetro de la búsqueda local, el k, que representa la cantidad de soluciones examinadas en la búsqueda local.

5.1.1. Generaciones, Tamaños de Población y su Efecto en el Tiempo Computacional

En esta sección revisaremos la influencia de el numero de generaciones y el tamaño de población y de cromosoma en el tiempo que tarda el algoritmo. El tamaño de población utilizado para las pruebas consideradas fue de 100 soluciones. Se hizo corridas con tamaños de 50 y de 700 soluciones. Con $N_{pop} = 50$ no se obtuvo la dispersion requerida; o por decirlo de otra manera, no se tuvo una diversificación deseable en la población, ya que las soluciones, desde un inicio, se parecían mucho entre ellas por lo que la solución final no resulto factible. La corrida de la simulación con $N_{pop} = 700$ soluciones, por otro lado, incremento casi en un 200 % el tiempo de procesamiento, llegando a ser de casi 5 horas sin un resultado muy diferente al conseguido con $N_{pop} = 100$. Es decir, no se mejoro la calidad de la respuesta con $N_{pop} = 700$ soluciones de tamaño de población. Se considero un tamaño estándar para las pruebas de 100 soluciones. En cuanto al tamaño de cromosoma, este se determina por el código del programa, ya que es el resultado de la cantidad de batches a programar. La cantidad de batches esta en directa relación a la cantidad de productos a programar. Además también entra a tallar el tamaño de horizonte utilizado. Siendo en las simulaciones, un horizonte de 7 dias, el tamaño de cromosoma varia entre 33 y 86 genes, dependiendo del tipo de producto y su cantidad. El factor determinante para el tiempo que toma el algoritmo en terminar es el número de generaciones que se aplica. El resultado de este efecto se probo para una corrida de 6 productos, con $N_{elite} = 25$ y $k = 5$, cuyos resultados se muestran en la tabla 5.1. Ahora examinara la dependencia de

Tabla 5.1: Tiempo computacional para los diferentes números de generaciones

Número de Generaciones	Tiempo Computacional (s)
50	31
100	73
500	427
5000	4826
10000	22680

la performance del algoritmo con respecto a la elección de N_{elite}. Luego de aplicar, como anteriormente, el algoritmo a la solución del problema con 6 productos,

$k = 5$, usando varios valores de N_{elite} (p.e., $N_{elite} = 0, 1, ..., 9$). El valor de k se especifico para $k = 5$. Para cada valor de N_{elite}, se itero la simulación 100 veces. Los resultados de la simulación se resumen en la tabla 5.2. Se puede ver que el

Tabla 5.2: Resultados de la simulación con varios valores de N_{elite} según mediciones de la función de fitness

Valores N_{elite}	0	1	3	10	25	30	50
Media	3463.14	3270.19	3095.26	2827.52	2609.96	2603.86	2344.32
Error tipico	17.85	16.48	22.25	18.33	20.59	21.79	32.83
Desviación estándar	177.64	164.06	221.39	182.41	204.91	216.81	326.71
Nivel de confianza (98 %)	42.22	38.99	52.62	43.35	48.703	51.53	77.65

algoritmo se deteriora según la elección del N_{elite} como $N_{elite} = 0$. Esto significa que las soluciones elite juegan un rol importante en el algoritmo híbrido. Se puede ver que el promedio de la función a minimizar decrece conforme aumentamos el número N_{elite}. El nivel de confianza crece conforme aumenta el N_{elite}, lo que me puede llevar a decidir que usare $N_{elite} = 25$ en mis futuras pruebas. Para ver lo que sucede en los valores de fitness, ver la figura 5.1. Se puede ver la diferencia para los parámetros $N_{elite} \geq 10$, que produce un decrecimiento de la función a minimizar.

5.1.2. Búsqueda Local

Finalmente analizaremos el parametro k en la performance del algoritmo. Hacemos las corridas con $N_{gen} = 100$ generaciones, $N_{elite} = 25$ y 6 productos. Como estos 6 productos representan, dada la demanda, 38 tareas o trabajos para realizar, se tiene $(n-1)^2$, $(38-1)^2$ soluciones vecinas que se pueden reproducir como máximo en la búsqueda local. De ahi que podemos escoger el valor de k entre [0, 1369]. Si se especifica $k = 0$, el algoritmo híbrido se reduce a un GA multiobjetivo sin procedimiento de búsqueda local. Por otra parte, si escogemos $k = 1369$, significa que se usa el procedimiento de búsqueda local estandar. La tabla 5.3 muestra los valores de k que fueron usados en las simulaciones para 6

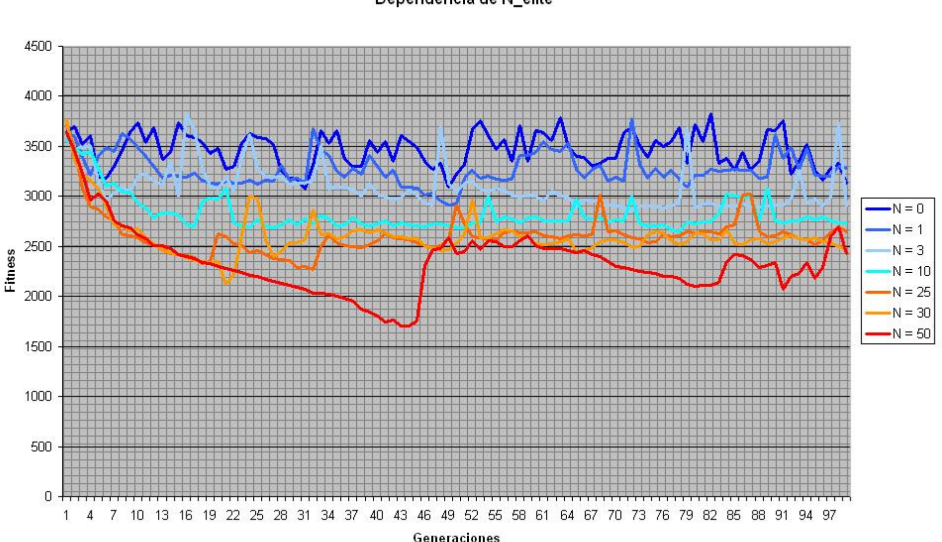


Figura 5.1: Función fitness y su dependencia al parámetro de N_{elite}

productos y su impacto en el tiempo computacional.

Tabla 5.3: Resultados de las simulaciones con varios valores de k

Número de k	Tiempo Computacional (s)
0	31
5	59
8	61
15	136
30	260
100	5479
200	29106

La tabla muestra un incremento exponencial de los tiempos de procesamiento computacionales según se incrementa k. Dado el incremento exponencial de los tiempos computacionales se aproximó a la ecuación:

$$T_c = 1031{,}4 \cdot \exp(0{,}0167 \cdot k) \tag{5.1.1}$$

Donde T_c es el tiempo computacional. De 5.1.1 se ve que para el valor máximo de k, $k = 1369$ para la prueba dada, el tiempo computacional invertido en el algoritmo es prohibitivo, en el orden de los 10^{12} segundos, que representan

meses de simulación.

5.2. Tendencias de los Objetivos

Como se menciono en las secciones anteriores, los objetivos principales escogidos para minimizar por el algoritmo fueron el *makespan* o tiempo total de duración y el *tardiness*, tiempo que suma las tardanzas, si es que las tienen, cada tarea con respecto a su fecha de entrega. En esta sección se presentarán los gráficos resultantes de estos objetivos y el comportamiento de las soluciones no dominantes.

5.2.1. Soluciones No Dominadas

En el capitulo anterior se presentaron los pasos que el algoritmo realizaba. Tomando el caso para programar 6 productos, con $N_{pop} = 100$, $N_{elite} = 25$, $k = 8$ y $N_{gen} = 100$. En el primer paso, se inicializan 100 soluciones aleatorias para la población. Cada cromosoma o solución comienza con tiempos de inicio de las tareas aleatorias, colocáncolas de esta manera el orden aleatorio. Ahora bien, se probó el impacto en de dos tipos de inicialisaciones aleatorias, cambiando el intervalo: un intervalo es para todo el horizonte de programación, en este caso, 7 dias: $random[0, 168]$ horas. El otro intervalo que se probo fue $random[12, duedate+24]$ para $duedate > 72$ y$random[1, duedate + 24]$ para $duedate < 72$; donde $duedate$ se refiere a la fecha de entrega del producto. Sin embargo, ninguna de las pruebas marco la diferencia una de la otra, por lo que se trabajo con el segundo intervalo de inicialisación. Luego de la inicialización, como se dijo en el capitulo anterior, se evalúa cada una de las soluciones. Si tenemos un $N_{pop} = 100$ y $N_{gen} = 100$ tendremos un total de 10000 soluciones a evaluar a lo largo de toda la simulación. El algoritmo híbrido también actualiza el conjunto tentativo de soluciones no dominadas en este paso. Se probo el problema de 6 productos para 3 N_{gen}, 50, 100 y 150.

