CIATEC

PROGRAMACIÓN DE OPERACIONES EN CONFIGURACIONES JOBSHOP FLEXIBLES UTILIZANDO UN ALGORITMO DE ESTIMACIONES DE DISTRIBUCIONES

Tesis

QUE PARA OBTENER EL GRADO ACADEMICO DE

Doctor en Ciencia y Tecnología en la Especialidad de Ingeniería Industrial y de Manufactura

PRESENTA

Ricardo Pérez Rodríguez

DIRECTOR DE TESIS

DR. JÖNS SÁNCHEZ AGUILAR

León, Guanajuato, Mayo del 2014

POSGRADO INTERINSTITUCIONAL EN CIENCIA Y TECNOLOGIA

AGRADECIMIENTOS

Aprovecho esta oportunidad para agradecer a todos los que me han apoyado en los momentos más importantes de mi vida académica.

En primer lugar agradezco a mis hermanos por su fuente inagotable de apoyo, estimulo e inspiración. Al igual que a mi esposa por su paciencia y comprensión en todo momento.

Expreso mi más profundo reconocimiento y gratitud a mi Director de Tesis, Jöns Sánchez, por ser una gran fuente de información y motivación, de sus invaluables consejos y apoyo, de sus discusiones interesantes y pensamientos que me ayudaron a culminar esta investigación.

Doy las gracias a todo los miembros del PICYT con sede CIATEC, A.C., a Antonio Quijas, Emma Acevedo y a todo el equipo de soporte por hacer de estos últimos años un momento agradable y memorable.

Ofrezco mi gratitud a los investigadores Arturo Hernández, Javier Yañez, Antonio Vázquez y Carlos Ochoa, por darme la oportunidad de aprender junto con ellos al igual de compartir sus importantes discusiones y comentarios. Y en especial a Arturo Hernández por sus pensamientos muy útiles y sugerencias.

Así mismo ofrezco mis agradecimientos a Jon Fournier, Martin Barnes y Tan Zhuoning por su soporte técnico incondicional en el lenguaje de simulación utilizado en esta investigación.

Finalmente estoy agradecido con el CONACYT por el soporte económico para solventar esta investigación al igual que al CONCYTEG por su apoyo en la infraestructura utilizada.

RESUMEN DEL AUTOR

Ricardo Pérez Rodríguez nació en la Cd. de México en Abril de 1976. Graduado del Instituto Politécnico Nacional IPN a través de la Unidad Profesional Interdisciplinaria de Ingeniería y Ciencias Sociales y Administrativas UPIICSA como Licenciado en Administración Industrial LAI en Marzo del año 2000. Obtuvo el grado de Maestro en Ingeniería –Investigación de Operaciones– por la División de Estudios de Posgrado de la Facultad de Ingeniería DEPFI de la Universidad Nacional Autónoma de México UNAM en Mayo del 2004. Actualmente es candidato a Doctor en Ciencia y Tecnología en la opción terminal de Ingeniería Industrial y Manufactura por parte del Posgrado Interinstitucional en Ciencia y Tecnología PICYT con sede en el Centro de Innovación Aplicada en Tecnologías Competitivas CIATEC, A.C. de la Cd. de León, Gto.

Su principal área de interés es la optimización de procesos de manufactura por medio de técnicas de Inteligencia Artificial y Simulación de Eventos Discretos.

Ha ejercido por más de 10 años como Ingeniero Industrial, Manufactura y de Procesos en diversas compañías manufactureras y de servicios tanto nacionales como extranjeras, entre las que destacan INVERLAT, GNP, VITRO, PEPSI Co., SEGLO Group, SSW Holding Co., MAGNA Automotive Systems, entre otras.

Adicional, ha laborado como profesor de cátedra en Instituciones de Educación Superior, además de participar en Congresos Internacionales y Nacionales relacionados a la Ingeniería Industrial y Simulación de Manufactura. Finalmente ha publicado artículos con alcance nacional e internacional en revistas de reconocido prestigio.

RESUMEN

Los Algoritmos Genéticos, algoritmos basados en poblaciones, se han utilizado en diversos problemas de programación complejos. Estos algoritmos se combinan dos procesos principales con el fin de encontrar mejores soluciones: la selección y la variación. Los operadores de cruza y mutación pertenecen al proceso de variación. El operador de cruza recombina soluciones parciales mediante la selección de los individuos que representan soluciones factibles del problema y la mutación hace cambios aleatorios en los descendientes que también son soluciones factibles del problema y que son el resultado de la variación. Sin embargo, a veces el proceso de variación puede distorsionar las soluciones y no tiene efectos positivos sobre la aptitud -capacidad de supervivencia- de cada individuo.

