UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESIS PARA LA MAESTRÍA EN AUTOMATIZACIÓN DE OFICINAS

ALGORITMOS EVOLUTIVOS AVANZADOS COMO SOPORTE DEL PROCESO PRODUCTIVO

Realizada por la Lic. Carolina Salto

Dirigida por el Dr. Raúl Gallard

Octubre de 2000

RESUMEN

El mundo de los negocios actuales está sufriendo muchos cambios, ya no basta con generar reportes y realizar una correcta planificación. Se deben incluir herramientas de optimización para crear soluciones de negocios adaptativas como por ejemplo para límites de créditos, precios y descuentos, y scheduling. Esto redundará en beneficios para la empresa ya sea en la disponibilidad de tecnología de avanzada como también en la disminución de los costos asociados a la toma de decisiones óptimas, también incrementará la capacidad para aprender de experiencias previas y para adaptar a cambios en el mercado.

En estos últimos años se han realizados muchos estudios de investigación respecto de la aplicación de las técnicas de computación evolutiva para la solución de problemas de scheduling. La principal ventaja de las técnicas evolutivas es su habilidad para proveer buenas soluciones a problemas extremadamente complejos usando tiempos razonables.

En este trabajo se hace un revisión de las clases y características de algoritmos evolutivos así como también algunas mejoras introducidas a los mismos. Entre estas últimas se pueden incluir múltiple crossover, multiplicidad de padres y prevención de incesto. Asimismo se presentan algunas variantes de algoritmos evolutivos planteados para la resolución de un problema particular de scheduling como lo es el problema de job shop scheduling.

TABLA DE CONTENIDOS

LISTA DE FIGURAS

LISTA DE TABLAS

INTRODUCCION

La década de los 90 se puede caracterizar como la década de la integración corporativa de los sistemas de información computarizados. Casi todas las empresas, independientemente de su rubro de actividad o tamaño han experimentado en los últimos años una asimilación significativa de tecnología de información tendiente al soporte de los procesos administrativos y operacionales. Probablemente, la próxima década tendrá como elemento distintivo la incorporación de sistemas avanzados para el soporte de decisiones que maximicen la utilidad de la enorme inversión realizada en recursos de tecnología de información pero que bajo cualquier indicador de productividad se encuentran hoy poco explotados.

Un área particularmente estratégica cuya importancia es ampliamente reconocida es el uso de herramientas computacionales avanzadas para la planificación, programación y control de la producción. Estudios realizados por consultoras internacionales resaltan a estas herramientas como diferenciadores claves de competitividad.

Este trabajo está basado en el análisis de técnicas evolutivas justamente para aportar soluciones al problema de programación de operaciones, el cual se puede traducir en la resolución de un problema de scheduling. Este último está relacionado con la asignación de recursos limitados a tareas sobre el tiempo; es un proceso de toma de decisiones que tiene por propósito la optimización de uno o más objetivos. El problema de scheduling existe en la mayoría de los sistemas de producción y manufactura como también en la mayoría de los ambientes de procesamiento de información. Existe también en los ambientes de transporte y distribución y en otros tipos de industrias de servicio.

El problema de job shop scheduling clásico es uno de los problemas de scheduling más conocidos. Informalmente, se puede describir por un conjunto de jobs y un conjunto de máquinas. Cada job consiste de una cadena de operaciones, las cuales necesitan ser procesadas durante un período de tiempo ininterrumpido sobre una determinada máquina. Cada máquina puede procesar una operación a la vez. Un schedule es la asignación de operaciones en intervalos de tiempo sobre las máquinas. El asunto es hallar un schedule de largo mínimo.

El problema de job shop scheduling es uno de los problemas de optimización más duros. Debido a su intratabilidad los procedimientos heurísticos son una buena

alternativa. La mayoría de los procedimientos heurísticos convencionales usan reglas de prioridad, es decir, una regla para elegir una operación desde un subconjunto de operaciones aún no asignadas. En los recientes años, se ha puesto énfasis especial en usar búsqueda local para resolver el problema de job shop scheduling debido al desarrollo de métodos de búsqueda local probabilísticos tal como simulated annealing [139], tabu search [31] y algoritmos evolutivos [10, 23, 26, 32, 54, 68, 70, 85, 108, 113, 122, 129, 136, 148, 138].

En este trabajo se pretende mostrar que los algoritmos evolutivos son una eficiente herramienta de optimización a disposición de los administradores de una organización en cuanto a la resolución del problema de asignación de recursos se trate. Por consiguiente, se dan las características de los algoritmos evolutivos y se introduce el problema de scheduling. En la parte final del trabajo se presentan resultados experimentales para respaldar tal situación, se realiza un análisis de los resultados obtenidos por tres algoritmos evolutivos distintos para la resolución de este problema.

El trabajo de tesis se organiza como se describe a continuación. En el capítulo 1 se provee una introducción respecto de los sistemas de información dentro de una organización, la clasificación de los mismos y además se pone en contexto a los sistemas de scheduling o de programación de operaciones. En el capítulo 2 se tratan los conceptos preliminares de scheduling: la notación y una revisión de los ambientes de máquinas. El capítulo 3 versa sobre los aspectos teóricos y operativos de los algoritmos evolutivos. Se describen diferentes variantes de algoritmos evolutivos tal como algoritmos genéticos (GA), estrategias evolutivas (ES), programación evolutiva (EP) y programación genética (GP). En el capítulo 4 se introducen los conceptos básicos de los algoritmos genéticos, se describe la estructura de los mismos a los efectos de reflejar las prácticas actuales de los métodos de algoritmos genéticos. En el capítulo 5 se hace una revisión de opciones desarrolladas para la recombinación y matting, tal como multiplicidad de padres y crossovers, y prevención de incesto. El capítulo 6 se concentra sobre los esquemas de representaciones propuestos para el job shop scheduling. En el capítulo 7 se presentan algunas opciones evolutivas desarrolladas para la resolución del problema de job shop scheduling, incluyendo descripciones de experimentos y análisis de resultados. La última sección de este trabajo presenta las conclusiones y trabajos futuros.

Capítulo I

SISTEMAS DE INFORMACIÓN

1.1. INTRODUCCIÓN

Los sistemas de información están creando muchas oportunidades para las empresas, pero son también fuentes de nuevos problemas y de cambios [94].

Aunque la tecnología de información está avanzando a pasos agigantados, no hay nada predeterminado o mecánico respecto de la construcción y uso de los sistemas de información. Hay cinco cambios claves a los cuales los gerentes de una empresa deben hacer frente:

1. *estrategia del negocio*: cómo usar la información para diseñar organizaciones que sean competitivas y eficientes?
2. *globalización*: cómo las firmas pueden entender los requerimientos de sistemas y de negocio de un ambiente económico global? El rápido crecimiento en el mercado internacional y el surgimiento de una economía global requieren sistemas de información que puedan soportar tanto la producción como la venta de productos en diferentes países.
3. *arquitectura de la información*: cómo una organización puede desarrollar una arquitectura de información que soporte sus objetivos de negocio? Los sistemas nuevos actuales frecuentemente requieren rediseñar la organización y desarrollar una nueva arquitectura de sistemas de información. Estos sistemas son la forma particular en que la tecnología de la información entra en una organización para cumplir los objetivos o funciones seleccionadas. La figura 1.1 (extraída de [94]) muestra los principales elementos de la arquitectura de los sistemas de información que los administradores necesitarán desarrollar. Aunque los sistemas base de computación los operan típicamente el personal técnico, la administración general deberá decidir cómo asignar los recursos de hardware, software y telecomunicaciones. El resto de los sistemas base de computación son los sistemas de aplicación de empresas. Como los gerentes y empleados directamente

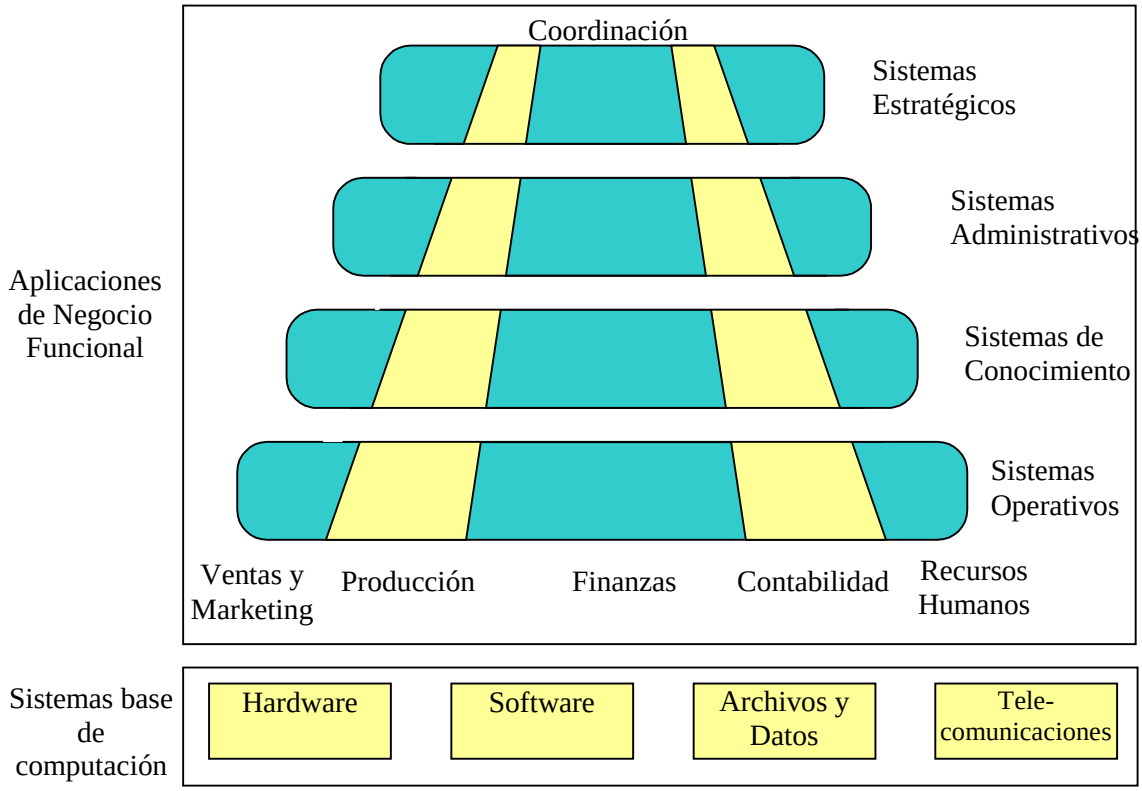


Figura 1.1 Organización de la arquitectura de los sistemas de información

interactúan con esos sistemas, es de especial interés para el éxito de la organización que los sistemas reúnan los requerimientos funcionales y de negocio.

4. *Inversión en sistemas de información*: cómo las organizaciones pueden determinar el valor de negocio de los sistemas de información?
5. *Responsabilidad y control*: cómo las organizaciones pueden diseñar sistemas que la gente pueda controlar y entender? Cómo las organizaciones aseguran que sus sistemas de información se usan de una manera ética y social? Los sistemas de información son esenciales para el negocio, el gobierno y la vida diaria, por lo tanto las organizaciones deben poner especial cuidado en asegurar su exactitud, confiabilidad y seguridad. Los sistemas de información se deben diseñar para que funcionen acorde con la intención inicial y además para que los responsables puedan controlar el proceso.

1.2. CLASES DE SISTEMAS

Dada la existencia de distintos intereses, especialidades y niveles dentro de una organización, hay diferentes clases de sistemas. Un único sistema puede proveer toda la

información que una organización necesita. La figura 1.2 [94] presenta una forma de describir las clases de sistemas halladas en una organización. La organización está dividida en los siguientes niveles:

- ✓ *Estratégico*.
- ✓ *Administrativo*.
- ✓ *Del conocimiento*.
- ✓ *Operativo*.

Estos a su vez, se dividen en áreas funcionales tales como marketing, manufactura, finanzas, contabilidad y recursos humanos. Los sistemas se construyen para servir esos diferentes intereses de la organización.

Se pueden reconocer cuatro tipos principales de sistemas de información que sirven a los diferentes niveles de la organización: sistemas a nivel operativo, sistemas a nivel de conocimiento, sistemas a nivel administrativo y sistemas a nivel estratégico.

Los sistemas a nivel operativo son sistemas que monitorean las actividades y transacciones elementales de la organización, tal como ventas, ingresos, depósitos, decisiones para asignar créditos, y el flujo de materiales en una empresa. Ejemplos de

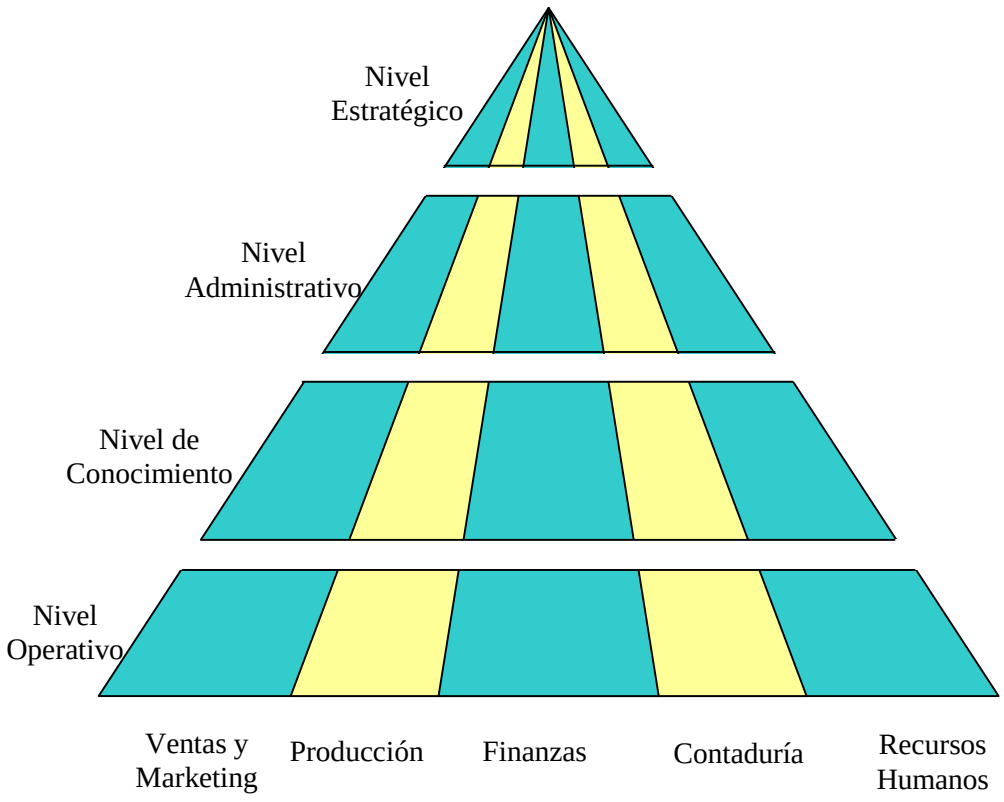


Figura 1.2 Tipos de sistemas de información

sistemas a nivel operativo incluyen sistemas para registrar los depósitos bancarios desde un cajero automático o sistemas para registrar el número de horas trabajadas cada día por los empleados en un piso de una fábrica.

El propósito de los sistemas a nivel de conocimiento es ayudar a la empresa a que integre nuevo conocimiento y a que la organización pueda controlar el flujo del trabajo.

Los sistemas a nivel administrativo se diseñan para servir al monitoreo, al control, a la toma de decisiones, y a las actividades administrativas de los gerentes intermedios. Los sistemas a nivel administrativo típicamente proveen reportes periódicos más que información instantánea de las operaciones.

Los sistemas a nivel estratégico ayudan a los gerentes senior a hacer frente y a dirigir asuntos estratégicos, tanto de la firma como del ambiente externo. Dan soporte a las actividades de planificación a largo plazo.

Los sistemas de información se pueden diferenciar por la especialidad funcional. Las funciones más importantes de una organización, tales como ventas y marketing, manufactura, finanzas, contabilidad, y recursos humanos, cuentan con sus propios sistemas de información. En organizaciones grandes, las subfunciones de cada una de esas funciones principales tienen a su vez sus propios sistemas de información. Por ejemplo, la función de manufactura podría tener sistemas para el manejo de inventario, control de procesos, planificación de operaciones, mantenimiento de planta, ingeniería asistida por computadoras, y planificación de requerimientos de material.

1.3. TIPOS DE SISTEMAS DE INFORMACIÓN

Diferentes organizaciones tienen diferentes sistemas de información para las mismas áreas funcionales. Como no hay dos organizaciones con exactamente los mismos objetivos, estructuras o intereses, los sistemas de información deben ser hechos a medida para atacar las características únicas de cada una.

La figura 1.3 [94] muestra los tipos específicos de sistemas de información que corresponden a cada nivel de la organización. La organización tiene sistemas de soporte ejecutivo (ESS) al nivel estratégico; sistemas de información administrativo (MIS) y sistemas de soporte de decisión al nivel administrativo; sistemas de *knowledge work* (KWS) y sistemas de automatización de oficinas (OAS) al nivel de conocimiento; y sistemas de procesamiento de transacciones (TPS) al nivel operativo. Los sistemas

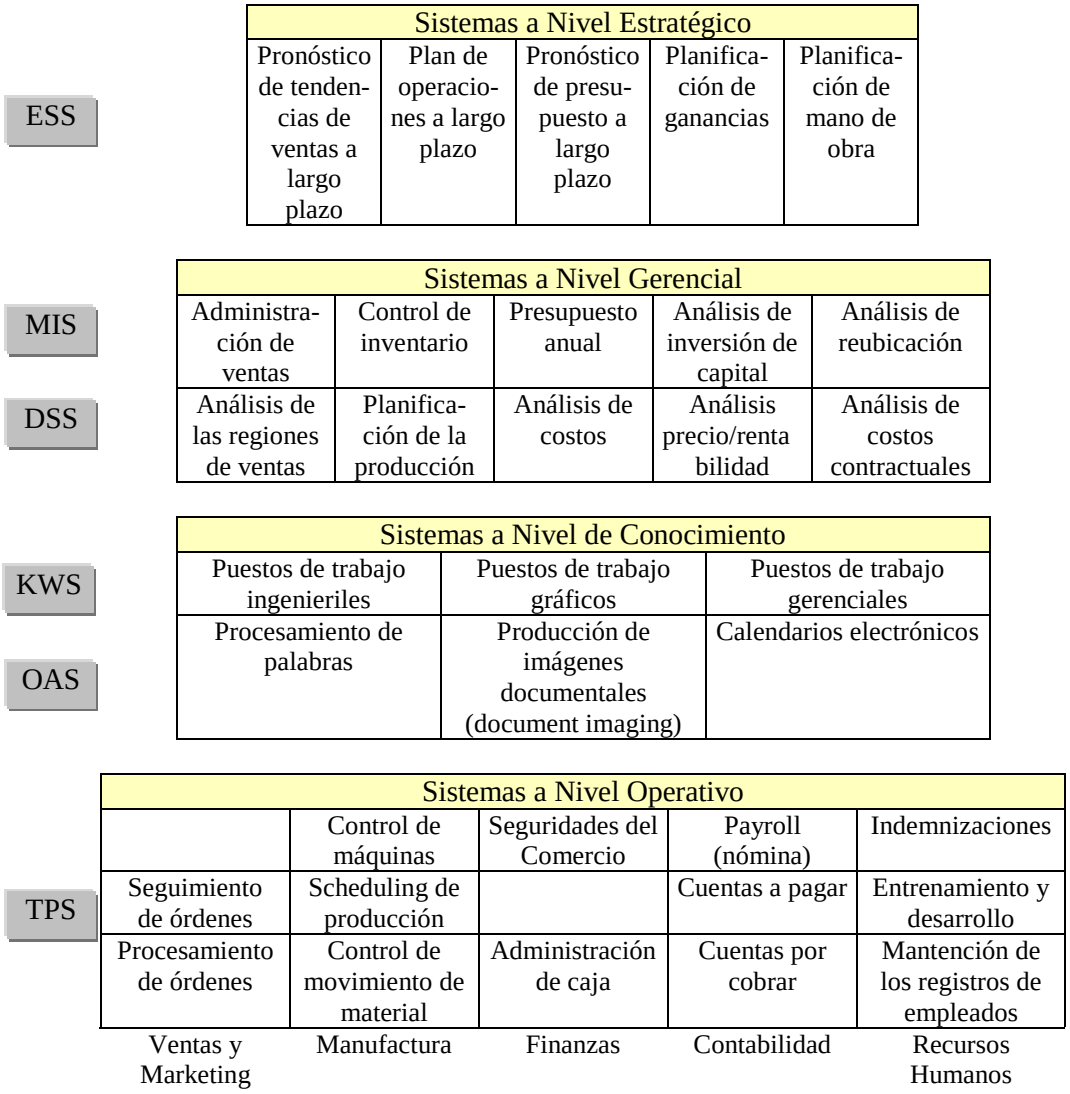


Figura 1.3 Tipos de sistemas de información.

correspondientes a cada nivel a su vez se especializan para servir a cada una de las principales áreas funcionales. Así, los sistemas típicos hallados en las organizaciones se diseñan para asistir a empleados o gerentes de cada nivel y en las funciones de ventas y marketing, producción, finanzas, contabilidad, y recursos humanos

1.3.1. SISTEMAS DE PROCESAMIENTO DE TRANSACCIONES

Los Sistemas de Procesamiento de Transacciones (TPS) son los sistemas básicos que sirven al nivel operativo de la organización. Un sistema de procesamiento de transacciones es un sistema computarizado que realiza y registra las transacciones de diarias necesarias para el que dirige la empresa.

A nivel operativo, las tareas, los recursos y los objetivos están predefinidos y altamente estructurados.

En la figura 1.4 [94] se identifican algunas aplicaciones típicas de un TPS. La figura muestra que hay cinco categorías funcionales de TPS: ventas/marketing, manufactura/producción, finanzas/contabilidad, recursos humanos, y otros tipos de TPS que son únicos a una industria particular.

Toda organización tiene cinco clases de TPS (aún en los sistemas manuales). Los sistemas de procesamiento de transacciones son frecuentemente tan centrales para una empresa que una falla por unas pocas horas puede producir mucho daño a una empresa.

Los gerentes necesitan TPS para monitorear el estado de las operaciones internas y las relaciones de la firma con el ambiente externo. Los TPS son los productores de información para los otros tipos de sistemas.

Tipos de TPS	**Principales funciones del sistema**	**Principales sistemas de aplicación**
Sistemas de Venta/Marketing	Administración de ventas Búsqueda de mercados Promoción Precios Nuevos productos	Sistemas de información de órdenes de ventas Sistemas de búsqueda de mercados Sistemas de precios
Sistemas de Manufactura/Producción	*Scheduling* *Compras* *Embarque/recepción* *Ingeniería* *Operaciones*	*Sistemas de planificación de recursos* *Sistema de planificación de operaciones* *Sistemas de control de órdenes de compra* *Sistemas ingenieriles* *Sistemas de control de calidad*
Sistemas de Finanzas/Contabilidad	Presupuesto Contabilidad Facturación Costos	Sistemas de contabilidad Cuentas por cobrar y por pagar Presupuestos Sistemas de administración de fondos
Sistemas de Recursos Humanos	Registros de personal Beneficios Indemnizaciones Relaciones laborales Entrenamiento	Payroll Registros de empleados Sistemas de beneficios Sistemas de ascensos del personal
Otros tipos (ej. universidad)	Admisión Registro de grados Registro de cursos Alumnado	Sistemas de registración Sistema de inscripción de alumnos Sistemas de control de clasificación de currículum Sistema de beneficios de alumnos

Figura 1.4 Aplicaciones típicas de TPS.

1.3.2. SISTEMAS DE AUTOMATIZACIÓN DE OFICINAS Y DE KNOWLEDGE WORK

Los sistemas de Automatización de Oficinas (OAS) y de Knowledge Work (KWS) proveen la información necesaria al nivel de conocimiento de la organización.

Los KWS, tal como estaciones de trabajo de diseño ingenieril o científico, promueven la creación de nuevo conocimiento y asegura que nuevo conocimiento y experiencia técnica se integre propiamente en la empresa.

Los OAS son aplicaciones de tecnología de información diseñadas para incrementar la productividad de los encargados del procesamiento de la información, al soportar las actividades de coordinación y de comunicación de la oficina típica.

1.3.3. SISTEMAS DE INFORMACIÓN GERENCIAL

Los Sistemas de Información Gerencial (MIS) se aplican en el nivel gerencial de las organizaciones, brindando reportes y, en algunos casos, con accesos on-line a los registros de performance actual de la organización y a los registros históricos. Típicamente están orientados a eventos internos, no a ambientales ni a externos. Primariamente MIS sirve para las funciones de planeamiento, control y toma de decisiones del nivel gerencial. Generalmente, estos sistemas dependen de los sistemas de procesamiento de transacciones subyacentes.

Las características de estos sistemas son las siguientes:

- ✓ Soportan decisiones estructuradas a los niveles de control administrativo y operativo. Sin embargo, son útiles para los gerentes senior para propósitos de planificación.
- ✓ Son orientados al control y reporte. Son diseñados para generar reportes sobre operaciones existentes y para realizar el control diario de las operaciones.
- ✓ Utilizan datos corporativos existentes y flujos de datos.
- ✓ Tienen poca capacidad analítica.
- ✓ Ayudan generalmente a la toma de decisiones usando datos pasados y presentes.
- ✓ Son relativamente poco flexibles.
- ✓ Tienen una orientación interna más que externa.

1.3.4. SISTEMAS DE SOPORTE DE DECISIÓN

Los Sistemas de Soporte de Decisión (DSS) ayudan al nivel gerencial de la organización. Los DSS ayudan a los gerentes a tomar decisiones que son semiestructuradas, únicas o que cambian rápidamente, y que no es posible especificarla de ante mano. Las características de los DSS son las siguientes:

- ✓ Ofrece a los usuarios flexibilidad, adaptabilidad y rápida respuesta.
- ✓ Opera con poca asistencia desde los profesionales de la computación.
- ✓ Provee soporte para decisiones y problemas cuyas soluciones no pueden ser especificadas en forma prematura.
- ✓ Usa análisis de datos y herramientas de modelado sofisticados.

1.3.5. SISTEMAS DE SOPORTE EJECUTIVO

Los gerentes senior usan una categoría de sistemas de información llamada Sistemas de Soporte Ejecutivo (ESS). Estos sirven al nivel estratégico de la organización. Permiten tomar decisiones no estructuras y crear ambientes de comunicaciones y computación generalizados. Se diseñan para incorporar datos de eventos externos tales como nuevas leyes de impuestos o competidores, pero también muestran información resumida desde MIS y DSS internos. Filtran, compactan y siguen el curso de datos críticos, poniendo énfasis en la reducción de tiempo y esfuerzo requeridos para brindar información útil a ejecutivos.

1.4. SISTEMA DE PLANIFICACIÓN DE OPERACIONES

El scheduling tiene lugar en una amplia gama de actividades. Se caracteriza por tener que definir de qué forma se lleva a cabo un conjunto de tareas que requieren la utilización óptima de recursos, generalmente compartidos y limitados en cantidad y/o disponibilidad, en un determinado período de tiempo, generalmente de corta longitud. Ejemplos típicos de este tipo de problemas son los de programación de la producción en las industrias de manufactura o de procesos batch, los de asignación de aviones a las distintas puertas de una terminal aérea, o de trenes a los andenes de una terminal ferroviaria, asignación y secuenciación de tareas de cómputos en una computadora compartida o a multiprocesador, secuencias de impresión en una impresora, asignación

de clases a las aulas en una Universidad, de asignación de cuadrillas a tareas de reparación en empresas eléctrica o telefónica, de programación de las actividades de una flota de transporte dedicada a tareas de distribución/recolección de productos, etc.

El problema de scheduling involucra la asignación temporal de órdenes de trabajo a un conjunto de recursos, tratando de alcanzar ciertas metas u objetivos, cumplimiento de fechas/horas pactados, maximizar el uso de recursos, y de satisfacer todas las restricciones que pudieran plantearse en relación a como y cuando se deben utilizar los recursos. Este problema es sumamente complejo y de naturaleza combinatoria. Su solución da lugar a la definición de un programa o agenda de trabajo, e implica la evaluación implícita de un número muy elevado de alternativas y la satisfacción de múltiples restricciones de diversas índoles.

Una característica común en muchos de esos problemas es que no se conocen algoritmos que aporten soluciones eficientes para la resolución de los mismos en forma óptima empleando tiempo polinomial.

Este trabajo de tesis está enfocado a proveer una herramienta aplicable en un sistema de planificación de operaciones (*scheduling*) para un sistema de manufactura/producción, categoría funcional de un TPS.

Capítulo II

SCHEDULING

2.1. INTRODUCCIÓN

El scheduling es una tarea extremadamente difícil con una importante necesidad de cálculo [21]. Los problemas de scheduling se pueden identificar en distintas áreas de aplicación. Diversos ítems están sujetos a scheduling, tal como operaciones de producción en una industria de manufactura, procesamiento computacional en un sistema operativo, movimiento de camiones en transporte, etc. La gran importancia práctica convierte al scheduling en un área activa de investigación.

Los problemas de scheduling son problemas de optimización combinatoria. La función del scheduling es la asignación de recursos limitados a tareas a lo largo del tiempo. Tiene como finalidad la optimización de uno o más objetivos.

Los recursos y las tareas pueden tomar muchas formas. Los recursos pueden ser máquinas en un taller, pistas en un aeropuerto, ladrillos en una construcción, unidades de procesamiento en un ambiente computacional, etc. Como tareas se pueden tener operaciones de un proceso de producción, despegues y aterrizajes en un aeropuerto, etapas de un proyecto de construcción, ejecuciones de un programa de computación, etc. Cada tarea puede tener diferentes niveles de prioridad, tiempos de posibles inicios, etc. Los objetivos pueden tomar varias formas: uno posible es minimizar los tiempos de finalización de la última tarea, otro minimizar el número de tareas luego de una fecha de entrega acordada, etc.

Los problemas de scheduling en la práctica poseen estructuras de problemas más complejas, pero en situaciones reales pueden ser relevantes diferentes restricciones, tal como planes de procesamiento alternativos para la fabricación de un producto, estructuras de producción especializadas, etc.

El scheduling puede ser un problema difícil desde el punto de vista técnico como de implementación [115]. El tipo de dificultades encontradas en los aspectos técnicos son similares a las encontradas en otras ramas de optimización combinatoria y modelado

estocástico. Las dificultades encontradas desde el punto de vista de la implementación son de distintas clases y están relacionadas al modelado de problemas de scheduling de mundo real y la recuperación de información.

Analizar un problema de scheduling y desarrollar un procedimiento para tratar con el problema es sólo una parte. El procedimiento tiene que estar embebido en un sistema que habilite la aplicación del scheduler. El sistema de scheduling se tiene que incorporar en el sistema de información de la empresa u organización, lo cual puede ser una tarea considerable.

2.2. NOTACIÓN

En todos los problemas de scheduling considerados, el número de máquinas es finito. El número de jobs se identifica con *n* y el número de máquinas con *m*. Usualmente, el subíndice *j* hace referencia a un job, mientras el subíndice *i*, a una máquina. Si un job necesita varios pasos de procesamiento u operaciones, entonces el par (*i*,*j*) indica la operación (paso de procesamiento) del job *j* sobre la máquina *i*. Con cada job *j* se asocian los siguientes datos:

- ✓ *Tiempo de procesamiento (p_{ij}).* Representa el tiempo de procesamiento del job *j* sobre la máquina *i*. El subíndice *i* se omite si el tiempo de procesamiento del job *j* no depende de la máquina o si el job *j* sólo se procesa en una determinada máquina.
- ✓ *Fecha de release (r_j).* La fecha de *release* r_j del job *j* es la fecha de listo. Es el tiempo en el que el job *j* arriba al sistema, es decir, es el tiempo más temprano en el cual se puede iniciar su procesamiento.
- ✓ *Fecha de finalización (d_j).* La fecha de finalización d_j del job *j* representa el *committed shipping* (fecha de terminación, embarque o entrega), la fecha en la cual se promete el job al cliente. Se permite la finalización de un job después de su fecha de terminación, pero se penaliza. Cuando se debe cumplir con la fecha de terminación esto se denomina como un *deadline*.
- ✓ *Peso (w_j).* El peso w_j del job *j* es básicamente un factor de prioridad, indicando la importancia relativa del job *j* con respecto a otros jobs del sistema. Por ejemplo, este peso puede representar el costo actual de mantener el job en el sistema.

2.2.1. DESCRIPCIÓN DE UN PROBLEMA DE SCHEDULING

Un problema de scheduling se describe por $\alpha|\beta|\gamma$. El campo α describe el ambiente de máquina y contiene una única entrada. El campo β provee detalles de las características de procesamiento y restricciones; puede tener una única entrada, múltiples entradas o ninguna entrada. El campo γ identifica el objetivo a ser minimizado y usualmente contiene una única entrada.

2.2.1.1. CAMPO α

Los posibles ambientes de máquina que se pueden especificar en el campo α son [95]:

- ✓ *Máquina única* (1). El caso de máquina única es el ambiente de máquina más simple y es el caso especial de todos los demás ambientes de máquina.
- ✓ *Máquinas idénticas en paralelo* (P_m). Hay *m* máquinas idénticas en paralelo. El job *j* necesita una única operación y se puede procesar en cualquiera de las *m* máquinas o sobre alguna de un conjunto especificado. Si no se permite el procesamiento del job *j* sobre cualquier máquina o sobre sólo alguna perteneciente a un subconjunto dado (el subconjunto M_j), entonces en el campo β aparece la entrada M_j.
- ✓ *Máquinas en paralelo con diferentes velocidades* (Q_m)**.** Hay *m* máquinas en paralelo con diferentes velocidades; v_i indica la velocidad de la máquina *i*. El tiempo de permanencia p_{ij} del job *j* en la máquina *i* es p_j/v_i, asumiendo que sólo se procesa en la máquina *i*. Este ambiente también se lo conoce como máquinas *uniformes*. Si todas las máquinas tienen la misma velocidad, es decir $v_i = 1$ para todo *i* y $p_{ij} = p_j$, entonces este ambiente es idéntico al previo.
- ✓ *Máquinas no relacionadas en paralelo* (R_m). Este ambiente es una generalización del anterior. Hay *m* máquinas diferentes en paralelo. La máquina *i* puede procesar el job *j* a una velocidad v_{ij}. El tiempo de permanencia p_{ij} del job *j* sobre la máquina *i* es p_j/v_{ij}, asumiendo que sólo se procesa sobre la máquina *i*. Si la velocidad de las máquinas son independientes de los jobs, es decir $v_i = v_{ij}$ para todo *i* y *j*, entonces el ambiente es idéntico al anterior.
- ✓ *Flow shop* (*Fm*). Hay *m* máquinas en serie, cada job se procesa en cada una de ellas. Todos los jobs tienen la misma trayectoria, es decir, primero se procesan sobre la máquina 1, luego sobre la máquina 2, y así sucesivamente. Cuando un job deja de

usar una máquina se agrega a la cola de la próxima máquina. Usualmente, se asume que todas las colas trabajan bajo la disciplina *primero en entrar primero en salir (FIFO)*. Si se aplica la disciplina FIFO, al flow shop se lo denomina como flow shop con permutación y el campo β incluye la entrada *prmu.*

- ✓ *Flexible flow shop* (*FFs*). Es una generalización del flow shop y del ambiente de máquinas paralelas. En lugar de *m* máquinas en serie, hay *s* etapas en serie con una determinada cantidad de máquinas en paralelo en cada una de ellas. Cada job se procesa primero en la etapa 1, luego en la etapa 2, y así siguiendo. Cada etapa funciona como un banco de máquinas paralelas; en cada etapa el job *j* necesita sólo una máquina y cualquier máquina puede procesar cualquier job. Las colas entre etapas distintas trabajan bajo una disciplina FIFO.
- ✓ *Open Shop* (*Om*) . Hay *m* máquinas y *n* jobs. Cada job se debe procesar en cada una de las *m* máquinas. Sin embargo, algunas de esos tiempos de procesamiento puede ser cero. No hay restricción en relación al ruteo de cada job a través del ambiente de máquinas. El scheduler determina la ruta de cada job, diferentes jobs pueden tener distintas rutas.
- ✓ *Job shop* (*Jm*). Hay *m* máquinas y *n* jobs, cada job tiene predeterminada su ruta. Se hace una distinción entre job shops donde cada job puede visitar alguna máquina al menos una vez y aquellos donde un job puede visitar una máquina más de una vez. En el último caso, el campo β contiene la entrada *recrc* para indicar *recirculación*.