La figura 5.3 y 5.2.1 muestran 4 pruebas diferentes, cada una de ellas muestra el conjunto de soluciones dominantes y dominadas, para la primera y ultima generación. En las figuras (a) y (b), el numero de generaciones fue de 50; siendo la diferencia el intervalo de inicializción. Se nota que la población final no se aparta de la inicial, se mezclan entre ellas ya que el numero de generaciones

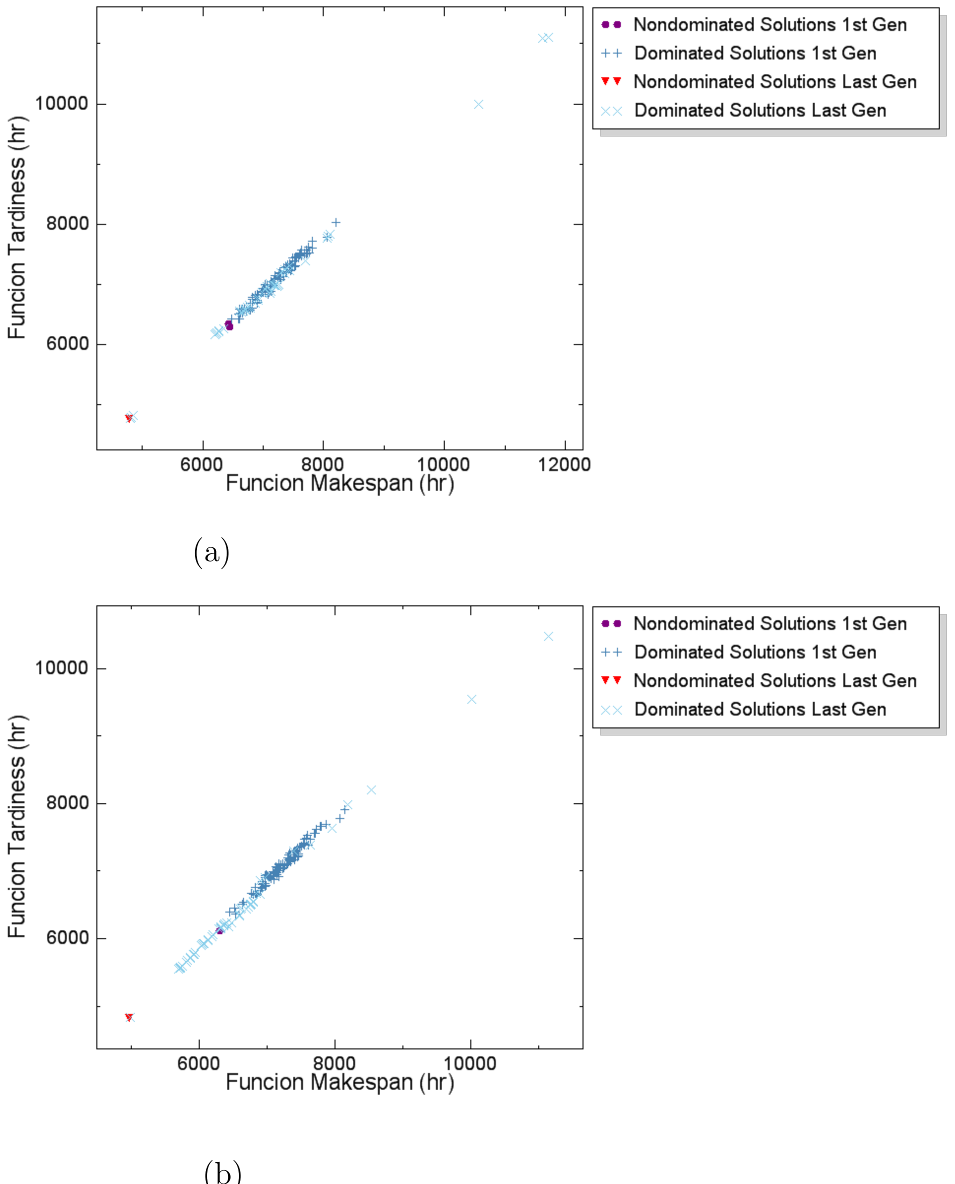


Figura 5.2: Para (a) se uso $N_{gen} = 50$, (b) $N_{gen} = 50$ con otro intervalo de inicialización.

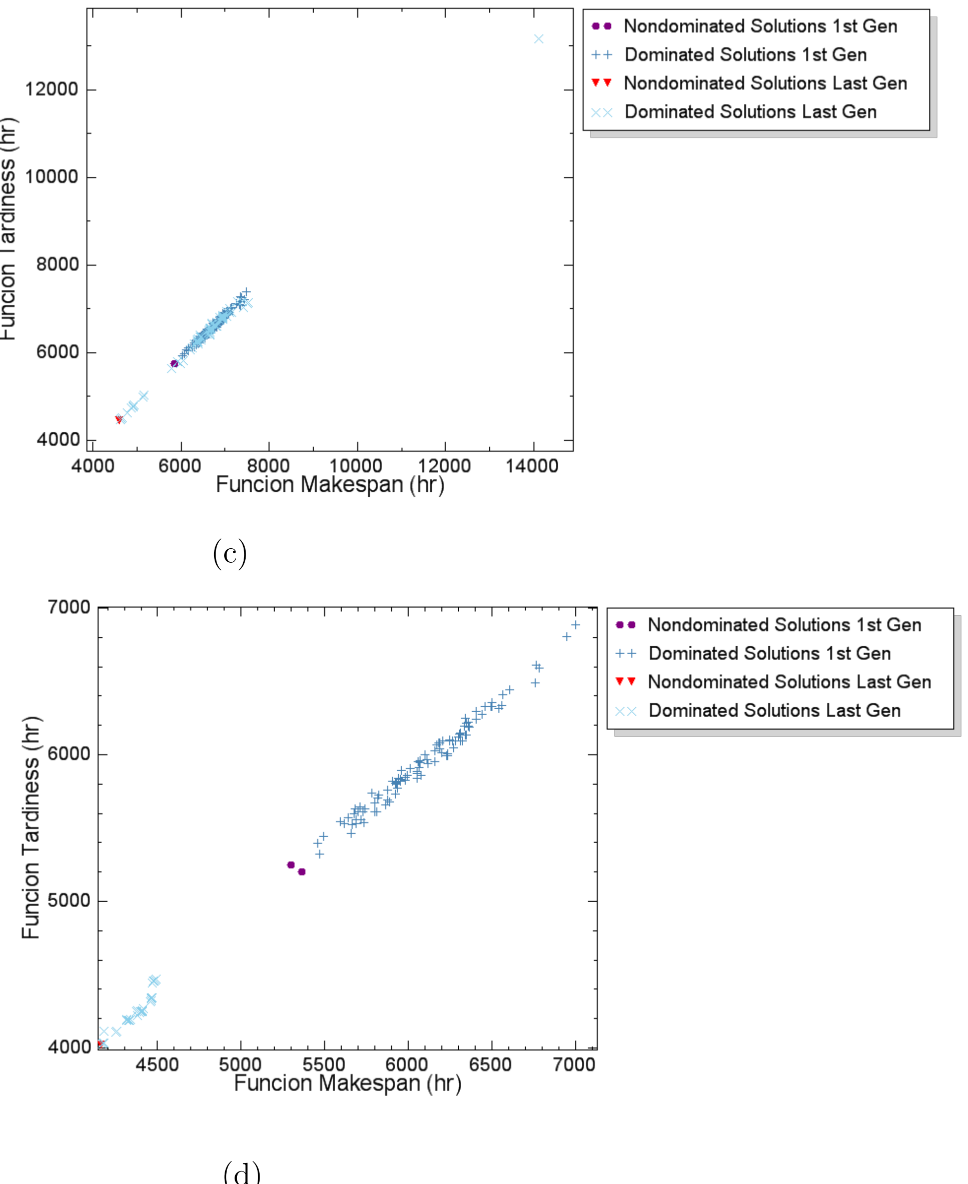


Figura 5.3: (c) uso $N_{gen}n = 100$ y (d) uso $N_{gen} = 150$.

no fue suficiente para encontrar una solución optima y el espacio de búsqueda no se llego a explorar del todo. Sin embargo, para (c) y (d), con $N_{gen} = 100$ y $N_{gen} = 150$ respectivamente, si se observa una separación y conjuntos marcados entre la población inicial y final. Nótese que la población final se compacta y se acerca mas a los mínimos del espacio de búsqueda. De ahi que para nuestras pruebas subsiguientes , escogiéramos $N_{gen} = 100$ como mínimo para garantizar una solución factible.

5.2.2. Escenarios

Aquí presentaremos las tendencias graficadas para los dos objetivos escogidos del problema, además de el gráfico de demanda no satisfecha para cada solución final propuesta. Como se menciono al inicio del capítulo, se realizaron 2 escenarios: uno con 6 productos (baja demanda) y otro con 16 productos a capacidad total de planta. Los parámetros se resumen en la tabla 5.4.

Tabla 5.4: Parámetros utilizados para cada escenario

Parametro	Escenario 1	Escenario 2
Productos	6	16
N_{gen}	100	150
k	8	25
N_{elite}	25	25
N_{pop}	100	100
Tiempo Computacional	109	587

Con dichos parametros, los resultados obtenidos para el escenario 1 y 2 se muestran en la figuras.