Los Algoritmos de Estimación Distribuciones también pertenecen a la clase de algoritmos de optimización basados en poblaciones. Estos están motivados por la idea de descubrir y explotar la interacción entre las variables en la solución. En estos algoritmos se estima una distribución de probabilidad de la población de soluciones y se toman muestras para generar la siguiente población. Muchos Algoritmos de Estimación Distribuciones utilizan modelos gráficos como técnicas probabilísticas de modelado para este propósito. En particular, los modelos gráficos dirigidos (redes bayesianas) han sido ampliamente utilizados en los Algoritmos de Estimación Distribuciones.

Esta tesis propone y describe un nuevo enfoque híbrido para hacer frente a problemas de programación y secuenciamiento de operaciones en configuración jobshop flexible, a través del uso de la Simulación de Eventos Discretos apoyado por un Algoritmo de Estimación de Distribuciones como método de optimización. Este fue capaz de calcular la dependencia condicional entre las secuencias de producción y los turnos de trabajo empleados para reducir la producción en proceso en un sistema de manufactura de puertas de acero. El método híbrido se comparó con un Algoritmo Genético sobre el sistema de manufactura mencionado. El enfoque híbrido propuesto demuestra encontrar secuencias y turnos más prometedores dentro del espacio de búsqueda combinatorio mediante el uso de tres modelos de gráficos probabilísticos en el mismo algoritmo.

CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

INTRODUCCIÓN

En la literatura concerniente a optimización, toda solución a un problema de este tipo consiste en encontrar la solución óptima o bien cerca de esta desde un conjunto de soluciones factibles utilizando alguna medida para evaluar cada solución individual. Un algoritmo que resuelve problemas de este tipo es llamado algoritmo de optimización. Esta tesis se enfoca sobre una clase de algoritmo de optimización general conocido como Algoritmo Evolutivo EA (por sus siglas en inglés Evolutionary Algorithm).

Los Algoritmos Evolutivos están inspirados por la teoría de Darwin sobre la evolución natural de las especies. En la naturaleza, las especies se desarrollan mejor a través de la selección natural y por medio de variaciones aleatorias. Los Algoritmos Evolutivos EAs siguen este enfoque y simulan la selección natural y las variaciones aleatorias para encontrar la mejor solución a un problema. Ambos, selección natural y variación aleatoria tienen un distinto rol en la ejecución de un EA. La selección genera la presión en la evolución hacia soluciones de alta calidad, mediante la selección de soluciones más aptas de una población de soluciones. La variación reproduce la siguiente generación de soluciones basado en la selección de soluciones más aptas y asegura una apropiada exploración al conjunto posible de soluciones. Las tres estrategias mejor conocidas como EAs, los Algoritmos Genéticos GAs (por sus siglas en inglés Genetic Algorithms) desarrollados por Holland (1975), las Estrategias Evolutivas ES (por sus siglas en inglés Evolution Strategies) por Rochenberg (1973) y la Programación Evolutiva EP (por sus siglas en inglés Evolutionary Programming) por Fogel (1962) mantienen los conceptos de la selección y la variación en su ejecución.

En un GA, existen dos operadores, la cruza y la mutación, estos son utilizados para simular la variación. El operador de cruza forma la nueva población al intercambiar algunas partes entre las soluciones seleccionadas. El operador de mutación modifica ligeramente algunas partes de las nuevas soluciones recién formadas. La cruza y mutación, enfoque tradicional de variación en un GA, se ha encontrado que es limitado para algunos problemas de optimización, por lo que las primeras investigaciones en GAs se centraron en la modificación de estos operadores para mejorar su rendimiento.
En años recientes, el enfoque probabilístico a la variación ha sido propuesto, reemplazando los operadores de la cruza y mutación por otros dos: la estimación de la distribución y el muestreo.
La estimación de la distribución es calcular una distribución de probabilidad de las soluciones de la población, y el muestreo genera nuevas soluciones a partir de dicha distribución. Los algoritmos que utilizan este enfoque son llamados Algoritmos de Estimación de Distribuciones EDAs (por sus siglas en inglés Estimation of Distribution Algorithms) introducidos por Mühlenbein & Paaβ (1996).

Los EDAs han sido reconocidos como una técnica poderosa para optimizar. Se les ha encontrado un mejor desempeño con respecto a los GAs, en problemas donde estos últimos no han dado resultados satisfactorios.
El desempeño de un EDA depende de que tan bien estima la distribución de probabilidad y de que tan bien muestrea con ella. Demasiadas investigaciones sobre EDAs están enfocadas en este punto. Particularmente, los modelos gráficos dirigidos (redes Bayesianas) han sido ampliamente estudiados y se han establecido como un método útil para estimar la distribución en los EDAs. En esta tesis se propone la combinación de tres modelos gráficos dirigidos, para estimar la distribución de un EDA que sistemáticamente resuelve un problema de optimización NP-hard (por sus siglas en inglés Non-deterministic Polynomial-time hard).