2.2.1.2.CAMPO β

Las restricciones de procesamiento especificadas en el campo β pueden incluir múltiples entradas. Las entradas posibles son:

- ✓ *Release time* (r_j). Si este símbolo está presente en el campo β, el job *j* no puede empezar su procesamiento antes de su release date r_j. Si r_j no aparece en el campo β, el procesamiento del job *j* puede comenzar en cualquier momento. En contraste con el release dates, los tiempos de entrega no se especifican en este campo. El tipo de función objetivo da información suficiente si se consideran tiempos de entrega o no.
- ✓ *Tiempos de setup dependientes de la secuencia* (s_{jk}). s_{jk} representa el tiempo de setup dependiente de la secuencia de los jobs *j* y *k*; s_{0k} indica el tiempo de setup para el job *k* si es el primero en la secuencia y s_{j0} hace referencia al tiempo de

clean-up luego del job *j* si éste es el último de la secuencia (tanto s_{0k} como s_{j0} pueden ser 0). Si el tiempo de setup entre los jobs *j* y *k* depende de la máquina, entonces se debe incluir el subíndice *i*, es decir s_{ijk}. Si s_{0k} no aparece en el campo β, se asume que todos los tiempos de setup son cero o independientes de la secuencia, en cuyo caso se pueden simplemente incorporar a los tiempos de procesamiento.

- ✓ *Restricciones de precedencia* (*prec*)**.** Las restricciones de precedencia pueden aparecer en un ambiente de máquina única o máquinas paralelas, exigiendo que uno o más jobs tengan que finalizar antes que otros comiencen su procesamiento. Hay varias formas de restricciones de precedencia. Si cada job tiene al menos un antecesor y un sucesor, las restricciones se denominan *cadenas*. Si cada job tiene al menos un sucesor, las restricciones se denominan *intree*. Si cada job tiene al menos un antecesor, las restricciones se denominan *outtree*. Si en el campo β no aparece *prec*, los jobs no están sujetos a restricciones de precedencia.
- ✓ *Permutaciones* (*prmu*)**.** Una restricción que puede aparecer en el ambiente flow shop es que la cola de cada máquina trabaje respetando la disciplina FIFO. Esto implica que el orden (o permutación) en la cual se asignan los jobs a la primer máquina se debe respetar en todas las demás.
- ✓ *Recirculación* (*recrc*). Puede ocurrir en job shop, cuando un job puede visitar más de una vez una máquina.

2.2.1.3.CAMPO γ

El objetivo es minimizar una función de tiempos de finalización de los jobs, los cuales dependen del schedule. El tiempo de finalización de la operación de un job *j* sobre la máquina *i* se denota como C_{ij}. El tiempo en el que el job *j* sale del sistema se indica como C_j. La función objetivo puede ser una función de tiempos de entrega. La *lateness* de un job *j* se define como:

$$L_j = C_j - d_j,$$

la cual es positiva cuando el job *j* finaliza tarde y negativo cuando se completa en forma temprana. La *tardiness* de un job *j* se define como:

$$T_j = \max(C_j - d_j, 0) = \max(L_j, 0)$$

La diferencia entre tardiness y lateness es que la primera nunca es negativa. La *unidad de penalidad* del job *j* se define como:

$$U_j = \begin{cases} 1 & \text{si } C_j > d_j \\ 0 & \text{en otro caso} \end{cases}$$

Lateness, tardiness y unidad de penalidad son las tres funciones de penalidad básicas relacionadas con tiempos de entrega.

Los siguientes son algunos ejemplos de funciones objetivos a ser minimizadas:

- ✓ *Makespan* (C_{max}). Se define como el max(C_1, C_2, …, C_n). Es equivalente al tiempo de finalización del último job en dejar el sistema. Un makespan mínimo usualmente implica una alta utilización de las máquinas.
- ✓ *Lateness máxima* (L_{max})**.** Se define como el max(L_1, L_2, …, L_n). Mide la peor violación de los tiempos de entrega.

2.2.2. EJEMPLOS

Los siguientes ejemplos ilustran la notación:

- ✓ $Jm \mid\mid C_{max}$, denota un problema de job shop con *m* máquinas. No hay recirculación, de modo que cada job puede visitar sólo una una vez cada máquina. El objetivo es minimizar el makespan.
- ✓ $Pm \mid prec \mid L_{max}$, denota un ambiente de máquinas paralelas idénticas con restricciones de precedencia entre los jobs y optimización del lateness máximo.

2.3. CLASES DE SCHEDULES

Se puede hacer una distinción entre una *secuencia*, un *schedule*, y una *política de scheduling*. Una secuencia es usualmente una permutación de un conjunto de jobs o un orden en el cual se deben procesar los jobs sobre una determinada máquina. Un schedule es una asignación de jobs a un ambiente de máquinas más complicados. El concepto de políticas de scheduling se usa frecuente en ambientes estocásticos: una política prescribe una acción apropiada para alguno de los estados en el que el sistema puede estar. En modelos determinísticos es importante las secuencias o schedules.

Hay distintas clases de schedules [72, 66]: *nondelay, activo y semiactivo*.

Un schedule factible es *nondelay* si ninguna máquina permanece ociosa mientras exista una operación disponible para su procesamiento.

Exigiendo que un schedule sea nondelay es equivalente a prohibir *unforced idleness*.

Ciertos procedimientos y algoritmos heurísticos se basan sobre la generación de schedules con propiedades especiales. Los schedules activos y semiactivos son importantes para la generación del schedule en procedimientos algorítmicos.

Un schedule factible es *activo* si no se puede adelantar la finalización de una operación por alteración de la secuencia de procesamiento sin que se retrase otra operación.

Un schedule nondelay es un schedule activo pero no es cierto lo inverso. El siguiente ejemplo describe un schedule que es activo pero no nondelay.

Ejemplo. Sea un problema de job shop con tres máquinas y dos jobs. El job 1 necesita una unidad de tiempo sobre la máquina 1 y tres sobre la máquina 2. El job 2 necesita dos unidades sobre la máquina 3 y tres sobre la máquina 2. La máquina 2 es la última en procesar ambos jobs. Sea el schedule que procesa el job 2 sobre la máquina 2 antes que el job 1, (figura 2.1). Este schedule es activo; invirtiendo la secuencia de los dos procesos sobre la máquina 2 se pospone el procesamiento del job 2. Sin embargo, el schedule no es un schedule nondelay. La máquina 2 permanece ociosa hasta el tiempo 2, mientras que hay un job disponible para procesamiento en el tiempo 1.

Un schedule factible se llama *semiactivo* si ninguna operación puede finalizar tempranamente sin alterar la secuencia de procesamiento de alguna de las máquinas.

Ejemplo. Sea un problema de job shop con tres máquinas y dos jobs [115]. Las rutas de los dos jobs son las mismas que en el ejemplo previo. El tiempo de procesamiento del job 1 sobre la máquina 1 y la máquina 2 es el mismo. El tiempo de procesamiento del job 2 sobre la máquina 2 y la máquina 3 es de dos. Sea el schedule bajo el cual el job 2 se procesa sobre la máquina 2 antes que el job1 (figura 2.2). Esto implica que el job 2 comienza su procesamiento sobre la máquina 2 en el tiempo dos y el job 1 comienza su procesamiento sobre la máquina 2 en el tiempo cuatro. Este schedule es semiactivo. Sin embargo, es no activo: el job 1 se puede procesar sobre la máquina 2 sin retrasar el procesamiento del job 2 sobre la misma máquina.

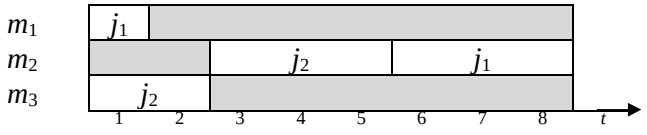


Figura 2.1 Schedule activo.

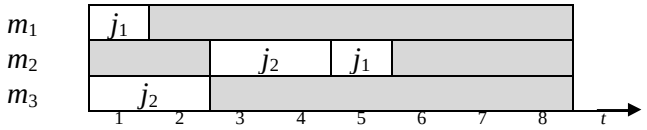


Figura 2.2 Schedule semiactivo.

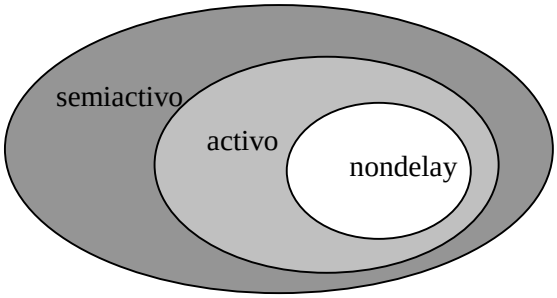


Figura 2.3 Relación entre las clases de schedules.

Se puede construir un ejemplo de schedule que no es semiactivo. Por ejemplo, posponer el comienzo del procesamiento del job 1 sobre la máquina 2 por una unidad de tiempo, es decir, la máquina 2 permanece ociosa por una unidad de tiempo entre el procesamiento del job 2 y del 1. Este schedule es no semiactivo.

La relación entre los distintos schedules se presenta en la figura 2.3. Un schedule óptimo está dentro del conjunto de schedules activos. Los schedules nondelay son más pequeños que los schedules activos, pero no hay garantía que el primero contenga el óptimo [18].

2.4. JERARQUÍA DE COMPLEJIDAD

Frecuentemente, un algoritmo para un problema de scheduling se puede aplicar a otros. Por ejemplo, $1||\sum C_j$ es un caso especial de $1||\sum w_j C_j$, y un procedimiento para $1||\sum w_j C_j$ se puede usar para $1||\sum C_j$ [115]. En terminología de complejidad se dice que $1||\sum C_j$ se reduce a $1||\sum w_j C_j$ y se indica como:

$$1||\sum C_j \propto 1||\sum w_j C_j$$

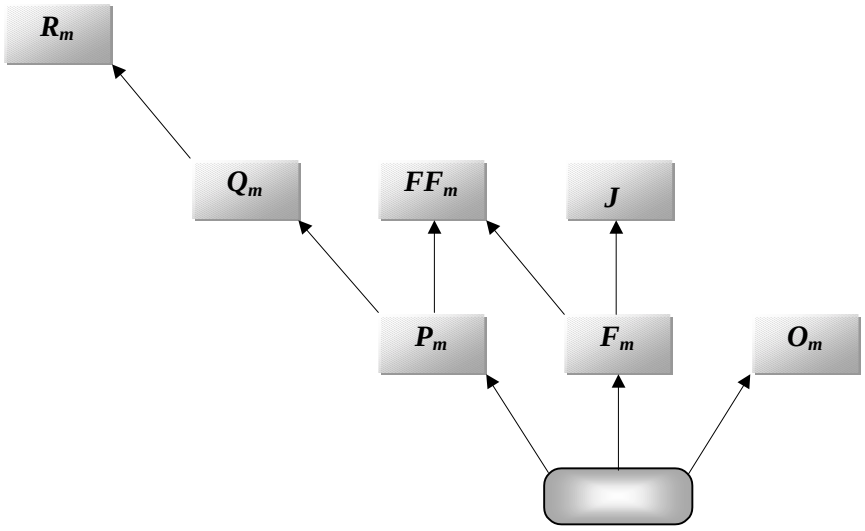
Basándose en este concepto, se puede establecer una cadena de reducciones. Por ejemplo,

$$1||\sum C_j \propto 1||\sum w_j C_j \propto P_m||\sum w_j C_j \propto Q_m|prec|\sum w_j C_j$$

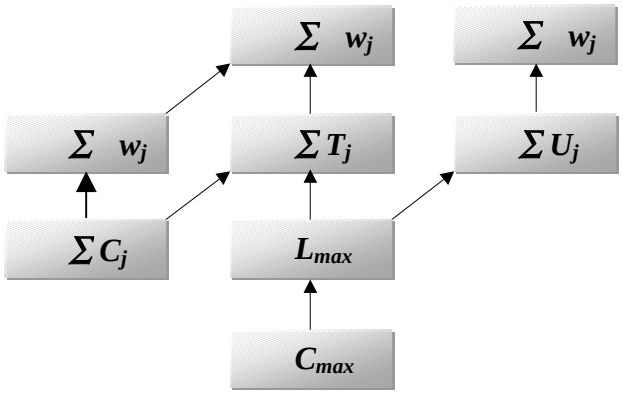
Hay varios problemas que no se pueden comparar entre sí. Por ejemplo, $P_m \parallel \sum w_j C_j$ y $J_m \parallel C_{max}$

Al comparar las complejidades de los diferentes problemas de scheduling, es de interés conocer cómo un cambio en un único elemento en la clasificación de un problema afecta su complejidad [117, 98, 93, 96]. En la figura 2.4, se muestra la jerarquía de complejidad de los problemas de scheduling determinístico [115].

Ha habido un gran desarrollado de investigación en scheduling determinístico para hallar un algoritmo eficaz que encuentre en tiempo polinomial una buena solución a los problemas de scheduling. Sin embargo, muchos problemas de scheduling no tienen algoritmos con tiempos polinomiales, por lo tanto esos problemas son problemas NP-duros.



(a) ambiente de máquina.



(b) funciones objetivo.

Figura 2.4 Jerarquía de complejidad de problemas de scheduling determinístico.

2.5. JOB SHOP SCHEDULING

Entre los ambientes de máquina presentados en la sección 2.1.1, este trabajo se enfoca en el problema de job shop scheduling clásico, el cual es uno de los problemas de scheduling más conocidos y el que con mayor frecuencia se encuentra dentro de un ambiente de producción. Se puede describir como sigue: hay *m* máquinas diferentes y *n* jobs diferentes a ser asignados. Cada job está compuesto por un conjunto de operaciones y el orden de las operaciones sobre las máquinas está preestablecido. Cada operación se caracteriza por la máquina requerida y el tiempo de procesamiento. Hay varias restricciones sobre jobs y máquinas:

- ✓ Un job no puede visitar una misma máquina dos veces.
- ✓ No hay restricciones de precedencia entre operaciones de distintos jobs.
- ✓ Las operaciones no se pueden interrumpir.
- ✓ Cada máquina puede procesar sólo un job a la vez.
- ✓ No se especifican ni *release times* (fecha de job listo a ser procesado)ni *due dates* (fecha de entrega).

El problema es determinar la secuencia de operaciones sobre las máquinas con el objetivo de minimizar el *makespan*, es decir, el tiempo necesario para completar todos los jobs.

El problema de job shop scheduling es uno de los problemas de optimización más duros. Debido a su dificultad para abordarlos, los procedimientos heurísticos son una alternativa muy atractiva. Los procedimientos heurísticos convencionales usan *reglas de prioridades*, es decir, una regla para elegir una operación desde un subconjunto especificado de operaciones aún no planificadas. Una de las técnicas más usadas para resolver este tipo de problema es la de búsqueda local, tal como simulated annealing, tabu search y algoritmos evolutivos. En el capítulo 7 se presentan distintas alternativas de algoritmos evolutivos para obtener soluciones para el problema de job shop scheduling.

Capítulo III

ALGORITMOS EVOLUTIVOS

3.1. INTRODUCCIÓN

Ha habido un interés creciente en algoritmos que se basan en el principio de evolución (supervivencia del mejor). Un término común, aceptado recientemente, para referirse a tales técnicas es *algoritmos evolutivos* (EA) o métodos de *computación evolutiva* (EC).

En general, cualquier tarea abstracta a ser realizada se puede pensar como la resolución de un problema, el cual, a su vez, se puede percibir como una búsqueda a través de un espacio de soluciones potenciales [24]. Como usualmente se busca la mejor solución, se puede pensar en esta tarea como un proceso de optimización. Para espacios pequeños, generalmente es suficiente usar los métodos exhaustivos clásicos; pero para espacios grandes se deben emplear técnicas especiales de inteligencia artificial. Los algoritmos evolutivos caen dentro de esas técnicas; son algoritmos estocásticos cuyos métodos de búsqueda modelan un fenómeno natural: la herencia genética y la rivalidad darwiniana para la supervivencia. Como se indica en [28]: "*la metáfora subyacente en algoritmos genéticos es la evolución natural. En evolución, el problema de cada especie es un problema de búsqueda de adaptaciones benéficas en ambientes cambiantes y complicados. El conocimiento que cada especie ha ganado se embebe en el cromosoma de sus miembros*".

Hay varias variantes de algoritmos evolutivos, las cuales incluyen:

- ✓ *programación evolutiva*: se hace evolucionar una población de máquinas de estados finitos sometiéndolas a transformaciones unitarias.
- ✓ *estrategias evolutivas*: se hace evolucionar una población de estructuras compuestas por un vector real y una variable aleatoria (parámetro) que codifican las posibles soluciones de un problema numérico y las dimensiones de los cambios o saltos. La selección es implícita.

- ✓ *programación genética*: se hace evolucionar una población de estructuras de datos sometiéndolas a una serie de transformaciones específicas y a un proceso de selección.
- ✓ *algoritmos genéticos*: se hace evolucionar una población de enteros binarios sometiéndolos a transformaciones unitarias y binarias genéticas y a un proceso de selección.

Hay también muchos sistemas híbridos los cuales incorporan varias características de los paradigmas antes mencionados y, consecuentemente, es difícil su clasificación; de todas maneras, se referirán a ellos como métodos de computación evolutiva. La estructura de cualquier algoritmo evolutivo es bastante similar; una estructura simple es la que se muestra en la figura 3.1 [24].

Los algoritmos evolutivos mantienen una población de individuos, $P(t) = \{x^t_1, x^t_2, \ldots, x^t_n\}$ en la iteración t, donde n es el tamaño de la población (*pop_size*). Cada individuo representa una solución potencial al problema, y se implementa como una estructura de datos *S*. Cada solución x^t_i se evalúa para dar alguna medida de su "fitness". Entonces, una nueva población (iteración $t + 1$) se forma al seleccionar los individuos más idóneos (paso seleccionar). Algunos miembros de la nueva población se someten a transformaciones (paso alterar) por medio de operadores

```
procedure: algoritmo evolutivo
  begin
    t ← 0;
    iniciar P(t);
    evaluar P(t);
    while no(condición de terminación) do
        t ← t + 1;
        seleccionar P(t) de P(t-1);
        alterar P(t);
        evaluar P(t);
    end while
  end procedure
```

Figura 3.1 Estructura de un algoritmo evolutivo.

"genéticos" para formar nuevas soluciones. Hay transformaciones unarias m_i (tipo mutación), las cuales crean nuevos individuos al producir pequeños cambios en un único individuo ($m_i : S \rightarrow S$), y transformaciones de orden superior c_j (tipo crossover), la cuales crean nuevos individuos al combinar varias partes de dos o más individuos ($m_i : S \times \ldots \times S \rightarrow S$). En la mayoría de los casos el crossover involucra sólo dos padres, sin embargo, no es necesariamente el caso. En [38] se ha investigado los méritos de "orgía", donde se involucran más de dos padres en el proceso de reproducción. Luego de algún número de generaciones, el algoritmo converge; es de esperar que el mejor individuo represente una solución cercana a la óptima.

A pesar de las similitudes entre los algoritmos evolutivos, hay también muchas diferencias entre ellos (a menudo ocultas a un bajo nivel de abstracción). Frecuentemente usan estructuras de datos *S* diferentes para la representación de los cromosomas, en consecuencia, los operadores genéticos también son diferentes. Pueden incorporar o no alguna otra información (para controlar el proceso de búsqueda) en sus genes. Hay también otras diferencias; por ejemplo, las dos líneas de la figura 1:

seleccionar *P(t)* de *P(t-1)*;

alterar *P(t)*;

pueden aparecer en orden inverso: en estrategias evolutivas primero se altera la población y luego se forma la nueva población por el proceso de selección. Hay muchos métodos para seleccionar individuos para supervivencia y reproducción. Esos métodos incluyen:

- ✓ *selección proporcional*: la probabilidad de selección es proporcional al fitness del individuo,
- ✓ *métodos de ranking*: todos los individuos de la población se ordenan de mejor a peor y las probabilidades de selección son fijas durante todo el proceso de evolución, y
- ✓ *selección por torneo*: algunos individuos (usualmente dos) compiten por ser seleccionados a la nueva generación; este paso de competencia (torneo) se repite *pop_size* veces.

La selección proporcional puede necesitar el uso de métodos de truncamiento o de escalamiento, hay diferentes formas para asignar probabilidades en métodos de ranking (distribuciones lineales o no lineales), el tamaño de un torneo juega un rol significativo en el método de selección por torneo.

Es importante determinar las políticas generacionales, es decir especificar la forma en la cual quedará conformada la próxima generación. Por ejemplo, es posible reemplazar la población entera por la población de hijos, o seleccionar los mejores individuos de las dos poblaciones (padres e hijos), esta selección se puede hacer de una manera determinística o no determinística. Es también posible producir pocos hijos (en particular, uno único), con los cuales reemplazar algunos individuos (sistemas basados en tales políticas generacionales son llamados “steady state”). En un modelo generacional el conjunto de los padres se mantiene fijo mientras se generan todos los hijos que formarán parte de la próxima generación. También, se puede usar un modelo elitista, el cual pasa el mejor individuo de una generación a otra, esto significa que si el mejor individuo de la generación actual corre el riesgo de perderse, ya sea por la selección o los operadores genéticos, el sistema fuerza a que quede en la próxima generación; tal modelo es muy útil para resolver muchas clases de problemas de optimización.

Las representaciones (estructuras de datos necesarias implementar un individuo en el espacio fenotípico) de las soluciones potenciales para un problema particular, junto con el conjunto de operadores genéticos, constituyen los componentes esenciales de cualquier algoritmo evolutivo. Esos son los elementos claves que permiten distinguir varios paradigmas de métodos evolutivos.

Los algoritmos evolutivos han recibido mucha atención tanto desde el punto de vista académico como industrial. Las herramientas basadas en EA tienen un impacto creciente en compañías (prediciendo el mercado financiero), en fábricas (scheduling) debido a su poder de búsqueda, optimización, adaptación y aprendizaje. Una de las áreas recientes de aplicación para esas técnicas es el campo de la ingeniería industrial, donde se incluye scheduling y ordenamiento en sistemas de manufactura, diseño asistido por computadora, facilidad de layout y problemas de ubicación, y muchos más. Se recurren a los algoritmos genéticos por su simplicidad, y facilidad de extensión.

Aunque los algoritmos evolutivos han sido exitosamente aplicados a muchos problemas prácticos y el número de aplicaciones se incrementa, también se presentan varias desventajas. Hay poco conocimiento respecto de que características del dominio los hacen apropiados o inapropiados. Se han sugerido varias modificaciones para mitigar las dificultades tanto en el manejo de la información codificada como en las

formas de representar el espacio del problema. Tres importantes afirmaciones se han hecho respecto de porque los algoritmos evolutivos trabajan bien:

- ✓ Las poblaciones de soluciones candidatas proveen muestreos independientes.
- ✓ La selección es un mecanismo que preserva buenas soluciones.
- ✓ Las soluciones parciales se pueden modificar eficientemente y combinar a través de varios operadores genéticos.

3.2. ALGORITMOS GENÉTICOS

La forma usual de un algoritmo genético (GA) fue descripta por Goldberg [75]. Los algoritmos genéticos (GAs) son técnicas de búsqueda estocásticas basadas en los mecanismos de selección y genética natural. Los GAs, a diferencia de las técnicas de búsqueda convencionales, comienzan con un conjunto inicial de soluciones, generado en forma aleatoria, llamado *población*. Cada individuo en la población se llama *cromosoma* y representa una solución al problema. Un cromosoma es un string de símbolos y generalmente, pero no necesariamente, un string de bits binario de longitud fijo. Los cromosomas evolucionan a través de sucesivas iteraciones, llamadas *generaciones*. Durante cada generación, los cromosomas se *evalúan*, usando alguna medida de *fitness* (aptitud) [62]. Para crear la próxima generación, se forman nuevos cromosomas, llamados *hijos*, ya sea:

- ✓ mezclando dos cromosomas de la generación actual usando un operador de crossover, ó,
- ✓ modificando un cromosoma por medio de un operador de mutación.

Una nueva generación se forma al seleccionar, acorde a los valores de fitness, alguno de los padres e hijos, y al rechazar algunos, a fin de mantener constante el tamaño de la población.

Los cromosomas mejor adaptados tienen alta probabilidad de ser seleccionados. Luego de varias generaciones, el algoritmo converge al mejor cromosoma, el cual se espera que represente una solución óptima o subóptima.

Los algoritmos genéticos tienen varias diferencias con los procedimientos convencionales de búsqueda y optimización. Goldberg [75] las ha resumido como sigue:

- ✓ Los algoritmos genéticos trabajan con un conjunto de soluciones codificadas, no con la solución en sí misma.
- ✓ Los algoritmos genéticos buscan en una población de soluciones, no con una única solución.
- ✓ Los algoritmos genéticos usan información de retribución (función de fitness), no necesitan conocimiento derivado o auxiliar.
- ✓ Los algoritmos genéticos usan reglas de transición probabilísticas, no reglas determinísticas.

Los GAs han recibido considerable atención como técnica de optimización en función de sus potenciales. Hay tres ventajas principales cuando se aplican GA para la optimización de problemas:

- ✓ Los GAs no tienen muchos requerimientos matemáticos de los problemas de optimización. Dada la naturaleza evolutiva, los GAs buscarán soluciones sin considerar el funcionamiento interno del problema. Un GA puede manejar cualquier clase de funciones objetivo y cualquier clase de restricciones (lineales o no lineales) definidas sobre espacios de búsqueda discretos, continuos o una mezcla de ellos.
- ✓ La naturaleza de los operadores de evolución hacen a los GAs muy efectivos para realizar búsquedas globales. Las opciones tradicionales realizan una búsqueda local con un procedimiento de convergencia, el cual compara los valores de puntos cercanos y se mueve a puntos relativamente óptimos.
- ✓ Los GAs proveen una gran flexibilidad para realizar híbridos con heurísticas dependientes del dominio, a fin de lograr una implementación efectiva para problemas específicos.

3.3. ESTRATEGIAS EVOLUTIVAS

Las estrategias evolutivas (ES) se desarrollaron como un método para resolver problemas de optimización paramétrica [127]; consecuentemente, un cromosoma representa un individuo como un par de vectores de valor flotante.

Las primeras estrategias evolutivas se basaban sobre una población de un único individuo. Se usaba un único operador genético en el proceso de evolución: la mutación. Sin embargo, la idea interesante fue representar un individuo como un par de vectores de valor flotante, es decir $v = (\mathbf{x}, \boldsymbol{\sigma})$. Aquí, el primer vector **x** representa un punto en el

espacio; el segundo vector σ es un vector de desviaciones estándares. La mutación se realizaba al reemplazar **x** por:

$$\mathbf{x}^{t+1} = \mathbf{x}^{t} + N(0, \boldsymbol{\sigma})$$

donde $N(0, \boldsymbol{\sigma})$ es un vector de números random gausianos independientes con una media de cero y una desviación estándar $\boldsymbol{\sigma}$ (esto está en armonía con la observación biológica que los cambios pequeños ocurren más frecuentemente que los grandes). El hijo (el individuo mutado) se acepta como un nuevo miembro de la población (reemplaza a su padre) si y solo si tiene fitness mejor y satisface todas las restricciones (si es que las hay). Por ejemplo, si f es la función objetivo sin restricciones a maximizar, un hijo $(\mathbf{x}^{t+1}, \boldsymbol{\sigma})$ reemplaza al padre $(\mathbf{x}^{t}, \boldsymbol{\sigma})$ si y solo si $f(\mathbf{x}^{t+1}) > f(\mathbf{x}^{t})$. En caso contrario, se elimina al hijo y la población permanece sin cambio.

El vector de desviaciones estándares σ permanece sin cambio durante el proceso de evolución. Si todos los componentes de este vector son idénticos, es decir, $\boldsymbol{\sigma} = (\sigma,\ldots,\sigma)$, y el problema de optimización es regular, es posible demostrar el teorema de convergencia [7]:

Teorema de Convergencia: para $\sigma > 0$ y un problema de optimización regular con $f_{opt} > -\infty$ (minimización) o $f_{opt} < \infty$ (maximización), se tiene

$$p\{\lim_{t\to\infty} f(x^{t}) = f_{opt}\} = 1$$

Las estrategias evolutivas evolucionaron más tarde [127] para madurar como:

$$(\mu+\lambda)\text{-ES y } (\mu,\lambda)\text{-ES}$$

la principal idea detrás de estas estrategias fue permitir el control de parámetros (como la varianza de mutación) para auto adaptación más que para cambiar sus valores por algún algoritmo determinístico.

En (μ+λ)-ES, μ individuos producen λ hijos. La nueva población de (μ+λ) individuos se reduce, por un proceso de selección, nuevamente a μ individuos. Por otra parte, en (μ,λ)-ES, los μ individuos producen λ hijos (λ>μ) y el proceso de selección elige una nueva población de μ individuos desde el conjunto de λ hijos. Al hacer esto, la vida de cada individuo se limita a una generación. Esto permite que (μ,λ)-ES trabaje mejor sobre problemas con un óptimo moviéndose sobre el tiempo, o en problemas donde la función objetivo presenta mucho ruido.

La selección (μ+λ), con la supervivencia garantizada de los mejores individuos, parece ser más efectiva, porque un curso monótono de evolución se realiza de esta

forma. Sin embargo, este mecanismo de selección tiene varias desventajas cuando se compara con la selección (μ,λ), la cual restringe el tiempo de vida de los individuos a una generación:

- ✓ En el caso de medio ambientes cambiantes la selección $(\mu+\lambda)$ preserva la solución y no es capaz de seguir un óptimo en movimiento.
- ✓ La capacidad de la selección (μ,λ) para olvidar buenas soluciones (en principio permite dejar óptimos locales pequeños) es ventajosa en el caso de topologías multimodales.
- ✓ La selección $(\mu+\lambda)$ obstaculiza el mecanismo de auto adaptación para que trabaje adecuadamente con respecto a los parámetros estratégicos, porque los parámetros estratégicos mal adaptados pueden sobrevivir por un gran número de generaciones cuando producen mejoras en el fitness.

Por consiguiente, se recomienda la selección (μ,λ). Resumiendo las condiciones para una exitosa auto adaptación de parámetros de estrategia, se obtiene la siguiente lista [6]:

- ✓ Se necesita la estrategia (μ,λ) a fin de facilitar la extinción de individuos mal adaptados.
- ✓ La presión selectiva puede no volverse muy fuerte.
- ✓ Es necesaria la recombinación de los parámetros estratégicos (generalmente, la recombinación intermedia da buenos resultados).

El operador usado en $(\mu+\lambda)$-ES y (μ,λ)-ES incorpora dos niveles de aprendizaje: su parámetro de control $\boldsymbol{\sigma}$ no es más constante, no se cambia por algún algoritmo determinístico (como la regla de éxito 1/5), pero se incorpora en la estructura del individuo y se somete al proceso de evolución. Para producir un hijo, el sistema actúa en varias etapas:

- ✓ Seleccionar dos individuos:

  $(\mathbf{x}^1, \boldsymbol{\sigma}^1) = (x^1_1, \ldots, x^1_n), (\sigma^1_1,\ldots, \sigma^1_n)$ y

  $(\mathbf{x}^2, \boldsymbol{\sigma}^2) = (x^2_2, \ldots, x^2_n), (\sigma^2_2,\ldots, \sigma^2_n)$.
- ✓ Aplicar un operador de recombinación (crossover). En ES, la recombinación se usa para la creación de todos los hijos, cuando $\mu > 1$. Tanto las variables objeto como los parámetros estratégicos están sujetos a recombinación y el operador de recombinación puede ser diferente para variables objeto y desviaciones estándar. Esto implica que la recombinación de esos grupos de información se realiza en

forma independiente unos de otros, es decir, no se restringe que los parámetros estratégicos se originen desde los mismos padres que los valores objeto. Los operadores de recombinación tradicional de ES se denominan :

✓ *discretos,* donde el nuevo hijo es

$(\mathbf{x}, \boldsymbol{\sigma}) = (x^{q_1}_1, \ldots, x^{q_n}_n), (\sigma^{q_1}_1, \ldots, \sigma^{q_n}_n)$ y

donde q_i=1 o q_i=2 (así cada componente proviene del primer o del segundo padre preseleccionados). Por cada componente del vector se decide en forma aleatoria desde cuál de los dos padres se copia el componente al hijo.

✓ *intermedios,* donde el nuevo hijo es

$(\mathbf{x}, \boldsymbol{\sigma}) = ((x^1_1 + x^2_2)/2, \ldots, (x^1_n + x^2_n)/2), ((\sigma^1_1 + \sigma^2_2)/2, \ldots, (\sigma^1_n + \sigma^2_n)/2)$

indica que los componentes de los hijos se obtienen al calcular la media aritmética de los componentes correspondientes de ambos padres.

Cada uno de estos operadores se pueden aplicar también en un modo global, donde el nuevo par de padres se selecciona para *cada* componente del vector del hijo.

✓ Aplicar mutación al hijo $(\mathbf{x}, \boldsymbol{\sigma})$ obtenido; el nuevo hijo resultante es $(\mathbf{x}', \boldsymbol{\sigma}')$, donde

$\boldsymbol{\sigma}' = \boldsymbol{\sigma} \cdot e^{N(0, \Delta\sigma)}$, y

$\mathbf{x}' = \mathbf{x} + N(0, \boldsymbol{\sigma}')$,

donde $\Delta\boldsymbol{\sigma}$ es un parámetro del método.

3.4. PROGRAMACIÓN EVOLUTIVA

La técnica original de programación evolutiva (EP) se desarrolló por Fogel [64]. Se proponía la evolución de inteligencia artificial en el sentido de desarrollar la habilidad para predecir cambios en un medio ambiente. El medio ambiente se describió como una secuencia de símbolos (desde un alfabeto finito) y el algoritmo suponía la producción por evolución, como una salida, de un nuevo símbolo. El símbolo de salida maximiza la función de retribución (payoff), la cual mide la exactitud de la predicción.

Por ejemplo, se puede considerar una serie de eventos, marcada por símbolos $a_1, a_2, \ldots, a_n$; un algoritmo debería predecir el próximo símbolo (no conocido), es decir a_{n+1} sobre las bases de los símbolos ya conocidos $a_1, a_2, \ldots, a_n$. La idea de programación evolutiva fue evolucionar tal algoritmo.

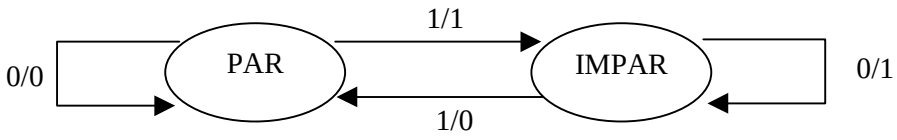


Figura 3.2 Una FSM para un chequeo de paridad.

Se seleccionaron máquinas de estado finito (FSM) para la representación del cromosoma de los individuos; luego, todas las máquinas de estado finito proveen una representación significativa del comportamiento basado sobre interpretación de símbolos. La figura 3.2 [24] presenta un ejemplo de un diagrama de transición de una máquina de estado finito simple para un chequeo de paridad. Tal diagrama de transición es un grafo dirigido que contiene un nodo en cada etapa y ejes que indican la transición desde un estado a otro, valores de entrada y salida (notación: *a*/*b* cercano a un arco que une el estado S_1 con el estado S_2, sugiere que el valor de entrada *a*, mientras la máquina está en el estado S_1, resulta en un valor de salida *b* y en el próximo estado S_2).

Hay dos estados PAR e IMPAR (la máquina comienza en el estado PAR); la máquina reconoce la paridad de un string binario.

La técnica de programación evolutiva mantiene una población de máquinas de estado finito (FMS); cada individuo representa una solución potencial al problema (representa un comportamiento particular). Cada FSM se evalúa para dar alguna medida de su fitness. Esto se hace de la siguiente forma: cada FSM está expuesta al ambiente en el sentido de que examina todos los símbolos vistos previamente. Por cada subsecuencia, $a_1, a_2, \ldots, a_i$ produce una salida a'_{i+1}, la cual se compara con el próximo símbolo observado, a_{i+1}. Por ejemplo, si se han visto *n* símbolos, un FSM hace *n* predicciones (una por cada uno de los substrings a_1, a_1, a_2 y así sucesivamente hasta $a_1, a_2, \ldots, a_n$); la función de fitness toma en cuenta la performance total.