También mostramos un objetivo indirecto del algoritmo, que es el volumen de demanda perdida o no satisfecha. Para esto hemos usado el siguiente criterio: toda demanda que se demora mucho para ser entregada se considera perdida, asi como toda demanda que se entrega con mucha anticipación. El criterio entonces es, dado los tiempos de inicio representados por numeros triangulares difusos,

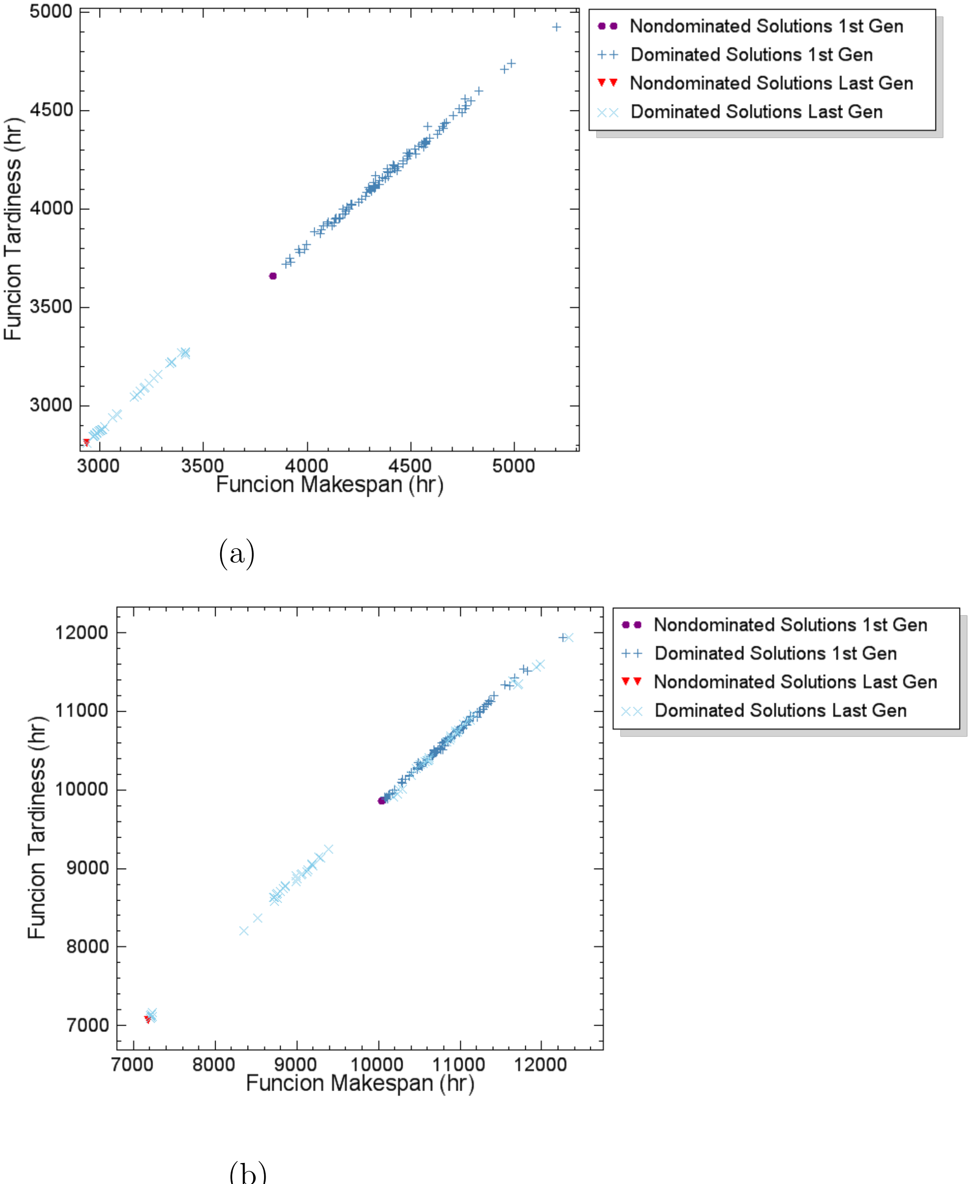


Figura 5.4: Soluciones no dominadas para (a) escenario 1 y (b) escenario 2.

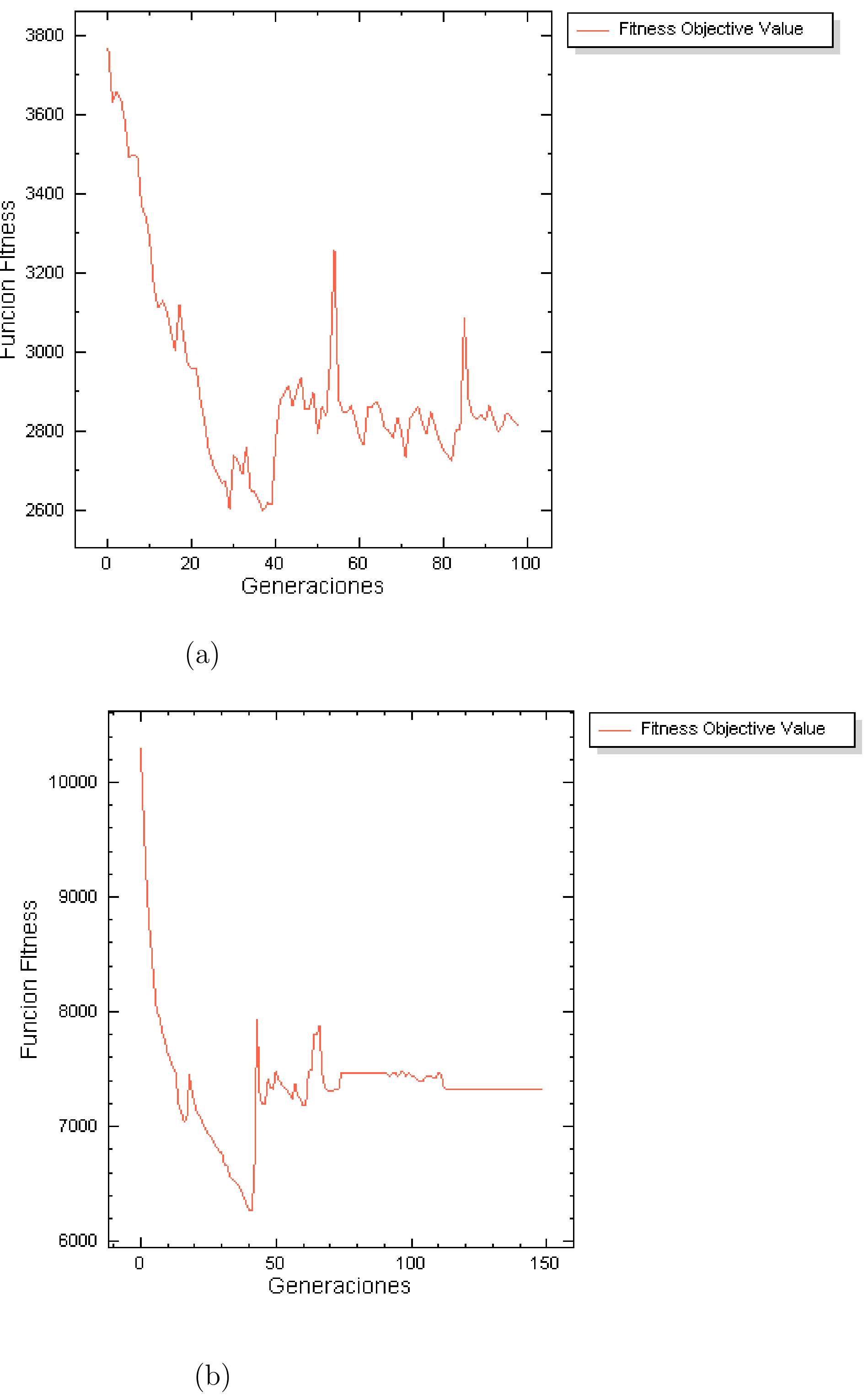
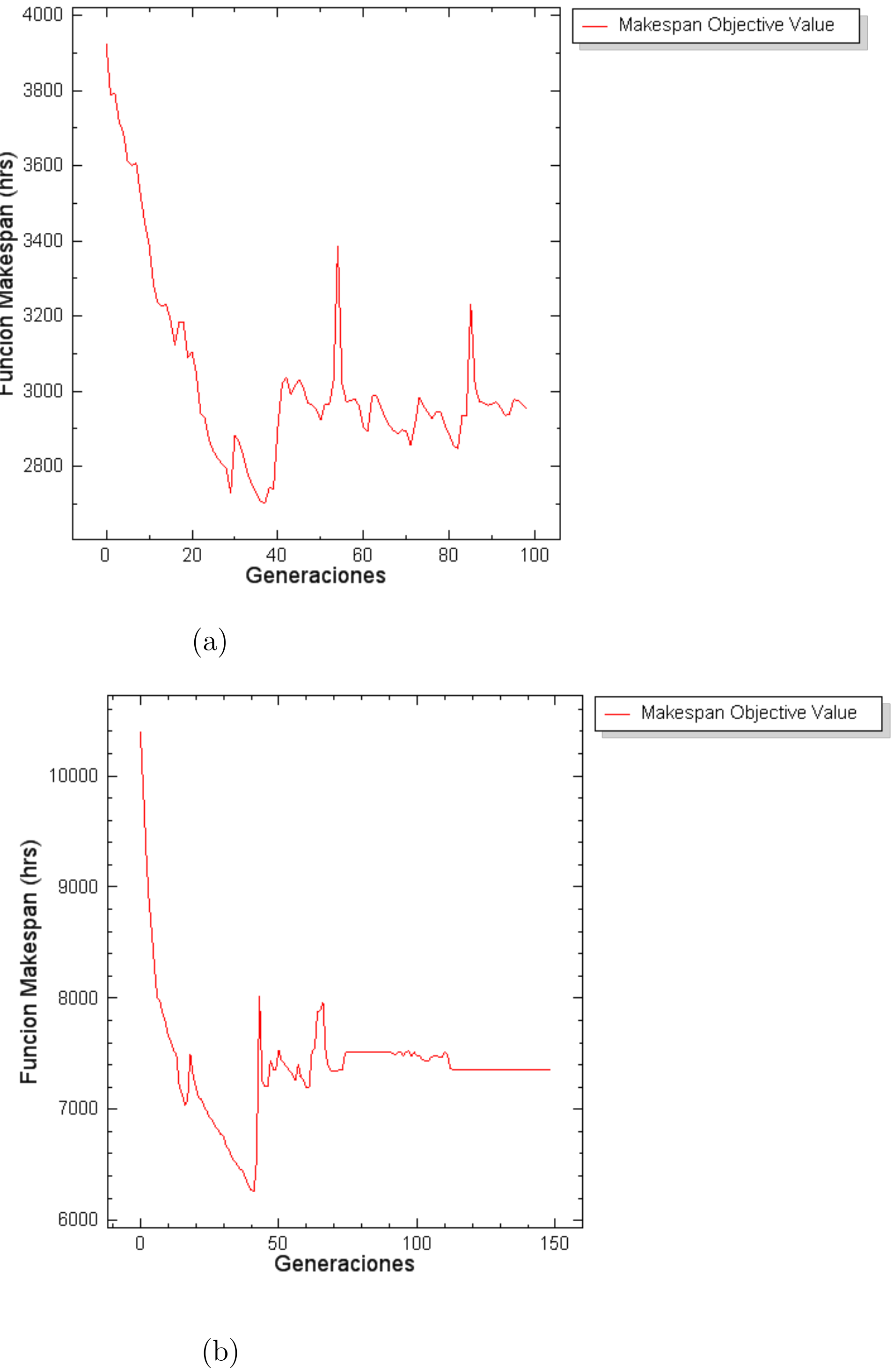


Figura 5.5: Función fitness para (a) escenario 1 y (b) escenario 2.