El problema de optimización NP-hard a resolver es la programación y secuenciamiento de operaciones en configuración jobshop flexible de un sistema de manufactura bajo estudio.
La programación y secuenciamiento de operaciones puede ser definida como la asignación de recursos a los trabajos a procesar en un tiempo definido. La programación juega un importante rol en sistemas de producción y manufactura. En los problemas relacionados con programación y secuenciamiento de operaciones, donde se busque una solución óptima (o cercana a esta) son en su mayoría NP-hard. Los planteamientos actuales para resolver problemas de programación pueden ser clasificados en determinísticos y estocásticos. La programación determinística generalmente ha utilizado el supuesto implícito de un ambiente de manufactura estático donde no se tiene ningún tipo de falla o bien un evento incierto. Desafortunadamente, la mayoría de los sistemas de manufactura operan en ambientes dinámicos, donde usualmente ocurren eventos inevitables e impredecibles y pueden causar un cambio en el programa de operaciones original, además de que la solución óptima (o cercana a esta) con respecto a la información original puede llegar a ser obsoleta cuando sea liberado el programa de operaciones al piso de producción, generando con esto la necesidad de una programación estocástica.
Ejemplos de eventos que ocurren inevitable e impredeciblemente son fallas en los equipos, máquinas, llegada de trabajos urgentes, cambios en las fechas y cantidades de entrega, cancelaciones de órdenes de producción, ausentismo, accidentes, errores de operación, calidad de materiales, suministros, componentes, etc.
MacCarthy & Liu (1993) se enfocaron a identificar la brecha entre la teoría de la programación y la programación en la práctica y la falla de la teoría de programación por responder a las necesidades de ambientes prácticos. Determinaron las tendencias de investigación en años recientes sobre programación que tratan de encontrar soluciones más relevantes y aplicables.

Una alternativa de solución relevante y aplicable a la programación y secuenciamiento de operaciones que incluya eventos de manera dinámica es la simulación. Los modelos de simulación que se construyen actualmente en diversos lenguajes, son utilizados para resolver problemas de programación y secuenciamiento de operaciones y ayudan a comprender mejor el entorno dinámico de cualquier sistema de manufactura. La simulación es una herramienta que se utiliza efectivamente para analizar sistemas de manufactura complejos. La simulación también permite manejar problemas estocásticos, donde la teoría de programación ha demostrado ser intratable si no se tienen simplificaciones importantes del problema en estudio. Las aplicaciones de la tecnología de simulación en programación y secuenciamiento de operaciones ha sido también un tópico de investigación popular (Tunali, 1997).

Esta investigación maximiza el potencial de utilizar un algoritmo de estimación de distribuciones en combinación con un modelo de simulación para determinar la mejor forma de programar y secuenciar las operaciones en configuración jobshop flexible de un sistema de manufactura.

1. JUSTIFICACIÓN

En esta tesis se trata la programación de operaciones, entendida de forma general como una disciplina cuyo objetivo es asignar y secuenciar diferentes órdenes de fabricación en un sistema productivo, y que se encuentra fuertemente ligada a técnicas que se desarrollan en el marco cuantitativo. Durante décadas se ha avanzado en forma significativa en esta disciplina. En los últimos años, se ha realizado un esfuerzo mayor en abordar problemas prácticos y se han desarrollado una serie de nuevas técnicas que se enfocan en resolver problemas del mundo real. Existen modelos basados en restricciones con gran flexibilidad en las representaciones y escalabilidad en la gestión de las restricciones. Igualmente,

existen herramientas de programación matemática capaces de afrontar problemas se escalas sin precedentes, y metaheurísticas que proporcionan capacidades robustas para la optimización de la programación de operaciones. A pesar de esto, solo un pequeño número de propuestas de solución atiende la etapa de implementación industrial o incluso pruebas en condiciones reales. Un primer paso fue hecho por la comunidad de investigación de operaciones (IO), la cual ha propuesto diversos estándares que permiten evaluar los distintos algoritmos que se diseñan para resolver problemas de optimización estáticos en manufactura. Los estándares se construyen comparando la salida de diferentes sistemas sobre un conjunto de datos de entrada para mejorar alguna medida de desempeño. Las ventajas de estos estándares son bien conocidas: bases de datos muy grandes utilizadas como generadores de casos y las mejores soluciones difundidas en la comunidad científica. Los problemas atendidos están relacionados a problemas de IO tradicionales. Así, estos estándares son útiles para evaluar la calidad de un método o algoritmo de programación con un conjunto de datos estructurados, es decir, un conjunto de datos completo, exacto y disponible en el momento inicial de la programación pero sin utilizar un control de retroalimentación. Como resultado, los datos manejados en estos estándares son cuantitativos y estáticos, lo que permite una clara comparación de los resultados en términos de alguna variable de interés.