Como en estrategias evolutivas, la técnica de programación evolutiva primero crea hijos y luego selecciona individuos para la próxima generación. Cada padre produce un único hijo; así se duplica el tamaño de la población intermedia (como en (*pop_size*, *pop_size*)-ES). Los hijos (una nueva FSM) se crean por mutaciones aleatorias de la población de padres. Hay cinco posibles operadores de mutación: cambio de un símbolo de salida, cambio de una transición de estado, adición de un estado, eliminación de un estado, y cambio de un estado inicial (hay algunas restricciones

iniciales sobre el número máximo o mínimo de estados). Esas mutaciones se eligen con respecto a alguna distribución de probabilidades (la cual puede cambiar durante el proceso de evolución); también es posible aplicar más de una mutación a un único padre (la decisión sobre el número de mutaciones de un individuo particular se hace con respecto a alguna otra distribución de probabilidades).

Los mejores *pop_size* individuos se retienen para la próxima generación, es decir, para calificar para la próxima generación, un individuo deberá estar por el encima del 50% de la población intermedia. En la versión original [64], este proceso se iteraba varias veces antes de que estuviera disponible la próxima salida de símbolos. Una vez que se encuentra disponible un nuevo símbolo, se adicionaba a la lista de símbolos conocidos, y se repite el proceso total.

Desde luego, el proceso detallado previamente se puede extender de muchas maneras; como se describe en [58].

Las técnicas de programación evolutiva fueron generalizadas para tratar con problemas de optimización numéricas [55, 58]. En [64, 56, 60, 61, 128, 100] se encuentran ejemplos de técnicas de programación evolutiva.

3.5. PROGRAMACIÓN GENÉTICA

Fue desarrollada por Koza [88, 89]. Koza sugiere que el programa deseado deberá evolucionar durante el proceso de evolución. En otras palabras, en lugar de resolver un problema, y de construir un programa evolutivo para resolver el problema, se deberá buscar en el espacio de posibles programas de computación el que mejor se adecue. Koza desarrolló una nueva metodología, llamada programación genética (GP), la cual provee una forma para realizar tal búsqueda.

Hay cinco pasos para usar programación genética en un problema particular. Ellos son:

- ✓ Selección de terminales.
- ✓ Selección de una función.
- ✓ Identificación de la función de evaluación.
- ✓ Selección de los parámetros del sistema.
- ✓ Selección de la condición de terminación.

La estructura que pasa por el proceso de evolución es la estructura jerárquica del programa de computación. El espacio de búsqueda es un hiper-espacio de programas válidos, los cuáles se pueden ver como un espacio de árboles. Cada árbol está compuesto de funciones y terminales apropiadas para el dominio de un problema particular; el conjunto de todas las funciones y terminales se selecciona a priori de tal forma que algunos de los árboles formados lleve a una solución.

Por ejemplo, dos estructuras e_1 y e_2 (figura 3.3) representan expresiones $2x + 2.11$ y $x \cdot \text{sen}(3.2)$, respectivamente. Un posible hijo e_3 (luego del crossover de e_1 y e_2) representa $x \cdot \text{sen}(2x)$ [24].

La población inicial está compuesta de tales árboles; la construcción de un árbol en forma aleatoria es sencilla. La función de evaluación asigna un valor de fitness el cual evalúa la performance de un árbol (programa). La evaluación se basa sobre un conjunto preseleccionado de casos de testeo; en general, la función de evaluación retorna la suma de distancias entre el resultado correcto y el obtenido sobre todos los casos de prueba.

La selección es proporcional; cada árbol tiene una probabilidad de ser seleccionado para la próxima generación proporcionalmente a su fitness. La operación primaria es el crossover que produce dos hijos desde dos padres seleccionados. El crossover crea hijos al intercambiar subárboles entre dos padres. Hay otros operadores como mutación, permutación, edición, etc. Por ejemplo, una mutación típica selecciona un nodo en un árbol y genera un nuevo subárbol el cual se origina en el nodo seleccionado.

Hay cinco pasos para construir un programa genético para un problema particular. Koza [91] recientemente considera la ventaja de agregar una característica adicional: un conjunto de procedimientos. Esos procedimientos se llaman funciones definidas

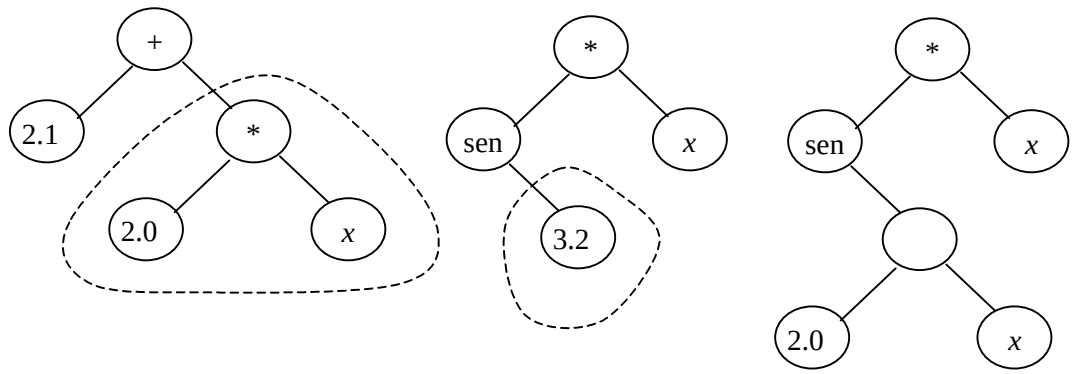


Figura 3.3 Expresión e_3: un hijo de e_1 y e_2. La línea con trazo cortado incluye área intercambiadas durante la operación de crossover.

automáticamente (ADF). Este parece ser un concepto extremadamente útil para técnicas de programación genética, con su mejor contribución al área de reusabilidad de código. ADF descubre y explota las regularidades, simetrías, similitudes, patrones, y modularidades del problema y el programa genético puede invocar a esos procedimientos en diferentes momentos de su ejecución.

El hecho de que los programas genéticos trabajen sobre programas de computación tiene aspectos muy interesantes. Por ejemplo, los operadores se pueden ver como programas, los cuales se pueden someter a una evolución separada durante la ejecución del sistema. Adicionalmente, un conjunto de funciones pueden consistir de varios programas, los cuales realizan tareas complejas; tales funciones pueden evolucionar durante la ejecución evolutiva. En general, un GP tiende a hacer un trabajo bastante bueno para hallar una solución general siempre que:

- ✓ Se le brinden casos de entrenamientos para ser capaz de deducir la solución general.
- ✓ Que la función y el conjunto de terminales hayan sido elegidos a fin de influir apropiadamente en el espacio de soluciones para un problema dado.

Es una de las áreas de desarrollo actual en el campo de la computación evolutiva [89, 91, 87, 3].

3.6. OTRAS TÉCNICAS

Los algoritmos evolutivos han sido modificados al agregar en el algoritmo conocimiento específico del problema. Varios trabajos han discutido técnicas de inicio, diferentes representaciones, técnicas de decodificación (traslación desde representaciones genéticas a representaciones fenotípicas), y el uso de heurísticas para operadores genéticos.

Tales sistemas no estándares o híbridos tienen significativa popularidad en la comunidad de computación evolutiva. Frecuentemente esos sistemas, extendidos por el conocimiento específico del problema, mejoran tanto a otros métodos evolutivos clásicos como a otras técnicas estándares [101, 102]. Por ejemplo, un sistema Genético-2N [101] construido para el problema de transporte no lineal usa una representación matricial para sus cromosomas, una mutación específica del problema (principal operador, con probabilidad 0.4) y crossover aritmético (operador de background, usado con probabilidad 0.05). Es difícil clasificar tal sistema: no es

realmente un algoritmo genético, ya que puede ejecutar sólo con el operador de mutación sin un significativo decremento de la calidad de los resultados. Sin embargo, todas las entradas en la matriz son valores de punto flotantes. No es una estrategia evolutiva, ya que no codifica ningún parámetro de control en la estructura del cromosoma. Claramente, no cae dentro de la programación genética ni de la programación evolutiva. Es una técnica de computación evolutiva apuntada a la resolución de un problema particular.

Hay pocas heurísticas para guiar al usuario en la selección de estructuras de datos y operadores apropiados para un problema particular. Es de conocimiento general que para un problema de optimización numérico se deberá usar una estrategia evolutiva o un algoritmo genético con representación de punto flotante, mientras algunas versiones de algoritmos genéticos deberán ser las mejores para controlar problemas de optimización combinatoria. Los programas genéticos son buenos para descubrir reglas dadas como un programa de computación, y las técnicas de programación evolutivas se pueden usar exitosamente para modelar el comportamiento de un sistema.

3.7. DIFERENCIAS ENTRE LAS DISTINTAS TÉCNICAS DE EA

Aunque todos los algoritmos comparten un meta-modelo, cada uno enfatiza diferentes características como las más importantes para el modelado exitoso del proceso evolutivo. Dejando de lado las cuestiones de representación, las diferencias más notorias se presentan en la interpretación del rol de los operadores genéticos. En programación evolutiva, no se usa recombinación, mientras que en algoritmos genéticos juega un rol dominante, abandonando la influencia de la mutación casi completamente. Las estrategias evolutivas usan ambos operadores, indicando un necesario uso de recombinación sobre los parámetros estratégicos para lograr una auto adaptación exitosa.

Tanto la programación evolutiva como los algoritmos genéticos insisten en la selección probabilística, aunque con diferencias en la implementación; el primer caso se refiere a la característica de selección en la naturaleza y en el segundo caso a la analogía con el multiarmed bandit (bandido de múltiples armas) [6]. En contraste, las estrategias evolutivas usan un mecanismo de selección estrictamente determinístico, siendo

extintivo en el sentido de excluir definitivamente de la selección a los peores individuos.

Mientras en programación evolutiva algunos individuos son excluidos de la selección, los algoritmos genéticos hacen uso de *preservación* en el sentido de que cada individuo recibe una probabilidad de selección distinta de cero. La propiedad elitista es implícita en la selección en programación evolutiva, mientras esto ocurre explícitamente en estrategias evolutivas ($(\mu+\lambda)$-ES) y algoritmos genéticos.

Capítulo IV

ALGORITMOS GENÉTICOS

4.1. INTRODUCCIÓN

La estructura de un algoritmo genético simple se corresponde con la estructura de cualquier programa evolutivo (ver figura 1.1). Durante la iteración *t*, un algoritmo genético mantiene una población de soluciones potenciales (cromosomas, vectores), $P(t) = \{x^t_1, ..., x^t_n\}$. Cada solución x^t_n se evalúa para dar alguna medida de su fitness. Entonces, se forma una nueva población (iteración $t + 1$) al seleccionar los individuos más adaptados. Algunos miembros de esta nueva población se someten a alteraciones por medio de crossover y mutación, para formar nuevas soluciones. El crossover combina las características de dos cromosomas padres para formar dos hijos similares al intercambiar segmentos entre los padres. La idea detrás de la aplicación del operador de crossover es el intercambio de información entre diferentes soluciones potenciales. La mutación altera arbitrariamente uno o más genes del cromosoma seleccionado, estos cambios aleatorios se realizan con probabilidad usualmente baja. La idea detrás del operador de mutación es introducir alguna variabilidad en la población.

Para resolver un problema particular, un algoritmo genético (como cualquier programa evolutivo) deberá tener los siguientes componentes:

- ✓ Una representación genética de las soluciones potenciales del problema.
- ✓ Una forma de crear la población inicial de soluciones potenciales.
- ✓ Una función de evaluación que juegue el papel de medio ambiente, y permita ordenar las soluciones de acuerdo a su fitness.
- ✓ Operadores genéticos que alteren la composición de los hijos.
- ✓ Valores de varios parámetros que el algoritmo genético utiliza.

Grefenstette y Baker [81] presentan una versión modificada de la descripción de un algoritmo genético. Esta descripción difiere del paradigma presentado por Holland, donde la selección se hace para obtener padres para recombinación [84].

procedure: Algoritmo Genético
begin
$t \leftarrow 0$;
iniciar *P(t);*
evaluar *P(t);*
while no(condición de terminación) **do**
recombinar *P(t)* para producir *C(t)*;
evaluar *C(t)*;
seleccionar *P(t+1)* de *P(t)* y *C(t)*;
$t \leftarrow t + 1$;
end while
end procedure

Figura 4.1 Estructura general de un algoritmo genético.

Sea $P(t)$ y $C(t)$ la población de padres e hijos en la generación actual t, la estructura general de un algoritmo genético se describe en la figura 4.1.

La iniciación generalmente se realiza en forma aleatoria. La recombinación típicamente involucra crossover y mutación para generar los hijos. De hecho, sólo hay dos clases de operaciones en algoritmos genéticos:

- ✓ *Operaciones genéticas*: crossover y mutación.
- ✓ *Operación de evolución*: selección.

Las operaciones genéticas imitan el proceso de herencia de genes para crear nuevos hijos en cada generación. La operación de evolución imita el proceso de evolución de Darwin para crear poblaciones de generación en generación.

4.1.1. VOCABULARIO GA

Como los GAs tienen sus raíces tanto en la genética natural como en la ciencia de la computación, la terminología usada en la literatura de algoritmos genéticos es una mezcla de la natural y de la artificial.

En un organismo biológico, la estructura que codifica la prescripción especificando como está construido el organismo se llama *cromosoma*. Se pueden necesitar uno o más

Algoritmos Genéticos	Explicación
Cromosoma (string, individuo)	Solución
Genes (bits)	Parte de una solución
Locus	Posición del gen
Alelos	Valores del gen
Fenotipo	Solución decodificada
Genotipo	Solución codificada

Tabla 4.1 Explicación de los términos de algoritmos genéticos

cromosomas para especificar el organismo completo. El conjunto de cromosomas se llama *genotipo*, y el organismo resultante, *fenotipo*. Cada cromosoma comprende un conjunto de estructuras individuales llamadas *genes*. Cada gen codifica una característica particular de un organismo, y la ubicación del gen, *locus*, dentro de la estructura del cromosoma determina la característica particular que representa el gen. En un locus particular, un gen puede codificar cualquiera de los distintos valores de la característica que representa. Los diferentes valores de un gen se denominan *alelos*.

La tabla 4.1 [69] muestra la correspondencia entre los términos de algoritmos genéticos y los de optimización.

4.1.2. REPRESENTACIÓN

Hay un conjunto de mecanismos de representación (también llamada codificación) que se usan para resolver distintos problemas de optimización en GAs, entre ellas se encuentran la codificación binaria, real, permutaciones, etc.

4.1.2.1. CODIFICACIÓN BINARIA

La codificación binaria es una elección excelente para problemas en los cuales un individuo se mapea naturalmente en un string de ceros y unos. Un string binario se usa como un cromosoma para representar valores reales de la variable x. El largo del vector depende de la precisión requerida. Por ejemplo E = [10001010] es un string binario con ocho genes. El locus del i-ésimo gen es simplemente el i-ésima posición en el string y su valor o alelo lo da $E[i]$.

Para muchos problemas una codificación binaria no es apropiada debido a [24]:

- ✓ *Epístasis*: significa una fuerte interacción entre genes en un cromosoma. En otras palabras, la epistasis mide el grado en el cual la contribución en el fitness de un gen depende de los valores de otros genes.
- ✓ *Representación natural*: el problema a ser resuelto requiere un conjunto de símbolos de orden mayor.
- ✓ *Soluciones ilegales*: los operadores genéticos pueden producir soluciones ilegales con una codificación binaria, ya que una codificación binaria puede no describir naturalmente un punto del espacio de búsqueda.

4.1.2.2. CODIFICACIÓN REAL

En problemas de optimización de funciones reales se da una función *n*-dimensional, por ejemplo $f(x, y, z)$. El objetivo de optimización es típicamente el requerimiento para ubicar el valor máximo (o mínimo) de la función en un dominio dado. En GAs clásicos, usados en problemas de optimización de funciones reales, una solución potencial al problema se codifica en un string de bits. En un ejemplo tridimensional $f(x, y, z)$, una solución potencial podrá ser $f(x_1, y_1, z_1)$ y x_1, y_1, z_1 se codificarán como substrings tridimensionales. Esos substrings se concatenarán para formar un único genotipo de strings de bits. Cada parámetro se puede codificar como un único string binario o como un número en punto flotante tradicional.

El principal objetivo detrás de esta implementación es que el algoritmo genético esté más cerca del espacio del problema. Esto fuerza, pero también permite, que los operadores sean más específicos del problema. Por ejemplo, esta representación tiene la propiedad que dos puntos cercanos en el espacio de representación pueden también ser cercanos en el espacio del problema, y viceversa. Esto no es generalmente verdad en la opción binaria, donde la distancia en una representación es normalmente definida por el número de posiciones de bits diferentes. Sin embargo, es posible reducir tal discrepancia usando *codificación Gray*.

4.1.2.3. CODIFICACIÓN CON PERMUTACIONES

Los problemas de permutaciones necesitan el arreglo óptimo de un conjunto de símbolos en una lista. El TSP es uno de esos problemas donde se puede usar un símbolo para identificar una ciudad, y la disposición de los símbolos en una lista representa el orden en el cual el vendedor visita cada ciudad para formar un circuito con todas las ciudades. Una codificación con permutaciones se puede representar por medio de una lista de valores enteros distintos, por ejemplo x = [4, 3, 0, 1, 2] [82, 80, 144, 119, 118, 120]. Cada valor entero en la lista codifica directamente el orden relativo de algún objeto específico del problema. Esta representación prohibe valores de alelos duplicados o perdidos; permite el uso de operadores genéticos de alta performance (tal como el *edge recombination operator* [144]); y facilita un mecanismo de decodificación simple desde el genotipo al fenotipo para el TPS.

4.1.2.4. OTRAS REPRESENTACIONES

Se han usado otras estrategias de codificación en algoritmos evolutivos. Entre ellas se puede incluir representación con árboles [90], matrices [22, 15, 140], y codificaciones con estructuras heterogéneas [71].

4.1.3. CROSSOVER

El crossover es el principal operador genético. Es un operador sexual. Trabaja sobre dos cromosomas a la vez y genera hijos al recombinar ambas características de los cromosomas. Una forma simple de realizar el crossover deberá consistir en elegir en forma aleatoria un punto de corte, y generar el hijo combinando el segmento de un padre a la izquierda del punto de corte con el segmento a la derecha del otro padre. Este método trabaja con la representación de strings de bits y se ha extendido a otras representaciones. La performance del algoritmo genético depende, en gran parte, del comportamiento del operador de crossover usado.

La *probabilidad de crossover* (indicada por p_c) se define como la relación entre el número de hijos producidos en cada generación y el tamaño de la población (generalmente indicado por *pop_size*). Esta probabilidad controla el número esperado de cromosomas que se someten a la operación de crossover. Una alta probabilidad de crossover permite una mayor exploración del espacio de soluciones, reduciendo las

chances de establecerse en un óptimo falso; pero si la probabilidad es muy alta, provoca un gran desperdicio en cuanto a cantidad de tiempo de computación en la exploración de regiones no prometedoras del espacio de soluciones. Los valores propuestos para la probabilidad de crossover son $p_c = 0.6$ [29], $p_c = 0.95$ [78], y $p_c \in [0.75,0.95]$ [124]. El crossover tradicional de un punto introducido por Holland [84] elige un posición de crossover $i \in \{1, ..., l\text{-}1\}$ dentro del string de bits (de longitud l) en forma aleatoria e intercambia los bits a la derecha de esa posición entre ambos individuos. Si los padres se representan por los siguientes vectores $(s_1, s_2,\ldots, s_n)$ y $(v_1, v_2,\ldots, v_n)$, entonces al cruzar los cromosomas luego del punto i se deberán producir los siguientes hijos:

$$(s_1, s_2,\ldots, s_{i-1}, s_i, v_{i+1}, \ldots, v_n) \text{ y } (v_1, v_2,\ldots, v_{i-1}, v_i, s_{i+1}, \ldots, s_n)$$

Una generalización a crossover *multi punto* [43] permite más de un punto de crossovers. En la figura 4.2 se esquematiza el efecto del crossover de 5 puntos.

Al incrementar el número de puntos de crossover, se obtiene el operador de *crossover uniforme* [134]. En el caso de crossover uniforme, el intercambio de segmentos se reduce a bits; por cada bit se decide en forma aleatoria si se realiza el intercambio o no. Comparando este operador con el crossover de un punto, Syswerda reporta mejores resultados con crossover uniforme sobre un conjunto de funciones de prueba.

4.1.4. MUTACIÓN

La mutación es un operador de background en los GAs el cual produce cambios aleatorios en varios cromosomas. Una forma simple de realizar la mutación deberá ser alterar uno o más genes. En un algoritmo genético, la mutación cumple el rol de reposición de genes perdidos en la población durante el proceso de selección y de provisión de aquellos genes que no están presentes en la población inicial.

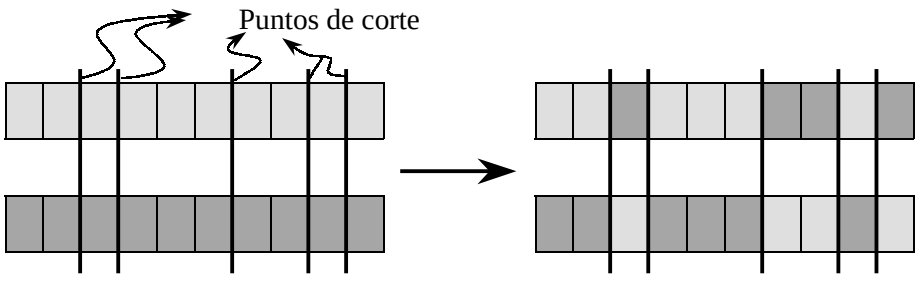


Figura 4.2 Aplicación del crossover de 5 puntos.

La *probabilidad de mutación* (indicada por p_m) se define como el número total de genes en la población que deben ser mutados. La probabilidad de mutación controla el porcentaje en el cual se introducen nuevos genes en la población. Si es muy baja, muchos genes que podrían haber sido producidos nunca se prueban. Si es muy alta, habrá mucha perturbación aleatoria, los hijos comenzarán a perder su parecido a los padres. El algoritmo perderá la habilidad de aprender de la historia de la búsqueda. Valores comunes para la probabilidad de crossover son $p_m = 0.001$ [29], $p_m = 0.01$ [78] y $p_m \in [0.005,0.01]$ [124].

Mientras la mutación en un algoritmo genético sirve como un operador para reintroducir alelos perdidos en la población, es decir posiciones de bits que convergen a un cierto valor en la población completa y por consiguiente no podrían ser recuperados nuevamente por medio de la recombinación, el operador de crossover es el operador de búsqueda más importante de un GA. La idea de crossover es que los segmentos significativos de diferentes padres se combinen para producir un nuevo individuo que se beneficie con combinaciones de bits ventajosas de ambos padres. De esta forma, se espera el surgimiento de segmentos grandes de alto fitness, finalmente trayendo a una solución general buena [84].

4.1.5. EXPLORACIÓN Y EXPLOTACIÓN

La búsqueda es una de los métodos más universales de resolución de problemas, donde no se puede determinar a priori la secuencia de pasos que lleva a la solución. La búsqueda se puede realizar con *estrategias ciegas* o con *estrategias heurísticas* [16]. Las estrategias de búsqueda ciegas no usan información respecto del dominio del problema. Las estrategias de búsqueda heurísticas usan información adicional para guiar la búsqueda. Hay dos cuestiones importantes en estrategias de búsqueda: explotar la mejor solución y explorar el espacio de búsqueda [18]. Michalewicz da una comparación de la búsqueda hill-climbing, búsqueda aleatoria y búsqueda genética [102]. Hill-climbing es un ejemplo de una estrategia que explota la mejor solución por una posible mejora, mientras ignora la exploración del espacio de búsqueda. La búsqueda aleatoria es un ejemplo de una estrategia que explora el espacio de búsqueda mientras ignora la explotación de las regiones prometedoras del espacio de búsqueda. Los algoritmos genéticos son una clase de métodos de búsqueda de propósito general, el

cual combina elementos de búsqueda estocástica y dirigida, las cuales pueden hacer un balance entre exploración y explotación del espacio de búsqueda. Al comienzo de la búsqueda genética existe una población diversa y aleatoria, el operador de crossover tiende a realizar una búsqueda general al explorar todo el espacio de soluciones. Mientras se desarrollan soluciones con fitness alto, el operador de crossover provee exploración en el vecindario de cada una de ellas. En otras palabras, la clase de búsqueda (exploración o explotación) que un operador de crossover realice deberá estar determinada por el ambiente del sistema genético (la diversidad de la población), pero no por el operador en sí mismo. Además, los operadores genéticos simples se diseñan como métodos de búsqueda de propósito general (métodos de búsqueda independientes del dominio), esencialmente realizan una búsqueda a ciegas y podrían no garantizar la producción de mejores hijos.

4.1.6. BÚSQUEDA BASADA EN LA POBLACIÓN

Generalmente, el algoritmo para resolver problemas de optimización es una secuencia de pasos los cuales convergen asintóticamente a la solución óptima. La mayoría de los métodos de optimización clásicos, generan una secuencia determinística de operaciones basadas sobre el gradiente o derivadas de ordenes superiores de la función objetivo. Los métodos se aplican a un único punto del espacio de búsqueda. El punto se mejora gradualmente con direcciones ascendentes y descendentes a través de las iteraciones. Este método punto a punto tiene la desventaja de caer en óptimos locales. Los GAs realizan una búsqueda en múltiples direcciones al mantener una población de soluciones potenciales. La opción población a población procura que la búsqueda escape de óptimos locales. La población se somete a una evolución simulada: en cada generación las soluciones relativamente buenas se reproducen, mientras las soluciones relativamente malas mueren. Los algoritmos genéticos usan reglas de transición probabilísticas para seleccionar quien se reproducirá y quien morirá.

4.1.7. META-HEURÍSTICAS

Al principio los GAs se crearon como una herramienta genérica para la resolución de muchos problemas difíciles. En la mayoría de los primeros trabajos en GAs se usó la

representación interna universal consistente de strings binarios de longitud fija, con operadores genéticos binarios para trabajar de una manera independiente del dominio sin tener conocimiento de la interpretación fenotípica del string. Esta universalidad se reflejó en un fuerte énfasis sobre el diseño de sistemas robustos adaptables para una gran gama de aplicaciones. Sin embargo, los GAs simples son difíciles de aplicar directa y exitosamente en muchos problemas de optimización de resolución complicada. Se han creado varias implementaciones no estándares para problemas particulares en las cuales los GAs se usan como *meta-heurísticas*.

4.2. CODIFICACIÓN DEL PROBLEMA

Cómo codificar en un cromosoma una solución del problema es una cuestión clave para un algoritmo genético. En el trabajo de Holland, la codificación se realiza por medio de strings binarios. Para muchas aplicaciones de GAs, especialmente para problemas de ingeniería industrial, el GA simple tiene una aplicación difícil en forma directa porque el string binario no es una codificación natural. Se han desarrollado muchas técnicas de codificación no string para problemas particulares, por ejemplo *codificación de números reales* para problemas de optimización con restricciones y *codificación con enteros* para problemas de optimización combinatoria. La elección de una representación apropiada de las soluciones candidatas al problema a manejar es fundamental para aplicar GAs, a fin de resolver problemas del mundo real. La representación condiciona todos los pasos subsecuentes del algoritmo genético. En muchos casos de aplicación, es necesario realizar un análisis cuidadoso para asegurar una representación apropiada de soluciones junto a operadores genéticos específicos y útiles al problema.

Una de las características básicas de los algoritmos genéticos es que trabajan sobre un *espacio codificado* y un *espacio de soluciones* en forma alternativa: los operadores genéticos trabajan sobre el espacio codificado, genotipos (cromosomas), mientras la evaluación y la selección lo hacen sobre el espacio de soluciones (fenotipos) (figura 4.3 [69]). La selección natural es la conexión entre cromosomas y la performance de sus soluciones decodificadas. Para la opción no string, surgen tres puntos vinculados con el codificado y decodificado entre cromosomas y soluciones (o la traslación entre fenotipo y genotipo):

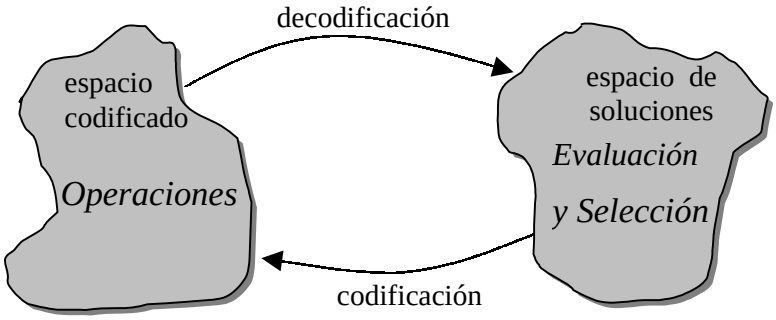


Figura 4.3 Espacio codificado y espacio de soluciones.

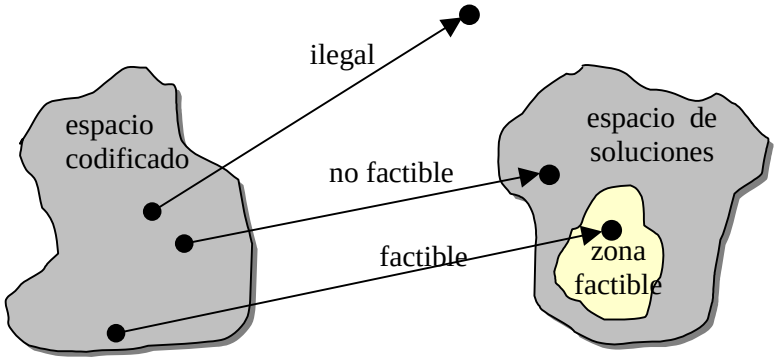


Figura 4.4 Factibilidad y legalidad.

- ✓ La factibilidad de un cromosoma.
- ✓ La legalidad de un cromosoma.
- ✓ La unicidad de la traslación.

La *factibilidad* hace referencia a si una solución decodificada cae en una región factible del problema dado. La *legalidad* hace mención al fenómeno de si un cromosoma representa una solución al problema dado (figura 4.4 [69]).

La no factibilidad de un cromosoma se origina por la naturaleza de los problemas de optimización con restricciones. Todos los métodos, los convencionales o los algoritmos genéticos, deberán manejar restricciones. Para muchos problemas de optimización, la región factible se puede representar por medio de un sistema de igualdades o desigualdades (lineales o no lineales). Para tales casos, se han propuesto muchos métodos de penalidades eficientes para tratar con cromosomas no factibles [67, 103, 130]. En problemas de optimización con restricciones, el óptimo típicamente se produce

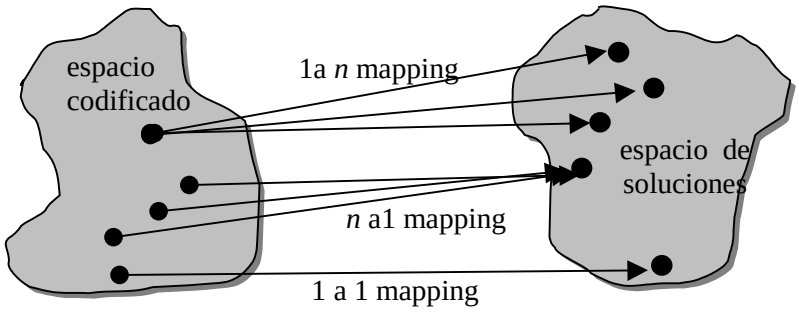


Figura 4.5 La transformación de cromosomas a soluciones.

en el límite entre áreas factibles y no factibles. Las opciones de penalidad forzarán la búsqueda genética para aproximarse al óptimo desde las regiones factibles y no factibles.

La ilegalidad de los cromosomas se origina por la naturaleza de las técnicas de codificación. En muchos problemas de optimización combinatoria, se usan codificaciones específicas del problema y generalmente producen un hijo ilegal por la simple operación de crossover de un punto. Como un cromosoma no se puede decodificar a una solución, esto significa que tampoco se puede evaluar; por lo tanto la opción con penalidades no es aplicable en esta situación. Usualmente, se adoptan técnicas reparadoras para convertir un cromosoma ilegal en uno legal. Por ejemplo, el operador PMX es esencialmente una clase de crossover de dos puntos para representaciones con permutaciones, junto con un procedimiento de reparación para resolver la ilegitimidad causada por el crossover de dos puntos. Orvosh y Davis [112] han mostrado que, para muchos problemas de optimización combinatoria, es relativamente fácil reparar un cromosoma ilegal o no factible; la estrategia de reparación supera a otras estrategias como las de rechazo o las de penalización.

La transformación de un cromosoma en una solución (decodificación) puede pertenecer a alguno de los siguientes tres casos (figura 4.5 [69]):

1. Maping 1 a 1.
2. Maping *n* a 1.
3. Maping 1 a *n*.

El mapping 1 a 1 es el mejor de los tres casos y el mapping 1 a *n* es el menos deseado.

4.3. CONVERGENCIA PREMATURA

La teoría de GAs provee algunas explicaciones de porqué, para una formulación del problema, se puede obtener una convergencia a un punto óptimo buscado. Desgraciadamente, las aplicaciones prácticas no siempre siguen la teoría, las razones principales son:

- ✓ La codificación del problema frecuentemente mueve al GA a trabajar en un espacio diferente al del problema en sí mismo.
- ✓ Hay un límite sobre el número de iteraciones hipotéticamente no limitadas.
- ✓ Hay un límite sobre el tamaño de la población hipotéticamente no limitada.

Una de las implicaciones de esas observaciones es la inhabilidad de los algoritmos genéticos, bajo ciertas condiciones, para hallar las soluciones óptimas; tal característica es la causa para una convergencia prematura a un óptimo local. La convergencia prematura es un problema común de los algoritmos genéticos y de otros algoritmos de optimización. Si la convergencia ocurre muy rápidamente, entonces frecuentemente se pierde el desarrollo de información valiosa en parte de la población. Las implementaciones de algoritmos genéticos son propensas a la convergencia prematura antes de hallar la solución óptima.

Algunos investigadores relacionan la convergencia prematura con dos puntos muy relacionadas:

- ✓ La magnitud y clase de errores introducidos por los mecanismos de muestreos.
- ✓ Las características de la función.

Hay dos cuestiones muy importantes en el proceso de evolución de la búsqueda genética: la *diversidad poblacional* y la *presión selectiva*. Esos factores están fuertemente relacionados: un incremento en la presión selectiva decrementa la diversidad de la población, y viceversa. En otras palabras, una fuerte presión selectiva origina la convergencia prematura de la búsqueda; una débil presión selectiva puede hacer la búsqueda inefectiva [102]. Es importante un balance entre esos dos factores; los mecanismos de muestreos intentan realizar este objetivo. En [144] se observa que esos factores son primarios en la búsqueda genética y en algún sentido, es otra variación de la idea de *exploración* versus *explotación* discutida por Holland y otros. Muchos de los parámetros que se usan para ajustar la búsqueda son realmente medios indirectos para afectar la presión selectiva y la diversidad en la población. Cuando se incrementa la

presión selectiva, la búsqueda se enfoca en los individuos más adaptados de la población, pero por la explotación se pierde diversidad genética. Reduciendo la presión selectiva (usando grandes poblaciones) se incrementa la exploración porque hay más genotipos involucrados en la búsqueda.

4.4. SELECCIÓN

El principio detrás del algoritmo genético es esencialmente la selección natural de Darwin. La selección provee la fuerza motora a un algoritmo genético. La presión selectiva es crítica. En un extremo, la búsqueda terminará prematuramente; mientras que en el otro, el progreso será más lento que el necesario. Típicamente, se sugiere una baja presión selectiva al comienzo de los algoritmos genéticos en favor de una amplia exploración del espacio de búsqueda, mientras se recomienda una alta presión selectiva al final, para explotar las regiones más prometedoras del espacio de búsqueda. La selección dirige la búsqueda del GA a través de regiones prometedoras del espacio de búsqueda.

Se presentan tres cuestiones básicas relacionadas con la fase de selección [68]:

- ✓ *Espacio de muestreo.*
- ✓ *Mecanismo de muestreo*
- ✓ *Probabilidad de selección.*

Cada una de ellas ejerce una influencia significativa sobre la *presión selectiva* y por consiguiente en el comportamiento del algoritmo genético.