(a)

(b)

Figura 5.6: Función Makespan para (a) escenario 1 y (b) escenario 2.

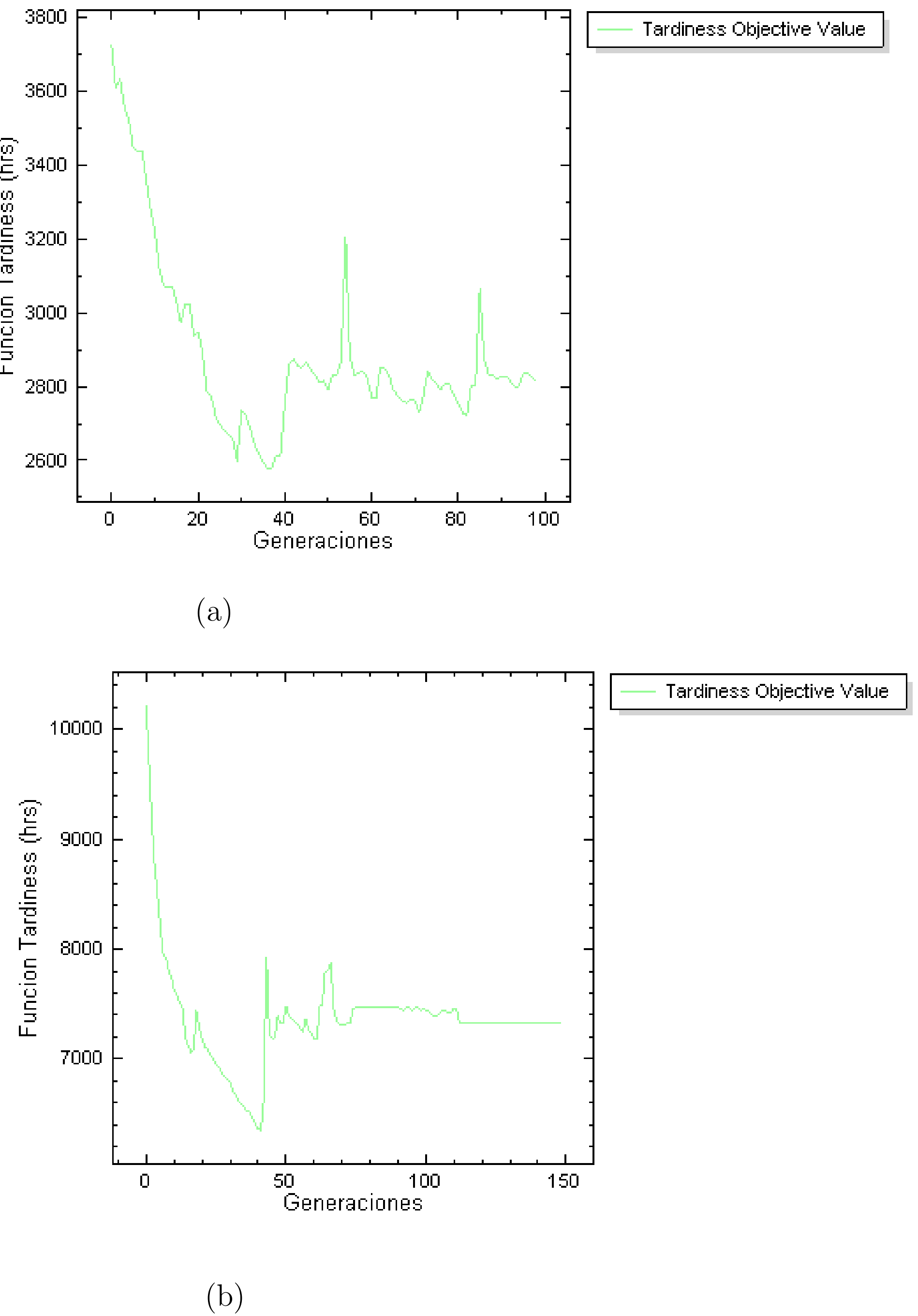


Figura 5.7: Función tardiness para (a) escenario 1 y (b) escenario 2.

podemos representarlos con los α-cuts.

$$\begin{array}{ccc} \left[\tilde{S}_{for}\right]_{\alpha}^{L} & \text{y} & \left[\tilde{S}_{for}\right]_{\alpha}^{U} \\ \left[\tilde{S}_{deso}\right]_{\alpha}^{L} & \text{y} & \left[\tilde{S}_{deso}\right]_{\alpha}^{U} \end{array}$$

donde S_{for} denota el inicio de la formulación y S_{deso} el inicio de la desodorización. Los superindices L y U denotan los limites inferior y superior de los α-cuts. Se cree conveniente que para un α, una demanda se considera perdida si:

$$\left[\tilde{S}_{deso}\right]_{\alpha}^{U} + 4 \quad > \quad D_{crispduedate} \tag{5.2.1}$$

$$\left[\tilde{S}_{for}\right]_{\alpha}^{U} + 48 \quad < \quad D_{crispduedate} \tag{5.2.2}$$

Se asume que una vez empezada la desodorización, se necesitará 4 horas como minimo para tener el producto listo. $D_{crispduedate}$ representa la fecha de entrega de planta. Es decir, dado un α-cut (en nuestras pruebas con $\alpha = 0{,}7$), si 4 horas después del limite superior del inicio de la desodorización ya se pasó la fecha de entrega, es una demanda perdida y se considera una falta de entrega. Asi mismo, dado el mismo α, si el limite superior del inicio de la formulación ocurre antes de dos dias o 48 horas que la fecha de entrega, se considera un desperdicio de recursos ya que no se debe guardar stocks en los almacenes, y el producto debe salir justo a tiempo de entrega, por lo que si se tiene producto listo esperando en almacén es un gasto inútil. La figura 5.8 explica gráficamente este criterio.

Para los escenarios presentados, el comportamiento del algoritmo con respecto al volumen perdido es el mostrado en la figura

Se puede ver como el algoritmo busca estabilizar las soluciones en 0 toneladas de producto perdido. Para complementar los resultados se muestra una parte del diagrama de Gantt de la producción para 6 productos en la figura 5.10.