Desde el punto de vista del control, estos estándares no permiten que el comportamiento dinámico de casi todo proceso de manufactura sea evaluado, ni modificar las decisiones basadas en las observaciones de eventos en tiempo real y sobre datos no estructurados. Además, esos estándares fueron diseñados principalmente desde un punto de vista teórico, con poca atención puesta en restricciones de sistemas de manufactura reales tales como capacidad de producción, almacenamiento, transporte, mantenimiento, espacio, ni en eventos dinámicos o datos como averías u órdenes canceladas o urgentes.

La comunidad industrial también ha propuesto estándares intentando la comparación coherente entre los resultados que se obtienen y la factibilidad de implementación al tomar en cuenta la dinámica del ambiente. Estos estándares son interesantes porque se intenta trabajar con el comportamiento dinámico del sistema a ser controlado, el cual es más difícil de formalizar en un simple modo cuantitativo. Si los datos dinámicos, consideraciones en tiempo real y eventos impredecibles deben ser manejados y sus impactos evaluados, esto incrementara drásticamente la complejidad para desarrollar un útil y claro estándar.

En base a lo expuesto, existen técnicas muy maduras, las más cercanas a los entornos estáticos y con pocos recursos en juego, y otros muy lejos aún de ser dominados, como los relacionados con entornos dinámicos, con incertidumbre o de gran complejidad.

Algunos de los retos más destacados en programación de operaciones son:

- La generación de técnicas para proporcionar programar en condiciones de alta complejidad en las restricciones y objetivos. Se trata de disponer de herramientas que funcionen en condiciones realistas, y que tengan en cuenta la escala y complejidad de los problemas.
- Desarrollo de técnicas para generar programas en entornos cambiantes. Se debe proponer herramientas que traten los entornos inciertos y que sean capaces de gestionar los cambios con la evolución del sistema.
- La integración de la planeación y programación. Existe una línea muy delgada entre ambas actividades o al menos una fuerte dependencia, por lo que se requiere dotar de herramientas que implemente dicha integración.
- Desarrollo de herramientas de modelado intuitivas para el usuario. Se trata de implementar productos que permitan a un usuario no experto modelar un problema y analizar los resultados del modelo sin necesidad de conocer el lenguaje de modelado matemático.

- La necesidad de disponer de técnicas que permitan incorporar los conocimientos ya desarrollados normalmente en forma de modelos o algoritmos, para crear nuevas herramientas dotadas de un alto grado de robustez, flexibilidad y escalables.

Buscando atender a los retos descritos previamente se opta por utilizar un modelo de simulación. Una de las razones para justificar el uso de un modelo de simulación en esta investigación para programar y secuenciar operaciones es el hecho de que los modelos matemáticos que están reportados en la literatura, frecuentemente, no son eficientes para resolver problemas de tamaño razonable. Además, estos se basan en supuestos que simplifican el proceso de modelado y que no siempre es válido para problemas prácticos de manufactura. Los supuestos, son diferentes dependiendo el modelo utilizado: algunos modelos asumen que las condiciones por ejemplo de transportación de la producción entre estaciones de trabajo, las entregas de material, los herramentales, los equipos, entre otras facilidades de manufactura no son restricciones a considerar en el modelo, ocasionando que se modele el piso de manufactura desde un punto de vista estático. Se asume que las actividades de programación y secuenciamiento de operaciones son realizadas con exactitud o bien que las interrupciones no son relevantes. Se dice entonces que los modelos convencionales para programar y secuenciar operaciones no son eficientes para representar un sistema de manufactura de manera realista, precisamente porque simplifican la complejidad del sistema bajo estudio. Adicionalmente, no siempre ha sido posible resolver dichos problemas eficientemente por métodos tradicionales.

Hoy en día, algunos investigadores han presentado los modelos de simulación como una herramienta para programar y secuenciar operaciones. Básicamente, la simulación es propuesta como una herramienta para evaluar las políticas de programación que se tiene en el piso de manufactura. Así, un modelo de simulación se construye con las condiciones tal como se presentan en el sistema bajo estudio. El modelo es capaz de inicializarse al estado actual que se tiene en el sistema.
La simulación es ciertamente más tratable que las formulaciones de programación matemática utilizadas en problemas de programación y secuenciamiento de operaciones. Con simulación, no hay ninguna preocupación sobre la flexibilidad del modelo, ya que no es necesario hacer ningún supuesto que simplifique el mismo. Además, los modelos de simulación pueden ser construidos tan idénticos como sea necesario sobre los procesos de manufactura a estudiar.
La simulación puede ser utilizada como una herramienta de soporte a las decisiones cuando hay la posibilidad de simular diferentes alternativas de decisión. Esto es debido a que a diferencia de los métodos clásicos y/o convencionales de programación y secuenciamiento de operaciones, la simulación puede manejar eventos inesperados en un ambiente de manufactura dinámico, como lo son casi todos. Eventos inesperados pueden ser fallas en máquinas, retrasos en entregas de material, ausentismo, entre otros. Entonces si un evento inesperado ocurre, el programa de producción y de operaciones definido previamente, debería tener ciertas modificaciones para lograr encontrar el mejor valor de la medida de desempeño que se esté evaluando.