4.4.1. ESPACIO DE MUESTREO

El procedimiento de selección puede crear una nueva población para la próxima generación basándose en todos los padres e hijos o parte de ellos. Esto da lugar al problema de espacio de muestreo. Un espacio de muestreo se caracteriza por dos factores: *tamaño* e *integrantes* (padres o hijos). Sea *pop_size* que indica el tamaño de la población y *off_size*, la cantidad de hijos producidos en cada generación. El espacio de muestreo regular tiene el tamaño de *pop_size* y contiene todos los hijos pero sólo una parte de los padres. El espacio de muestreo extendido tiene el tamaño de *pop_size* + *off_size* y contiene el conjunto total de padres e hijos.

4.4.1.1.ESPACIO DE MUESTREO REGULAR

En el algoritmo genético original de Holland, los padres se reemplazan por sus hijos tan pronto estos últimos nacen. Esto se denomina *reemplazo generacional*. Como las operaciones genéticas son ciegas por naturaleza, los hijos pueden ser peores que sus padres. Con la estrategia de reemplazar cada padre por su hijo directamente, algunos cromosomas con fitness alto se pueden perder en el proceso de evolución. Para solucionar este problema, se propusieron varias *estrategias de reemplazo*. Holland sugiere que cada vez que nace un hijo, éste reemplace un cromosoma de la población elegido en forma aleatoria [84]. De Jong propone una estrategia de *crowding* [29]. En el modelo de crowding, cuando nace un hijo, se selecciona un padre para morir, y es aquel que más se parezca al nuevo hijo. En el trabajo de Holland, la selección hace referencia a la elección de padres para recombinación; se forma una nueva población al reemplazar los padres con sus hijos. Esto recibe el nombre de plan reproductivo. Desde el trabajo de Grefenstette y Baker [81], la selección se usa para formar la próxima generación, usualmente con un mecanismo probabilístico. Michalewicz [102] da una descripción detallada de algoritmos genéticos simples donde los hijos reemplazan a sus padres en cada generación; la próxima generación se forma utilizando *selección por ruleta* (*roulette wheel selection*) [102].

4.4.1.2.ESPACIO DE MUESTREO EXTENDIDO

Cuando la selección trabaja sobre un espacio de muestreo extendido, tanto los padres como los hijos tienen las mismas chances de competir por la supervivencia. El caso típico es la selección $(\mu + \lambda)$ [57]. Esta estrategia fue inicialmente usada en estrategias evolutivas [126, 127]. Bäck y Hoffmeister introdujeron este tipo de selección en su algoritmo genético [5, 9]. Con esta estrategia, μ padres y λ hijos compiten por la supervivencia y se seleccionan los μ mejores individuos entre los hijos y los padres como padres de la próxima generación. Otro caso desde las estrategias evolutivas es la selección (μ,λ), la cual selecciona los μ mejores hijos como padres de la próxima generación $(\mu < \lambda)$. Ambos métodos son completamente determinísticos y se pueden convertir en métodos probabilísticos. Aunque la mayoría de los métodos de selección se basan en el espacio de muestreo regular, es fácil implementarlos sobre espacios de

muestreos extendidos. Una ventaja que presenta esta opción es que se puede mejorar la performance del GA al incrementar las probabilidades de crossover y mutación.

4.4.2. MECANISMOS DE MUESTREO

Los mecanismos de muestreos están relacionados con el problema de cómo seleccionar cromosomas del espacio de muestreo. Se han usado tres opciones básicas para sacar muestras de cromosomas [69]:

- ✓ *Muestreos estocásticos.*
- ✓ *Muestreos determinísticos.*
- ✓ *Muestreos mixtos.*

4.4.2.1. MUESTREOS ESTOCÁSTICOS

Una característica común de los métodos encasillados en esta categoría es que la fase de selección determina el número de copias que cada cromosoma recibirá basándose en su probabilidad de supervivencia [11]. De este modo la fase de selección está compuesta de dos partes:

- ✓ Determinar el valor esperado de hijos del cromosoma.
- ✓ Convertir el valor esperado en un número de hijos.

Un valor esperado de hijos del cromosoma es un número real que indica el número promedio de hijos que un cromosoma deberá recibir. El procedimiento de muestreo se usa para convertir el valor esperado real en número de hijos.

El método más conocido en esta clasificación es la *selección proporcional* de Holland o *roulette wheel selection*. La idea es determinar la *probabilidad de selección* (también llamada probabilidad *de supervivencia*). Para el cromosoma *k* con fitness f_k, su probabilidad de selección p_k se calcula de la siguiente manera:

$$p_k = \frac{f_k}{\sum_{j=1}^{pop\_size} f_j}$$

Entonces se puede construir una ruleta en función a esas probabilidades. El proceso de selección está basado en hacer girar la rueda de ruleta *pop_size* veces; cada vez, se selecciona un único cromosoma para la nueva población.

Este mecanismo falla en el caso de fitness negativo o tareas de minimización, si *f* representa los valores de la función objetivo sin realizar un escalamiento apropiado. Por otra parte, cualquier técnica de escalamiento manipula las probabilidades de selección y por consiguiente el mecanismo de selección, de tal forma que los fundamentos teóricos de la selección proporcional se vuelven cuestionables en estos casos.

Baker propone *stochastic universal sampling* [11], el cual usa una única pasada por la rueda (*wheel spin*). La rueda se construye como una ruleta y se gira con un conjunto de marcadores separados a igual distancia, la cantidad de marcadores es igual al tamaño de la población. El valor esperado e_k para el cromosoma k se calcula como $e_k = pop\_size \times p_k$. El procedimiento de la figura 4.6 describe el stochastic universal sampling.

La consideración básica de esta opción es tomar el número esperado de copias de cada cromosoma en la próxima generación. Un punto interesante es la *prohibición de cromosomas duplicados* en la población. Hay dos razones para usar esta estrategia:

- ✓ Prevenir que *super cromosomas* dominen la población al mantener muchas copias en la población. Esto es una causa rápida de convergencia a un óptimo local.

```
procedure: stochastic universal sampling
begin
   sum ← 0;
   ptr ← rand(); // rand() retorna un número real random distribuido
                    uniformemente dentro del rango [0,1).
   for k ← 1 to pop_size do
        sum ← sum + e_k;
        while (sum > ptr) do
             seleccionar cromosoma k;
             ptr ← ptr + 1;
        end while
   end for
end procedure
```

Figura 4.6 Procedimiento del Stochastic Universal Sampling

Un problema relacionado es que cuando se descartan los cromosomas duplicados, el tamaño de la población formada con los padres y los hijos puede ser menor que el tamaño predefinido *pop_size*. En este caso, se usa el procedimiento de iniciación para llenar el espacio vacante del pool de población.

4.4.2.2.MUESTREOS DETERMINÍSTICOS

Esta opción generalmente selecciona los mejores *pop_size* cromosomas desde el espacio de muestreo. Tanto la selección $(\mu + \lambda)$ como la (μ, λ) pertenecen este método. Ambas opciones prohiben que cromosomas duplicados entren en la población durante la selección.

Truncation selection y *block selection* pertenecen a este método, los cuales ordenan todos los cromosomas de acuerdo a su fitness y selecciona el mejor como padre [137]. En truncation selection se define un umbral *T* tal que se seleccionan los *T%* mejores cromosomas y cada uno recibe alrededor de 100/*T* copias. Block selection es equivalente a truncation selection ya que para una población dada de tamaño *pop_size*, se dan simplemente *s* copias a los *pop_size/s* mejores cromosomas.

La s*elección elitista* asegura que el mejor cromosoma se mantiene a través de las generaciones si no se selecciona a través del proceso de selección. Bajo ciertas condiciones muy generales, la introducción del elitismo garantiza la convergencia teórica al óptimo global; en la práctica, mejora la velocidad de convergencia de los GAs cuando la función de evaluación es unimodal, es decir, no existen subóptimos, sin embargo la velocidad de convergencia empeora con funciones fuertemente multimodales.

El muestreo determinístico de Brindle está basado sobre el concepto de *número esperado* [20]. La probabilidad de selección para cada cromosoma se calcula de la forma usual, $p_k = f_k / \sum f_k$. El número esperado de cada cromosoma se calcula como $e_k = p_k \times pop\_size$. A cada cromosoma se le asignan muestras en relación a la parte entera del número esperado, y entonces la población se ordena en función a la parte fraccional del número esperado. Los cromosomas restantes necesarios para completar se sacan de la parte superior de la lista ordenada.

El *reemplazo generacional* (reemplazo del conjunto entero de padres por sus hijos) se puede ver como otra versión de una opción determinística. Una modificación al

método es reemplazar los *n* cromosomas más viejos y peores con hijos (*n* es el número de hijos) [143, 102]. Los cromosomas a reemplazar por los hijos se seleccionan en función de su probabilidad de supervivencia. Los cromosomas peores en performance que la media tienen las chances más altas de ser seleccionados para morir.

4.4.2.3. MUESTREOS MIXTOS.

Esta opción tiene tanto características determinísticas como aleatorias. Un ejemplo típico es *selección por torneo* presentado por Goldberg [76]. Este método elige en forma aleatoria un conjunto de cromosomas y escoge el mejor del conjunto de reproducción. El número de cromosomas en el conjunto se llama *tamaño del torneo*. Un tamaño de torneo común es 2. Esto se denomina *torneo binario*.

Stochastic tournament selection fue sugerido por Wetzel [142]. En este método, las probabilidades de selección se calculan en la forma usual y se muestrean pares consecutivos de cromosomas usando selección por ruleta. Luego de sortear un par, esos cromosomas con alto fitness se insertan en la nueva población. El proceso continúa hasta que se completa la población.

Remainder stochastic sampling propuesto por Brindle es una versión modificada del muestreo determinístico [20]. En este método, a cada cromosoma se le asignan muestras acorde a la parte entera del número esperado, y entonces los cromosomas compiten en función de la parte fraccional del número esperado para los lugares restantes en la población.

4.4.3. PROBABILIDAD DE SELECCIÓN

En este punto lo más importante es determinar la probabilidad de selección de cada cromosoma. En el procedimiento de selección proporcional, la probabilidad de selección de un cromosoma es proporcional a su fitness. Este esquema simple exhibe algunas propiedades no deseables. Por ejemplo, en generaciones tempranas, hay una tendencia a que super individuos dominen el proceso de selección; en las generaciones finales, cuando la población ha convergido, la competencia entre cromosomas es menos fuerte y se producirá un comportamiento aleatorio de búsqueda donde la selección no concede mayor preeminencia a uno u otro individuo. En una población infinita los

procesos de muestreo son siempre imperfectos, pudiendo favorecer más de lo que le corresponde a individuos ocasionalmente más aptos; debido a que la población es finita y como consecuencia de esos errores estocásticos en los muestreos, un individuo puede acabar expulsando de la población a los restantes, haciendo que el GA converja prematuramente hacia tal individuo afortunado.

Para mitigar esos problemas se proponen los *mecanismos de escalamiento* y *ranking*. Los métodos de escalamiento transforman los valores crudos de la función objetivo a algún valor real positivo, y la probabilidad de supervivencia de cada cromosoma se determina en función de ese nuevo valor. Los métodos de ranking ignoran los valores de la función objetivo actual y en su lugar usan el ranking del cromosoma para determinar la probabilidad de supervivencia. El escalamiento del fitness tiene las siguientes intenciones:

1. Mantener una diferencia razonable entre los niveles de fitness relativo de los cromosomas.
2. Prevenir un copamiento muy rápido de algunos super cromosomas para reunir los requerimientos de limitar la competencia temprana y estimular la tardía.

Para la mayoría de los métodos de escalamiento, los parámetros de escalamiento son dependientes del problema. El ranking de fitness tiene un efecto similar al escalamiento de fitness, pero evita la necesidad de parámetros de escalamiento extras [116].

En general, el fitness escalado f'_k desde el fitness crudo (valor de la función objetivo) f'_k para el cromosoma k se puede expresar como [81]:

$$f'_k = g(f'_k)$$

donde la función $g(\cdot)$ transforma el fitness crudo en un fitness escalado. La función $g(\cdot)$ puede tomar diferentes formas para producir diferentes métodos de escalamiento, tal como escalamiento lineal, *sigma trunction* [75], *power law scaling* [73], *logarithmic scaling* [81], etc.

Baker introduce la noción de *ranking selection* para algoritmos genéticos [4, 76, 83, 54]. La idea es muy simple: ordenar la población del mejor al peor y asignar la probabilidad de selección de cada cromosoma acorde al ranking pero no ya a su fitness crudo. Hay dos métodos de uso común el *ranking lineal* y el *exponencial*.

4.4.4. PRESIÓN SELECTIVA

Para controlar la diversidad genética se debe actuar sobre el mecanismo de asignación de probabilidades de supervivencia (o de descendientes, en su caso): cuando más se favorezca a los más aptos menor variedad se obtendrá y, como consecuencia, se aumentará el riesgo de perder la información de valor desarrollada por los individuos moderadamente aptos, entendiendo por información de valor toda la que podría servir posteriormente para hallar el óptimo global. Por el contrario, si no se favorece especialmente a los individuos más aptos, se obtendrá más diversidad en la población, pero a costo de hacer las etapas de selección mucho menos eficientes, lo que empeora la eficiencia global del GA.

A la mayor o menor tendencia del mecanismo de asignación de probabilidades de supervivencia por favorecer a los individuos más aptos se le llama *presión selectiva*. Lo ideal sería que en las primeras fases de la evolución de un GA hubiera poca presión selectiva con el fin de que la búsqueda fuera lo más global posible y en las últimas fases de la evolución, una vez que ya sido localizada la mejor solución, se incrementará fuertemente para encontrar la mejor solución cuanto antes.

Desde el punto de vista Neo-Darwiniana, el proceso de evolución se puede dividir en tres categorías [99]:

- ✓ *Selección estabilizante.*
- ✓ *Selección direccional.*
- ✓ *Selección disruptiva.*

La *selección estabilizante* también se la llama como *selección normalizada*, porque tiende a eliminar cromosomas con valores extremos. La *selección direccional* tiene el efecto de incrementar o decrementar el valor medio de la población. La *selección disruptiva* tiende a eliminar cromosomas con valores moderados. La mayoría de los métodos de selección están basados en selección direccional.

4.5. OTROS COMPONENTES

4.5.1. ELECCIÓN DE LA POBLACIÓN INICIAL

En términos generales, la población inicial debe ser lo más variada posible. Es necesario que la distribución de aptitudes sea uniforme para evitar la convergencia

prematura. En la práctica, por falta de mayor información, se elige la población inicial al azar. El conocimiento específico puede ayudar a formar una población inicial factible con algún individuo cercano al óptimo.

La iniciación de la población en GA se realiza por muestreos aleatorios de una variable binaria, l . *pop_size* veces (muestreos independientes), donde l es la longitud del cromosoma; de esta forma se determina cada bit de la población inicial $P(0)$ en forma aleatoria.

4.5.2. CRITERIOS DE TERMINACIÓN

Los criterios de finalización de los GAs se identifican frecuentemente con un número máximo de generaciones. Sin embargo, algunas veces también se controla por la medida de la diversidad genética que permite calcular la convergencia promedio de todas las posiciones de bits en toda la población, es decir, la medida *bias* definida por Grefenstette [79]. Denotando un único individuo por $x_i=(x_{i,1},...,x_{i,l})$ ($\forall i \in \{1,\ldots, pop\_size\}$) para distinguir sus bits, el bias $b(P(t))$ de una población se calcula como:

$$b(P(t)) = \frac{1}{l \cdot pop\_size} \sum_{j=1}^{l} \max\left\{ \sum_{i=1}^{pop\_size} (1 - a_{i,j}), \sum_{i=1}^{pop\_size} (a_{i,j}) \right\} \in [0.5,1.0]$$

El valor $b \in [0.5, 1.0]$ indica el porcentaje promedio de los valores predominantes en cada posición de los individuos. Un valor pequeño de b indica una alta diversidad genotípica, mientras que un valor grande implica una baja diversidad.

4.6. LIMITACIONES DE LOS GAS E INCONVENIENTES ASOCIADOS

Las limitaciones principales de un GA básico se pueden describir de un modo general de la siguiente manera [116]:

- ✓ El espacio de búsqueda se puede representar solamente por cadenas binarias.
- ✓ El mecanismo de un algoritmo genético básico no considera las restricciones posibles que se pueden agregar a la búsqueda.

- ✓ El algoritmo genético básico es un algoritmo de búsqueda ciego, es decir, sólo está guiado por la aptitud de los individuos, y no incorpora ningún otro conocimiento específico del problema en cuestión.
- ✓ A efectos prácticos, el GA simple sólo puede trabajar con poblaciones finitas no muy grandes.

Los inconvenientes que producen las dos primeras limitaciones no se salen de lo previsible; mientras que las dos últimas son causa original de muchos inconvenientes que se deberán atacar por separado. De manera esquemática, se puede decir que todos esos inconvenientes tienen su base en dos hechos:

1. *Debilidad*.
2. *Problemas de diversidad* derivados del uso de poblaciones finitas.

4.6.1. EL PROBLEMA DE LA DEBILIDAD DE LOS GAS

El inconveniente más visible de la debilidad es la *ineficiencia*: los métodos débiles de resolución de problemas presentan las ventajas de ser muy poco específicos, poco exigentes y notablemente eficaces; sin embargo, todos ellos son ineficientes en mayor o menor grado, y especialmente se les compara con métodos heurísticos. Los GAs sólo se pueden considerar eficientes en comparación con otros métodos estocásticos de búsqueda ciega, pero se puede garantizar que si se encuentra un método heurístico para resolver un problema, este siempre lo hará mucho más eficientemente que un GA.

Otro de los inconvenientes prácticos de la debilidad es la tendencia al *extravío* de la búsqueda. El GA simple realiza la búsqueda de los mejores puntos utilizando únicamente la aptitud de los individuos para recombinar internamente los bloques constructivos; en ocasiones ocurre que la información proporcionada resulta insuficiente para orientar correctamente al algoritmo en su búsqueda del óptimo. A esto se le llama *desorientación*. En el ámbito interno esto se concreta con la no verificación de la hipótesis de los bloques constructivos, es decir, ciertas combinaciones válidas de buenos bloques constructivos originan individuos de baja aptitud. En el peor de los casos puede ocurrir que este fenómeno sea preponderante e impida que el GA converja al óptimo global, desviándolo finalmente hacia un óptimo local. Esto es el extravío propiamente dicho. Afortunadamente, aunque un GA se desoriente no necesariamente se va a

extraviar; para eso es necesario partir de una mala población inicial o plantear un problema especialmente diseñado para que se extravíe.

Para que el mecanismo básico de los GAs funcione correctamente es necesario proporcionarle una mínima cantidad y calidad de información. Si no se le proporciona ese mínimo el mecanismo no funciona con propiedad, y el GA evolucionará incorrectamente.

Como es de suponer, el factor que más contribuye a la existencia de desorientación es la forma de la función de fitness. Típicamente, la desorientación es frecuente en problemas en los que los puntos óptimos están aislados y rodeados de puntos de muy bajo fitness. Sin embargo, la desorientación suele estar fuertemente estimulada por la mala codificación que oculte la ya de por sí escasa información disponible. De modo recíproco, una buena codificación reduce la probabilidad de extravío.

Un caso especialmente desfavorable ocurre cuando hay una fuerte interacción entre dos o más atributos, de tal forma que la contribución a la aptitud de un individuo que realiza cierto gen depende grandemente de los valores que tomen otros. A este fenómeno se denomina *epistasis* o acoplamiento y su presencia garantiza la desorientación dado que en esas condiciones resulta muy difícil constituir buenos bloques constructivos.

4.6.2. EL PROBLEMA DE LA DIVERSIDAD EN LOS GAS

Es esencial para el buen funcionamiento de un GA tener controlada en todo momento la diversidad de la población. En un sentido general se entiende por diversidad a la variedad de individuos y en particular a la variedad de aptitudes.

La necesidad de la diversidad de individuos se debe a que con poca variedad el operador de crossover pierde casi por completo la capacidad de intercambio de información útil entre individuos, y en definitiva, la búsqueda se estanca. La necesidad de tener controlada la diversidad de aptitudes (de individuos) radica en la imposibilidad práctica de trabajar con una población infinita.

Con poca diversidad de aptitudes, todos los individuos tendrán más o menos las mismas posibilidades de sobrevivir, la selección no incrementará la diversidad por lo que el peso de la búsqueda recaerá en los operadores genéticos, lo que traerá como

consecuencia una búsqueda aleatoria. Para que la selección resulte efectiva, la población debe contener en todo momento una cierta variedad de aptitudes.

Cuando la población es finita tampoco se puede tener gran disparidad de aptitudes, pues ello suele afectar muy negativamente a la diversidad de la población.

Es imprescindible tener control sobre la diversidad de aptitudes de la población para evitar que se produzca una convergencia prematura ya sea por mucha diversidad (superindividuos que son aptos en un cierto momento pero no los más aptos absolutos) o incluso por demasiado poca (deriva genética).

Capítulo V

ALGORITMOS EVOLUTIVOS AVANZADOS

5.1. INTRODUCCIÓN

En este capítulo se presentan algunas modificaciones al diseño de un algoritmo evolutivo para mejorar su performance. El hecho de innovar en alguna de las características de los algoritmos evolutivos, a fin de acrecentar su calidad y aplicabilidad, los transforman en algoritmos evolutivos avanzados. Estas modificaciones pueden afectar tanto a los operadores genéticos como al mecanismo de selección y tienen por objetivo balancear la explotación y la exploración a fin de evitar la convergencia prematura durante el proceso de búsqueda.

En los siguientes apartados se presentan tres técnicas que combinadas minimizan el riesgo de la convergencia prematura, a lo cual se le suma que la población final se encuentre centrada alrededor de la solución óptima al problema en cuestión. Las técnicas son:

- ✓ Múltiples padres.
- ✓ Múltiples crossovers.
- ✓ Prevención de incesto.

5.2. ALGORITMOS EVOLUTIVOS AVANZADOS CON MÚLTIPLES PADRES Y MÚLTIPLES CROSSOVERS

En la evolución simulada, es decir con algoritmos evolutivos, muchas características técnicas se inspiran por los mecanismos naturales. En particular, variantes abstractas de la reproducción sexual y asexual se implementan como operadores de búsqueda. Algunas técnicas evolutivas, por ejemplo programación evolutiva, trabajan exclusivamente con mutación (es decir, implementan una simplificación de la reproducción asexual), mientras otras, por ejemplo algoritmos genéticos y estrategias

evolutivas, usan recombinación (implementan una simplificación de la reproducción sexual) y mutación. Hay varios trabajos que investigan las ventajas y desventajas de la mutación con respecto al crossover [41, 60, 63, 86, 125,131].

Técnicamente, la cuestión se centra en la aridad de los operadores de reproducción. La mutación y el crossover tienen aridad uno y dos, respectivamente, y la pregunta es si los operadores unarios o binarios son preferibles para instancias típicas de problemas de optimización de relevancia práctica. Desde el punto de vista técnico, no hay necesidad de restringir la aridad de los operadores de reproducción en uno o dos. En general, un operador de reproducción puede tener una aridad desde uno al tamaño de la población (o aún más, si se permite la repetición entre los padres). Durante el desarrollo de la computación evolutiva se han propuesto varios operadores de recombinación, algunos alrededor de los sesenta [19]. En estrategias evolutivas, la recombinación global puede mezclar información proveniente de más de dos padres en un hijo [8, 126]. En algoritmos genéticos, la recombinación *p-sexual voting* de Mühlenbein [106], *scanning crossover* y *diagonal crossover* [37] y *recombinación gene-pool* [107, 141] son ejemplos de operadores *multi-parents*.

5.2.1. EVOLUCIÓN DE LA OPCIÓN MULTIPARENT

Un proceso de evolución mantiene una población de cromosomas a efectos de buscar a través del espacio de búsqueda de soluciones potenciales. Se debe lograr un balance entre [18]: la *explotación* de la mejor solución y la *exploración* del espacio de búsqueda.

Un factor a tener en cuenta, a la hora de mantener un buen balance, es el efecto causado por los mecanismos de selección: una fuerte presión selectiva puede producir una convergencia prematura a un óptimo local, mientras que una presión selectiva suave puede llevar a que la búsqueda se vuelva inefectiva.

El crossover contribuye en gran medida al intercambio de material genético durante la ejecución de un algoritmo evolutivo. Este operador combina las características de dos cromosomas para formar dos hijos similares al intercambiar los segmentos correspondientes de los padres. La idea detrás de la aplicación del operador de crossover es el intercambio de información entre diferentes soluciones potenciales. Muchos investigadores estudiaron el efecto de distintos tipos de crossover para mejorar la

performance de un GA en cuanto a exactitud y velocidad. Esas investigaciones abarcan varios tipos de crossover (crossover de un punto, crossover de dos puntos, crossover multi punto y crossover uniforme) y crossover especializados los cuales dependen de la estructura de datos particular usada para representar un cromosoma. Por ejemplo, Davis [25], Goldberg [74] y Oliver [111] crearon diversas variantes de crossover tal como OX, PMX y CX cuando trataban el problema del viajante (TSP) usando algoritmos genéticos. Se propusieron diversos operadores de crossover los cuales investigan el espacio de soluciones de diferentes maneras. Frantz [65], Syswerda [134] y Davis [27] propusieron el crossover *multi punto*, *uniforme* y *guaranted-uniform crossover*, respectivamente. Más recientemente Eshelman y Schaffer [39] estudiaron las características de algoritmos genéticos probando diversas variantes de crossover.

Independientemente del tipo de crossover aplicado, con la opción de crossover convencional se aplica el operador sólo una vez sobre la pareja de padres seleccionada para crear dos hijos. Por ejemplo, crossover de un punto, de dos puntos, uniforme y otras variaciones producen un par de hijos por pareja. Tal procedimiento se denomina como *single crossover per couple* (SCPC).

5.2.1.1. MULTIPLE CROSSOVER PER COUPLE

En [44] y [45] se presenta una opción diferente: *permitir múltiples hijos por pareja*, como frecuentemente ocurre en la naturaleza. La cantidad de hijos varía desde cero a algún número máximo predefinido. Este método de crossover se conoce como *multiple crossover per couple* (MCPC).

Esta opción se probó con un conjunto de funciones de testeo bien conocidas (funciones de De Jong F_1, F_2, F_3 [29], Schaffer F_1 [124] y otras funciones). El hecho de realizar múltiples crossover por pareja sobre un par de padres seleccionados provee beneficios extras en cuanto a tiempo de procesamiento y calidades similares en las soluciones halladas cuando se contrasta contra la opción convencional SCPC. Con la opción MCPC se observa que [49]:

- ✓ En algunos casos MCPC halla resultados que son mejores que los hallados por SCPC.
- ✓ MCMP halla una reducción del tiempo de procesamiento cuando se incrementa el número de crossover realizados por pareja.

- ✓ Se obtiene una mejora en la calidad de los resultados permitiendo entre 2 y 4 crossover por pareja.

MCPC se aplicó para optimizar funciones de testeo clásicas y algunas funciones duras (no lineales, no separables). Las bondades de esta opción prevalecen bajo todos los testeos y revelan que, cuando MCPC se aplica con 2, 3 y 4 crossovers por pareja, los resultados son tan buenos como se pueden esperar bajo SCPC, pero con un beneficio adicional en tiempo de procesamiento. La performance se obtiene a través de la habilidad mostrada por MCPC para explotar la recombinación de las soluciones buenas halladas.

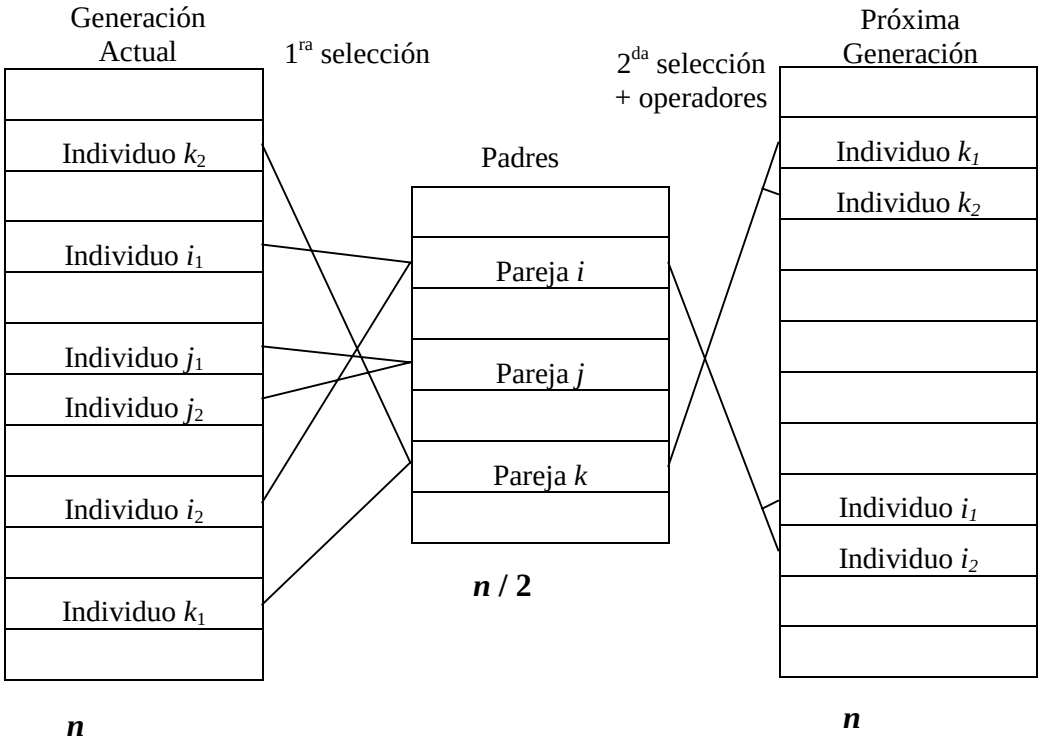
Los efectos de MCPC son: una economía en el esfuerzo computacional y una gran explotación en la recombinación de las soluciones buenas previamente encontradas. Pero por otra parte, esos experimentos mostraron que, en algunos casos, el método incrementa el riesgo de convergencia prematura dada la pérdida de diversidad genética. Para solucionar este problema se presenta una estrategia para combinar la explotación introducida por MCPC con la exploración de nuevas soluciones en el espacio del problema. Esto se logra al introducir un nuevo operador de selección [48]: *fitness proportional couple selection* (FPCS).

FPCS divide el proceso de selección total en dos etapas [51]. La primera selecciona los individuos de la población actual para crear un grupo intermedio de parejas, y posteriormente, la segunda etapa somete a esa población de parejas a la selección para cruzar aquellos pares con el fitness más alto (figura 5.1) [46]. Ambos pasos de selección se basan en el esquema de selección proporcional.

El operador de selección se puede modificar para favorecer, con más hijos, a aquellas parejas que tienen características específicas. No hay problema respecto del fitness individual, pero se han elegido dos criterios para asignar el fitness a las parejas: una basada sobre miembros con fitness disímiles ($FPCS_{DIFF}$) y el otro, sobre fitness promedio ($FPCS_{AVG}$).

El método se puede describir de la siguiente manera:

1. seleccionar individuos usando selección proporcional para construir la población intermedia de padres.
2. asignar a cada pareja un valor de fitness de pareja, calculado de acuerdo al criterio de fitness de la pareja.



1ra Selección: selección proporcional al fitness del individuo
2da Selección + operadores: selección proporcional al fitness de la pareja más operadores genéticos clásicos.
Fitness de la Pareja = |fitness.padre1 – fitness.padre2|
n: tamaño de la población.

Figura 5.1 Esquema del proceso de selección de parejas.

3. seleccionar las parejas para reproducción por medio de selección proporcional. Por cada pareja, se controla el proceso de producción de hijos para no sobrepasar el tamaño de la población.

Combinando las dos técnicas, MCPC y FPCS, se intenta disminuir la presión selectiva introducida por MCPC, buscando alta calidad de soluciones. Además se intenta mejorar la diversidad genética en forma indirecta al asignar un fitness alto a parejas disímiles esperando consecuentemente que los padres se encuentren a gran distancia en el *fitness landscape*.

Se obtienen buenos resultados para todas las funciones de optimización cuando se contrastan con los resultados obtenidos solamente bajo la opción MCPC [48]. Estos buenos resultados se logran permitiendo entre 2 y 3 crossover por pareja; pero determinar el valor adecuado de crossover por pareja es una tarea bastante tediosa.

Como una alternativa para hallar un número óptimo, se incorpora a cada individuo de la población el número de crossovers [47]. Así con el EA se realiza una auto-

adaptación de este parámetro dando la cantidad más apropiada de la repetición de crossovers.

La auto-adaptación es un nuevo campo en computación evolutiva la cual aconseja la actualización de parámetros del algoritmo al evolucionarlos como parte de la estructura del cromosoma. La auto-adaptación de los parámetros es una de las tres técnicas mencionadas en [36] para cambiar dinámicamente el valor de un parámetro mientras el algoritmo se está ejecutando. En [36] se dan las siguientes categorías de control de parámetros:

- ✓ *Control de parámetros determinístico*: es el caso en el cual los valores de los parámetros se modifican acorde con una regla determinística, sin retroalimentación alguna del proceso de búsqueda realizado por la estrategia.
- ✓ *Control de parámetros adaptables*: en este caso, se usa alguna información de retroalimentación del proceso de búsqueda para determinar la dirección y magnitud del cambio en los parámetros.
- ✓ *Control de parámetros auto-adaptables*: aquí los parámetros a ser adaptados se codifican dentro del cromosoma y se someten a los operadores genéticos. Los mejores individuos de la población tienen mejores chances de supervivencia y reproducción. Aquí es de esperar que los mejores valores de parámetros se propaguen intensivamente.

Como el número de crossovers a ser aplicado a una pareja en MCPC es uno de los parámetros del algoritmo, incluidos como parte del individuo, la opción pertenece a la última categoría mencionada.

Como en [47] se usa una representación binaria del cromosoma, el número de crossovers permitido por un individuo se codifica en un campo el cual ocupa los últimos $log_2(max\_cross + 1)$ bits de cada individuo para hallar el óptimo esperado. Esos últimos bits se denominan *ncross_field*. En algunos experimentos se permite un máximo de tres y en otros un máximo de siete crossovers por pareja. De modo que dos o tres bits extras son suficientes para tal propósito.

De esta forma se tienen dos espacios de búsqueda: uno correspondiente a la función objetivo, y otro asociado al número de crossovers a aplicar.

Cada individuo preserva información respecto del número de crossover originalmente aplicados a sus padres. De esta forma es de esperar que, basados en el

principio de *supervivencia por fitness*, las buenas soluciones lleven información del número de crossover aplicados a sus ancestros y que este número sea apropiado.

De acuerdo a Spears [132], se usa una técnica de auto-adaptación. Una vez que se selecciona la pareja, se chequea el correspondiente número de crossovers arrastrado por cada padre y:

- ✓ si son iguales, entonces se aplica el operador de crossover tantas veces como lo especificado por *ncross_field*,
- ✓ en caso contrario, se usa un número en el rango permitido en lugar del número *ncross_field*.

En la segunda situación, y siguiendo la opción de Spears, cuando el número decodificado de crossover es diferente, se viola la intención de preservar información porque los hijos no podrán tomar el número de crossovers por el cual fueron creados. Si el crossover y la mutación no alteran el *ncross_field* (y este evento tiene una baja probabilidad de ocurrencia) entonces el hijo retiene información de sus padres, pero no preserva información respecto de cómo fueron creados.

Para retener la información respecto de cuantos individuos fueron creados o cómo los padres fueron creados, en [47] se presentan dos opciones:

- ✓ *E1*: en cualquier situación, el intercambio de información de los padres a los hijos se hace en la forma tradicional aplicando el operador genético con sus correspondientes probabilidades. En el caso de que los valores del campo *ncross_field* no fueran iguales, esta opción, preservando la información de los padres, fuerza la diversidad poblacional en el espacio de búsqueda del parámetro, porque la mayoría de las veces un hijo hereda características desde un padre y el otro hijo hereda característica desde el otro padre.
- ✓ *E2*: si los valores especificados en *ncross_field* no fueran coincidentes, entonces un nuevo valor aleatorio para el número de crossover se inserta primero en el campo *ncross_field* del padre, y luego se realiza el crossover por el número de veces especificado por ese valor generado al azar. Al preservar la información, esta opción crea individuos más similares en el espacio de búsqueda del parámetro e incrementa la pérdida de diversidad genética.