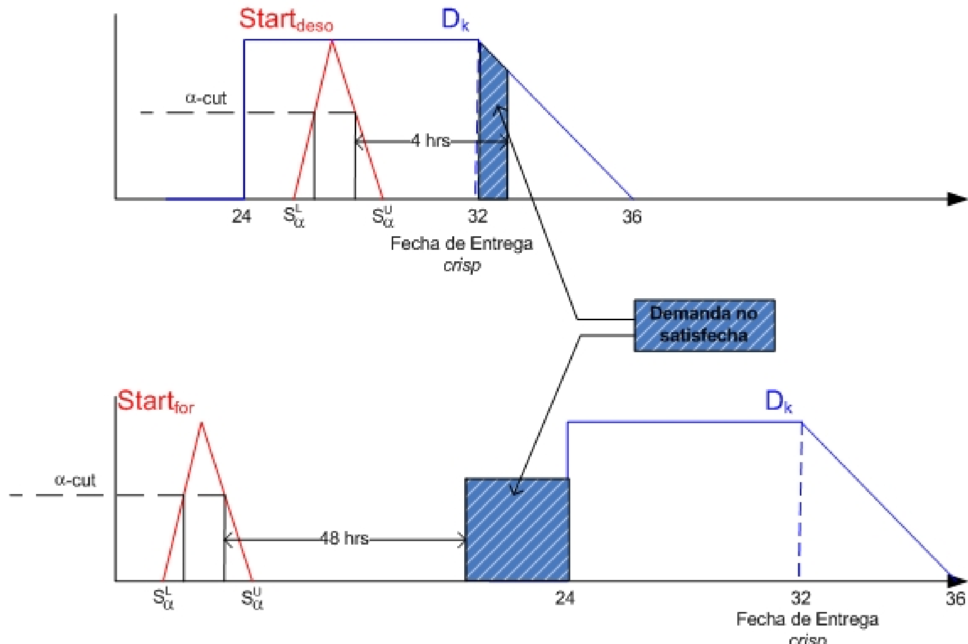


Figura 5.8: Criterio para la demanda no satisfecha

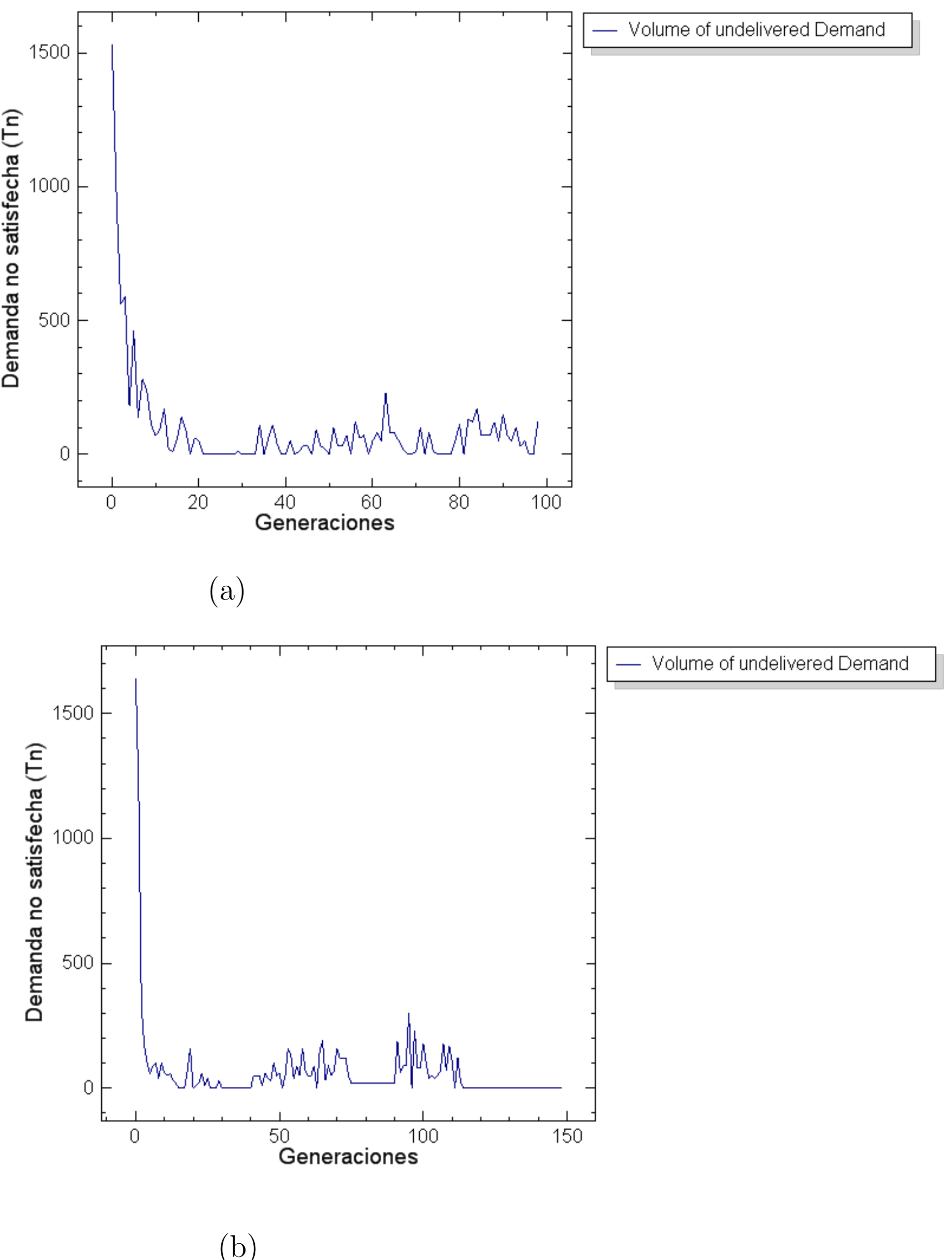


Figura 5.9: Volumen de demanda perdida o no satisfecha en toneladas para (a) escenario 1 y (b) escenario 2.

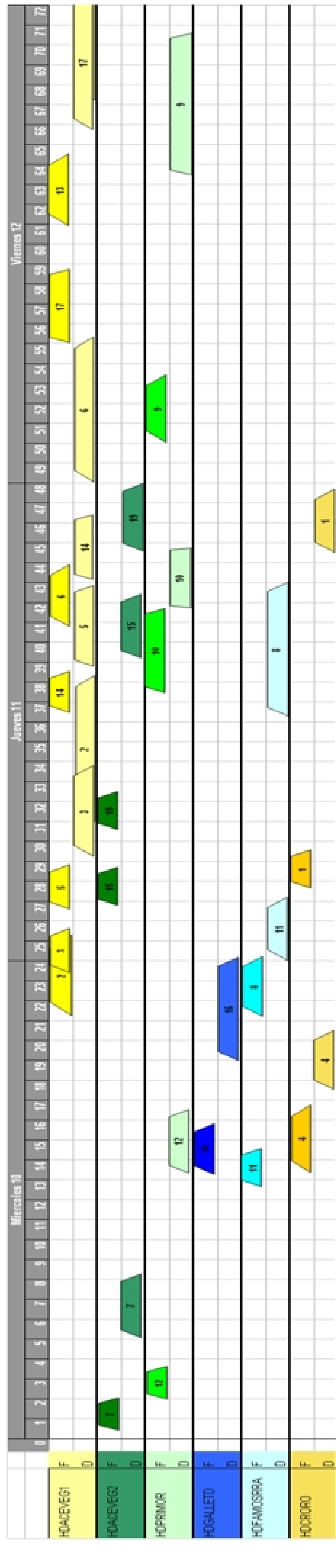


Figura 5.10: Diagrama de Gantt para la producción de 6 productos

CONCLUSIONES

La tesis concluye con un resumen corto de las contribuciones y algunas recomendaciones para investigaciones del futuro que son principalmente los problemas que no han sido resueltos en este trabajo de tesis.

Conclusiones

En el presente trabajo se propuso un algoritmo genético multiobjetivo de búsqueda local. En el algoritmo propuesto, se aplico un procedimiento de búsqueda local para cada solución generada por las operaciones genéticas. Se demostró un alto desempeño del algoritmo mediante simulaciones por computadora de un entorno real de programación *flowshop*.

Las cualidades características del algoritmo híbrido propuesto se puede resumir:

1. Se usa una suma con pesos de los múltiples objetivos como función *fitness* en la operación de selección. Los valores de los pesos en la función de *fitness* se especifican aleatoriamente cada vez que un par de soluciones padre se seleccionan. Esto es, se realiza una selección con un vector de pesos diferente.

2. Se aplica un procedimiento de búsqueda local a cada nueva solución generada por las operaciones genéticas (p.e., selección, crossover, y mutación). La búsqueda local para una nueva solución se realiza para minimizar la función *fitness* que fue usada para seleccionar a sus soluciones padre. Esto es, cada nueva solución tiene su propia dirección de búsqueda local en el espacio de objetivos.

3. En la búsqueda local, no todas las soluciones vecinas son examinadas para cada movimiento. Esto es, el numero de soluciones vecinas examinadas de la solución actual se restringe en la búsqueda local. Esto es para prevenir a la búsqueda local de gastar la mayoría del tiempo computacional disponible.

4. Un conjunto tentativo de soluciones no dominadas se guarda y se actualiza en cada generación. El conjunto tentativo se guarda separadamente de la población actual. Se seleccionan unas pocas soluciones aleatoriamente del conjunto tentativo y se usan como soluciones *elite*, es decir, las soluciones que prevalescerán para la proxima generación.

El algoritmo propuesto es simple, y su tiempo computacional es casi el mismo que cualquier otro GA sin búsqueda local. Esto también representa una ventaja para el algoritmo híbrido propuesto.

El algoritmo genético resuelve el problema de optimización al establecer el secuenciamiento de una programación de la producción multiobjetivo, en una planta multi-producto, con dos etapas de máquinas en serie- una formulación y un desodorizador. La planta tiene restricciones de recursos y opera por batch, continuamente evitando estar parada entre batch y batch.

El tipo de representación indirecta utilizada en el GA es un arreglo de listas indexadas. Cada gen del cromosoma representa un batch u orden de producción.

Luego de analizar los resultados, se puede decir que la representación escogida tiene excelentes niveles de performance- muchas veces cerca al óptimo- en el caso que el objetivo sea suplir la demanda - la performance es satisfactoria y un sobresaliente desempeño desde el punto de vista de la minimización de la tardanza de la entrega de ordenes y la duración total de una producción.

Recomendaciones para Trabajos Futuros

El GA utilizado sólo se encarga del secuenciamiento de las tareas, mas no de la asignación de recursos. Para un trabajo futuro, un algoritmo de co-evolución podría ser implementado, de tal manera que se tenga cromosoma que represente la asignación de recursos, en este caso, los tanques y desodorizador y sea capaz de encontrar la asignación optima. Aquí se pueden usar las reglas de precedencia de la planta de acuerdo a la calidad de los productos. Por ejemplo, una regla es que nunca se comienza la producción con los productos de alta calidad.

El trabajo futuro debe analizar el impacto económico que la planificación y programación automatica y optimizada puede causar. Esto se preocuparia de la implementación práctica del programa, evaluando aspectos de ergonomía para los operadores y flexibilidad de uso.

Finalmente se puede realizar un trabajo dentro de un esquema de inteligencia artificial distribuida (DAI), en donde este algoritmo puede representar un módulo, que, a su vez, representar un agente en un sistema multi agente, con el fin de realizar la integración de la planificación y programación de la producción. Esta investigación daría una respuesta avanzada ante un entorno altamente incierto y cambiante.

APÉNDICE A

Glosario

Adaptive Iterative Construction Search (AICS) Metodo de búsqueda estocástica local que se basa en iteraciones multiples de un procedimiento de **búsqueda constructiva**; la construccion es guiada por información heuristica y por experiencia ganada de las iteraciones pasadas. Esta ultima es representada en la forma de *pesos* asociados con decisiones elementarias que pueden tomarse durante el proceso de construcción; estos pesos se adaptan durante la búsqueda. AICS es conceptualmente similar al **Ant Colony Optimzation** (ACO), y los rastros de feromona en ACO corresponde al peso en el AICS.

Ant Colony Optimization (ACO) La Optimización de Colonia de Hormigas es un metodo de búsqueda estocástica inspirado por el rastro de feromona que guía el comportamiento de algunas especies de hormigas. En analogia con el ejemplo biologico, ACO se basa en la comunicación indirecta de una colonia de agentes simples, llamados *hormigas* , mediados por *rastros de feromona*. Los rastros de feromona en ACO sirven como una memoria numerica, distribuida que las hormigas usan probabilísticamente para construir **soluciones candidatas** para la instancia dada del problema; como los pesos en **AICS**, los rastros de feromona se adaptan durante el proceso de búsqueda y representan la experiencia colectiva de las hormigas. Se ha propuesto un gran numero de algoritmos ACO, incluyendo el **MAX-MIN Ant System** y el **Ant Colony System**. La mayoria de los ACO con mejor performance para **problemas combinatorios** utilizan **búsquedas locales** subsidiarias para mejorar las **soluciones candidatas** construidas por las hormigas.