Ahora bien, dentro de los Algoritmos Evolutivos utilizados para resolver problemas de programación y secuenciamiento de operaciones están los Algoritmos Genéticos. Diversos investigadores han usado los algoritmos genéticos para programar de manera dinámica sistemas de manufactura con la presencia de fallas en los equipos, los procesamientos de las operaciones en tiempo real, las llegadas de nuevos materiales, la no disponibilidad de partes para ensamblar debido a fallas o a retrasos, herramientas no disponibles o mantenimiento no programado. Donde un evento dinámico ocurre, los Algoritmos Genéticos han sido propuestos como una alternativa de programación obteniendo resultados satisfactorios por encima de otras técnicas heurísticas. Sin embargo, si el problema es incierto y contiene aspectos difíciles de modelar, estos resultados desaparecen, al igual que cuando se incrementa el tamaño del problema y no son eficientes para encontrar una solución cercana al óptimo en un tiempo

razonable (Bierwirth & Mattfeld, 1999). La razón de esto es diversa, pero principalmente se debe a que los operadores del algoritmo genético no intentan aprender o aprovechar la interacción entre variables que pudiese existir en el programa de operaciones. La naturaleza aleatoria de los operadores puede algunas veces trastornar los valores que ofrece la interacción entre las variables del problema evitando así obtener efectos positivos en encontrar la solución. Por ello, utilizar un Algoritmo de Estimación de Distribuciones es la propuesta de esta investigación, que se enfoca en descubrir y aprovechar la interacción entre variables asociadas al problema en estudio. La idea es aprender y aprovechar la interacción entre variables mediante la estimación de una distribución de la población y muestrear a partir de dicha distribución los descendientes. Entonces los Algoritmos Genéticos (y por ende los Algoritmos de Estimación de Distribuciones) pueden ser utilizados para resolver problemas de programación en ambientes dinámicos de manufactura y trabajarlos en conjunto con otras técnicas para alcanzar buenos resultados (Husbands, 1994).

Finalmente, la integración de un modelo de simulación con un algoritmo evolutivo es ideal debido a que se puede guiar a una serie de simulaciones en busca de soluciones de alta calidad en ausencia de estructuras matemáticas tratables. Mientras ambos algoritmos generan un conjunto de soluciones iniciales factibles, el modelo de simulación es el encargado de evaluar dichas soluciones.

2. PROBLEMÁTICA

El problema de programación *Jobshop flexible* permite la asignación de cada operación de cada trabajo para ser procesado por más de una máquina. La idea es asignar la secuencia de procesamiento de operaciones en las máquinas y la asignación de las operaciones en máquinas tales que los objetivos del sistema se pueden optimizar. Sobre la asignación se ha mencionado que es una tarea difícil de implementar en los entornos de fabricación reales, porque hay muchas suposiciones para satisfacer, sobre todo cuando la cantidad de trabajo no es constante ni suficiente para mantener el proceso de fabricación ocupado durante mucho tiempo, haciendo que los tiempos de actividad sean intermitentes.

Con el enfoque propuesto en esta investigación, el funcionamiento del proceso se puede mejorar notablemente cuando diferentes máquinas son asignadas a diferentes horarios. Se define la construcción de un modelo de simulación y dos algoritmos evolutivos para encontrar las mejores secuencias de producción en la manufactura de puertas de acero. Este sistema de manufactura tiene una gran oportunidad clave de mejora: la secuencias de producción de puertas y los turnos de trabajo a programar. El interés entonces es encontrar el mejor programa de producción que está compuesto tanto de la mejor secuencia como del mejor calendario laboral para manufacturar puertas de acero.

La combinación del modelo de simulación con ambos algoritmos es aplicada a un sistema de manufactura de puertas de acero, donde las operaciones son extensas y variadas. Representar la gran cantidad y tipo de detalles que el sistema presenta es el reto a modelar, tales como tiempos de preparación de equipos, el programa de mantenimiento, los procesos de carga y descarga de material, el empaque, la transferencia de partes entre estaciones de trabajo, obedeciendo las reglas de despacho que se tienen en el sistema, las reglas de almacenamiento en los diversos puntos de transferencia, los retrasos en entregas de material, los turnos de trabajo, los paros programados, los almuerzos, los recursos para el manejo de material, los procesos, entre otros detalles.