La auto-adaptación se analiza desde los resultados obtenidos por dos funciones de testeo bastante duras: función Easom y la función 7 de Schweffel (función multimodal).

Para contrastar los resultados se usa un EA canónico con los mismos valores para los parámetros.

Las variables con las cuales se mide la performance son [47]:

- ✓ *Cr_Avg* que mide el número de crossover medio por pareja.
- ✓ *Calidad* de las soluciones encontradas.

En ambos casos, la *calidad* alcanza el valor 1 (al menos una solución fue óptima) luego de 80 a 152 generaciones respectivamente y los valores de *CrAvg* varían entre 2.5 y 2.8 luego de unas pocas generaciones bajo E1 y entre 2.3 y 2.6 bajo E2.

Aquí el comportamiento del mecanismo de control de parámetro adaptable es sencillo: cuando la diversidad genética en el espacio de búsqueda del parámetro es baja entonces se permiten menos cantidad de crossovers y viceversa. Este comportamiento favorece el proceso evolutivo.

5.2.1.2. MULTIPLE CROSSOVERS PER MATING ACTION

Como una consecuencia de la investigación relacionada a problemas de optimización multicriterio [50], MCPC se extiende a *multiple crossovers per mating action* (MCPMA) para aplicar el método a un conjunto formado por padres. Esto sirve para determinar que el *multiparent approach* (propuesto por Eiben) en un estudio de optimalidad Pareto [38, 34, 33] mitiga la posible pérdida de diversidad generada por MCPC y no requiere ajustes extras. Consecuentemente, se puede balancear la explotación y la exploración del espacio del problema.

MCPMA provee un medio para explotar buenas características de más de dos padres seleccionados en función de su fitness al aplicar repetidamente el método de crossover seleccionado. Una vez seleccionados, los padres se someten al crossover n_1 veces (especificado como argumento), y genera n_1 hijos.

Eiben usa, en su opción de múltiple padres, operadores genéticos multi-parent [37]. Esos operadores usan dos o más padres para generar hijos. El mecanismo básico usado para definir operadores multi-parent es *gene-scanning*. Las técnicas de scanning generan un hijo desde n_2 padres con $n_2 \geq 2$. Sobre diferentes funciones de optimización, diferentes versiones de scanning muestras diferentes comportamientos. Usando un gran número de padres n_2, la creación de hijos está basada sobre un mayor muestreo del espacio de búsqueda y consecuentemente se suministra una mayor diversidad. Esto

puede ayudar a evitar la convergencia prematura. Como ha reportado en su trabajo, es difícil determinar el número óptimo de padres, pero ha concluido que la mejor performance se obtiene cuando se trabaja entre 2 y 4 padres, e incrementado el número de padres deteriora la performance.

5.2.1.3. MULTIPLE CROSSOVERS ON MULTIPLE PARENTS (MCMP)

Multiple crossovers on multiple parents (MCMP) presentado en [49], permite la múltiple recombinación de múltiples padres bajo crossover uniforme, esperando que la explotación y la exploración del espacio del problema estén adecuadamente balanceadas.

MCMP provee un medio para explotar buenas características de más de dos padres seleccionados acorde a su fitness y aplicando repetidamente una de las variantes de SX: un número n_1 de crossovers se aplica sobre un número n_2 de padres seleccionados. De los n_1 hijos producidos se seleccionan n_3, acorde a algún criterio, para ser insertados en la próxima generación.

Bajo esta opción la velocidad de convergencia, medida en el número de generaciones, se aumenta sin incrementar el riesgo de convergencia prematura. Consecuentemente, la calidad de los resultados es muy buena.

Adicionalmente, cuando se observa la población final, se detecta que todos los individuos están mucho más concentrados alrededor del óptimo. Esta propiedad se detecta en la optimización multimodal. Esto es un aspecto importante para aplicaciones que necesitan soluciones alternativas cercanas al óptimo.

Aunque no se puede ser concluyente, por medio de esta asociación, múltiples crossovers sobre múltiples padres, el espacio de búsqueda se explota eficientemente por la aplicación de múltiples crossovers y se explora eficientemente por el gran número de muestreos provistos por múltiples padres.

5.2.2. OPERADORES GENÉTICOS MULTI-PARENT

En esa sección se describen varios operadores multi-parents. Esos operadores usan dos o más padres para generar un hijo.

5.2.2.1. TÉCNICAS DE GENE-SCANNING

El mecanismo general para scanning [35] es asignar un marcador tanto a cada padre como al hijo. El marcador del hijo atraviesa todas sus posiciones de izquierda a derecha, una por vez. En cada paso, el marcador de los padres se actualiza de modo que cuando se debe elegir un valor para el gen marcado en el hijo, los marcadores de los padres muestren las posibles elecciones. El algoritmo correspondiente se muestra en la figura 5.2.

Las dos características más importantes de todos los procedimientos de gene scanning son:

- ✓ el mecanismo de actualización del marcador del padre, y
- ✓ la forma en la cual se elige un valor de los genes marcados. Si se varía la forma en la cual se eligen los valores a ser insertados en el hijo se tienen tres técnicas de scanning específicas: *uniform, ocurrence based* y *fitness based*.
- ✓

5.2.2.1.1 UNIFORM SCANNING

Uniform scanning (USX) [35] es una extensión natural del número de padres para el crossover uniforme. El crossover uniforme se define de la siguiente manera:

- ✓ recorrer los dos padres y los dos hijos de izquierda a derecha,
- ✓ por cada posición en el hijo 1, elegir en forma aleatoria si se hereda del padre 1 o del padre 2.

```
procedure gene-scanning
begin
    Iniciar el marcador de los padres
    for marcador_hijo = 1 to l do
            Actualizar el marcador de los padres
            child.allele[marcador_hijo] = elegir un valor desde los indicados por los
                marcadores de los padres
  end for
end procedure
```

Figura 5.2 Procedimiento de gene-scanning

Padre 1	1	0	1	1	0	1	0	1	1	1
Padre 2	1	0	0	0	0	1	0	1	0	1
Padre 3	0	1	0	1	1	0	1	0	0	0
Padre 4	0	1	0	1	0	1	0	1	0	1
Hijo	1	0	0	1	0	1	0	1	0	1

Figura 5.3 OSX sobre un patrón de bits.

Para USX, se crea un único hijo. Cada alelo en el hijo se elige por un mecanismo aleatorio uniforme, cada padre tiene igual chance de ser elegido para aportar un valor. Esto significa que la posibilidad de heredar desde el padre *i*, P(*i*), se define como:

$$P(i) = \frac{1}{número\_de\_padres}$$

y el número esperado de genes para heredar desde el padre como:

$$E(i) := P(i) \times CHROMOSOME\_LENGTH$$

5.2.2.1.2 OCURRENCE BASED SCANNING

Ocurrence based scanning (OSX) [35] está basado sobre la premisa de que los valores que se repiten con mayor ocurrencia en la mayoría de los padres (los cuales se seleccionan en base a su fitness), en una posición particular, es el mejor valor a elegir. La elección entre los valores marcados se hace por una función de mayoría. Si no hay mayoría entre los valores marcados, se elige el valor del primer padre. La figura 5.3 muestra el funcionamiento del OSX para una representación binaria.

5.2.2.1.3 FITNESS BASED SCANNING

Fitness based scanning (FSX) [35] elige el valor a ser heredado en forma proporcional al valor de fitness de los padres. Por ejemplo, para un problema de maximización donde el padre *i* tiene un fitness *f(i)*, la probabilidad *P(i)* de elegir un valor desde esos padres es (selección por ruleta):

$$P(i) = \frac{f(i)}{\sum f(i)}$$

Como para OSX, esto significa que el número esperado de genes heredado del padre *i* es:

$$E(i) = P(i) \times CHROMOSOME\_LENGHT$$

5.2.2.1.4 ADAPTANDO EL SCANNING A DIFERENTES TIPOS DE REPRESENTACIONES

Es posible definir procedimientos de scanning para diferentes tipos de representaciones al cambiar el mecanismo de actualización de los marcadores [35]. Para representaciones donde todos las posiciones son independientes (no hay epistasis), el mecanismo de actualización del marcador es simple, los marcadores de los padres se inician en la primer posición de cada uno de ellos, y a cada paso todos se incrementan en uno (así recorren los padres de izquierda a derecha). En problemas donde está presente la epistasis se necesitan mecanismos de actualización de los marcadores más sofisticados.

5.2.2.2 ADJACENCY BASED CROSSOVER

Adjacency based crossover (ABC) [35] es un caso especial de scanning, diseñado para representaciones basadas en permutaciones donde las posiciones relativas de los valores es importante, por ejemplo en TSP. La principal diferencia entre los procedimientos de scanning ya descriptos y ABC es la forma en la cual se selecciona el primer valor y el mecanismo de actualización del marcador. El primer valor del gen del hijo se hereda siempre del primer valor del gen del primer padre. El mecanismo de actualización es el siguiente: por cada padre, se setea su marcador al primer sucesor del valor previamente seleccionado, el cual no está en el hijo.

Un crossover similar a ABC se propone en [146]. La principal diferencia se produce cuando todos los sucesores inmediatos a la ciudad se han heredado en el hijo.

Se han definido dos tipos de ABC: *ocurrence based* (OB-ABC) y *fitness based* (FB-ABC). Ambos usan el mismo mecanismo de selección de los valores a ser heredados por el hijo como occurrence based (figura 5.4) y fitness based scanning, la diferencia es el mecanismo de actualización del marcador.

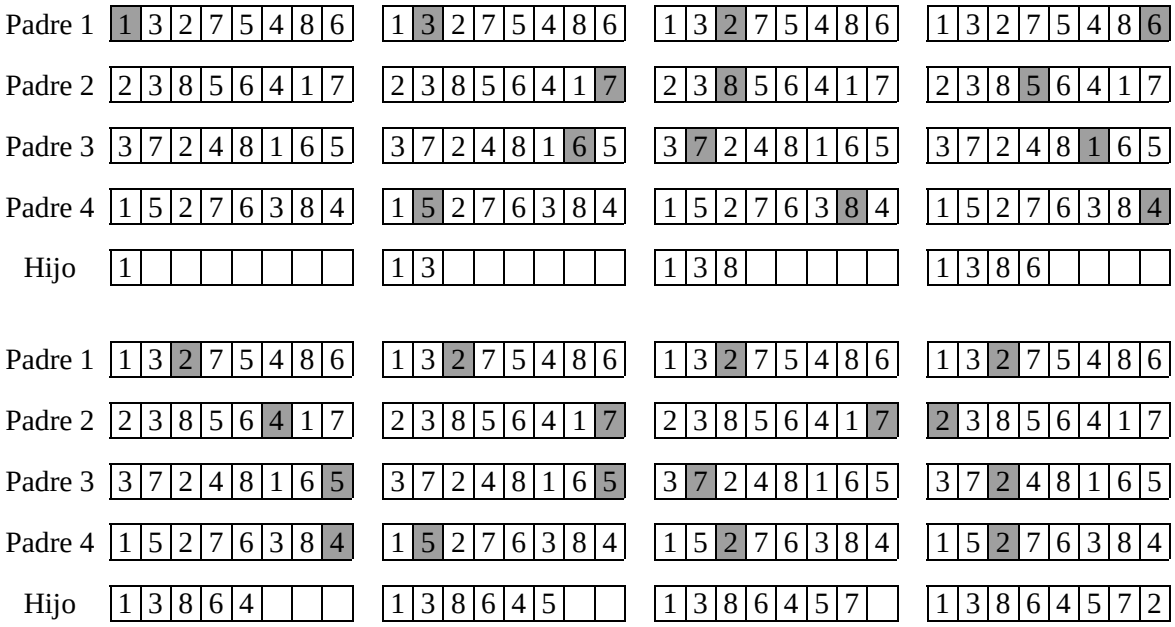


Figura 5.4 Ejemplo del crossover OB-ABC.

5.2.2.3 CROSSOVER DIAGONAL

El crossover tradicional crea dos hijos desde dos padres al empalmar los padres entre el único punto de crossover e intercambiando los substrings. La idea básica detrás del crossover diagonal es generalizar este mecanismo para un crossover *n*-ario de $(n-1)$-point. El crossover diagonal selecciona $(n-1)$ puntos de crossover produciendo *n* segmentos del cromosoma y formando *n* hijos al tomar las piezas desde los padres a través de las diagonales. Por ejemplo, el primer hijo se compone tomando el primer $substring_1$ del $padre_1$, el segundo $substring_2$ del $padre_2$, etc. Mientras el segundo hijo deberá tener el primer $substring_1$ del $padre_2$, primer $substring_2$ del $padre_3$, etc. La figura 5.5 ejemplifica esta idea.

5.3. ALGORITMOS EVOLUTIVOS AVANZADOS CON PREVENCIÓN DE INCESTO

Se han sugerido varios mecanismos para combatir la convergencia prematura en algoritmos genéticos. Muchos de ellos, están sujetos a la objeción de que tienden a socavar el paralelismo implícito del algoritmo, la propiedad que distingue a los algoritmos genéticos de otros mecanismos de búsqueda.

Padre 1	1a	1*b*	1*c*
Padre 2	2a	2*b*	2c
Padre 3	3a	3*b*	3*c*
Hijo 1	1*a*	2*b*	3*c*
Hijo 2	2*a*	3*b*	1*c*
Hijo 3	3*a*	1*b*	2*c*

Figura 5.5 Crossover diagonal para tres padres.

La convergencia prematura –pérdida de diversidad poblacional antes de alcanzar el óptimo o al menos un valor satisfactorio- se reconoce como una falla seria de los algoritmos genéticos. En [40] se examinan tres estrategias para combatir la convergencia prematura:

- ✓ Una estrategia de mating, *prevención de incesto,*
- ✓ Una estrategia de crossover, *uniform crossover*, y
- ✓ Una estrategia de manejo de la población, *eliminación de duplicados*.

Todas esas estrategias, pero especialmente las dos primeras, están sujetas a la objeción de volver al operador de crossover más destructor de esquemas, debilitando el paralelismo implícito de los algoritmos genéticos.

La idea de prevención de incesto, fue inicialmente propuesta por Eshelman y Schaffer [40] y muestra sus beneficios para evitar la convergencia prematura. Este método evita el cruzamiento de padres que muestren similitudes, las cuales se determinan en función de la distancia de Hamming de los padres. La prevención de incesto fue extendida en [2] al mantener información de los ancestros dentro del cromosoma. Modifica el mecanismo de selección para reproducción al prevenir el cruzamiento de individuos pertenecientes a la misma familia, por un número predefinido de generaciones. Esta opción fue testeada sobre un conjunto de funciones multimodales.

5.3.1. OPCIÓN DE ESHELMAN Y SCHAFFER

Los padres se pueden seleccionar de modo de mantener la diversidad de la población. Usualmente, las estrategias de mating se consideran en el contexto de especialización, donde el objetivo es prevenir que individuos disímiles se crucen [17,

147]. Todo lo demás sigue igual, los hijos producidos por crossover de diferentes padres tenderán a ser más diversos. Las *funciones de sharing* de Goldberg, al reducir el fitness de los individuos como una función de cuán similares son a los otros individuos en la población, se pueden pensar como una estrategia de mating indirecta [77]. Aunque Goldberg presente funciones de sharing en el contexto de especialización, tienen el efecto de reprimir más que promover el mating dentro de especies, al menos para la especie dominante. En lugar de evitar el mating de individuos diversos, los individuos similares pertenecientes a la especie más grande tienen menor probabilidad de ser seleccionados para reproducción con los demás. El mecanismo de prevención de incesto de Eshelman es una opción más directa para prevenir el mating de individuos similares [42]. Los individuos se recombinan aleatoriamente, pero sólo se cruzan si su distancia Hamming es superior a un cierto umbral. De este modo sólo una parte de la población se cruza para producir nuevos hijos en cualquier generación. El umbral se establece inicialmente a una distancia Hamming promedio esperada de la población inicial, y entonces se disminuye a medida que la población converge. Aunque este mecanismo no previene explícitamente que se crucen hermanos o ancestros cercanos, este mecanismo tiene su efecto en la medida en que tales individuos tiendan a ser similares.

5.3.2. PREVENCIÓN DE INCESTO EXTENDIDA

La extensión del concepto de incesto está fuertemente relacionada al cruzamiento de miembros de la misma familia. Para prevenir el incesto, la *prevención de incesto extendida* (EIP) [2] permite sólo la recombinación de individuos que no compartan ancestros. Para construir la nueva población, los individuos se eligen de la previa en función a la selección proporcional al fitness, pero se permite su cruzamiento si no se detectan ancestros comunes en las generaciones cercanas. De esta forma, el intercambio de material genético se reduce y la diversidad de la población se mantiene en algún grado conveniente. Por lo tanto cada individuo mantiene información respecto de sus ancestros.

Permitiendo el crossover sólo entre individuos que no son familiares, se modifica el efecto del mecanismo de selección en la población. Sin embargo, la selección es el único operador de un EA donde el fitness del individuo afecta el proceso de evolución. En tal proceso hay dos puntos fuertemente relacionados: diversidad poblacional y

presión selectiva forzadas por el mecanismo. La selección juega un rol importante porque una fuerte presión selectiva puede producir convergencia prematura y una presión selectiva suave puede provocar que la búsqueda se torne inefectiva [84].

El pseudo-código que se muestra en la figura 5.6 delinea un procedimiento para prevenir el incesto entre miembros de una misma generación o consecutivas, es decir entre hermanos y entre padres e hijos.

En lugar de medir la similitud entre individuos a través de su distancia Hamming para prevenir la recombinación, EIP evita el cruzamiento entre individuos pertenecientes a la misma familia en un número limitado de generaciones.

Esta opción muestra una buena performance cuando se la contrasta con un EA tradicional para la optimización de funciones multimodales de variada dificultad. Los valores óptimos se alcanzan en la mayoría de las corridas de cada una de las series.

Esta variante presenta también un criterio alternativo para prevenir el incesto que es independiente de la representación del individuo y consecuentemente, se puede aplicar a representaciones con enteros, reales, vectores y otras sin necesidad de realizar cambios. Esta capacidad no está presente en el enfoque de Eshelman.

```
procedure selección de padres
begin
  for 1 to pop_size
     seleccionar indiv-1 C(t)
     seleccionar indiv-2 C(t)
     while ((padre(indiv-1) = padre(indiv-2)) or
          (indiv-1 = padre(indiv-2)) or
          (indiv-2 = padre(indiv-1)))
             seleccionar indiv-2 C(t)
     end while
     recombinar y mutar individuos en C(t) construyendo C'(t)
  end for
end procedure
```

Figura 5.6 Procedimiento para EIP

En [2] se reportan resultados para dos variantes de EIP, se realiza la prevención de incesto durante dos (EIP2) y tres generaciones (EIP3) sucesivas. Cuando se usa la opción EIP3, se presentan algunas dificultades y se deben a que a medida que la población evoluciona se vuelve más difícil encontrar individuos para formar una pareja que no pertenezcan a la misma familia, para salvar esta dificultad es necesario considerar un tamaño de la población mayor al tomado para EIP2.

Se realizaron experimentos para un gran número de generaciones consecutivas, pero el proceso muestra un alto overhead cuando se buscan individuos que posean ancestros distintos cuando se realiza la búsqueda localizada. La prevención de incesto manteniendo la información ancestral por tres generaciones consecutivas presenta mejores resultados que cuando se aplica a sólo dos generaciones sucesivas.

EIP ha sido puesta a prueba sobre un amplio conjunto de funciones de testeo y aunque adiciona algún overhead dado el control, específicamente, cuando la diversidad genética de la población inicial no es alta, los resultados son prometedores.

5.4. EA AVANZADOS CON MULTIPLICIDAD Y PREVENCIÓN DE INCESTO

En [104] y [105] se presenta un algoritmo evolutivo el cual combina múltiples crossovers, múltiples padres y prevención de incesto. Esos trabajos mostraron que se mejora considerablemente el proceso de búsqueda de un algoritmo evolutivo.

Cuando se usa una cantidad de padres $n_2 > 2$, el pseudo código para la opción EIP se muestra en la figura 5.7.

Como se puede observar, cuando se aplica MCMP y mutación bit flip se obtienen n_1 hijos, una selección siguiente elige el hijo con el fitness más alto para insertarlo en la próxima generación.

Esta opción se ha analizado para un conjunto de funciones de testeo. Los resultados indican que esta opción mitiga la posible pérdidas de diversidad generada por la aplicación de múltiples crossovers sobre un par de padres. Consecuentemente la calidad de los resultados es al menos tan buena como las opciones previas más complejas. Adicionalmente, cuando se observa la población final se detecta que todos los individuos están mucho más centrados alrededor del óptimo. Es una característica importante cuando la aplicación necesita de múltiples soluciones alternativas cercanas al

```
procedure múltiple selección de padres
begin
  int mating_pool[cant_parents]  //vector para almacenar los padres seleccionados
  int children_pool[cant_cross] //vector para almacenar los hijos creados
  for 1 to pop_size
    seleccionar indiv-1 C(t)
    mating_pool[1] = indiv-1
    i=2
    while (i <= cant_parent)
      repeat
        seleccionar indiv-i C(t)
      until(is_relative(mating_pool, indiv-i)) //control de no existencia de
        ancestros comunes y unicidad de padres en el mating pool
      matting_pool[i] = indiv-i
      i=i+1
    end while
    recombinar usando MCMP y mutar individuos del mating_pool a
      children_pool
    seleccionar los mejores individuos de children_pool construyendo C'(t)
  end for
end procedure
```

Figura 5.7 Modificación del procedimiento EIP para incorporar MCMP

óptimo para hacer frente a sistemas dinámicos. También la velocidad de convergencia, medida en número de generaciones, se aumenta sin incrementar el riesgo de convergencia prematura.

Capítulo VI

MÉTODOS DE SOLUCIÓN PARA JOB SHOP SCHEDULING

6.1. HEURÍSTICAS CONVENCIONALES

El problema de job shop scheduling (JSSP) es un problema muy importante [69]; está entre los problemas de optimización combinatoria más duros, por lo tanto existen métodos de aproximación que producen soluciones aceptables empleando un tiempo razonable. Los procedimientos heurísticos para un problema de job shop se pueden clasificar en dos clases:

- ✓ *Heurísticas de una pasada*: construyen una única solución fijando una operación por vez en el schedule basándose en *reglas de prioridad de despacho*. Hay muchas reglas para elegir una operación de un subconjunto para ser asignada próximamente. Esta heurística es rápida.
- ✓ *Heurísticas de múltiples pasadas*: se construyen al repetir una heurística de una pasada para obtener schedules mejores incurriendo en algún costo extra de computación.

6.1.1. HEURÍSTICAS DE PRIORIDAD DE DESPACHO

Las reglas de prioridad son probablemente las reglas heurísticas aplicadas con mayor frecuencia para resolver problemas de scheduling, esto se debe a su fácil implementación y a su baja complejidad de tiempo. Los algoritmos de Giffler y Thompson se pueden considerar como la base de todas las heurísticas basadas en reglas de prioridad [133]. Giffler y Thompsom han propuesto dos algoritmos para generar schedules: procedimientos de generación de *schedules activos* y *schedules nondelay* [72]. Un schedule nondelay tiene la propiedad de que ninguna máquina permanece ociosa si hay un job disponible para su procesamiento. Un schedule activo tiene la propiedad de que ninguna operación se puede iniciar tempranamente sin retrasar otro

job. Los schedules activos forman un conjunto más grande que incluye como subconjunto a los schedules nondelay. El procedimiento de generación de Giffler y Thompson explora el espacio de búsqueda por medio de una estructura de árbol. Los nodos en el árbol representan los schedules parciales, los arcos representan las posibles elecciones, y las hojas del árbol son el conjunto de schedules. Para un schedule parcial, el algoritmo esencialmente identifica todos los conflictos de procesamiento, es decir, identifica operaciones que compiten por la misma máquina, y en cada etapa usa un procedimiento de enumeración para resolver esos conflictos bajo todas las formas posibles. En contraste, las heurísticas resuelven esos conflictos con reglas de prioridad de despacho; las heurísticas especifican una regla de prioridad para seleccionar una operación entre las operaciones en conflicto.

Los procedimientos de generación trabajan con un conjunto de operaciones planificables en cada etapa. Las operaciones planificables son aquellas que no se han asignado y que tienen predecesoras planificadas; este conjunto se puede determinar desde la estructura de precedencia. El número de etapas para un procedimiento de una pasada es igual al número de operaciones $m \times n$. En cada etapa, se selecciona una operación para agregar al *schedule parcial*. Los conflictos entre operaciones se resuelven por reglas de prioridad de despacho.

Para un schedule parcial activo dado, el posible tiempo de inicio σ_i se determina por el tiempo de finalización del antecesor directo de la operación i y el tiempo de finalización sobre la máquina requerida por la operación i. La mayor de esas dos cantidades es σ_i. El posible tiempo de finalización es simplemente $\sigma_i + t_i$, donde t_i es el tiempo de procesamiento de la operación i. Dada la siguiente notación:

- ✓ PS_t -un schedule parcial conteniendo t operaciones ya planificadas.
- ✓ S_t - el conjunto de operaciones planificables en la etapa t, correspondiente al PS_t dado.
- ✓ σ_i- el tiempo más temprano en el cual puede comenzar la operación $i \in S_t$.
- ✓ ϕ_i - el tiempo más temprano en el cual la operación $i \in S_t$ puede finalizar.

el procedimiento para generar un schedule activo consta de los siguientes pasos:

1. Sea $t = 0$ y comenzar con PS_t como el schedule parcial nulo. Inicialmente S_t incluye todas las operaciones sin antecesoras.

2. Determinar $\phi^*_t = \min_{i \in S_t} \{\phi_i\}$ y la máquina m^* sobre la cual se puede realizar ϕ^*_t.
3. Para cada operación $i \in S_t$ que necesita la máquina m^* y para la cual $\sigma_i < \phi^*_i$, calcular un índice de prioridad de acuerdo a la regla de prioridad. Hallar la operación con el índice más pequeño y agregarla a PS_t tan pronto como sea posible, creando así un nuevo schedule parcial PS_{t+1}.
4. Para PS_{t+1} actualizar los datos como sigue:
   1. Eliminar la operación *i* desde S_t.
   2. Formar S_{t+1} al adicionar el sucesor inmediato de la operación *j* a S_t.
   3. Incrementar *t* en uno.
5. Retornar al paso 2 hasta que se haya generado un schedule completo.

El problema que resta es identificar una regla de prioridad efectiva. A continuación se listan algunas de las reglas más comúnmente usadas en la práctica [69]:

- ✓ **SPT** (*shortest processing time*): seleccionar una operación con el tiempo de procesamiento más corto.
- ✓ **LPT** (*longest processing time*): seleccionar una operación con el tiempo de procesamiento más largo.
- ✓ **MWR** (*most work remaining*): seleccionar una operación para el job con el mayor tiempo de procesamiento pendiente.
- ✓ **LWR** (*least work remaining*): seleccionar una operación para el job con el menor tiempo de procesamiento pendiente.
- ✓ **MOR** (*most operations remaining*): seleccionar una operación para el job con el mayor número de operaciones pendientes.
- ✓ **LOR** (*least operations remaining*): seleccionar una operación para el job con el menor número de operaciones pendientes.
- ✓ **EDD** (*earliest due date*): seleccionar un job con la fecha de entrega más temprana.
- ✓ **FCSF** (*first come, first served*): seleccionar la primer operación en la cola de jobs para la misma máquina.
- ✓ **RND** (*random*): seleccionar una operación en forma aleatoria.

6.1.2. HEURÍSTICA DE DESPACHO ALEATORIO

Mientras las heurísticas de una pasada se limitan a la construcción de una única solución, las heurísticas de multipasada (también denominadas heurísticas de búsqueda) tratan de obtener muchas soluciones al generar varias de ellas, usualmente a expensas de mucho tiempo de computación. Técnicas tales como branch-and-bound y programación dinámica pueden garantizar una solución óptima, pero no son prácticas para problemas a gran escala.

Las heurísticas aleatorias son un intento para proveer soluciones más exactas [12]. La idea de despacho aleatorio es comenzar con una familia de reglas de despacho. Por cada selección de una operación para ejecutar, se elige la regla de despacho en forma aleatoria, esto se repite a lo largo de la generación del schedule. El proceso se reitera varias veces y se elige el mejor resultado. El procedimiento para generar un schedule activo tiene los siguientes pasos:

0. Sea *BS* el mejor schedule, iniciarlo como un schedule nulo.
1. Sea $t = 0$ y PS_t el schedule parcial nulo. Inicialmente S_t incluye todas las operaciones sin antecesoras.
2. Determinar $\phi^*_t = \min_{i \in St} \{\phi_i\}$ y la máquina m^* sobre la cual se realizará ϕ^*_t.
3. Seleccionar una regla de despacho aleatoriamente desde la familia de reglas. Para cada operación $i \in S_t$ que necesita la máquina m^* y para la cual $\sigma_i < \phi^*_i$, calcular un índice de prioridad de acuerdo a la regla de prioridad específica. Hallar la operación con el índice más pequeño y agregar esta operación a PS_t tan pronto como sea posible, creando así un nuevo schedule parcial PS_{t+1}.
4. Para PS_{t+1} actualizar los datos como sigue:
    1. Eliminar la operación *i* desde S_t.
    2. Formar S_{t+1} al adicionar el sucesor directo de la operación *j* a S_t.
    3. Incrementar *t* en uno.
5. Retornar al paso 2 hasta que se haya generado un schedule completo.
6. Si el schedule generado es los pasos anteriores es mejor que el mejor hallado hasta el momento, copiar éste en *BS*. Regresar al paso 1 hasta que la cantidad de iteraciones iguale las veces predeterminadas.

Varios investigadores han tratado de obtener mejoras a la opción aleatoria. Un cambio es introducir un proceso de aprendizaje de modo que las reglas de despacho más exitosas tengan más chances de ser seleccionadas en el futuro.

6.2. ALGORITMOS EVOLUTIVOS PARA EL PROBLEMA DE JOB SHOP SCHEDULING

6.2.1. REPRESENTACIÓN

Por la existencia de restricciones de precedencia de las operaciones, en JSSP no es tan fácil determinar una representación natural como en el problema del viajante (*traveling salesman problem* - TSP). No hay una buena representación que utilice un sistema de desigualdades para las restricciones de precedencia. Así, la opción con penalidades no es de fácil aplicación para manejar esos tipos de restricciones. Orvosh y Davis [112] han mostrado que para muchos problemas de optimización combinatoria, es relativamente fácil reparar un cromosoma ilegal o no factible, y la estrategia de reparación realmente supera estrategias tales como la de rechazo o la de penalización. La mayoría de los investigadores de JSSP y de algoritmos genéticos prefieren usar la estrategia de reparación para manejar la no factibilidad o ilegalidad. Una muy buena opción para la construcción de un algoritmo evolutivo para el problema de job shop scheduling es inventar una representación apropiada de soluciones junto con operadores genéticos específicos del problema, de manera tal que todos los cromosomas generados en la fase inicial o en el proceso evolutivo produzcan schedules factibles. Durante los últimos años se han propuesto varias representaciones para el problema de job shop scheduling, las cuales se pueden clasificar en:

- ✓ *Opciones directas*: un schedule (la solución al JSSP) se codifica en un cromosoma, y el algoritmo genético se usa para evolucionar esos cromosomas a fin de determinar un schedule mejor. Dentro de esta clase se encuentran las siguientes representaciones:
  - Representación basada en operaciones.
  - Representación basada en jobs.
  - Representación basada en relación de pares de jobs.
  - Representación basada en tiempo de finalización.

– Representación random key.

✓ *Opciones indirectas*: tal como la representación basada en reglas de prioridad, dentro del cromosoma se codifica una secuencia de reglas de despacho, los algoritmos genéticos se usan para determinar una mejor secuencia de reglas de despacho. Un schedule se construye con la secuencia de reglas de despacho. A esta clase pertenecen las siguientes representaciones:

– Representación basada en lista de preferencias.

– Representación basada en reglas de prioridad.

– Representación basada en grafos disjunctivas.

– Representación basada en máquinas.

6.2.1.1.REPRESENTACIONES DIRECTAS

6.2.1.1.1. REPRESENTACIÓN BASADA EN OPERACIONES

Esta representación codifica un schedule como una secuencia de operaciones, cada gen representa una operación. Hay dos posibles formas de nombrar cada operación. Una forma natural es usar números naturales para nombrar cada operación. Desafortunadamente, por la existencia de restricciones de precedencia, no todas las permutaciones de números naturales definen schedules factibles. Gen, Tsujimura, y Kubota propusieron una forma alternativa: nombrar a todas las operaciones de un mismo job con el mismo símbolo y entonces interpretarlas de acuerdo al orden de ocurrencia en la secuencia para un cromosoma dado [30, 92]. Para un problema de n jobs y m máquinas, un cromosoma contiene $n \times m$ genes. Cada job aparece en el cromosoma m veces, y cada repetición no indica una operación concreta de un job sino que se refiere a una operación la cuál es dependiente del contexto. Es fácil ver que cualquier permutación del cromosoma siempre produce un schedule factible.

Consideremos el problema de tres jobs y tres máquinas utilizado en [69] y presentado en la tabla 6.1. Supongamos el siguiente cromosoma [3 2 2 1 1 2 3 1 3], donde 1 indica el job j_1, 2 el job j_2, 3 el job j_3. Como cada job tiene tres operaciones, tiene exactamente tres ocurrencias en el cromosoma. Por ejemplo, en el cromosoma hay tres 2, lo cual representa las tres operaciones del job j_2:

✓ el primer 2 corresponde a la primer operación del job j_2 la cual se procesará sobre la máquina 1,

	Tiempo de Procesamiento				Secuencia de Máquinas		
	Operaciones				Operaciones		
Job	1	2	3	Job	1	2	3
j_1	3	3	2	j_1	m_1	m_2	m_3
j_2	1	5	3	j_2	m_1	m_3	m_2
j_3	3	2	3	j_3	m_2	m_1	m_3

Tabla 6.1 Ejemplo de un problema de tres jobs y tres máquinas.

- ✓ el segundo 2 corresponde a la segunda operación del job j_2 la cual se procesará sobre la máquina 3, y
- ✓ el tercer 2 corresponde a la tercer operación del job j_2 la cual se procesará sobre la máquina 2.

Como se puede observar, todas las operaciones para el job j_2 se denominan con el mismo símbolo 2 y se interpretan acorde al orden de ocurrencia en la secuencia del cromosoma. Las relaciones de ocurrencia correspondientes de las operaciones de jobs y máquinas de procesamiento se muestran en la figura 6.1.

Acorde a esas relaciones, podemos obtener la lista de máquinas [2 1 3 1 2 2 1 3 3], como se muestra en la figura 6.2. De esta figura se puede ver que el orden de procesamiento de jobs para la máquina 1 es 2 – 1 – 3 (indicado con el sombreado), para la máquina 2 es 3 – 1 – 2, y para máquina 3 es 2 – 1 – 3. En la figura 6.3, se muestra un schedule factible.

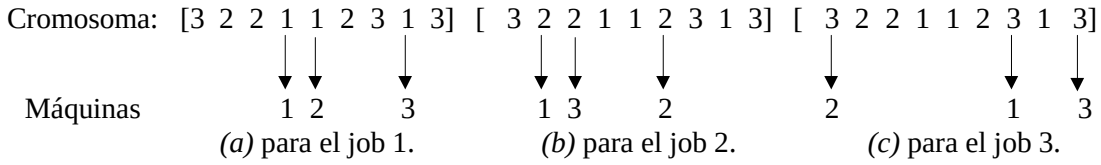


Figura 6.1 Operaciones de los jobs y correspondencia con las máquinas.

cromosoma	[3	2	2	1	1	2	3	1	3]
máquinas	2	1	3	1	2	2	1	3	3

Figura 6.2 El orden de procesamiento de los jobs sobre la máquina 1.