Ant Colony System (ACS) Un algoritmo **Ant Colony Optimization** que usa un procedimiento agresivo de construcción en el cual, con alta probabilidad , se realizan elecciones determinísticas en vez de las usuales estocásticas elecciones. Además, a diferencia de los algoritmos ACO, ACS realiza actualizaciones adicionales de los rastros de feromona durante la construcción de **soluciones candidatas**.

Approximate (Optimization) Algorithm Algoritmo de búsqueda incompleta para un **problema de optimización**, esto es, un algoritmo para resolver un problema de optimizacion que no garantiza encontrar una solucion optima, aun si se corre por un largo tiempo (finito).

Backlog Si el **surplus** es la diferencia entre la producción acumulada y la demanda acumulada, el *backlog* viene a ser su valor negativo. Viene a representar las órdenes de compra/manufactura que aun no han sido procesadas.

Backtracking Técnica algorítmica en la cual un algoritmo de búsqueda, a partir de encontrar un çamino sin salida"desde el cual mayor búsqueda es imposible o desprometedora, revierte hacia un punto anterior en su trayectoria de búsqueda. Varios algoritmos de búsquedas sistemáticas están basados en combinaciones de métodos de búsquedas constructivas con backtracking.

Branch & Bound método de búsqueda sistemático para problemas combinatorios de optimización que exploran *técnicas de frontera superior e inferior* para *podar* efectivamente el árbol de búsqueda de una instancia dada del problema. En particular, la búsqueda puede ser podada en cualquier solución parcial en la cual el limite inferior exceda el limite superior actual en calidad de una solución óptima para la instancia del problema dado (en el caso de un problema de minimización). En el contexto de algoritmos de branch & bound para problemas de minimización, pueden ser usados los algoritmos de búsqueda local estocástica para obtener limites superiores en la calidad de la solución óptima de una instancia del problema dado. Los algoritmos de branch & bound están entre los mejor realizados métodos de búsqueda sistemática para muchos problemas de optimización combinatorios del tipo "hard", tal como el Problema de la Máxima Satisfacción.

Branch & Cut Método para resolver problemas de programación entera o integer programming que trabaja resolviendo una serie de relajaciones de programación lineal. En cada etapa, los cortes o *cuts*, que son, inecuaciones lineales que son satisfechas por todas las soluciones enteras factibles pero no por la solución optimal no entera de la relajación actual, son adicionadas al problema lineal de optimización; esto hace que la relajación se aproxime mas cerca a la solución optimal del problema original de programación entera. Este proceso se itera hasta encontrar que los "buenosçortes se conviertan en "hard", punto en el cual el problema actual se divide en dos sub-problemas, al cual se le aplica branch & cut recursivamente. Para muchos problemas, tales como el Problema del Vendedor Viajante, los algoritmos de branch & cut actualmente están entre los mejor realizados algoritmos completos de búsqueda.

Combinatorial Problem Problema en el cual, dado un conjunto de *soluciones componentes*, el objetivo es encontrar una combinación de estos componentes con ciertas propiedades. Para problemas combinatorios de decision, tales como el Satisfiability Problem (SAT), se establecen las propiedades deseadas en la forma de condiciones lógicas, mientras que en el caso de problemas de optimización combinatoria, tales como el del vendedor viajante (TSP), las propiedades deseadas mayormente consisten de objetivos de optimización que puede estar acompañados de condiciones lógicas adicionales. Muchos (pero no todos) los problemas combinatorios son difíciles de calcular y se resuelven buscando espacios exponencialmente largos de **soluciones candidatas**.

Completion Time O Tiempo de Terminación. Tiempo en el cual una operación o trabajo termina su procesamiento en un problema de programación.

Constraint Satisfaction Problem (CSP) problema de decisión prominentemente combinatorio en inteligencia artificial; dado un conjunto de variables y un conjunto de relaciones (**restricciones**) entre dichas variables, el objetivo de la variante de decisión es decidir si existe una asignación de valores a las variables tal que todas las restricciones se satisfagan simultáneamente - llamándose tal asignación *satisfying tuple*. El *Finite discrete CSP*, el caso

especial en el cual todas las variables tienen dominios finitos y discretos, es un problema NP-completo ampliamente estudiado de gran interés teórico y práctico.

Crossover Tipo de mecanismo de **recombinación** que se basa en el ensamble de piezas o fragmentos de una representación lineal de dos o más **soluciones candidatas** (*padres*) en uno o más nuevas **soluciones candidatas** (*offspring*). Crossover se usa en muchos **Evolutionary Algorithms**, a veces, el término es usado ampliamente para referirse a cualquier tipo de mecanismo de recombinación.

Earliness Termino contrario al **tardiness**, que denota la cantidad de tiempo en que se termina un trabajo, dado un programa de producción, antes de la fecha de entrega. En algunas aplicaciones, el hecho de entregar el trabajo antes de la fecha no incurre ningún costo adicional.

Espacio de Búsqueda Conjunto de **soluciones candidatas** de una instancia de un problema dado. El tamaño del espacio de búsqueda, que es, el número de soluciones candidatas, típicamente escala, por lo menos exponencialmente, el tamaño de la instancia de una familia de instancias del problema.

Evolutionary Algorithms (EAs) Clase amplia y diversa de los llamados **population based SLS algorithms** que se inspiran por el proceso de evolución biológico a través de la **selección**, **mutación** y **recombinación**. Los algoritmos evolucionarios son algoritmos iterativos que se inician con una población inicial de **soluciones candidatas** y luego aplican repetidamente una seria de *operadores genéticos* de selección, mutación y recombinación. Usando estos operadores, en cada iteración de un EA, la población actual se reemplazada (completa o parcialmente) por un nuevo conjunto de *individuos*, que son, las soluciones candidatas. Históricamente, varios tipos de EA se pueden distinguir: *genetic algorithms*, *evolutionary programming methods* y *evolution strategies*; pero más recientemente, las diferencias entre estos tipos se vuelven más borrosas. Muchos EA de alto rendimiento para **problemas combinatorios** usan un procedimiento de **búsqueda local** subsidiaria para mejorar las soluciones candidatas en cada iteración de el proceso de búsqueda; en muchos casos, estos algoritmos híbridos se basan

en algoritmos genéticos y por esto se refieren como *genetic local search method*. Ligeramente más general, los EAs usan procedimientos de búsqueda local subsidiaria también se llaman *memetic search methods* o *memetic algorithms*.

Fecha de Entrega fecha límite para la terminación de un trabajo en un problema de programación. Los problemas con fechas de entrega frecuentemente usan un número (prorateado) de los trabajos que son terminados después de sus respectivas fechas de entrega o la suma de sus retrasos prorateados como una función objetivo.

Búsqueda local (Local Search) método algorítmico para buscar un espacio dado de **soluciones candidatas** que comienza con una solución candidata inicial y luego se mueve iterativamente de una solución candidata a otra de su vecindario directo, basada en información local, hasta que una condición de terminación se satisfaga.

Diversificación Propiedad importante de un proceso de búsqueda; los *mecanismos de diversificación* ayudan a asegurar suficientemente una gran exploración del **espacio de búsqueda** para evadir la **stagnation** y estancamiento del proceso de búsqueda en regiones del espacio de búsqueda que no contiene **soluciones candidatas** de suficiente calidad. Es un asunto clave al momento de diseñar algoritmos estocásticos, llegar a un equilibrio entre diversificación y intensificación.

Intensificación Propiedad importante de un proceso de búsqueda; los *mecanismos de intensificación* buscan examinar cuidadosamente un área específica de un **espacio de búsqueda** dado para encontrar una solución o soluciones candidatas de mejor calidad. Las estrategias de intensificación se basan fuertemente en mecanismos heurísticos guía avaros.

Makespan Función objetivo comúnmente usada para **problemas de programación** que mide el **completion time** máximo de cualquier trabajo bajo un programa o secuencia candidata.

***MAX-MIN* Ant System (*MMAS*)** Clase de algoritmos **Ant COlony Optimization** que usa límites en los valores factibles de los rastros de la fe-

romona asi como mecanismos adicionales de **intensificación** y **diversificación** . *MMAS* forma la base para muchas aplicaciones exitosas de los algoritmos ACO a problemas de optimización NP-hard.

Memetic Algorithm (MAs) Clase de **population-based SLS algorithms** que combina los operadores evolucionarios de **mutación**, **recombinación** y **selección** con una **búsquea local** subsidiaria o, más generalmente, información heurística más específica del problema. MAs que usan la búsqueda local subsidiaria pueden ser vistos como algoritmos evolutivos que buscan un espacio de **soluciones candidatas** optimales locales.

Mutación Operación básica usada en los **Evolutionary Algorithms** para introducir modificaciones a un miembro de la población, esto es, a una **solución candidata**.

Permutación Orden de un conjunto de objetos; formalmente, una permutación sobre un conjunto de S con n elementos puede definirse como un mapeo de los enteros $1, ..., n$ a elementos de S tal que cada entero es mapeado a un único elemento en S.

Population Based SLS Method Clase de algoritmos de búsqueda local estocástica que mantiene una *población*, que es, un conjunto de **soluciones candidatas** para un problema dado. En cada paso de búsqueda, uno o más elementos de la población (p.e., soluciones candidatas individuales) pueden ser modificadas. El uso de una población de soluciones candidatas ayuda frecuentemente a lograr una **diversifciación** adecuada del procesos de búsqueda. Algunos ejemplos de este tipo de algoritmos son los **Evolutionary Algorithm** ,**Memetic Algorithm** y **Ant Colony Optimization**.