El sistema de manufactura de puertas de acero es complejo y amplio para modelar. Con más de 50 productos que son posibles de producir en una mezcla de estaciones de trabajo y/o celdas de manufactura en configuración por procesos y bajo una configuración tipo jobshop flexible. Este

sistema contiene más de 100 tipos de operaciones diferentes que pueden ser realizadas a lo largo de los diversos departamentos involucrados. Cada tipo de puerta tiene una diferente ruta de construcción que puede ser realizada en diversas celdas equivalentes. Además, el contenido de trabajo entre cada tipo de puerta es sensiblemente distinto.
Como muchos sistemas de manufactura, una vez que se inicia la producción, la secuencia de esta puede ser modificada debido a diferentes eventos que pueden ocurrir, originando con ello producción en proceso que se almacena hasta una nueva indicación.
Debido a las condiciones de comercialización, los compromisos en entregas deben cumplirse, de lo contrario la compañía incurre en costos de maquila o bien en tiempo extra para cumplir con los requerimientos del cliente.
Desde el punto de vista computacional, encontrar una secuencia que sea mejor que las demás sobre un conjunto de soluciones es una tarea extensa. Tan solo el hecho de buscar secuencias de producción para una cantidad limitada de trabajos requiere un tiempo computacionalmente importante. Un ejemplo, es tan solo con 30 trabajos que deben ser procesados en cualquier sistema de manufactura, requiere evaluar $2.65X10^{32}$ posibles secuencias. Esto es bajo el supuesto de que el producto se procese bajo una misma ruta de producción, entonces la evaluación se centra en las posibles permutaciones de los trabajos. Donde cada permutación de trabajos es una secuencia de producción definida. La evaluación entonces se centra en $n!$ posibles secuencias. Donde n es el número de trabajos a secuenciar. Sin embargo, en sistemas de manufactura como el mencionado, cada trabajo tiene una ruta distinta o diferente, lo que conlleva a considerar la cantidad de máquinas o equipos involucrados en el proceso. La evaluación ahora es $(n!)^m$ para calcular las posibles secuencias, donde m es el número de equipos o máquinas en el sistema.
En vez de evaluar todas las posibles soluciones, se pretende explorar el espacio de soluciones a través de un algoritmo que rápidamente encuentre buenas soluciones y garantice haber revisado la amplia región de factibilidad del problema.

Desde el punto de vista práctico, encontrar buenas soluciones, es decir, secuencias que optimicen alguna medida de desempeño es el fin. Como medida de desempeño a evaluar se encuentra la producción en proceso, la cual representa en esta investigación a aquellas puertas de acero que no han sido concluidas en su proceso de manufactura y que generan costos por mantenerlas (espacio, inventario, control y administración del material) dentro del sistema. Entonces, encontrar el mejor programa de producción que está compuesto tanto de la mejor secuencia como del mejor calendario laboral para manufacturar puertas de acero que reduzcan o minimicen la producción en proceso es la meta. Sin embargo, encontrar la relación entre secuencias de producción ideales y un calendario laboral que permita minimizar dicha producción en proceso es el reto a modelar a través del Algoritmo de Estimación de Distribuciones.

El proceso de manufactura de puertas de acero, donde las operaciones son extensas y diversas, pertenece a la configuración jobshop flexible. Wang *et al* (2012) explica la formulación para esta configuración.
El problema de programación en configuración jobshop flexible FJSP (por sus siglas en ingles Flexible Jobshop Scheduling Problem) es comúnmente definido como sigue: existen n trabajos $J = \{J_1, J_2, \dots, J_n\}$ para ser procesados sobre m máquinas $M = \{M_1, M_2, \dots, M_n\}$. Un trabajo J_i es constituido por un conjunto de n_i operaciones $\{O_{i,1}, O_{i,2}, \dots, O_{i,n_i}\}$ para ser ejecutadas una después de otra acorde a una secuencia dada. La ejecución de la operación $O_{i,j}$ requiere una máquina $m_{i,j}$ del conjunto de máquinas $M_{i,j} \subseteq M$. Cada operación debe ser procesada sin interrupción una vez que esta inicia.
Para simplificar la notación del problema de programación en configuración jobshop flexible FJSP en esta investigación, es conveniente identificar cada operación $O_{i,j}$ mediante números 1,…, N donde

$N \coloneqq \sum_{i=1}^{n} n_i$. El tiempo de procesamiento de la operación i sobre la máquina $k \in M_i$ está definido por $t_{i,k}$. Además, el conjunto de operaciones es establecido como O.
Sea $J(i)$ el trabajo al cual la operación i pertenece y sea $P(i)$ ser la posición que ocupa la operación i en la secuencia de operaciones que pertenece al trabajo $J(i)$ iniciando con uno, es decir, $P(i) = 1$ si la operación i es la primera operación de un trabajo. Además, el conjunto I_k definido por $I_k \coloneqq \{i \in O | k \in M_i\}$ denota las operaciones $i \in O$ que pueden ser procesadas sobre la máquina k. Por lo que existen $|I_k|$ posiciones sobre la máquina k.