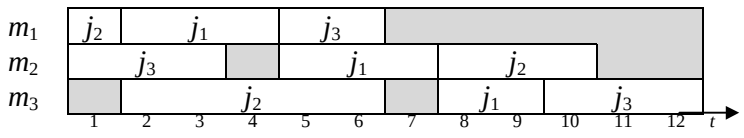
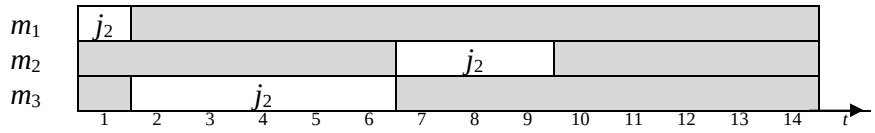


Figura 6.3 Un schedule factible.

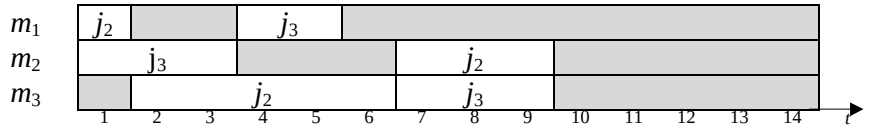
6.2.1.1.2. REPRESENTACIÓN BASADA EN JOBS

Esta representación consiste de una lista de *n* jobs y un schedule se construye en función de la secuencia de jobs. Para una secuencia de jobs dada, todas las operaciones del primer job se planifican primero, luego se consideran las operaciones del segundo job, y así sucesivamente. La primer operación del job en cuestión se asigna en el mejor tiempo de procesamiento disponible en la máquina que la operación necesite; luego se asigna la segunda operación, y así sucesivamente, hasta que se asignan todas las operaciones del job. El proceso se repite con cada uno de los jobs en la lista considerada en la secuencia apropiada. Toda permutación de jobs corresponde a schedules factibles.

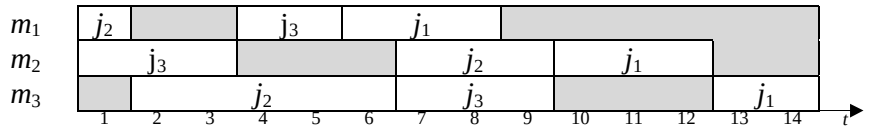
Usando el mismo ejemplo del problema de tres jobs y tres máquinas enunciado en la tabla 6.1, se supone el cromosoma [2 3 1]. El primer job a ser procesado es el job j_2. Las restricciones de precedencia de operaciones para j_2 son [m_1, m_3, m_2] y el tiempo de procesamiento correspondiente a cada máquina es [1 5 3]. El job j_2 se asigna como se muestra en la figura 6.4(a). Luego se procesa al job j_3, su precedencia de operaciones en las máquinas es [m_2, m_1, m_3] y los tiempos de procesamiento correspondientes en cada máquina son [3 2 3]. Cada una de sus operaciones se planifican en el mejor tiempo de procesamiento disponible en las máquinas que las operaciones necesitan, como se muestra en la figura 6.4(b). Finalmente, se planifica el job j_1 como se muestra en la figura 6.4(c).



(a) para el primer job j_2.



(b) para el segundo job j_3..



(c) para el último job j_1.

Figura 6.4 Deducir un schedule desde una codificación basada en jobs.

Precedencia de Operaciones				Schedule Factible			
job	Secuencia de Máquinas			máq.	Secuencia de Jobs		
j_1	m_1	m_2	m_3	m_1	j_2	j_1	j_3
j_2	m_1	m_3	m_2	m_2	j_3	j_1	j_2
j_3	m_2	m_1	m_3	m_3	j_2	j_1	j_3

Tabla 6.2 Ejemplo de un problema de tres jobs y tres máquinas.

6.2.1.1.3. REPRESENTACIÓN BASADA EN LA RELACIÓN ENTRE PARES DE JOBS.

Nakano y Yamada usan una matriz binaria para codificar un schedule, la cual se determina sobre la base de la relación de precedencia entre pares de jobs sobre las máquinas correspondientes [108].

Considerando el problema de scheduling de tres máquinas y tres jobs (tabla 6.1), en la tabla 6.2 se muestran las restricciones de precedencia de las operaciones de los jobs y un schedule factible para el problema.

Se define una variable binaria para indicar la relación de precedencia entre un par de jobs:

$$x_{ijm} = \begin{cases} 1, & \text{si el job } i \text{ es procesado antes al job } j \text{ sobre la máquina } m \\ 0, & \text{en cualquier otro caso.} \end{cases}$$

Examinando la relación de precedencia para el par de jobs (j_1 , j_2) sobre las máquinas (m_1, m_2, m_3), de acuerdo al schedule dado, se tiene (x_{121} x_{122} x_{123}) = (0 1 0). Para el par (j_1 , j_3) se tiene (x_{131} x_{132} x_{133}) = (1 0 1). Para un par de jobs (j_2 , j_3) la relación de precedencia sobre las máquinas (m_1, m_3, m_2) son (x_{231} x_{233} x_{232}) = (1 0 1). El orden de la variable x_{ijm} para un par de jobs deberá guardar consistencia con el orden de las operaciones del primer job *i*. Por ejemplo, para el par de jobs (j_2 , j_3), la secuencia de operaciones del job j_2 es (m_1, m_3, m_2), por lo tanto la secuencia de las variables relativas son (x_{231} x_{232} x_{233}). Para resumir los resultados, se muestra la matriz binaria para el schedule factible:

$$\begin{matrix} (j_1, j_2) \text{ sobre } (m_1, m_2, m_3) \\ (j_1, j_3) \text{ sobre } (m_1, m_2, m_3) : \\ (j_2, j_3) \text{ sobre } (m_1, m_3, m_2) \end{matrix} \begin{pmatrix} x_{121} & x_{122} & x_{123} \\ x_{131} & x_{132} & x_{133} \\ x_{231} & x_{233} & x_{232} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Esta representación es tal vez la más compleja. Es altamente redundante [53]. Además de su complejidad, los cromosomas producidos por procedimientos de inicialización o por operadores genéticos son, en general, ilegales.

6.2.1.1.4. REPRESENTACIÓN BASADA EN TIEMPOS DE FINALIZACIÓN

Yamada y Nakano propusieron la representación basada en tiempo de finalización [148]. Un cromosoma es una lista ordenada de tiempos de finalización de las operaciones. Para el mismo ejemplo dado en la tabla 6.1, el cromosoma se puede representar como:

$$[c_{111}\, c_{122}\, c_{133}\, c_{211}\, c_{223}\, c_{232}\, c_{312}\, c_{312}\, c_{333}\,]$$

donde c_{jir} indica el tiempo de finalización para la operación *i* del job *j* sobre la máquina *r*. Tal representación no es adecuada para la mayoría de los operadores genéticos porque se producirán schedules ilegales. Yamada y Nakano diseñaron un operador de crossover especial para esta representación.

6.2.1.1.5. REPRESENTATION RANDOM KEY

La representacion *random key* fue introducida por Bean [14]. Con esta técnica, las operaciones genéticas pueden producir hijos factibles sin producir overhead adicional en una amplia variedad de problemas de optimización y de secuenciamiento. Norman y Bean generalizaron con éxito la opción para el problema de job shop scheduling [109, 110].

Esta representación codifica una solución con un *número random*. Esos valores se utilizan como *claves* de ordenamiento para decodificar la solución. Para un problema de scheduling de *n* jobs y *m* máquinas, cada gen consiste de dos partes: un entero en el conjunto {1, 2, . . ., *m*} y una fracción generada aleatoriamente en el rango (0,1). La parte entera de cualquier clave random se interpreta como la asignación de máquina para ese job. El ordenamiento de las partes fraccionarias provee la secuencia de jobs en cada máquina. Consideremos el ejemplo dado en la tabla 6.1. Supongamos que el cromosoma es:

[1.34 1.09 1.88 2.66 2.91 2.01 3.23 3.21 3. 44]

Al ordenar las claves (dentro de cada máquina) en orden ascendente se obtienen las siguentes secuencias de jobs en cada máquina:

- ✓ máquina 1: secuencia de jobs $2 \rightarrow 1 \rightarrow 3$,
- ✓ máquina 2: secuencia de jobs $3 \rightarrow 1 \rightarrow 2$,
- ✓ máquina 3: secuencia de jobs $2 \rightarrow 1 \rightarrow 3$.

El cromosoma se puede trasladar a una única lista de operaciones ordenados como sigue:

$$[\, o_{21}\ o_{11}\ o_{31}\ o_{32}\ o_{12}\ o_{22}\ o_{23}\ o_{13}\ o_{33}]$$

donde o_{jm} indica el job j sobre la máquina m. La secuencias de jobs dados puede violar las restricciones de precedencia.

6.2.1.2. REPRESENTACIONES INDIRECTAS

6.2.1.2.1 REPRESENTACIÓN BASADA EN LISTAS DE PREFERENCIA

Esta representación fue propuesta en primer término por Davis para una clase de problema de scheduling [26]. Falkenauer y Bouffoix usaron esta representación para tratar un problema de job shop scheduling con tiempos de vencimiento [53]. Croce, Tadei y Volta aplicaron esta representación al problema de scheduling clásico [23].

Para un problema de job shop scheduling de n jobs y m máquinas, un cromosoma consiste de m subcromosomas, uno por máquina. Cada subcromosoma es un string de símbolos de longitud n, cada símbolo identifica una operación a ser procesada sobre la máquina correspondiente. Los subcromosomas no describen una secuencia de operaciones sobre la máquina sino que son *listas de preferencia*, cada máquina tiene su propia lista de preferencia. El schedule actual se deduce del cromosoma a través de una simulación, la cual analiza el estado de la cola de espera frente a cada máquina y, si es necesario, usa la lista de preferencia para determinar el schedule; es decir, se elige aquella operación que aparece primero en la lista de preferencia.

En el siguiente ejemplo se muestra cómo deducir un schedule desde un cromosoma tomando como base el ejemplo presentado en la tabla 6.1. Se supone que el cromosoma es [(2 3 1) (1 3 2) (2 1 3)]. El primer gen (2 3 1) es la lista de preferencia para la máquina m_1, (1 3 2) es la lista para la máquina m_2 y (2 1 3) para la máquina m_3. Desde esas listas de preferencia se puede ver que las primeras operaciones preferenciales son: job j_2 sobre la máquina m_1, j_1 sobre m_2, j_2 sobre m_3. Acorde a las restricciones de preferencia de las operaciones sólo se puede planificar j_2 sobre m_1, de este modo el primer schedule es el que se muestra en la figura 6.5(a). La próxima operación planificable es j_2 sobre la máquina m_3 como se muestra en la figura 6.5(b). En este punto, las operaciones preferenciales son j_3 sobre la máquina m_1 y j_1 sobre m_2 y m_3. Como no son planificables en este momento, se toma la segunda operación preferencial

en cada lista: j_1 sobre m_1 y j_3 sobre m_2.y m_3. Las operaciones a planificar son j_1 sobre m_1 y j_3 sobre m_2 y Se asignan como se muestra en la figura 6.5(c). Las próximas operaciones planificables son j_3 sobre m_1 y j_1 sobre m_2, como lo muestra la figura 6.5(d). Luego, las operaciones planificables son j_2 sobre m_2 y j_1 sobre m_3 como se muestra en la figura 6.5(e). La última operación planificable es j_3 sobre m_3 (figura 6.5(f)). Ahora se completa la deducción de un schedule desde el cromosoma y se obtiene un schedule factible con un makespan de 12 unidades de tiempo. Con el procedimiento de decodificación, todos los posibles cromosomas producen un schedule factible.

6.2.1.2.2 REPRESENTACIÓN BASADA EN REGLAS DE PRIORIDAD

Dorndorg y Pesch propusieron un algoritmo evolutivo basado en reglas de prioridad [32], un cromosoma se codifica como una secuencia de reglas de despacho para una asignación de jobs. Un schedule se construye con heurísticas de despacho de prioridad en base a la secuencia de reglas de despacho. Los algoritmos evolutivos se usan para evolucionar esos cromosomas que ayudarán a producir una mejor secuencia de reglas de despacho.

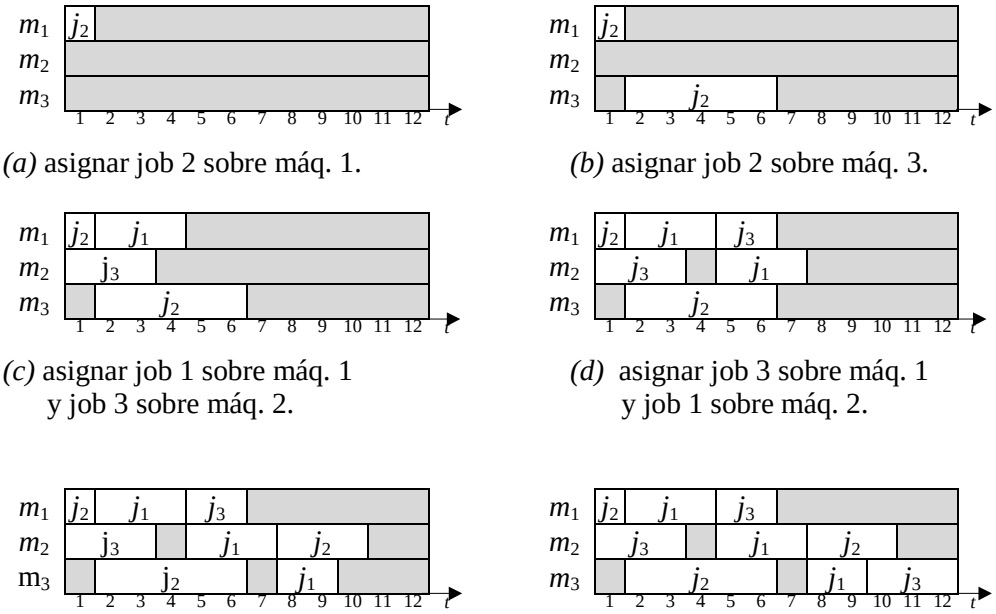


Figura 6.5 Deducir un schedule desde una codificación basada en listas de preferencias.

Las reglas de prioridad de despacho son las heurísticas que se usan con mayor frecuencia para resolver problemas de scheduling en la práctica, debido a su fácil implementación y a su bajo tiempo de complejidad. Los algoritmos de Giffler y Thompson se pueden considerar como la base para todas las heurísticas de reglas de prioridad [72, 133]. El problema es identificar una regla de prioridad efectiva.

Para un problema de n jobs y m máquinas, un cromosoma es un string de $n \times m$ entradas ($p_1, p_2, \ldots, p_{nm}$). Una entrada p_i representa una regla de un conjunto de reglas de prioridades de despacho. La entrada en la posición i-ésima indica que un conflicto en la iteración i-ésima del algoritmo de Giffler y Thompson se resolverá usando la regla de prioridad p_i. Una operación desde el conjunto de conflicto se selecciona por la regla p_i; en caso de existir más de una operación dentro de ese conjunto de conflicto, la decisión se hace en forma aleatoria.

Dada la siguiente notación:

- ✓ PS_t - un schedule parcial conteniendo t operaciones asignadas.
- ✓ S_t - el conjunto de operaciones asignables en la etapa t, correspondiente al PS_t.
- ✓ σ_i - el tiempo más temprano en el cual se deberá iniciar la operación $i \in S_t$.
- ✓ ϕ_i -el tiempo más temprano en el cual se deberá completar la operación $i \in S_t$
- ✓ C_t-el conjunto de operaciones en conflicto en la iteración t

el procedimiento para deducir un schedule desde un cromosoma dado ($p_1, p_2, \ldots, p_{nm}$) es el siguiente:

1. Sea $t = 1$ y comenzar con PS_t como el schedule parcial nulo. Inicialmente S_t incluye todas las operaciones sin predecesoras.
2. Determinar $\phi^*_t = \min_{i \in S_t} \{\phi_i\}$ y la máquina m^* sobre la cual se realizará ϕ^*_t. Si existe más de una máquina, la selección se determina en forma aleatoria.
3. Formar el conjunto C_t el cual incluye todas las operaciones $i \in S_t$ que necesita la máquina m^*. Seleccionar una operación de C_t usando la regla de prioridad p_t y adicionar esta operación a PS_t tan pronto como sea posible, creando así un nuevo schedule parcial PS_{t+1}. Si existe más de una operación acorde a la prioridad p_t, entonces esta situación se resuelve en forma aleatoria.
4. Actualizar PS_{t+1} al sacar la operación seleccionada desde S_t y agregar el sucesor directo de la operación j a S_t. Incrementar t en uno.
5. Retornar al paso 2 hasta que se haya generado un schedule completo.

Ejemplo. Dado [69]: las reglas SPT, LPT, MWR, LWR, con identificadores 1, 2, 3 y 4 respectivamente, el cromosoma [1 2 2 1 4 4 2 1 3] e indicando las operaciones como o_{jom} (donde el índice *jom* se refiere a job, operación, máquina). En el paso inicial (t = 1), se tiene:

$S_1 = \{ o_{111} , o_{211} , o_{312} \}$

$\phi_1^* = \min \{3, 1, 3\} = 1$

$m^* = 1$

$C_1 = \{ o_{111} , o_{211} \}$

Las operaciones o_{111} y o_{211} compiten por m_1. Como el primer gen del cromosoma es 1 (el cual significa la regla de prioridad SPT), la operación 211 es planificada en m_1, como se muestra en la figura 6.6(a). Después de actualizar tenemos (t =2),

$S_2 = \{ o_{111} , o_{223} , o_{312} \}$

$\phi_2^* = \min \{4, 6, 3\} = 3$

$m^* = 2$

$C_2 = \{ o_{312} \}$

Se planifica la operación o_{312} en m_2 como se muestra en la figura 6.6(b). Después de actualizar tenemos (en t =3),

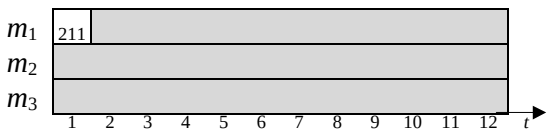
$S_3 = \{ o_{111} , o_{223} , o_{321} \}$

$\phi_3^* = \min \{4, 6, 5\} = 5$

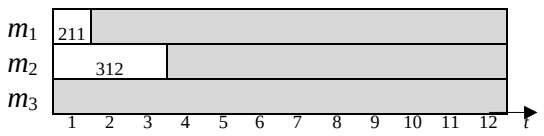
$m^* = 1$

$C_3 = \{ o_{111} , o_{321} \}$

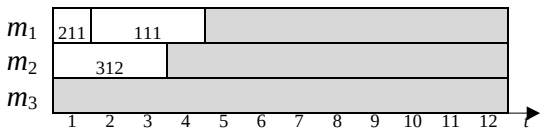
Las operaciones o_{111} y o_{321} compiten por m_1. Como el tercer gen del cromosoma es 2 (el cual representa a la regla de prioridad LPT), la operación o_{111} gana y se la planifica en m_1, como se muestra en la figura 6.6(c). Estos pasos se repiten hasta que se forma el schedule desde el cromosoma (figura 6.6(d)).



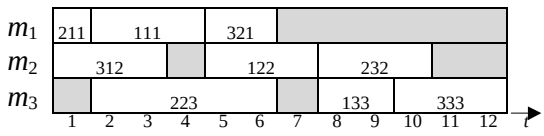
(a) planificación de la operación 211 en m_1.



(b) planificación de la operación 312 en m_1.



(c) planificación de la operación 211 en m_1.



(d) schedule final.

Figura 6.6 Deduciendo un schedule desde una codificación basada en reglas de prioridad.

6.2.1.2.3 REPRESENTACIÓN BASADA EN GRAFOS DISJUNCTIVOS.

Tamaki y Nishikawa proponen una representación basada en grafos disjunctivos [135], la cual se puede considerar como una clase de la representación basada en pares de jobs. El problema de job shop scheduling se puede representar con un grafo disjunctivo [13, 121]. El grafo disjunctivo $G = (N, A, E)$ se define como:

- ✓ N: contiene los nodos representado todas las operaciones.
- ✓ A: contiene los arcos que conectan operaciones consecutivas del mismo job.
- ✓ E: contiene los arcos disjuntos que conectan las operaciones a ser procesadas por la misma máquina.

Las restricciones disjunctivas se representan por un eje en E. Se puede establecer un arco disjunctivo por sus dos posibles orientaciones. La construcción de un schedule establece las orientaciones de todos los arcos disjunctivos así como también determina la secuencia de operaciones sobre la misma máquina. Una vez que se determina una secuencia para una máquina se reemplazan los arcos disjunctivos por arcos conjunctivos normales. La figura 6.7, muestra el grafo disjunctivo para el ejemplo de tres jobs y tres máquinas introducido en la tabla 6.1. Un cromosoma consiste de un string binario correspondiente a una lista de orden de arcos disjunctivos en E como se muestra en la figura 6.8, donde e_{ij} indica el arco disjunctivo entre los nodos i y j y se define como sigue:

$$e_{ij} = \begin{cases} 1, & \text{establecer la orientación del arco disjunctive desde el nodo } j \text{ al nodo } i \\ 0, & \text{establecer la orientación del arco disjunctive desde el nodo } i \text{ al nodo } j \end{cases}$$

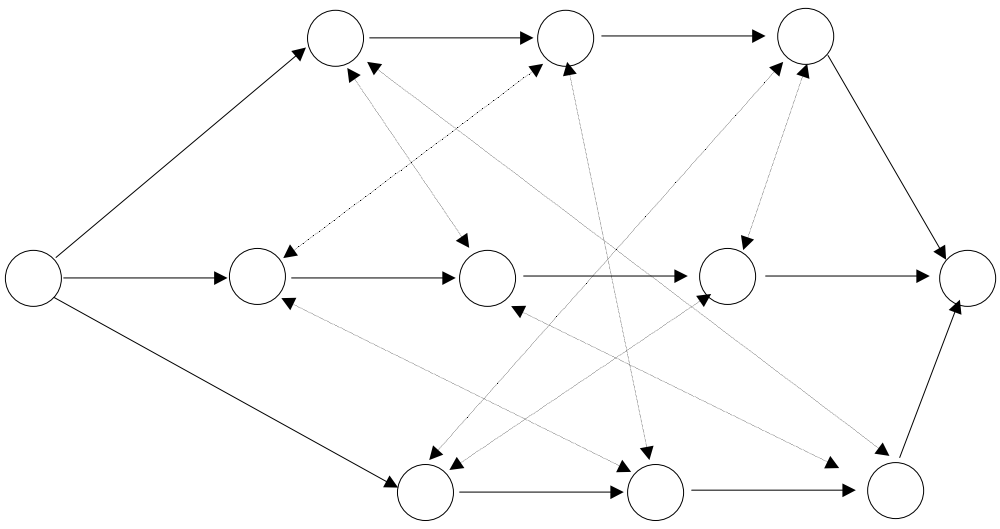


Figura 6.7 Grafo disjunctivo para el problema de tres máquinas y tres jobs.

Lista de orden de arcos disjunctivos	e_{15}	e_{19}	e_{59}	e_{24}	e_{28}	e_{48}	e_{36}	e_{37}	e_{67}
cromosoma	0	0	1	1	0	0	0	1	1

Figura 6.8 Representación basada en grafos disjunctivos.

El problema de job shop es encontrar un orden en las operaciones sobre cada máquina, es decir, fijar la orientación de los arcos disjunctivos tal que el grafo resultante no contenga ciclos para garantizar la no existencia de conflictos de precedencia entre las operaciones. Un cromosoma arbitrario puede provocar un grafo cíclico, lo cual significa que el schedule no es factible. Durante el proceso de deducción, cuando ocurre un conflicto de dos nodos (operaciones) sobre una máquina, se usa el bit correspondiente del cromosoma para fijar el orden de procesamiento de dos operaciones, es decir, fijar la orientación del arco disjunctive entre dos nodos.

6.2.1.2.4 REPRESENTACIÓN BASADA EN MÁQUINAS

Dornforf y Pesch propusieron un algoritmo evolutivo basado en máquinas [32] donde un cromosoma se codifica como una secuencia de máquinas y un schedule se construye con una heurística basada en corrimientos del cuello de botella sobre la secuencia.

La heurística de corrimiento del cuello de botella, propuesto por Adams, Balas y Zawack [1], es probablemente el procedimiento más poderoso entre todas las heurísticas para el problema de job shop scheduling. Las máquinas se secuencian una a una, sucesivamente, tomando aquella máquina identificada como el cuello de botella entre las máquinas aún no secuenciadas. Cuando se secuencia una nueva máquina, todas las secuencias establecidas previamente se reoptimizan localmente. Tanto los procedimientos de identificación del cuello de botella y de reoptimización local se basan en la resolución repetida de un cierto problema de scheduling de una única máquina que es una simplificación del problema original. La principal contribución de esta opción es la forma en que se usa esta simplificación para decidir respecto del orden en el cual se secuencian las máquinas.

Las heurísticas de corrimiento del cuello de botella se basan en la idea de dar prioridad a las máquinas con cuellos de botella. La calidad de los schedules obtenidos

por la heurística depende de la secuencia de máquinas identificadas como cuello de botella. Adams, Balas y Zawack también propusieron una versión enumerativa de la heurística para considerar diferentes secuencias de máquinas.

En lugar de una búsqueda enumerativa en un árbol, Dorndorf y Pesch propusieron una estrategia genética para determinar la mejor secuencia de máquinas para la heurística de corrimiento de cuello de botella. Un cromosoma es una lista de máquinas ordenadas. Los algoritmos genéticos evolucionan esos cromosomas que hallarán una mejor secuencia de máquinas que la heurística. La diferencia entre la heurística de corrimiento de cuello de botella y el algoritmo evolutivo es que no se usan más los cuellos de botella como criterio de decisión para elegir la próxima máquina, la cual se controla por el cromosoma dado.

Sea M_0 el conjunto de máquinas secuenciadas y $[m_1, m_2, \ldots, m_m]$ el cromosoma. El procedimiento para deducir un schedule desde el cromosoma es el siguiente:

1. Sea $M_0 \leftarrow \{\phi\}$ y $i \leftarrow 1$ y $[m_1, m_2, \ldots, m_m]$ el cromosoma.
2. Secuenciar óptimamente la máquina m_i. Actualizar el conjunto $M_0 \leftarrow M_0 \cup \{ m_i \}$.
3. Reoptimizar la secuencia de cada máquina crítica $m_i \in M_0$, mientras se toma las otras secuencias fijas.
4. $i \leftarrow i + 1$. Entonces si $i > m$, parar; en caso contrario ir al paso 2.

Capítulo VII

ALGORITMOS EVOLUTIVOS AVANZADOS PARA JOB SHOP SCHEDULING

7.1. INTRODUCCIÓN

En este capítulo se presentan tres distintas implementaciones planteadas para resolver el problema de job shop scheduling usando algoritmos evolutivos. Principalmente, esas distintas opciones difieren en cuanto el tipo de representación de cromosoma utilizado; esto trae como consecuencia la selección de los operadores genéticos factibles de aplicar. En la primer propuesta, se introduce un algoritmo evolutivo con características de multirecombinación y prevención de incesto para la obtención de resultados para esta clase de problema. Otra de las soluciones analiza el comportamiento de una implementación con representación basada en reglas de prioridad en la cual se incorporan semillas generadas por heurísticas a la población inicial. Finalmente, se presenta un algoritmo evolutivo con representación basada en operaciones usando el método de crossover de *parcial exchange*.

7.2. OPCIÓN CON MULTIPLICIDAD Y PREVENCIÓN DE INCESTO

Una solución posible al problema del job shop scheduling es usar conjuntamente, en un algoritmo evolutivo, multiplicidad y prevención de incesto [123].

En un problema de job shop scheduling, una representación basada en jobs consiste de una lista de jobs, donde el schedule se construye acorde a esa secuencia de jobs. En esta representación se consideran permutaciones de jobs, por lo tanto se deben usar operadores genéticos adecuados para una permutación, tal como *partially-mapped crossover*, *order crossover* y *cycle crossover*. Otra forma de encarar el problema cuando hay permutaciones es usando *decodificadores*. Aquí un cromosoma se

Cromosoma	Permutación
1 1 1 1 1 1	1 2 3 4 5 6
4 3 4 1 1 1	4 3 6 1 2 5
4 2 3 2 1 1	4 2 5 3 1 6

Tabla 7.1 Correspondencia entre cromosomas y permutaciones.

representa como una lista de *n* enteros; el *i*-ésimo componente del vector es un número entero en el rango [1 .. (*n* – *i* + 1)]. La idea detrás de esta representación es la siguiente: hay una lista ordenada, la cual sirve como punto de referencia para decodificar el vector al seleccionar el ítem apropiado desde la lista actual. Por ejemplo, para una lista de ítems L = (1,2,3,4,5,6), en la tabla 7.1 se observa la correspondencia entre cromosomas y permutaciones.

Un decodificador es una transformación desde el espacio de representaciones a una zona del espacio de soluciones factible, el cual incluye transformaciones entre representaciones que son sometidas al proceso de evolución y representaciones que constituyen la entrada a la función de evaluación. Esto simplifica la implementación y produce hijos factibles bajo ciertos métodos de crossover convencionales, evitando el uso de penalidades o acciones de reparación.

El método de multiplicidad usado en esta opción es MCMP (*múltiples crossovers sobre múltiples padres*), permite la múltiple recombinación de múltiples padres bajo el uso de algunas de las variantes de SX, esperando un adecuado balance entre la explotación y la exploración del espacio del problema.

En cuanto al método de prevención de incesto utilizado es EIP (*prevención de incesto extendida*) en donde se previene el incesto entre individuos que pertenecen a la misma familia. Este método combinado con multiplicidad genera un algoritmo al que denominaremos MCMPIP (MCMP *Incest Prevention*).

En la figura 7.1 se muestra pseudocódigo de un procedimiento que tiene en cuenta multiplicidad (cuando se usa un número de padres $n_2 > 2$) y prevención de incesto entre miembros de una misma generación o una generación consecutiva, es decir entre hermanos ó entre padres e hijos.

7.2.1. DESCRIPCIÓN DEL EXPERIMENTO

Tanto MCMP como MCMPIP fueron contrastados para un conjunto de instancias seleccionadas para el JSSP, con el uso de una representación con decodificadores. Para

```
procedure multiple  parent, multiple crossover, incest prevention
    begin
     int mating_pool[number_of_parents]
        //arreglo para almacenar los padres seleccionado
     int children_pool[number_of_cross]
      //arreglo para almacenar los hijos creados
     for 1 to pop_size
      select indiv-1 C(t)
      mating_pool[1] = indiv-1
      i = 2
      while (i <= nro_de_padres)
        repeat
           select indiv-i C(t)
        until (no_es_pariente(mating_pool, indiv-i))
        //control de ancestros no comunes y de unicidad de los padres en
        //mating_pool
        matting_pool[i] = indiv-i
        i = i + 1
      end while
      recombinar usando MCMP y mutar
      asignar individuos desde mating_pool a children_pool
      selecccionar el mejor individuo de children_pool
      construir C'(t)
     end for
    end procedure
```

Figura 7.1 Procedimiento que une MCMP y EIP

cada instancia, se realizaron 15 corridas diferentes. Los experimentos corresponden a diferentes combinaciones de cantidades de crossovers n_1 y cantidades de padres n_2, diferentes tipos de SX y métodos de selección de los hijos generados. En los algoritmos evolutivos se usó selección proporcional al fitness del individuo para la selección de los individuos que conformarán el matting pool. Los mejores individuos se retuvieron por elitismo. En ambos métodos se fijó los mismos parámetros a excepción del tamaño de la población. En los experimentos se consideró un tamaño de la población de 50

individuos para MCMP y de 120 individuos para MCMPIP. Esta diferencia en el tamaño de la población se basa en el hecho de que en MCMPIP, cada vez que se elige un padre para adicionar al *matting_pool*, se debe controlar que ya no sea parte del mismo y que además no mantenga alguna relación ancestral con los individuos ya incorporados al matting pool, por lo tanto para poder llevar a cabo este control es necesario contar con mayor cantidad de individuos para no caer en un bucle infinito, debido a que ningún individuo cumple con las restricciones impuestas. Se implementaron tres métodos de scanning crossovers y para la inserción en la próxima generación se optó por $n_3 = 1$; se contrastaron dos métodos de selección de un hijo entre los n_1 generados en cada cruzamiento: elegir el mejor hijo y realizar la elección en forma aleatoria. El número de crossovers y padres se definió en $1 \leq n_1 \leq 4$ y $3 \leq n_2 \leq 5$, respectivamente. Se usó la mutación *big-creep* como operador de mutación, la cual consiste en reemplazar el valor de un gen por otro dentro del rango permitido. El número máximo de generaciones se determinó en 500 y las probabilidades de crossover y mutación se fijaron en 0.8 y 0.01, respectivamente. Estos valores se precisaron como la mejor combinación de probabilidades luego de muchos ensayos de prueba y error. Se usaron 6 instancias [97] con óptimos bien conocidos (tabla 7.2).

Las siguientes variables de performance fueron elegidas para contrastar los resultados encontrados por cada algoritmo:

- ✓ ***ebest*** = (Abs(*opt_val* - best value)/*opt_val*)100

  Error porcentual de los mejores individuos hallados cuando se comparan con el valor óptimo de makespan conocido o estimado, *opt_val*. Da una medida de cuán lejano está el fitness del mejor individuo al compararlo con *opt_val*.
- ✓ ***epop*** = (Abs(*opt_val*- pop mean fitness)/*opt_val*)100

  Error porcentual del fitness medio de la población cuando se compara con *opt_val*. Da una medida de cuán lejos está el fitness medio de *opt_val*.

7.2.2. RESULTADOS

Para todas las instancias y en todas las combinaciones posibles de (n_1 ,n_2), cuando se debe decidir por seleccionar un hijo en forma aleatoria o retener el mejor generado de los n_3 generados, la mejor perfomance de los algoritmos se logra al escoger el mejor

Instancia	Tamaño	Óptimo
la06	15×5	926
la12	20×5	1039
la15	20×5	1207
la16	10×10	945
abz6	10×10	943
abz7	20×15	657

Tabla 7.2 Instancias.

hijo obtenido de los n_1 generados en cada cruzamiento. Por lo tanto los resultados que se muestran de aquí en adelante son los obtenidos bajo este criterio de selección.

En las tablas 7.3, 7.4 y 7.5 se detallan los resultados para la instancia *la06*: los valores correspondientes al makespan de los mejores individuos hallados, al *ebest* medio y al *epop* medio, respectivamente. En la tabla 7.3 se puede observar que, para esta instancia, tanto MCMP y MCMPIP hallan el óptimo bajo algún método de SX en el 70% de los experimentos realizados. No se alcanza el óptimo en el caso en el cual se aplica un único crossover. En general, los mejores resultados se obtienen cuando se incrementa el número n_1 de crossovers, independientemente del número n_2 de padres usados en el cruzamiento. En la tabla 7.4 se puede observar que los valores de *ebest*

Mejores Valores de Makespan						
n_2/n_1	MCMP			MCMPIP		
	USX	FBSX	OBSX	USX	FBSX	OBSX
3/1	933	929	930	934	932	934
3/2	926	926	926	926	926	926
3/3	926	926	926	926	926	926
3/4	926	926	934	926	926	926
4/1	928	936	926	934	930	930
4/2	926	926	929	926	926	926
4/3	926	926	926	926	926	926
4/4	926	926	926	926	926	929
5/1	936	930	926	935	934	930
5/2	930	926	931	926	926	926
5/3	926	926	926	926	926	928
5/4	926	926	926	926	926	926
Min	926	926	926	926	926	926
Aver.	927.9	927.4	927.7	928.1	927.5	927.7
Max	936	936	934	935	934	934

Tabla 7.3 Valores de makespan de los mejores individuos hallados bajo cada método para diferentes combinaciones de (n_1, n_2) para la instancia *la06*.

ebest						
n_2/n_1	MCMP			MCMPIP		
	USX	FBSX	OBSX	USX	FBSX	OBSX
3/1	0.76	0.32	0.43	0.86	0.65	0.86
3/2	0.00	0.00	0.00	0.00	0.00	0.00
3/3	0.00	0.00	0.00	0.00	0.00	0.00
3/4	0.00	0.00	0.86	0.00	0.00	0.00
4/1	0.22	1.08	0.00	0.86	0.43	0.43
4/2	0.00	0.00	0.32	0.00	0.00	0.00
4/3	0.00	0.00	0.00	0.00	0.00	0.00
4/4	0.00	0.00	0.00	0.00	0.00	0.32
5/1	1.08	0.43	0.00	0.9719	0.8639	0.43
5/2	0.43	0.00	0.54	0.00	0.00	0.00
5/3	0.00	0.00	0.00	0.00	0.00	0.22
5/4	0.00	0.00	0.00	0.00	0.00	0.00
Min	0.00	0.00	0.00	0.00	0.00	0.00
Aver.	0.21	0.15	0.18	0.22	0.16	0.19
Max	1.08	1.08	0.86	0.97	0.86	0.86

Tabla 7.4 Valores de *ebest* hallados bajo cada método para diferentes combinaciones de (n_1, n_2) para la instancia *la06*.

epop						
n_2/n_1	MCMP			MCMPIP		
	USX	FBSX	OBSX	USX	FBSX	OBSX
3/1	0.13	0.14	0.13	0.14	0.14	0.14
3/2	0.12	0.12	0.12	0.13	0.13	0.12
3/3	0.12	0.11	0.11	0.11	0.12	0.11
3/4	0.11	0.11	0.11	0.11	0.11	0.11
4/1	0.14	0.14	0.13	0.14	0.14	0.13
4/2	0.12	0.12	0.12	0.13	0.12	0.12
4/3	0.12	0.12	0.11	0.12	0.12	0.11
4/4	0.11	0.11	0.11	0.11	0.11	0.11
5/1	0.13	0.14	0.13	0.14	0.14	0.13
5/2	0.12	0.12	0.12	0.12	0.13	0.12
5/3	0.12	0.12	0.11	0.12	0.12	0.11
5/4	0.11	0.11	0.11	0.11	0.11	0.11
Min	0.11	0.11	0.11	0.11	0.11	0.11
Aver.	0.12	0.12	0.12	0.12	0.12	0.12
Max	0.14	0.14	0.13	0.14	0.14	0.14

Tabla 7.5 Valores de *epop* hallados bajo cada método para diferentes combinaciones de (n_1, n_2) para la instancia *la06*.

varían entre 0 y 1.08, la mejor performance es para MCMP con FBSX seguido por MCMPIP con el mismo método de crossover. El peor comportamiento de ambos algoritmos se obtiene bajo el método USX. Los valores de *epop* están dentro del rango 1.48 a 28.31. Los valores mínimos de *epop* varían entre 1.48 a 3.84 y se hallan con $n_1 = 4$ y $n_2 = 5$, esto significa que la población entera está más concentrada alrededor del mejor individuo hallado cuando el n_1 y n_2 son grandes.