Recombinación Operación fundamental en un **Evolutionary Algorithm** o **Memetic Algorithm** que genera una o más nuevas **soluciones candidatas** (*offspring*) mediante la combinación de partes (componentes de solución) de dos o más de soluciones candidatas existentes (*parents*). En muchos casos, se usa un tipo especial de recombinación llamada **crossover**.

Restricción Relación entre los valores que ciertas variables se permiten tomar en una instancia del CST o **Constraint Satisfaction Problem.** El termino

es usado también para referirse a las condiciones lógicas en las propiedades de una **solución candidata** o solución componente para un **problema combinatorio** dado. *Hard constraints* representan condiciones que cualquier solución candidata necesita satisfacer par ser considerara útil en un contexto de aplicación dado, mientras que *soft constratints* captan las metas de optimización, no todas las cuales se satisfacen simultaneamente.

Selección Operación fundamental en un **Evolutionary Algorithm** o **Memetic Algorithm** usado para seleccionar individuos (p.e. soluciones candidatas) de una población actual para ser retenidas para la siguiente iteración.

Soluciones candidatas Elemento del **espacio de búsqueda** de un **problema combinatorio** dado; tipicamente un agrupamiento, orden o asignamiento de soluciones componentes. Para muchos problemas combinatorios, hay una distinción natural entre *solución candidata parcial*, la cual puede ser extendida con soluciones componentes adicionales (sin violar la estructura fundamental de una solución candidata para el problema dado), y *soluciones candidatas completas*, para las cuales no es ete el caso. Por ejemplo, en el caso de Satisfiability Problem, donde la soluciones componente naturales son asignaciones de valores verdaderos a variables proposicionales individuales, las soluciones parciales candidatas pueden asignar valores solo a algunas de las variables en la formula dada, mientras que las soluciones candidatas completas especifican valores verdaderos para todas las variables. Las soluciones candidatas también se llaman frecuentemente *posiciones de búsqueda*; algunos otros términos usados en literatura a veces son *estado de (búsqueda)* y *configuración de (búsqueda)*

Stagnation Situación en el cual la *eficiencia* de un proceso de búsqueda decrece, donde se define eficiencia formalmente basado en la taza de crecimiento a lo largo del tiempo en la probabilidad de solución. En el caso de algoritmos estocásticos, la Stagnation ocurre cuando el proceso de búsqueda se estanca (permanentemente o por una cantidad de tiempo sustancial) en un area de no solución de un espacio de búsqueda.

Surplus Es la diferencia entre la producción acumulada y la demanda acumulada.

Tardiness Cantidad de tiempo por el cual un trabajo en un problema de programaón se completa después de la fecha de entrega. En aplicaciones practicas, el tardiness incurre costos adicionales, tal como las penalidades de entregas fuera de fecha.

Tiempo Computacional (CPU time) Medida del tiempo de corrida o run time de un algoritmo que se basa en el tiempo real de computación usado por las respectiva implementación (proceso) de un procesador dado, típicamente medido en segundos, donde un segundo de CPU corresponde a un segundo de reloj de pared durante el cual el CPU ejecuta sólo el proceso dado. El CPU time se mayormente independiente de otros procesos corriendo al mismo tiempo en un sistema operativo multi tarea, pero depende del tipo de procesador (modelo, velocidad, memoria cache) y puede ser afectado por otros aspectos del entorno de ejecución, tal como memoria principal, compilador y características del sistema operativo. Cuando se reporta runtimes en segundos de CPU, se debe, por lo menos, mencionar el modelo del procesador y velocidad, la cantidad de memoria RAM, y también el tipo de sistema operativo y al version deben ser especificadas.

Tiempo de Terminación Tiempo en el cual una operación o un trabajo (Job) termina de procesar en un problema de programación.

BIBLIOGRAFÍA

[1] J.H.M. Korst y P.J.M. van Laarhoven Aarts, E.H.L., *Simulated annealing*, Local Search in Combinatorial Optimization (Wiley, Chichester) (E. Aarts and J. Lenstra, eds.), 1997, pp. 91–120.

[2] P.Laarhoven et al. Aarts, E., *Job shop scheduling by simulated annealing*, Technical Report OS-R8809, Centre for matematics and computer science, Amsterdam, 1988.

[3] Hezberger J. Alefeld, G., *Introduction to interval computations*, Academic Press, New York, 1983.

[4] Stefan Voss Andreas Fink, *Solving continuous flow-shop scheduling problem by metaheuristics*, Tech. report, Technical University of Braunschweig, Department of Business Administration, Informatio Systems and Information Managment, Abt-Jeruisalem Str. 7, 38106 Braunschweig, Germany, January 2001.

[5] T. Bäck, *Evolutionary algorithms in theory and practice*, Oxford Univeristy Press, New York, 1996.

[6] S. Uckum et al. Bagchi, S., *Exploring problem-specific recombination operators for job shop scheduling*, Proceedings of the Fifth International Conference on Genetic Algorithms, 1991.

[7] J. Adams et al. Balas, E., *The shifting bottleneck procedure for job shop scheduling*, Management Science **34** (1988), no. 3, 391–401.

[8] R. Battiti, *Modern heuristics search methods*, chichester ed., ch. Reactive search: Toward self-tuning heuristics, pp. 61–83, V. Rayward-Smith, I. Osman, C. Reeves, y G. Smith, Wiley, 1996.

[9] F. F. Boctor, *Some efficient multi-heuristic procedures for resource-constrained project scheduling*, European Journal of Operation Research, vol. 49, 1990, pp. 3–13.

[10] Sunwon Park Bok, Jin-Kwang, *Continuous-time modeling for short-term shceduling of multipurpose pipeles plants*, Industrial & Engineering Chemistry Research **37** (1998), 3652–3659.

[11] R. Bruns, *Direct chromosome representation and advanced genetic operators for production scheduling*, International Conference on Genetic Algorithms, 1993, pp. 352–359.

[12] L. Lei C-Y Lee and M. Pinedo., *Current trend in deterministic scheduling*, Annals of Operations Research **70** (1997), 1–41.

[13] J. Carlier and E.Ñeron, *An exact method for solving the multi-processor flowshop*, RAIRO-Recherche Operationnelle **34** (2000), no. 1, 1–25.

[14] A.P.; Matos H.A. & Novais A. Q. Castro, P.M.; Barbosa-Povoa, *Simple continuous-time formulation for short-term scheduling of batch and continuous processes*, Industrial and Engineeering Chemistry Research, vol. 43, 2004, pp. 105–118.

[15] Barbosa-Póvoa A.P.F.D & Matos H. Castro, P., *An improved rtn continuous-time formulation for the short-term scheduling of batch plants*, Industrial and Engineering Chemistry Research, vol. 40, 2001, pp. 2059–2068.

[16] V. Cerny, *Thermodynamical approach to teh traveling salesman problem: An efficient simulation algorithm*, Journal of Optimization Theory and Applications **45** (1985), 41–51.

[17] G.A. Cleveland and S.F.Smith, *Using genetic algorithms to schedule flow shop releases*, Proceedings of the Third International Conference on Genetic Algorithms., 1989, pp. 264–270.

[18] E. W. y J. H. Patterson. Davis, *A comparison of heuristic and optimum solutions in resource-constrained project scheduling*, Management Science, vol. 21, 1975, pp. 944–955.

[19] E.W. y G. E. Heidorn. Davis, *An algorithm for optimal project scheduling under multiple resource constraints*, Management Science, vol. 17, 1971, pp. B–803–b817.

[20] L. Davis, *Job shop scheduling with genetic algorithms*, Proceedings of an International Conference on Genetic Algorithms and their Applications (Pittsburg) (Lawrence Erlbaum Associates, ed.), 1985.

[21] D.B.Fogel, *An introduction to simulated evolutionary optimization*, IEEE Transactions on Neural Networks **5** (1994), no. 1, 3–14.

[22] Goldberg D. E., *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, 1989.

[23] M. L. Espinouse, *Flowshop et extensions: chevauchement des taches, indisponibilite des machines et systeme de transport*, Phd thesis, University of Joseph Fourier-Grenoble 1, 1998.

[24] L.J. Fogel., *Toward inductive inference automata*, Proceedings of the International Federation for Information Processing Congress (Munich), 1962, pp. 395–399.

[25] y S. F. Smith Fox, M. S., *Isis-a knowledge-based system for factory scheduling*, Expert Systems **1** (1984), no. 1, 25–49.

[26] F. y M. Laguna Glover, *Tabu search*, Kluwer, Dordrecht, 1997.

[27] O. Martin H. Ramalhino Lourenco and T. Stützle, *Iterated local search*, Handbook of Metaheuristics, http://www.intellektik.informatik.tu-darmstadt.de/ tom/pub.html, f. glover and g. kochenberger ed., to appear.

[28] M.Mori H. Tamaki and M. Araki, *Generation of a set of pareto-optimal solutions by genetics algorithms*, Transactions Soc. Instrum. Contr. Eng. **31** (1995), no. 8, 1185–1192.

[29] y M. L. Ginsberg Harvey, W. D., *Limited discrepancy search*, CIRL, Eugene, OR, USA, 1995.

[30] M. Held and R. M. Karp, *A dynamic programming approach to sequencing problems*, Journal of the Society for Industrial and Applied Mathematics **10** (1962), no. 1, 196–210.

[31] A. Hertz and D.Kobler, *A framework for the description of evolutionary algorithms*, European Journal of Operational Research **126** (2000), 1–12.

[32] D. Hildum, *Flexibility in a knowledge-based system for solving dynamic resource-constrained scheduling problems*, Umass CMPSCI Technical Report 94-97 (1994).

[33] G.G. Liepins et al. Hilliard, M.R., *Machine learning applications to job shop scheduling*, Proceedings of the AAAI-SIGMAN Workshop on Production Planning and Scheduling, 1988.