Para modelar la asignación de las operaciones sobre las maquinas, variables binarias de asignación $x_{i,k,p}$ para todo $p_k = 1, \dots, |I_k|$, $k = 1, \dots, m$, $i \in O$ son introducidas. Si $x_{i,k,p}$ = 1 significa que la operación i es programada para la posición p sobre la maquina k. Además, S_i es definido como el tiempo de inicio para la operación i.
Para cada trabajo, las operaciones correspondientes tienen que ser procesadas en el orden definido, esto es, el tiempo de inicio de una operación no debe ser antes que el punto en cual la operación precedente en la secuencia de operaciones del respectivo trabajo es completado. Esta restricción es impuesta simultáneamente sobre todos los pares apropiados de operaciones, definidos por el conjunto $C \coloneqq \{(i,j) | \, i,j \in O : J(i) = J(j) \wedge P(j) = P(i) + 1\}$. Las restricciones de precedencia están dadas por

$$S_i + \sum_{k \in M_i} \sum_{p=1}^{|I_k|} x_{i,k,p} t_{i,k} \leq S_j \text{ para todo } (i,j) \in C. \quad \text{(Ec. 0.1)}$$

Además, cada operación tiene que ser asignada exactamente a una sola posición, esto se asegura por medio de

$$\sum_{k=1}^{m} \sum_{p=1}^{|I_k|} x_{i,k,p} = 1 \text{ para toda } i \in O. \quad \text{(Ec. 0.2)}$$

Adicional, solamente una operación puede ser asignada a cada posición, debido a las restricciones

$$\sum_{i \in O} x_{i,k,p} \leq 1 \text{ para toda } p = 1, \dots, |I_k|, k = 1, \dots, m. \quad \text{(Ec. 0.3)}$$

Las posiciones sobre cada máquina tienen que ser asignadas una después de otra, esto es, una operación se asigna a una posición si la posición precedente ya está asignada. Esta condición es asegurada por

$$\sum_{i \in O} x_{i,k,p} \leq \sum_{i \in O} x_{i,k,p-1} \text{ para toda } p = 2, \dots, |I_k|, k = 1, \dots, m. \quad \text{(Ec. 0.4)}$$

Con el fin de interconectar las variables de posición-asignación de la máquina con las variables de tiempo de inicio y obtener un programa factible, restricciones de no traslape son definidas por medio de

$$S_i + t_{i,k} - M\left(2 - x_{i,k,p-1} - x_{j,k,p}\right) \leq S_j \quad \text{(Ec. 0.5)}$$

para toda $p = 2, \dots, |I_k|$, $i \neq j \in I_k$, $k = 1, \dots, m$. Si las operaciones i y j son asignadas a la misma máquina k para posiciones consecutivas $p-1$ y p, entonces el tiempo de inicio S_j de la operación j no debe ser antes que el tiempo de terminación $S_i + t_{i,k}$ de la operación i. M es una constante suficientemente grande para garantizar las restricciones (Ec. 0.5). Si al menos una de las variables de posición $x_{i,k,p}$ y $x_{i,k,p-1}$ es cero, es decir, las operaciones i y j no están asignadas en posiciones consecutivas sobre la misma máquina, la restricción no tiene que ser tomada en cuenta.

Diferentes tiempos de inicio para cada máquina son considerados en esta investigación. Sea T_k el tiempo libre o inactivo para cada máquina k, entonces el tiempo de inicio para la primer operación sobre cada máquina no debe ser antes que T_k, esto es asegurado por

$$T_k + S_i + \sum_{p=1} x_{i,k,p} t_{i,k} \leq S_j \text{ para toda } i \in I_k, k = 1, \dots, m. \quad \text{(Ec. 0.6)}$$

Sea H el tiempo de un turno de trabajo para el proceso de manufactura. Entonces, el turno real de operación R para cada máquina k se define como

$$T_k + H = R_k \text{ para toda } k = 1, \dots, m. \quad \text{(Ec. 0.6)}$$

El tiempo de culminación C_k sobre cada máquina k es definido como el tiempo total requerido para concluir todas las operaciones programadas, el cual se asegura a través de

$$max_{i \in I_k}\{S_i + t_{i,k}\} \leq C_k \text{ para toda } k = 1, \dots, m. \quad \text{(Ec. 0.7)}$$