En las figuras 7.2 a 7.5 se resume la información sobre las instancias *la06*, *la12*, *la15* y *abz6*. Las abreviaturas M-U, M-F y M-O indican la combinación MCMP con USX, FBSX y OBSX, respectivamente; mientras que MI-U, MI-F y MI-O indican MCMPIP con USX, FBSX y OBSX, respectivamente.

De la observación de las figuras se ve que a medida que las instancias se tornan más complejas se observa una pérdida de performance sobre ambos métodos. Este hecho refleja que a medida que la complejidad del espacio de búsqueda se incrementa es más difícil para ambos métodos hallar schedules cercanos al óptimo. Esto es particularmente cierto cuando se incrementa el número de máquinas, por ejemplo en las instancias *la16*, *abz6* y *abz7*. Este efecto se puede esperar porque el espacio codificado (permutaciones de jobs) corresponde a sólo una parte del espacio de soluciones total y es el precio a ser pagado por producir hijos que sean factibles. A pesar de esto, la performance es mejor que la de usar algoritmos evolutivos simples y se puede mejorar usando representaciones alternativas.

De las figuras 7.2 y 7.3 se observa que los valores de *ebest* varían entre 0.00 a 1.08 para la instancia *la06*, de 0.77 a 4.81 para *la12*, desde 2.07 a 9.78 para *la15* y desde 11.03 a 11.98 para *abz6*. Se detectan pequeñas variaciones a favor de MCMPIP en cuanto a la performance, pero para todos los experimentos los mejores resultados se obtienen bajo ambas opciones cuando se aumenta el número de crossovers para una determinada cantidad de padres.

Se observa con mayor frecuencia un rango pequeño de valores para los métodos de crossover USX y OBSX independientemente de la opción de recombinación utilizada.

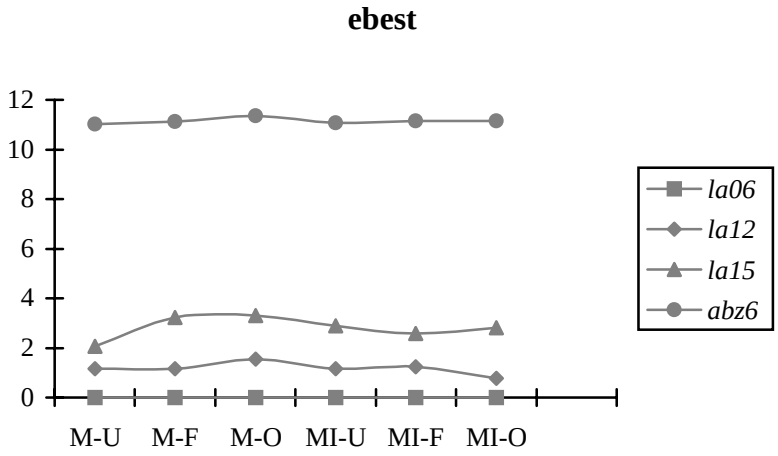


Figura 7.2 Mínimo *ebest* bajo MCMP y MCMPIP.

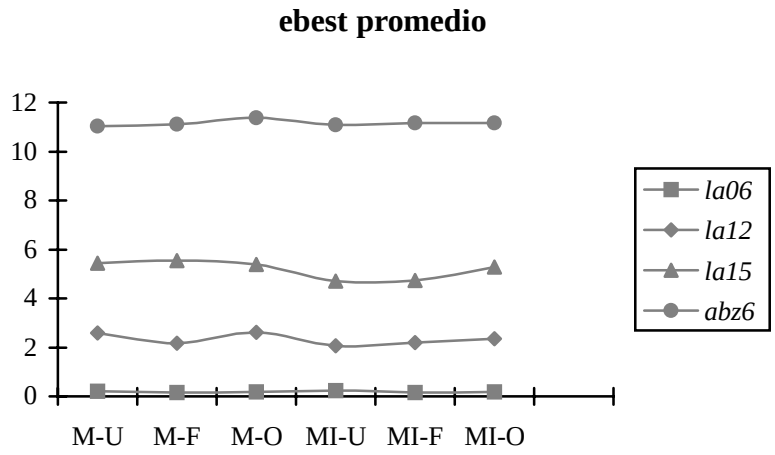


Figura 7.3 *ebest* promedio bajo MCMP y MCMPIP.

Analizando las curvas tanto de *epop* como de *epop* promedio de las figuras 7.4 y 7.5, se observa que estos valores varían desde 1.48 a 28.31 para *la06*, desde 3.85 a 29.92 para *la12*, desde 2.82 a 29.40 para *la15* y desde 14.50 a 47.99 para *abz6*. Nuevamente, los mejores resultados se obtienen bajo ambas combinaciones cuando se aumenta el número de crossovers dado un número determinado de padres. MCMPIP presenta una mejora en los errores poblacionales respecto a MCMP, salvo para la instancia *abz6*.

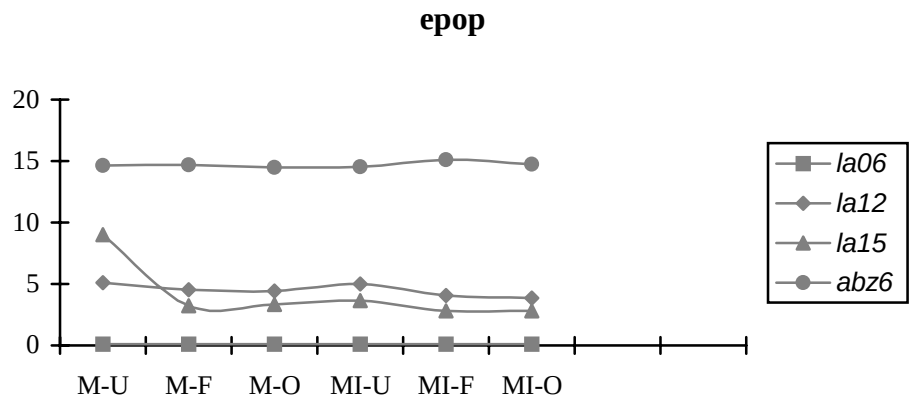


Figura 7.4 Mínimo *epop* bajo MCMP y MCMPIP.

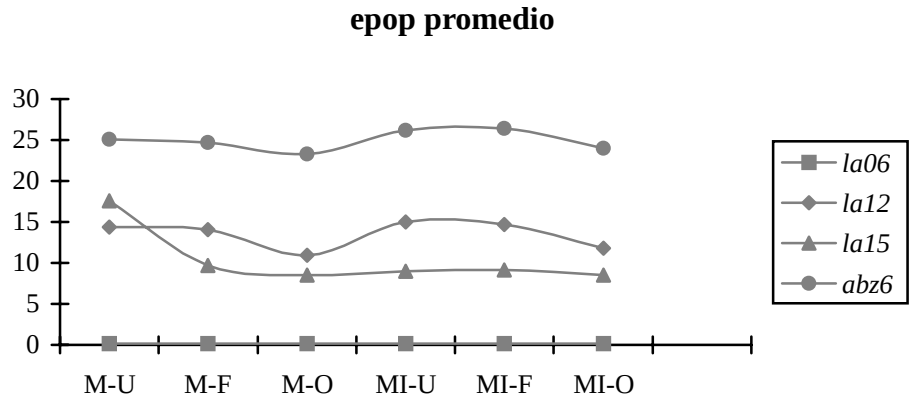


Figura 7.5 *epop* promedio bajo MCMP y MCMPIP.

7.2.3. CONCLUSIONES

La opción presentada introduce una opción con multirecombinación MCMP y su combinación con prevención de incesto MCMPIP, para resolver el problema de job shop scheduling. El uso de decodificadores para problemas con permutaciones permite la implementación directa del crossover convencional evitando la necesidad de penalidades o acciones de reparación. Esto es fundamental para la aplicación de las opciones de multirecombinación ya que el método SX, como inicialmente se define, no es directamente aplicable a representaciones con permutaciones. Luego de una serie de experimentos se puede concluir que:

- ✓ Se obtienen mejores resultados cuando se aumenta el número n_1 de crossovers para un número n_2 de padres. En muchos casos la población final está centrada alrededor

del individuo con fitness más alto, proveyendo un conjunto de schedules alternativos, los cuales pueden ayudar a tratar con sistemas dinámicos.

- ✓ Para cualquier asociación (n_1, n_2) USX y OBSX proveen las diferencias más pequeñas, entre los menores y mayores valores, cuando se lo compara con FBSX.
- ✓ Para la instancia *la06*, ambos métodos hallan el óptimo usando cualquiera de los métodos SX. Para problemas más grandes, hay un incremento en el error porcentual que siempre permanece debajo del 12%.

7.3. OPCIÓN CON REGLAS DE PRIORIDAD DE DESPACHO

Otra solución posible al problema del job shop scheduling es una implementación de algoritmos evolutivos utilizando reglas de despacho. Una particularidad de este algoritmo es la de utilizar semillas generadas por otro método heurístico, las cuales se insertan en la población inicial, para obtener una mejora en la eficiencia del mismo.

Para un problema clásico de job shop scheduling de *n* jobs y *m* máquinas, un cromosoma codifica una secuencia de reglas de despacho. Se incorpora al algoritmo evolutivo (EA) el algoritmo de Giffler y Thompson, donde el EA se usa para evolucionar una secuencia de reglas de prioridad de despacho y el algoritmo de Giffler y Thompson se usa para deducir un schedule desde la codificación de reglas de prioridad de despacho.

La heurística de reglas de despacho, descripta en 6.1.1, se utiliza para generar individuos, los cuales serán incorporados en la población inicial como semillas, en alguna posición determinada en forma aleatoria. En cada población inicial, se incorpora sólo una semilla. El procedimiento de generación de la población inicial de un EA sufrirá una pequeña modificación (la inclusión de la semilla), el pseudocódigo se presenta en la figura 7.6.

Al momento de determinar como será la conformación de un determinado cromosoma en la población inicial en referencia a las reglas de despacho, se elige, para cada gen del cromosoma una regla de prioridad de despacho en forma aleatoria, dentro del conjunto de reglas de despacho consideradas.

Una consideración adicional para elevar la performance de un algoritmo evolutivo es utilizar una heurística convencional en la generación inicial para producir buenos

```
procedure genera-pop-inicial
begin
 pos = nro_aleatorio(1, pop_size ); //la función aleatorio genera un número entre dos
                                    dados
for i = 1 to pop_size
   begin
   if not( i = pos)
     begin
       generar indiv-i
       insertar indiv-i en C(0)
     end
   else
     crear las semillas  //utiliza la heurística de prioridad de despacho
     incorporar semilla en C(0)
end for
end procedure
```

Figura 7.6 Modificación del proceso de generación de la población inicial

cromosomas que serán usados como semillas. Un cromosoma se genera con la heurística de prioridad de despacho, mientras que los restantes cromosomas se generan en forma aleatoria. A esta opción se la denomina PR-Sem-EA y se compara a esta opción híbrida con una en la cual no incorpora semillas en la población inicial, PR-EA.

7.3.1. DESCRIPCIÓN DEL EXPERIMENTO

En este experimento se contrasta el comportamiento dos algoritmos evolutivos, PR-EA y PR-Sem-EA, ambos utilizando representación basada en reglas de despacho.

Ambos algoritmos se analizaron para un conjunto de instancias seleccionadas para el JSSP. Se realizaron 15 corridas diferentes para cada instancia. Para la selección de los individuos que conformarán el matting pool, en el algoritmo evolutivo se usó selección proporcional al fitness del individuo. El mejor individuo se retuvo por elitismo. En ambos algoritmos se consideraron los mismos valores para los parámetros. En los experimentos el tamaño de la población fue de 50 individuos. El operador de crossover

utilizado es uno de los más sencillos, one-point crossover, el cual para esta representación siempre produce hijos factibles. El operador de mutación fue el *big-creep*, el cual consiste en reemplazar el valor de un gen por otro generado aleatoriamente. El número máximo de generaciones fue fijado en 500 y las probabilidades de crossover y mutación fueron establecidas en 0.65 y 0.01, respectivamente. Se usaron 7 instancias [97] con óptimos bien conocidos (tabla 7.6). Las reglas de despacho consideradas son las que se mencionan en la sección 6.1, a excepción de EDD, la cual no fue implementada.

Las variables de performance elegidas para contrastar los resultados encontrados por cada algoritmo son *ebest* y *epop*, además de incorpora:

- ✓ ***gbest*** = Identifica la generación donde el individuo (retenido por elitismo) fue encontrado.

7.3.2. ANÁLISIS DE LOS RESULTADOS

En la tabla 7.7 se muestran las veces que cada algoritmo ha encontrado el óptimo en cada instancia. En la instancia *ft6* ambos algoritmos encuentran la misma cantidad de veces el óptimo; en *la01*, PR-Sem-EA encuentra en una oportunidad más el óptimo que PR-EA; en *la06*, en el 100% de los casos ambos algoritmos hallan el óptimo; en *la12* en el 87% de los casos PR-EA encuentra el óptimo para esta instancia contra el 93% de PR-Sem-EA. En instancias más complejas como *abz6* y *abz7*, los algoritmos no encuentran en ninguna oportunidad el óptimo correspondiente a la instancia.

Instancia	Tamaño	Óptimo
ft6	6×6	66
la01	10×5	666
la06	15×5	926
la12	20×5	1039
la15	20×5	1207
abz6	10×10	943
abz7	20×15	657

Tabla 7.6 Instancias.

	PR-EA	PR-Sem-EA
ft6	9	9
la01	1	2
la06	15	15
la12	13	14
la15	0	0
abz6	0	0
abz7	0	0

Tabla 7.7 Cantidad de veces que cada algoritmo encuentra los óptimos de cada instancia.

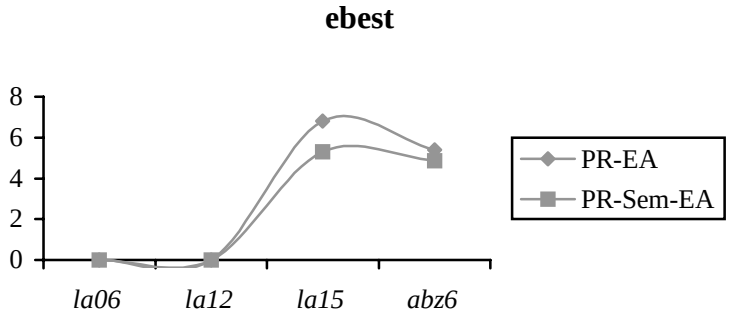


Figura 7.7 Mínimo *ebest* bajo PR-EA y PR-SEM-EA.

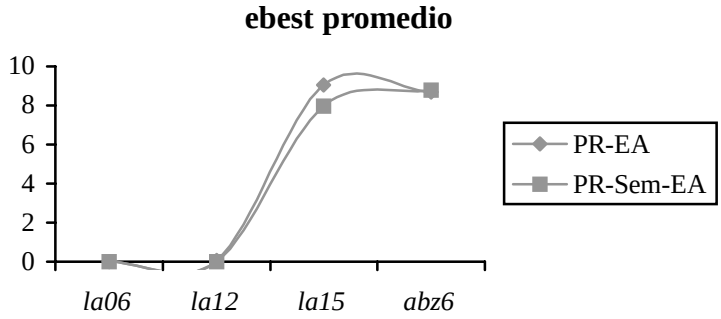


Figura 7.8 *ebest* promedio bajo PR-EA y PR-Sem-EA.

De las figuras 7.7 y 7.8 se pueden analizar las curvas correspondientes al *ebest* y *ebest* promedio tanto para PR-EA como para PR-Sem-EA. Para las instancias *ft6, la01, la06* y *la12*, el *ebest* es del 0%, para ambos algoritmos; lo cual es lógico observando la tabla correspondiente a la cantidad de veces que un algoritmo encuentra el óptimo: en todas estas instancias en alguna oportunidad se ha encontrado el óptimo. Para las restantes instancias, a medida que aumenta la complejidad de las mismas, el *ebest* se

hace más considerable; llegando a un 22.9833% para la instancia *abz7*. Para la instancia *la15* se observa una diferencia a favor de PR-Sem-EA, con un valor de 5.3024 contra 6.7937 de PR-EA, esta misma observación se puede hacer sobre la instancia *abz6* (4.8780 contra 5.4083). En cuanto al *ebest* promedio, para la instancia *la06* es de 0%, tal valor corrobora el porcentaje de éxito para esta instancia, independientemente del hecho de haber incorporado semilla en la población inicial o no. En las restantes instancias, el comportamiento de PR-Sem-EA ha sido levemente mejor que PR-EA; por ejemplo en la instancia *la12* el *ebest* promedio para PR-EA es de 0.1476 y para PR-Sem-EA es de 0.0706 y en la01 el *ebest* promedio para PR-EA es de 2.3123 y para PR-Sem-EA es de 2.2823.

Si se consideran en forma conjunta las curvas de *ebest* y *ebest* promedio, se obtiene una superposición de las mismas cuando se consideran las primeras instancias, lo que significa que los restantes mejores individuos hallados en cada corrida no están tan alejados, en promedio, al óptimo de cada instancia. En tanto que para las instancias *la15, abz6* y *abz7* las diferencias entre *ebest* y *ebest* promedio son un poco más significativas, por ejemplo, bajo la opción PR-Sem-EA el *ebest* para *abz7* es de 22.9833 mientras que el *ebest* promedio es 25.9361.

Analizando las curvas tanto de *epop* como de *epop* promedio de las figuras 7.9 y 7.10, nuevamente, se obtienen resultados bastantes aproximados entre ambas opciones, pero notándose una diferencia a favor de PR-Sem-EA.

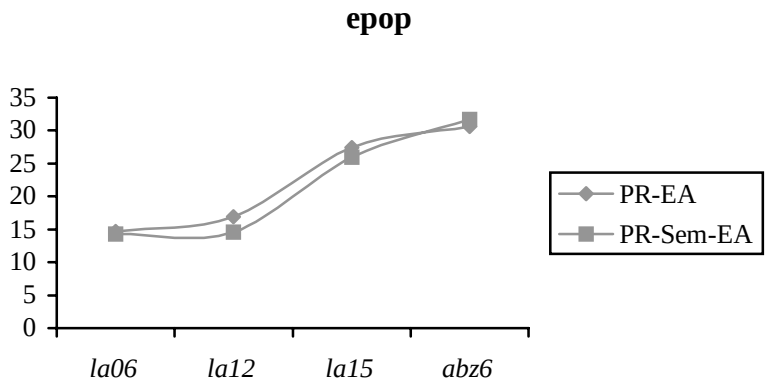


Figura 7.9 Mínimo *epop* bajo PR-EA y PR-SEM-EA.

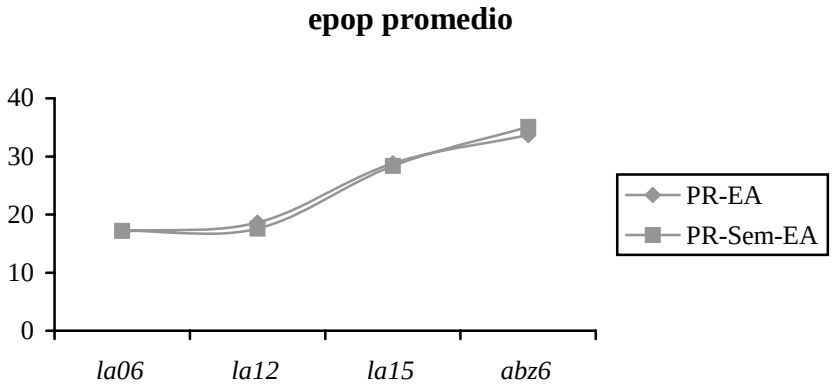


Figura 7.10 *epop* promedio bajo PR-EA y PR-SEM-EA.

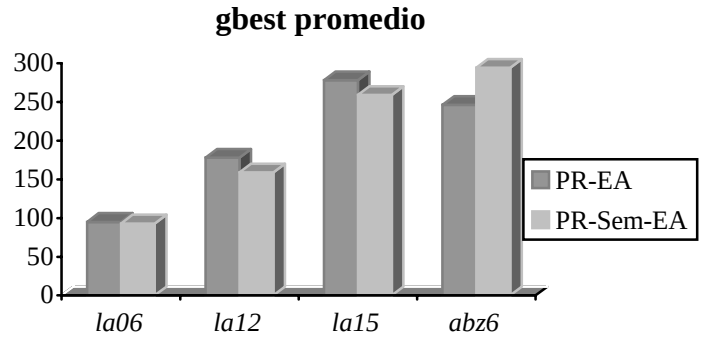


Figura 7.11 *gbest* promedio para todas las instancias.

En la figura 7.11 se observa que para las instancias donde se encuentra los óptimos (*ft6*, *la01*, *la06* y *la12*) e incluyendo a *la15*, PR-Sem-EA llega a encontrar los mejores valores en generaciones más tempranas que PR-EA (para la instancia *ft6* se necesitan 208 generaciones en el caso de PR-EA y 182 para PR-Sem-EA, *la01* tarda 315 generaciones para el caso de PR-EA y 216 para PR-Sem-EA). En el resto de las instancias, la población en PR-Sem-EA necesita evolucionar por más generaciones que en PR-EA para llegar a un valor aproximado al óptimo.

Observando la tabla 7.7, las gráficas correspondientes a *ebest* y la figura 7.11, se puede deducir que aquella población en la cual se introducen semillas parece arribar a sus soluciones finales más rápidamente, sin observar una disminución en la calidad de las soluciones.

7.3.3. CONCLUSIONES

En este caso se presenta otra solución posible al problema del job shop scheduling haciendo uso de algoritmos evolutivos el cual emplea reglas de despacho. El uso de estas últimas en la codificación de un cromosoma permite la aplicación del crossover más sencillo como lo es el one-point crossover, con lo cual siempre se obtendrán hijos factibles sin necesidad de incluir técnicas de penalidades o de reparación. Una particularidad de este algoritmo es la de utilizar semillas generadas por otro método heurístico convencional, las cuales se insertan en la población inicial, para obtener una mejora en la eficiencia del mismo.

Del análisis realizado sobre los resultados obtenidos bajo esta opción se puede resaltar:

- ✓ Que la incorporación de semillas en la población inicial aporta por lo general buenos resultados, permitiendo al algoritmo encontrar individuos más cercanos al óptimo y con poblaciones que permanecen más centradas respecto de ese individuo que en el caso de no incorporar semillas a la población inicial.
- ✓ Además para algunas de las instancias se encuentra el óptimo en más oportunidades cuando se incorporan individuos más adaptados a la población inicial que en el caso de no hacerlo. Además, si se analizan las generaciones promedio en las cuales cada algoritmo encuentra los mejores individuos para cada instancia y la cantidad de veces que para esa instancia se halló en óptimo, se puede concluir que el algoritmo que introduce semillas halla los mejores individuos en forma más rápida, sin que esto implique la pérdida en la calidad de soluciones y la diversidad genética.

7.4. OPCIÓN CON EL MÉTODO DE PARCIAL EXCHANGE

Gen, Tsujimura, y Kubota propusieron una implementación de un algoritmo genético para resolver un problema de job shop scheduling [70]. Usaron una representación del cromosoma basada en operaciones y diseñaron un operador de crossover *partial schedule exchange* para esta representación. El nuevo operador considera schedules parciales como bloques de construcción natural, y tal crossover se utiliza para mantener los bloques en los hijos. El schedule parcial se identifica con el

mismo job en la primer y última posición del schedule parcial. Por ejemplo, si tenemos dos cromosomas padres p_1 y p_2 [69]:

p_1 [3 2 1 2 3 4 1 2 4 4 1 3 4 1 2 3]

p_2 [4 1 3 1 1 3 4 1 2 3 4 2 2 2 3 4]

Los schedules parciales se eligen aleatoriamente de la siguiente manera:

(a) Elegir una posición en el padre p_1 aleatoriamente. Supongamos que es la 6ta posición en la cuál está ubicado el job 4.

(b) Hallar el job 4 más próximo en el mismo padre p_1, es cual está en la posición 9. Ahora tomar el *schedule parcial 1* como (4 1 2 4).

(c) El próximo schedule parcial no es generado aleatoriamente en el padre p_2. Este deberá comenzar y finalizar con el job 4. Note que los dos jobs 4 son el primer y segundo job 4 en el padre p_1. Así el *schedule parcial 2* está formado por los genes entre el primer y segundo job para el job 4 en el padre p_2, (4 1 3 1 1 3 4).

Una ejemplificación gráfica del procedimiento de selección de schedules parciales se muestra a continuación:

schedule parcial 1

p_1 [3 2 1 2 3 4 1 2 4 4 1 3 4 1 2 3]

p_2 [4 1 3 1 1 3 4 1 2 3 4 2 2 2 3 4]

schedule parcial 2

A continuación se muestra el intercambio de schedules parciales:

schedule parcial 2

o_1 [3 2 1 2 3 4 1 3 1 1 3 4 4 1 3 4 1 2 3]

o_2 [4 1 2 4 1 2 3 4 2 2 2 3 4]

schedule parcial 1

Por lo general, los schedules parciales están formados por distintas cantidades de genes, lo que puede provocar la ilegalidad de los hijos que resultan del intercambio. Los genes que se pierden y los que están en exceso en los hijos se pueden determinar de la siguiente manera:

genes perdidos y en exceso para o_1

schedule parcial 1 4 1 2 4 genes perdidos 2

schedule parcial 2 4 1 3 1 1 3 4 genes en exceso 3 1 1 3

genes perdidos y en exceso para o_2

schedule parcial 2 4 1 3 1 1 3 4 genes perdidos 3 1 1 3

schedule parcial 1 4 1 2 4 genes en exceso 2

En el próximo paso se legalizan los hijos al suprimir los genes en exceso y adicionar aquellos genes perdidos. Para el hijo o_1, sus genes en exceso son (3 1 1 3). El hijo o_1, recibe el *schedule parcial 2* desde el padre p_2, pero deberá borrar los jobs 3 y 1 antes del schedule parcial en o_1, para conseguir el mismo orden que en el padre p_2. La razón de lo anterior es que los genes en la representación basada en operaciones se refieren a operaciones las cuales son dependientes del contexto. Si los genes del *schedule parcial 2* en el hijo o_1 toman el mismo orden que en el padre p_2, esos genes se deberán referir a las mismas operaciones en ambos casos; en caso contrario se referirán a operaciones diferentes. Se espera que el hijo o_1 herede el mismo schedule parcial identificado por *schedule parcial 2* en el padre p_2. Debemos dejar los genes del *schedule parcial 2* en el mismo orden tanto en o_1 como en p_2. Para el hijo o_1, el gen que se pierde es 2. Como no hay ningún job 2 en *schedule parcial 2*, se inserta el job 2 en algún lugar (excepto dentro del *schedule parcial* 2), con lo cual no se cambia el orden de los genes en el *schedule parcial 2*. El proceso de legalización del hijo o_1 es el siguiente:

(1) *Borra los genes en exceso* 3 1 1 3

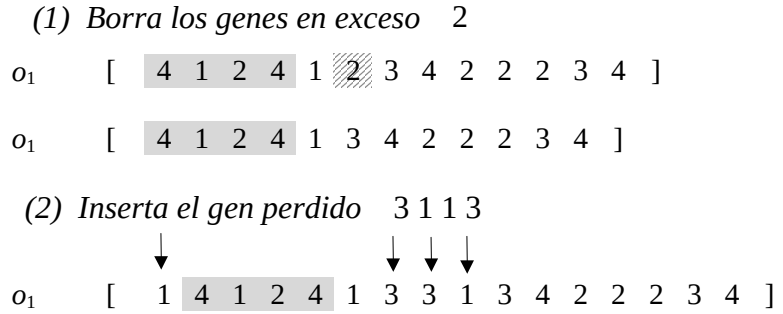
o_1 [3 2 1 2 3 4 1 3 1 1 3 4 4 1 3 4 1 2 3]

o_1 [2 2 4 1 3 1 1 3 4 4 3 4 1 2 3]

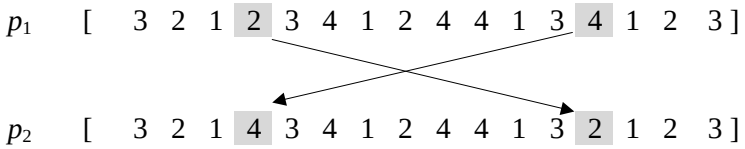
(2) *Inserta el gen perdido* 2

o_1 [2 2 4 1 3 1 1 3 4 2 4 3 4 1 2 3]

Para el hijo o_2, el gen en exceso es el job 2. Se puede borrar algún job 2 en el hijo o_2 excepto dentro del *schedule parcial 1*. Sus genes perdidos son (3 1 1 3). Se debe insertar un job 1 antes del *schedule parcial 1* y un job 1 luego del *schedule parcial 1* en el hijo o_2 de modo de conseguir que el job 1 del *schedule parcial 1* sea el segundo job 1 en el hijo o_2. Como no hay job 3 en el *schedule parcial 1*, se inserta un job 3 en algún lugar (excepto dentro del *schedule parcial 1*). Esto se puede apreciar en el siguiente esquema:



Gen Tsujimura, y Kubota usaron una mutación de intercambio de pares de jobs; es decir, elegir aleatoriamente dos jobs distintos y entonces intercambiar sus posiciones:



Para la representación basada en operación, la relación de mapping entre cromosomas y schedule es muchos a uno, y un hijo obtenido al intercambiar dos jobs cercanos puede traer el mismo schedule de los padres. De este modo la mayor separación entre dos jobs distintos es lo mejor.

7.4.1. DESCRIPCIÓN DEL EXPERIMENTO

En este experimento se contrasta el comportamiento de dos algoritmos evolutivos:

- ✓ **PR-EA** que utiliza la representación basada en reglas de prioridad, descripta en la sección anterior, y sobre la cual se aplica el one-point crossover y la mutación big-creep.
- ✓ **OR-EA** que utiliza la representación basada en operaciones, donde el operador de crossover utilizado es el *parcial schedule exchange crossover* y la mutación consiste en el intercambio de pares de jobs.

Ambos algoritmos se analizaron para un conjunto de instancias seleccionadas para el JSSP. Se realizaron 15 corridas diferentes para cada instancia. Para la selección de los individuos que conformarán el matting pool, en el algoritmo evolutivo se usó selección proporcional al fitness del individuo. Los mejores individuos se retuvieron por elitismo. En los experimentos el tamaño de la población fue de 50 individuos. El número máximo de generaciones fue fijado en 500 y las probabilidades de crossover y mutación fueron

establecidas en 0.65 y 0.01, respectivamente. Se usaron 7 instancias [97] con óptimos bien conocidos (tabla 7.6).

Las variables de performance elegidas para contrastar los resultados encontrados por cada algoritmo son *ebest*, *epop* y *gbest*.

7.4.2. ANÁLISIS DE LOS RESULTADOS

En la tabla 7.8 se muestran las veces que cada algoritmo ha encontrado el óptimo en cada instancia. En la instancia *la06*, en el 100% de los casos ambos algoritmos hallan el óptimo; en la instancia *ft6*, PR-EA encuentra en el 60% de los casos el óptimo contra el 100% de OR-EA; en *la01*, PR-EA encuentra en una única oportunidad el óptimo mientras que OR-EA lo hace en 9 de las 15 ejecuciones; en *la12* en el 87% de los casos PR-EA encuentra el óptimo para esta instancia contra el 100% de OR-EA. En instancias más complejas como *la15*, *abz6* y *abz7*, los algoritmos no encuentran en ninguna oportunidad el óptimo correspondiente a la instancia.

	PR-EA	OR-EA
ft6	9	15
la01	1	9
la06	15	15
la12	13	15
la15	0	0
abz6	0	0
abz7	0	0

Tabla 7.8 Cantidad de veces que cada algoritmo encuentra los óptimos de cada instancia.

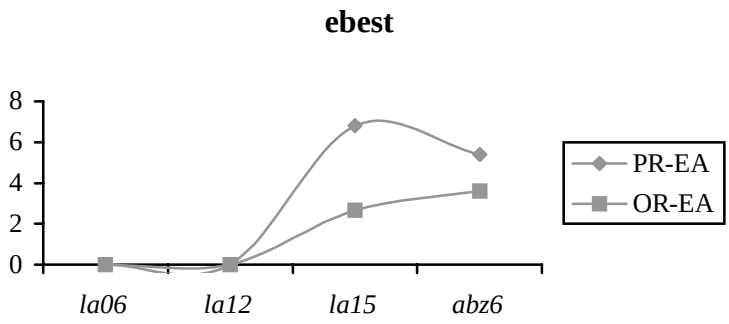


Figura 7.12 *ebest* para PR-EA y OR-EA.

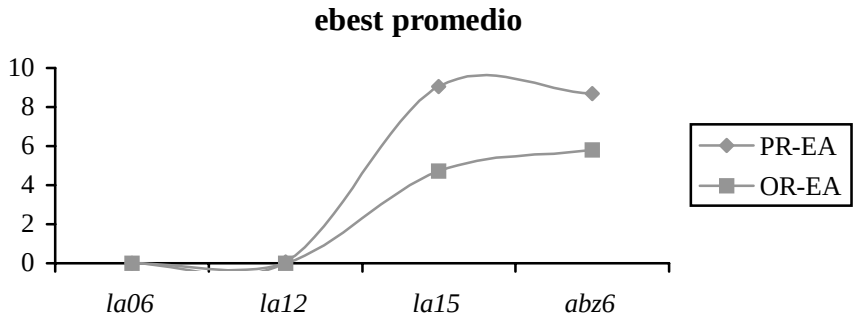


Figura 7.13 *ebest* promedio para PR-EA y OR-EA.

Analizando la figura 7.12 se puede observar que para las instancias *ft6*, *la01*, *la06* y *la12* ambos algoritmos logran un *ebest* del 0%, significando que en al menos una oportunidad se halla el valor óptimo. Para el resto de las instancias consideradas, el *ebest* en el que incurre OR-EA es considerablemente menor que PR-EA, por ejemplo para la instancia *la15*, el *ebest* de OR-EA es 2.6512% y el de PR-EA es 6.7937%.