[34] y Tadahiko Murata Hisao Ishibuchi, *A multi-objective genetic local search algorithm and its application to flowshop scheduling*, IEEE Transactions on Systems, Man, and Cybernetics-Part C:Applications and Reviews **28** (1998), no. 3, 392–402.

[35] J.H. Holland, *Adaption in natural and artificial systems*, Ann Harbor, MI, 1975.

[36] Gupta A. Hui, C.-W. and H.A.J. van der Meulen, *A novel milp formulation for short-term scheduling of multi-stage multi-product batch plants with sequence-dependent cosntraints.*, Computers and Chemical Engineering **24** (2000), no. 2705.

[37] P. Husbands, *Genetic algorithm for scheduling*, AISB Quaterly, 89 ed., 1996.

[38] P. Husbands and F.Mill, *Simulated co-evolution as the mechanism for emergent planning and scheduling*, Proceegind from the International Conference on Genetic Algorithm and their Applications, 1991, pp. 264–270.

[39] H. Itoh, T. y Ishii, *Fuzzy due-date scheduling problem with fuzzy processing time*, APORS 97 (1997).

[40] y D. E. Goldberg J. Horn, N.Ñafpliotis, *A niched pareto genetic algorithm for multi-objective optimization*, 1st IEEE Int. Conf. Evolutionary Computation, 1994, pp. 82–87.

[41] C.R. Aragon L.A. McGeoch y C. Schevon Johnson, D.S., *Optimization by simulated annealing: an experimental evaluation*, Operations Research, vol. 37, 1989, pp. 865–892.

[42] J.J. Kanet and V.Sridharan, *Progenitor: A genetic algorithm for production scheduling*, Tech. report, Wirtschaftsinformatik, 1991.

[43] C. Gelatt Jr. Kirkpatrick, S. and M. Vecchi, *Optimization by simulated annealing*, Science, no. 220, 1983, pp. 671–680.

[44] W. H. Kohler and K. Steiglitz, *Exact, approximate, and guaranteed accuracy algorithms for the flowshop problem n/2/f/f.*, Journal ACM **22** (1975), no. 1, 106–114.

[45] F. Kursawe, *A variant of evolution strategies for vector optimization*, ch. Parallel Problem Solving from Nature, pp. 193–197, Springer Verlag, Berlin, Germany, 1991.

[46] S. R. y T. E. Morton. Lawrence, *Resource-constrained multi-project scheduling with tardy costs: Comparing myopic, bottleneck, and resource pricingn heuristics.*, European Journal of Operacional Research, vol. 64, 1993, pp. 168–187.

[47] S. Chung H. K. Lee I. B. Lee Lee, K. H., *Continuous time formulation of short-term scheduling for pipeless batch plants*, Journal of Chemical Engineering of Japan **34** (2001), no. 10, 1267–1278.

[48] A.J. Owens L.J.Fogel and M.J.Walsh, *Artificial intelligence through simulated evolution*, Wiley, 1966.

[49] M. A. Carvalcanti Pacheco M. Rodriguez de Almeida, *Applying genetic algorithms to the production scheduling of a petroleum refinery*, Technical report, Pontificia Universidad Católica do Rio de Janeiro, Rio de Janeiro, Brasil, 2003.

[50] C.M. McDonnald and I. Karimi, *Planning and scheduling of parallel semi-continuous process. 2:short-term scheduling*, Industrial and Engineering Chemistry Research **36** (1997), no. 2701.

[51] G.P. & Cerdá J. Méndez, C.A.;Henning, *An milp continuous-time approach to short-term scheduling of resource-constrained multistage flowshop batch facilities*, Computers and Chemical Engineering **25** (2001), 701–711.

[52] J. Méndez, C.A. & Cerdá, *An milp continuous-time framework for short-term scheduling of multipurpose batch processes under different operation strategies.*, Optimization & Engineering **4** (2003a), 7–22.

[53] ______, *Dynamic scheduling in multiproduct batch plants*, Computer and Chemical Engineering **27** (2003b), 1247–1259.

[54] ______, *An milp framework for batch reactive scheduling with limited discrete resources*, Computers and Chemical Engineering **28** (2004a), 1059–1068.

[55] ______, *Short-term scheduling of multistage batch processes subject to limited finite resources.*, Computer-Aided Chemical Engineering **15B** (2004b), 984–989.

[56] Michalewicz, *Genetic algorithm + data structures = evolution programs*, Springer, Berlin, 1994.

[57] Z. Michalewicz and M. Michalewicz, *Evolutionary computation techniques and their applications*, Proceedings of the IEEE International Conference on Intelligent Processing Systems (Beijing, China), Insititute of Electrical & Electronics Engineers, 1997, Incorporated, pp. 14–24.

[58] R.Ñakano, *Conventional genetic algorithm for job shop scheduling*, Fifth International Conference on Genetic Algorithm, Morgan Kaufmann Publishers, 1991.

[59] Norbis and Smith, *Two level heuristic for the resource constrained scheduling problem*, Int. J. Production Res, vol. 24, 1986, pp. 1203–1219.

[60] A.Hertz D.Kobler P. Calegari, G.Coray and P.Kuonen, *A taxonomy of evolutionary algorithms in combinatorial optimization*, Journal of Heuristics **5** (1999), 145–158.

[61] y D. V. Gucht P. Jog, J. Y. Suh, *The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the travelling salesman problem*, 3rd. Int. Conf. Genetic Algorithms, 1989, pp. 110–115.

[62] C.C. Pantelides, *Unified frameworks for optimal process planning and scheduling.*, Foundations of Computer-Aided Process Operations (New York) (Davis JF Rippin DWT, Hale JC, ed.), Cache Publications, 1994, pp. 253–274.

[63] S. S. y W. Iskander Panwalker, *A survey of scheduling rules*, Operation Research, vol. 25, 1977, pp. 45–61.

[64] J. H. Patterson, *A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem*, Managment Science, vol. 30, 1984, p. 854.

[65] M. Pinedo (ed.), *Scheduling:theory, algorithms and systems*, Prentice-Hall, 1995.

[66] J. Pinto and I.E. Grossmann, *A continuous time mixed integer linear programming model for short-term scheduling of multistage batch plants*, Industrial and Engineering Chemistry Research **34** (1995), no. 3037.

[67] ———, *An alternate milp model for short-term scheduling of batch plants with pre-ordering constraints.*, Industrial and Engineering Chemistry Research **35** (1996), no. 338.

[68] W. Maxwell R. Conway and L. Miller, *Theory of scheduling*, Addison-Wesley, 1967.

[69] J. K. Lenstra R. L. Graham, E. L. Lawler and A. H. G. Rinnooy Kan, *Optimization and approximation in deterministic sequencing and scheduling: a survey*, Annals of Discrete Mathematics **5** (1979), 287–326.

[70] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution*, Frommann-Holzboog, 1973.

[71] R. Z. Rios-Mercado and J. F. Bard, *Computational experience with a branch-and-cut algorithm for flowshop scheduling for flowshop scheduling setups.*, Computers and Operations Research **25** (1998), no. 5, 351–366.

[72] ______, *A branch-and-bound algorithm for permutation flow shops with sequence.*, IIE Transactions **31** (1999), no. 8, 721–731.

[73] N. Sadeh, *Look-ahead techniques for micro-opportunistic job shop scheduling*, Phd thesis, School of Computer Science, Carnegie Mellon University, Pittsburg, PA, March 1991.

[74] S. E. y E. N. Weiss Sampson, *Local search techniques for the generalized resource constrained project scheduling problem*, Naval Research Logistics, vol. 40, 1993, p. 665.

[75] S. Sayin and S. Karabati, *A bicriteria approach to the two-machine flow shop scheduling problem*, European Journal of Operational Research **113** (1999), 435–449.

[76] J. D. Schaffer, *Multi-objective optimization with vector evaluated genetic algorithms*, 1st Int. Conf. Genetic Algorithms, 1986, pp. 93–100.

[77] Stephen Smith and Peng Si Ow, *The use of multiple problem decompositions in time constrained planning tasks*, Tech. Report CMU-RI-TR-85-11, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 1985.

[78] A. Sprecher and A. Drexl, *Manuskripte aus den instituten für betriebswirtschaftslehre der universität kiel.*, Universität Kiel., 1996.

[79] G. Syswerda, *The application of genetic algorithm to resource scheduling*, Proceedings from the Fourth International Conference on Genetic Algorithms, 1990, pp. 502–508.

[80] ______, *Schedule optimization using genetic algorithms*, handbook of genetic algorithm 21, New York, 1991.

[81] H. Ishibuchi T. Murata, *Perfgormance evaluation of genetic algorithms for flowshop scheduling problems*, 1st IEEE Int. Conf. Evolutionary Computation, 1994, pp. 812–817.

[82] V. Tkindt and J.C. Billaut, *Multicriteria scheduling - theory, models and algorithms*, Springer-Verlag, 2002.

[83] P.J.M. van Laarhoven and E.H.L. Aarts, *Simulated annealing: Theory and applications*, Dordrecht, 1987.

[84] D. L. y E. Zenel Woodruff, *Hashing vectors for tabu search*, Annals of Operations Research, vol. 41, 1993, pp. 123–138.

[85] T. Murata y H. Ishibuchi, *Moga: Multi-objective genetic algorithm*, 2nd IEEE Int. Conf. Evolutionary Computat., 1995, pp. 289–294.

[86] C. M. Fonseca y P. J. Fleming, *Genetic algorithms in multiobjective optimization: Formulation, discussion and generalization*, 5th Int. Conf. Genetic Algorithms, 1993, pp. 416–423.

[87] ———, *Genetic algorithms in multiobjective optimization*, Evolutionary Computation, vol. 3, 1995, pp. 1–16.

[88] R. R. Yager, *A procedure for ordering fuzzy subsets of the unit interval*, Information Sciences, vol. 24, 1981, pp. 143–161.