Como se puede observar en el gráfico correspondiente a *ebest* promedio (figura 7.13), OR-EA encuentra en promedio individuos más cercanos al óptimo que PR-EA. En las instancias *ft6*, *la06* y *la12* se puede observar que el error producido por OR-EA es del 0%, lo cual indica que en todas las ejecuciones el algoritmo ha encontrado el valor óptimo correspondiente a cada instancia; en tanto que para la instancia *la01* el *ebest* promedio para este mismo algoritmo es cercano a 0 (0.5%). En el caso de PR-EA, los errores promedios para las mismas instancias *ft6*, *la01* y *la12* oscilan entre 2.3% (en el caso de *la01*) y 0.07% (para *la12*). Cabe destacar que ambos algoritmos obtienen en todas las ejecuciones el valor óptimo de la instancia *la06*. En los casos de las instancias *la15*, *abz6* y *abz7* se producen errores promedios más altos pero siempre se observa una mejor performance para OR-EA.

Al analizar la figura 7.14, se puede observar que la población final a la que arriba el algoritmo OR-EA está más centrada alrededor del valor óptimo que en el caso de PR-EA; en particular en las instancias *ft6* y *la01*, donde los errores cometidos son menores al 0.8%. En estas mismas instancias y en *abz6* es donde se observan las mayores diferencias entre los *epop* que presentan ambos algoritmos, por ejemplo en la instancia *ft6* el *epop* producido por PR-EA es 24.7637% mientras que OR-EA produce un *epop* de 0.001%.

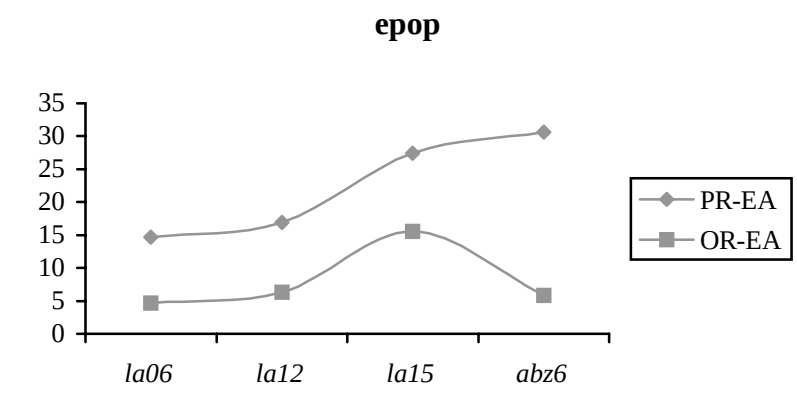


Figura 7.14 *epop* para PR-EA y OR-EA.

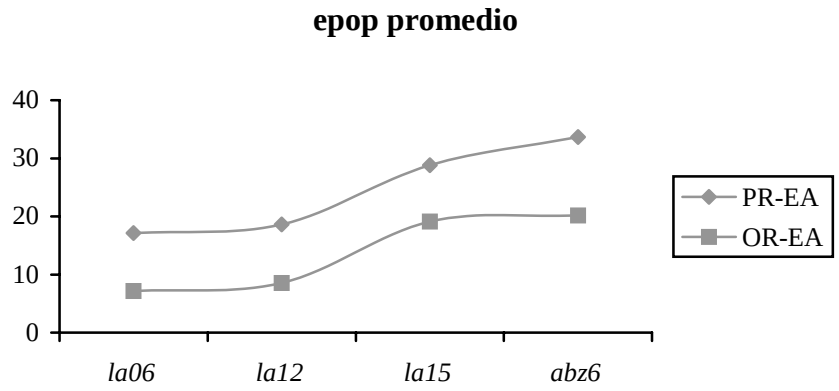


Figura 7.15 *epop* promedio para PR-EA y OR-EA.

En la figura 7.16, se puede observar que para la instancia *la06* donde ambos algoritmos encuentran en el 100% de las ejecuciones el valor óptimo, OR-EA lo hace usando una notable menor cantidad de generaciones (5 generaciones en promedio para OR-EA y 95 para PR-EA). En las instancias *ft6*, *la01* y *la12*, donde el *ebest* es muy cercano a cero, la cantidad de generaciones que OR-EA necesita para llegar a un valor óptimo o muy cernano a él está muy por debajo de las requeridas por PR-EA. Cuando la complejidad de las instancias, medida en la cantidad de máquinas empleadas, se incrementa repercute en la cantidad de generaciones que OR-EA emplea para hallar el valor más cercano al óptimo, en particular para *abz6* y *abz7* necesitan aproximandamente 50 generaciones más que PR-EA, pero de todos modos los errores tanto poblacionales como los de los mejores individuos de OR-EA son siempre inferiores a PR-EA.

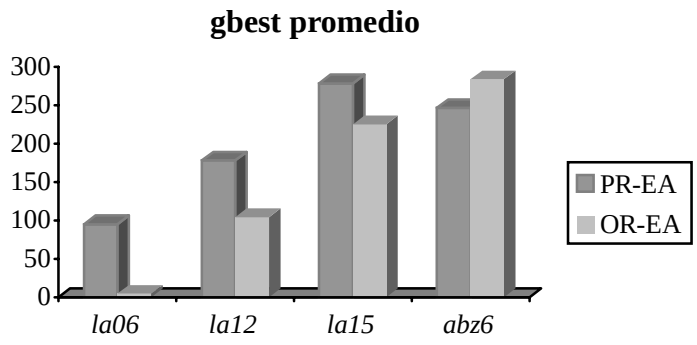


Figura 7.16 Gbest para PR-EA y OR-EA

7.4.3. CONCLUSIONES

En este caso se presenta otra alternativa para la solución del problema de job shop scheduling usando un algoritmo evolutivo, pero aquí con una representación basada en operaciones e incorporando un operador de crossover adecuado para tal representación como lo es el parcial schedule exchange crossover.

De los análisis previos realizados sobre los resultados obtenidos se puede concluir que con un algoritmo evolutivo usando la representación basada en operaciones, la performance es considerablemente mejor que cuando se usa una representación con reglas de despacho; esa performance se mide ya sea en la calidad de soluciones halladas en forma individual como en el error de población final respecto del valor óptimo para cada instancia. Además una característica importante a remarcar es que las poblaciones finales a las que arriba OR-EA permanecen más centradas respecto del mejor individuo hallado que en el caso de PR-EA.

Es importante destacar que en las instancias *la01*, *la06* y *la12*, OR-EA ha encontrado en todas las pruebas realizadas el valor óptimo de cada instancia; a medida que la complejidad de las instancias se incrementa esta cantidad disminuye pero de todos modos el error promedio porcentual cometido por los mejores individuos hallados no es signifivo.

REFERENCIAS

[1] Adams, J., Balas, E., y Zawack, D., The Shifting Bottleneck Procedure for Job Shop Scheduling, en *International Journal of Flexible manufacturing Systems*, vol. 34, nro. 3, pag. 391-401, 1987.

[2] Alfonso, H., Cesán, P., Fernandez, N., Minetti, G., Salto, C., Velazco, L., y Gallard, R., Improving Evolutionary Algorithms Performance by Extending Incest Prevention, en *Anales del Cuarto Congreso Argentino de Ciencias de la Computación* (CACIC'98), Universidad Nacional del Comahue, pag. 323-334, Abstracts del Congreso, pag. 69, 1998.

[3] Angeline, P.J y Kinnear, K.E. (editores), *Advances in Genetic Programming II*, MIT Press, Cambridge, MA, 1996.

[4] Bäck, J., Reducing Bias and Inefficiency in the Selection Algorithm, en Grefenstette, J. (editor), *Proceedings of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, pag. 14-21, Hillsdale, NJ, 1985.

[5] Bäck, T., Selective Pressure in Evolutionay Algorithms: a Characterisation of Selection Mechanisms, en Fogel, D. (editor), *Proceedings of the First IEEE Conference on Evolutionary Computation*, IEEE Press, pag. 57-62, 1995.

[6] Bäck, T., *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, New York, 1996.

[7] Bäck, T., Hoffmeister, F., y Schwefel, H., A Survey of Evolution Strategies, en Belew, R. y Booker, L. (editores), en *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pag. 2-9, Los Altos, CA, 1991.

[8] Bäck, T., y Schwefel, H., An Overview of Evolutionary Algorithms for Parameter Optimization, en *Evolutionary Computation*, vol. 1, nro. 1, pag. 1-23, 1993.

[9] Bäck, T., y Hoffmeister, T., Extended Selection Mechanisms in Genetic Algorithms, en Belew, R., y Booker, L. (editores), *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pag 92-99, San Mateo, CA, 1991.

[10] Baghi, S., Uckun, S., Miyabe, Y., y Kawamura, K., Exploring problem-specific recombination operators for job shop scheduling, en Belw, R. y Boolker. L. (editores), *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pag. 10-17, San Mateo, CA, 1991.

[11] Baker, J., Adaptive Selection Methods for Genetic Algorithm, en Grefenstette, J. (editor), *Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, pag. 100-111, NJ, 1987.

[12] Baker, K., *Introduction to Sequencing and Scheduling*, John Wiley & Sons, New York, 1974.

[13] Balas, E., Machine Sequencing via Disjunctive Graphs: an Implicit Enumeration Algorithm, en *Operations Research*, vol. 17, pag. 941-957, 1969.

[14] Bean, J., Genetic Algorithms and Random Keys for Sequencing and Optimization, en *ORSA Journal on Computing*, vol. 6, nro. 2, pag. 154-160, 1994.

[15] Beasley, D., Bull, D., y Martin, R., Reducing Epistasis in Combinatorial Problems by Expansive Coding, en Forrest, S. (editor), *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pag. 400-407, San mateo, CA, 1993.

[16] Bolc, L. y Cytowski, J., *Search Methods for Artificial Intelligence*, Academic Press, London, 1992.

[17] Booker, L.B., Intelligent Behaviour as an Adaptation to the Task Environment, Ph.D. Dissertation, Department Computer and Communication Sciences, University of Michigan, Ann Arbor, MI, 1982.

[18] Booker, L.B., Improving Search in Genetic Algorithms, en Davis, L. (editor), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, pag. 61-73, Los Altos, CA, 1987.

[19] Bremermann, H., Rogson, M., y Salaff, S., Global Properties of Evolution Processes, en Patee, H., Edlsack, E., Fein, L., y Callahan, A., (editores), *Natural Automata and Useful Simulations*, Spartan Books, pag. 3-41, 1966.

[20] Brindle, A., Genetic Algorithms for Function Optimization, Ph.D. thesis, University of Alberta, Edmonton, 1981.

[21] Bruns, Ralf, Scheduling, en Th. Bäck, D. B. Fogel, y Z. Michalewicz (editores), *Handbook of Evolutionary Computation*, capítulo F1.5, pages F1.5:1-F1.5:9. Oxford University Press, New York, y Institute of Physics Publishing, Bristol, 1997.

[22] Cohoon, J. y Paris, W., Genetic Placement, en *IEEE Transactions on Computer-Aided Design*, vol. 6, nro. 6, pag. 1272-1277, 1987.

[23] Croce, F., Tadei, R., y Volta, G., A genetic Algorithm for the job shop problem, en *Computers and Operations Research*, vol. 22, pag. 15-24, 1995.

[24] Dasgupta, D., y Michalewicz, Z., *Evolutionary Algorithms in Engineering Applications*, Dasgupta, D., y Michalewicz, Z. (editores), Springer_Verlag, pag. 3-28. 1997.

[25] Davis, L., Applying adaptive algorithms to domains, en *Proceedings of the International Joint Conference on Artificial Intelligence*, pag. 162-164, 1985.

[26] Davis, L., Job Shop Scheduling with Genetic Algorithms, en Grefenstette, J. (editor), *Proceedings of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, pag. 136-140, 1985.

[27] Davis, L., Adapting Operators Probabilities in Genetic Algorithms, en Schaffer, J. (editor), *Proceeding of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pag. 61-69, San Mateo, CA, 1989.

[28] Davis, L. y Steenstrup, M., Genetic Algorithms and Simulated Annealing: An Overview, en *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, pag. 1-11, Los Altos, CA, 1987.

[29] De Jong, K., An Analysis of the Behaviour of a Class of Genetic Adaptive Systems, Ph.D. Thesis, University of Michigan, Ann Arbor, 1975.

[30] De Jong, K., *Genetic algorithms: a 25 Years Perspective, in Computational Intelligence: Imitating Life*, IEEE Press, New York, pag. 125-134, 1994.

[31] Dell'Amico, M., y Trubian, M., Applying Tabu Search to the Job-Shop Scheduling Problem, en *Annals of Operations Research*, vol. 40, pag. 231-252, 1993.

[32] Dorndorf, U., y Pesch, E., Evolution Based Learning in a Job Shop Scheduling Environment, en *Computers and Operations Research*, vol. 22, pag. 25-40, 1995.

[33] Eiben, A.E y van Kemenade, C.H.M, Diagonal Crossover in Genetic Algorithms for Numerical Optimization, en *Journal of Control and Cybernetics*, vol. 26, nro. 3, pag. 447-465, 1997.

[34] Eiben, A.E. y Bäck Th., An Empirical Investigation of Multi-Parent Recombination Operators in Evolution Strategies, en *Evolutionary Computacion*, vol. 5, nro. 3, pag. 347-365, 1997.

[35] Eiben, A.E., A Method for Designing Decision Support Systems for Operational Planning, PhD Thesis, Eindohoven University of Technology, 1991.

[36] Eiben, A.E., Hinterding, R., y Michalewicz, Z., Parameter Control in Evolutionary Algorithms, Technical Report, UNC – Charlotte, 1998.

[37] Eiben, A.E., Raué P.E., y Ruttkay Zs. Genetic Algorithms with Multi-parent recombination, en Davidor, H.-P. Schwefel, y R. Männer (editores), *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, nro 866 in LNCS, pag. 78-87, Springer-Verlag, 1994.

[38] Eiben, A.E., van Kemenade, C.H.M, y Kok, J.N., Orgy in the Computer: Multi-Parent Reproduction in Genetic Algorithms, en F. Moran, A. Moreno, J.J. Merelo, y P. Chacon, editors, *Proceedings of the Third European Conference on Artificial Life*, number 929 in LNAI, pag. 934-945, Springer-Verlag, 1995.

[39] Eschelman, L.J., y Schaffer, D.J., Crossover Niche, en Stephanie Forrest (editor), *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pag. 9-14, 1993.

[40] Eshelman L.J., Schaffer, J.D., Preventing Premature Convergence in Genetic Algorithms by Preventing Incest, en Belew, R. y Booker, L. (editores), *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers,, pag. 115-122, San Mateo, California, USA, 1991.

[41] Eshelman, L, y Schaffer, J., Crossover's niche, en Forrest, S. (editor), *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pag. 9-17, 1993.

[42] Eshelman, L.J., The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Non-traditional Genetic Recombination, en Rawlins, G.J.E. (editor), *Foundations of Genetic Algorithms and Classifier Systems*, Morgan Kaufmann Publishers, pag. 265-283, 1991.

[43] Eshelman, L.J.k Caruana, R.A., y Schaffer, J.D., Biases in the Crossover Landscape, en Schaffer, J. (editor), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pag. 10-19, San Mateo, CA, 1989.

[44] Esquivel, S., Gallard, R., y Michalewicz, Z., MCMP: Another Approach to Crossover in Genetic Algorithms, en *Anales del Primer Congreso Argentino de Ciencias de la Computación*, Universidad Nacional de San Luis, pag. 141-150, 1995.

[45] Esquivel, S., Leiva, A. y Gallard, R., Multiple Crossover per Couple in Genetic Algorithms, en *Proceedings of the Fourth IEEE International Conference on Evolutionary Computation* (ICEC'97), pag. 103-106, Indianapolis, USA, April 1997.

[46] Esquivel, S., Leiva, H.A., y Gallard, R.H., Multiple Crossover per Couple and Fitness Proportional Couple Selection in Genetic Algorithms, en *Anales del Tercer Congreso Argentino de Ciencias de la Computación*, CACIC'97, Universidad Nacional de La Plata, vol 1., pag. 180-191, 1997

[47] Esquivel, S., Leiva, H.A., y Gallard, R.H., Self-Adaption of Parameters for MCPC in Genetic Algorithms, en *Anales del Cuarto Congreso Argentino de Ciencias de la Computación* (CACIC'98), Universidad Nacional del Comahue, pag. 419-426, 1998.

[48] Esquivel, S., Leiva, A. y Gallard, R., Couple Fitness Based Selection with Multiple Crossover per Couple in Genetic Algorithms, en Alpaydin, E. (editor), *Proceedings of the International Symposium on Engineering of Intelligent Systems* (EIS'98), La Laguna, Tenerife, Spain, vol. 1, pag. 235-241, published by ICSC Academic Press, Canada/Switzerland, 1998.

[49] Esquivel, S., Leiva, A. y Gallard, R., Multiple Crossovers between Multiple Parents to Improve Search in Evolutionary Algorithms, en *Proceedings of the 1999 Congress on Evolutionary Computation (IEEE)*, vol. 2, pag. 1589-1594, Washington DC., 1999.

[50] Esquivel, S., Leiva, A. y Gallard, R., Multiplicity in Genetic Algorithms to Face Multicriteria Optimization, en *Proceedings of the 1999 Congress on Evolutionary Computation*, IEEE Service Center, pag. 85-90, Washington, D.C., 1999.

[51] Esquivel, S., Leiva, H.A., y Gallard, R.H., The MCPC Evolution in Evolutionary Computation, en *Anales Workshop de Investigadores en Ciencias de la Computación*, Universidad Nacional de San Juan, vol. 2, trabajo 12, 1999.

[52] Falkenauer, E. y Bouffoix, S., A Genetic Algorithm for Job Shop, en *Proceedings of the IEEE International Conference on Robotics and Automation*, pag 824-829, 1991.

[53] Fang, H., Ross, P., y Corne, D., A Promising Genetic Algorithm Approach to Job-Shop Scheduling, Rescheduling, and Open-Shop Scheduling Problems, en *Proceedings of the Fifth International Conference on Genetic Algorithms*, 2nd Ed., Prentice Hall, pag. 375-382, 1992.

[54] Fogarty, T. (editor), *Evolutionary Computing*, Springer-Verlag, Berlin, 1994.

[55] Fogel, D.B., Evolving Artificial Intelligence, Ph.D. Thesis, University of California, San Diego, 1992.

[56] Fogel, D.B., Evolving Behaviours in the Iterated Prisoner's Dilemma, en *Evolutionary Computation*, vol. 1, nro. 1, pag. 77-97, 1993.

[57] Fogel, D.B., An Introduction to Simulated Evolutionary Optimization, en *IEEE Transactions on Neural Networks*, vol. 5, pag. 3-14., 1994.

[58] Fogel, D.B., *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEE Press, Piscataway, NJ, 1995.

[59] Fogel, D. y Atmar, J., Comparing Genetic Operators with Guassian mutations in Simulated Evolutionary Processes Using Linear Systems, *Biological Cybernetics*, nro. 63, pag. 111-114, 1990.

[60] Fogel, D.B. y Atmar, W. (editores), *Proceedings of the First Annual Conference on Evolutionary Programming*, La Jolla, CA, Evolutionary Programming Society, 1992.

[61] Fogel, D.B. y Atmar, W., (editores) *Proceedings of the Second Annual Conference on Evolutionary Programming*, Evolutionary Programming Society, La Jolla, CA, 1993.

[62] Fogel, D. y Ghozeil, A., Using Fitness Distributions to Design more Efficient Evolutionary Computations, en Fogel, D. (editor), *Proceedings of the Third IEEE Conference on Evolutionary Computation*, IEEE Press, pag. 11-19. Nagoya, Japan, 1996.

[63] Fogel, D. y Stayton, L., On the Effectiveness of Crossover in Simulated Evolutionary Optimization, en *Biosystems*, nro 32, pag. 171-182, 1994.

[64] Fogel, L.J., Walsh, M.J., y Owns, A.J., *Artificial Intelligence Through Simulated Evolution*, John Wiley, Chichester, UK, 1966.

[65] Frantz, D. R., Non-linearities in Genetic Adaptive Search, en Dissertation Abstracts International, vol. 33, nro. 11, pag. 5240B-5241B.

[66] French, S., *Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop*, Horwood, Chichester, 1982.

[67] Gen, M., y Cheng, R., A Survey of Penalty Techniques in Genetic Algorithms, en Fogel, D. (editor), *Proceedings of the Third IEEE Conference on Evolutionary Computation*, IEEE Press, pag. 804-809, Nagoya, Japan, 1996.

[68] Gen, M., y Kobayashi, T.(editores), *Proceedings of the 16th International Conference on Computers and Industrial Engineering*, Ashikaga, Japan, 1994.

[69] Gen, M. y Runwei, C., *Genetic Algorithms and Engineering Design*, Wiley-Interscience Publication John Wiley & Sons, Inc, 1997.

[70] Gen, M., Tsujmura, Y., y Kubota, E., Solving Job-Shop Scheduling Problem Using Genetic Algorithms, en Gen, M., y Kobayashi, T. (editores), *Proceedings of the 16th International Conference on Computers and Industrial Engineering*, Ashikaga, Japan, pag. 576-579, 1994.

[71] Gibson, G., *Application of Genetic Algorithms to Visual Interactive Simulation Optimisation*, PhD thesis, University of South Australia, 1995.

[72] Giffler, B. y Thompson, G., Algorithms for Solving Production Scheduling Problems, en *Operations Research*, vol. 8, nro. 4, pag. 487-503, 1960.

[73] Gillies, A., Machine Learning Procedures for Generating Image Domain Feature Detectors, Ph.D. thesis, University of Michigan, Ann Arbor, 1985.

[74] Goldberg, D. y Lingle, R., Alleles, loci and the traveling salesman problem, en Grefenstette, J. (editor), *Proceedings of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, pag. 154-159, Hillsdale, NJ, 1985.

[75] Goldberg, D., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[76] Goldberg, D., Korb, B., y Deb, K., A Comparative Analysis of Selection Schemes Used in Genetic Algorithms, en Rawlings, G. (editor), *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, pag. 69-93, San Mateo, CA, 1991.

[77] Goldberg, D.E. y Richardson, J., Genetic Algorithms with Sharing for Multimodal Function Optimization, en *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pag. 41-49, 1987.

[78] Grefenstette, J.J., Optimization of Control Parameters for Genetic Algorithms, en *IEEE Transactions on Systems, Man and Cybernetics*, SMC-16(1), pag. 122-128, 1986.

[79] Grefenstette, J.J., *A User's Guide to GENESIS*. Navy Center for Applied Research in Artificial Intelligence, Washington, D.C., 1987.

[80] Grefenstette, J.J., Incorporating problem specific knowledge into genetic algorithms. En L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, Pitman Publishing, pag. 42-60, London, 1987.

[81] Grefenstette, J., y Baker, J., How Genetic Algorithms Work: a Critical Look at Implicit Parallelism, en Schaffer, J. (editor), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pag. 20-27, San Mateo, CA, 1989.

[82] Grefenstette, J., Gopal, R., Rosmaita, B., y Gucht, D., Genetic Algorithms for the Travelling Salesman Problem, en Grefenstette, J. (editor), *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pag. 160-168, 1985.

[83] Hancock, P., An Empirical Comparison of Selection Methods in Evolutionary Algorithms, en Fogarty, T. (editor), *Evolutionary Computing*, Springer-Verlag, pag. 80-95, Berlín, 1994.

[84] Holland, J.H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[85] Holsapple, C., Jacob, V., Pakath, R., y Zaveri, J., A Genetics-Based Hybrid Scheduler for Generating Static Schedules in Flexible Manufacturing Contexts, en *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, pag. 953-971, 1993.

[86] Hordijk, W. y Manderich, B., The usefulness of Recombination, en Morán, F., Moreno, A., Merelo, J.J. y Chacón, P. (editores), *Advances in Artificial Life. Third International Conference on Artificial Life*, volume 929 of *Lecture in Artificial Intelligence*, Springer-Verlag, pag. 908-919, Berlin, 1995.

[87] Kinner, K.E. (editor), *Advances in Genetic Programming*, MIT Press, Cambridge, MA, 1994.

[88] Koza, J.R., Genetic Programming: a Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems, Report Nro. STAN-CS-90-1314, Stanford University, 1990.

[89] Koza, J.R., *Genetic Programming*, MIT Press, Cambridge, MA, 1992.

[90] Koza, J., *Genetic Programming: on Programming Computers by Means of natural Selection and Genetics*. The MIT Press, Camdridge, MA, 1992.

[91] Koza, J.R., *Genetic Programming – 2*, MIT Press, Cambridge, MA, 1994.

[92] Kubota, A., Study on Optimal Scheduling for Manufacturing System by Genetic Algorithms, Tesis de Maestría, Ashikaga Institute of Technology, Ashikaga, Japan, 1995.

[93] Lageweg, B., Lenstra, J., Lawler, E., Rinnooy Kan, A., *Computer-Aided Complexity Classification of Combinatorial Problems*. Communications of the ACM, vol. 25, pag 817-822, 1982.

[94] Laudon, K. y Laudon, J., *Essentials of Management Information Systems*, Prentice Hall, Third Edition, 1999.

[95] Lawler, E., Lenstra, J., y Rinnooy Kan, A., Recent Developments in Deterministic Sequencing and Scheduling: A Survey, en Dempster, M.A.H., Lenstra, J.K., y RinnoyKan, A.H.G. (editores), *Deterministic and Stochastic Scheduling*, pag. 35-75, 1982.

[96] Lawler, E., Lenstra, J., Rinnooy Kan, A., y Shmoys, D., Sequencing and Scheduling: Algorithms and Complexity, en Graves, S.S., Rinnooy Kan, A.HG., y Zipking, P. (editores), *Handbooks in Operations Research and Management Science*, vol 3: Logistics of Production and Inventory, pag. 445-522, North-Holland, New York, 1993.

[97] Lawrence, S., Resource Constrained Project Scheduling: an Experimental Investigation of Heuristic Scheduling Techniques, (Supplement), Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1984.

[98] Lenstra, J., *Sequencing by Enumerative Methods, Mathematical*, Centre Tracts 69, Centre for Mathematics and Computer Science, Amsterdam, 1977.

[99] Manly, B., *The Statistics of Natural Selection on Animal Populations*, Chapman & Hall, London, 1984.

[100] McDonnell, J.R., Reynolds, R.G., y Fogel, D.B. (editores), *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, The MIT Pres, 1995.

[101] Michalewicz, Z., A Hierarchy of Evolution Programs: An Experimental Study, en *Evolutionary Computation*, vol 1, nro. 1, pag. 51-76, 1993.

[102] Michalewicz, Z., *Genetic Algorithms + Data Structure = Evolution Programs*, 2nd ed., Springer-Verlag, New York, 1994.

[103] Michalewicz, Z., A Survey of Constraint Handling Techniques in Evolutionary Computation Methods, en McDonnell, J., Reynolds, R., y Fogel, D. (editores), *Evolutionary Programming IV*, MIT Press, pag. 135-155, 1995.

[104] Minetti, G., Salto, C., Alfonso, H., y Gallard, R., Combining Incest Prevention and Multiplicity in Evolutionary Algorithms, en *Anales del Quinto Congreso Argentino de Ciencias de la Computación*, CACIC'99, Universidad Nacional del Centro, 1999.

[105] Minetti, G., Salto, C., Alfonso, H., y Gallard, R., Multimodal Optimization via Multiplicity and Incest Prevention in Genetic Algorithms, en *Proceedings of the Second ICSC Symposium on Engineering of Intelligent Systems*, University of Paisley, pag. 445-450, Scotland, 2000.

[106] Mühlenbein, H., Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization, en Schaffer, J. (editor), *Proceeding of de Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pag. 416-421, San Mateo, CA, 1989.

[107] Mühlenbein, H., y Voigt, H.-M., Gene Pool Recombination in Genetic Algorithms, en Osman, I. y Kelly, J. (editores), *Meta-Heuristics: Theory and Applications*, pag. 53-62, 1996.

[108] Nakano, R., y Yamada, T., Conventional Genetic Algorithms for Job-Shop Problems, en Belew, R., y Booker, L. (editores), *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pag. 477-479, San Mateo, CA, 1991.

[109] Norman, B., y Bean, J., Random Keys Genetic Algorithm for Job-Shop Scheduling: Unabridged Version, Technical report, University of Michigan, Ann Arbor, 1995.

[110] Norman, B., y Bean, J., Random Keys Genetic Algorithm for Scheduling, Technical Report, University of Michigan, Ann Arbor, 1995.

[111] Oliver, I., Smith, D., y Holland, J., A study of permutation Crossover Operators on the Traveling Salesman Problem, en Grefenstette, J. (editor), *Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, pag. 224-230, NJ, 1987.

[112] Orvosh, D. and Davis, L., Using a Genetic Algorithm to Optimize Problems with Feasibility Constraints, en *Proceedings of the First IEEE Conference on Evolutionary Computation*, IEEE Press, pag. 548-552, 1995.

[113] Paredis, J., Explotiting Constraints as Background Knowledge for Genetic Algorithms: a Case-Study for Scheduling, en Männer, R., y Manderick, B. (editores), *Parallel Problem Solving form Nature PPSN II*, Elsevier Science Publishers, pag. 281-290, North-Holland, 1992.

[114] Perez Serrada, A., *Una Introducción a la Computación Evolutiva*, 1996.

[115] Pinedo, M, *Scheduling: Theory, Algorithms and Systems*, Prentice Hall, 1995.

[116] Reeves, D., Diversity and Diversification in Algorithms: Some Connections with Tabu Search, en Albrecht, R., Reeves, C., y Steele, N. (editores), *Artificial Neural Nets and Genetic Algorithms*, Springer-Verlag, pag. 344-351, New York, 1993.

[117] Rinnooy Kan, A. *Machine Scheduling problems: Classification, Complexity and Computations*, Nijhoff, The Hague, 1976.

[118] Ronald, S., Preventing Diversity Loss in a Routing Genetic Algorithm with Hash Tagging, en R. Stonier and Xing Huo Yu (editores), *Complex Systems: Mechanism of Adaptation*, IOS Press, pag. 1663-140, Amsterdam, 1994.

[119] Ronald, S., *Genetic Algorithms and Permutation-enconding Problems. Diversity Preservation and a Study of Multi-Modality*. PhD thesis, University South Australia. The Department of Computer and Information Science, 1995.

[120] Ronald, S., Asenstorfer, J., y Vincent. M., Representational Redundancy in Evolutionary Algorithms, en Fogel, D. (editor), *Proceedings of the 1995 IEEE International Conference on Evolutionary Computation*, IEEE Press, pag. 631-637, New York, 1995.

[121] Roy, B., y Sussmann, B., Les Problems d'Ordonnancemente avec Constintes Disjonctives, Technical Report 9 , SEMA, Note D.S, Paris, 1964.

[122] Sakawa, M., Kato, K., y Mori, T., Flexible Scheduling in a Machining Center through Genetic Algorithms, *Computers and Industrial Engineering*, vol. 30, nor. 4, pag. 931-940, 1996.

[123] Salto, C., Alfonso, H., Gallard, R., Multiplicity and Incest Prevention in Evolutionary Algorithms to Deal with de Job Shop Problem, en *Proceedings of the Second ICSC Symposium on Engineering of Intelligent Systems*, University of Paisley, Scotland, pag, 451-457, 2000.

[124] Schaffer, H., Caruana, R., Eshelman, R., y Das, R., A Study of Control Parameters Affecting Online Performance of Genetic Algorithms Optimization, en *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pag. 51-60, San Mateo, CA, 1989.

[125] Schaffer, J. y Eshelman, L., On Crossover as an Evolutionary Viable Strategy, en Belew, R. y Booker, L. (editores), *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pag. 61-68, San Mateo, CA, 1991.

[126] Schwefel, H., *Evolution and Optimum Seeking*, John Wiley & Sons, New York, 1994.

[127] Schwefel, H., On the Evolution of Evolutionary Computation, en Aurada, H., Marks, R., y Robinson, C. (editores), *Computational Intelligence: Imitating Life*, IEEE Press, pag. 116-124, 1994.

[128] Sebald, A.V., y Fogel, L.J., (editores) *Proceedings of the Third Annual Conference on Evolutionary Programming*, San Diego, CA, 1994, World Scientific.

[129] Shi, G., Iima, H., y Sannomiya, N., A method for Constructing Genetic Algorithm in Job Shop Problems, en *Proceedings of 8th SICE Symposium Decentralized Autonomous System*, pag. 175-178, 1996.

[130] Smith, A., y Tate, D., Genetic Optimization using a Penalty Function, en Forrest, S. (editor), *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pag. 499-505, San Mateo, CA, 1993.

[131] Spears, W., Crossover or Mutation?, en Whitley, L. (editor), *Foundations of Genetic Algorithms-2*, Morgan Kaufmann Publishers, pag. 221-238, 1993.

[132] Spears, W.M., Adapting Crossover in a Genetic Algorithm, en J. R. McDonnell, R. G. Reynolds y D. B. Fogel (editores), *Proceedings of the Fifth Conference on Evolutionary Programming*, Cambridge: MIT Press, pag. 367-384, 1995.

[133] Storer, R., Wu, S., y Vaccari, R., New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling, en *Management Science*, vol. 38, nro. 10, pag. 1495-1510, 1992.

[134] Syswerda, G., Uniform Crossover in Genetic Algorithms, en *Proceeding of the International Conference on Genetic Algorithms*, pag. 2-9, 1989.

[135] Tamaki, H., Mori, M., y Araki, M., Generation of a Set of Pareto-Optimal Solutions by Genetic Algorithms, en *Transactions of the Society of Instrument and Control Engineers*, vol. 31, nro. 8, pag. 1185-1192, 1995.

[136] Tamaki, H. y Nishikawa, Y., A paralleled genetic Algorithm Based on a Neighborhook Model and its Application to the jobshop scheduling, en Männer, R. y Manderick, B. (editores), *Parallel Problem Solving from Nature: PPSN II*, Elsevier Science Publishers, North-Holland, pag. 573-582, 1992.

[137] Thierens, D., y Goldberg, D., Convergence Models of Genetic Algorithm Selection Schemes, en Davidor, Y., Schwefel, H., y Männer, R., (editores), *Parallel Problem Solving from Nature: PPSN III*, Springer-Verlag, pag. 119-129, Berlin, 1994.

[138] Tsujimura, Y. y Gen, M., Genetic Algorithms for Solving Multi-Processor Scheduling Problems, en Yao, X., Kim, J.H., y Furuhashi, T. (editores), *Proceedings of the First Asia-Pacific Conference on Simulated Evolution and Learning*, Taejon, 1996.

[139] Van Laarhoven, P., Aarts, E., y Lenstra, J., Job Shop Scheduling by Simulated Annealing, en *Operations Research*, vol. 40, nro. 1, pag. 113-125, 1992.

[140] Vignaux, G. Y Michalewicz, Z., Genetic Algorithms for the Transportation problem, en *Methodologies for Intelligent Systems*, vol. 4, pag. 252-259, 1989.

[141] Voigt, H.-M., y Mühlenbein, H., Gene Pool Recombination and Utilization of Covariances for the Breeder Genetic Algorithm, en *Proceedings of de Second IEEE Conference on Evolutionary Computation*, pag. 172-177, 1995.

[142] Wetzel, A., Evaluation of the Effectiveness of Genetic Algorithms, en Combinatorial Optimization, Technical Report, University of Pittsburgh, 1983,

[143] Whitley, D., GENITOR: a Different Genetic Algorithm, en *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, Denver, 1989.

[144] Whitley, D., The GENITOR Algorithm and Selection Pressure: Why Rank-based Allocation of Reproductive Trials is Best¸ en Schaffer, J., (editor), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pag. 116-121, San Mateo, CA, 1989.

[145] Whitley, D., Starkweather, T., y Fuquay, D., Scheduling Problems and Travelling Salesmen: The Genetic Edge Recombination Operator, en Schaffer, J. (editor), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pag. 133-139, San Mateo, CA, 1989.

[146] Whitley, D., Starkweather, T., y Shaner, D., The Travelling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination, en *Handbook of Genetic Algorithms*, pag. 350-372, Van Nostrand Reinhol, New York, 1991.

[147] Wilson, S.W., Classifier System Learning of a Boolean Function, RIS 27r, The Rowland Institute for Science, Cambridge, MA, 1986.

[148] Yamada, T. y Nakano, R., A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems, en Männer, R. y Manderick, B. (editores), *Parallel Problem Solving from Nature*, PPSN II, Elsevier Science Publishers, North-Holland, pag. 281-290, 1992.