Se considera minimizar la producción en proceso al final de cada turno real sobre cada máquina, es decir, la diferencia en tiempo entre el tiempo de culminación y el turno real de cada máquina, denotado por

$$Min\ WIP = \sum_{k=1}^{m} \max\{C_k - R_k, 0\} \quad \text{(Ec. 0.8)}$$

Así, el modelo matemático para el proceso está dado por

$$Min\ WIP = \sum_{k=1}^{m} \max\{C_k - R_k, 0\}$$

s.a.

$$max_{i \in I_k}\{S_i + t_{i,k}\} \leq C_k \qquad \text{para toda } k = 1, \dots, m.$$

$$S_i + \sum_{k \in M_i} \sum_{p=1}^{|I_k|} x_{i,k,p} t_{i,k} \leq S_j \qquad \text{para todo } (i,j) \in C.$$

$$\sum_{k=1}^{m} \sum_{p=1}^{|I_k|} x_{i,k,p} = 1 \qquad \text{para toda } i \in O.$$

$$\sum_{i \in O} x_{i,k,p} \leq 1 \qquad \text{para toda } p = 1, \dots, |I_k|, k = 1, \dots, m.$$

$$\sum_{i \in O} x_{i,k,p} \leq \sum_{i \in O} x_{i,k,p-1} \qquad \text{para toda } p = 2, \dots, |I_k|, k = 1, \dots, m.$$

$$S_i + t_{i,k} - M(2 - x_{i,k,p-1} - x_{j,k,p}) \leq S_j \quad \text{para toda } p = 2, \dots, |I_k|, i \neq j \in I_k, k = 1, \dots, m.$$

$$T_k + S_i + \sum_{p=1} x_{i,k,p} t_{i,k} \leq S_j \qquad \text{para toda } i \in O, k = 1, \dots, m.$$

$$T_k + H = R_k \qquad \text{para toda } k = 1, \dots, m.$$

$$S_i \geq 0 \qquad \text{para toda } i \in O.$$

$$T_k \geq 0 \qquad \text{para toda } k = 1, \dots, m.$$

$$x_{i,k,p} \in \{0,1\} \qquad \text{para toda } p = 1, \dots, |I_k|, k \in M_i, i \in O.$$

H constante

Aunque la formulación matemática del problema ha sido explicada claramente, esta incluye supuestos que no son relevantes o consistentes con el proceso de manufactura mencionado. Algunos de estos supuestos simplemente no pueden ser aplicados en el proceso de manufactura de puertas de acero desafortunadamente. Esta situación impide utilizar EDAs directamente. Algunos supuestos que no se cumplen son:

1.-Las máquinas se configuran en serie. En el proceso de manufactura de puertas de acero esto no es posible, especialmente por un layout que no logra una configuración de ese tipo.

2.-El almacén o las capacidades de los buffers entre sucesivas máquinas pueden ser virtualmente ilimitados. Desafortunadamente, esto no ocurre con el proceso de manufactura actual. Cuando los productos son físicamente tan grandes como las puertas de acero, el espacio en el buffer entre dos sucesivas máquinas tiene una capacidad limitada, causando bloqueo. Cuando esto ocurre, el trabajo tiene que permanecer en la máquina, ocasionando un trabajo en cola.

3.-Cualquier trabajo puede procesarse por cualquier máquina en cualquier etapa. Aunque algunas estaciones de trabajo en el proceso de manufactura de puertas de acero tienen máquinas paralelas para procesar cualquier trabajo que se presente en estas, el impacto en el desempeño puede ser totalmente diferente usando una máquina a otra. La principal razón es porque hay un número limitado de trabajadores calificados para utilizar esas máquinas paralelas.

4.-Las operaciones no pueden ser interrumpidas. Existen muchas razones para interrumpir las operaciones en cualquier estación de trabajo o máquina tales como fallas, ajustes, preparaciones incorrectas, desperdicios, y trabajos con prioridad.

5.-Cada máquina puede procesar solamente una operación a la vez. En el proceso de manufactura actual, el sistema de horno de curado actúa como una máquina, procesando más de cien puertas (de diferentes trabajos) al mismo tiempo.

6.-Un tipo de máquina está disponible. Cuando los trabajos han ingresado al piso de producción, estos son producidos acorde a una específica ruta como en cualquier configuración tipo jobshop normalmente; sin embargo, existen grupos de trabajadores que pueden no estar disponibles debido a la programación de los turnos de trabajo.

7.-El tiempo para transferir trabajos entre máquinas no es relevante. En el proceso mencionado, la mayoría de los trabajos necesitan ser transferidos entre máquinas o estaciones de trabajo por dollies, plataformas o montacargas para continuar el proceso, y estas transferencias consumen tiempo.

8.- El tiempo de procesamiento es fijo o conocido de antemano. La mayoría de las piezas requieren un procesamiento similar, pero las características específicas requeridas para cada modelo causan una cierta variación en el contenido de trabajo real. Debido a esta singularidad, las piezas requieren diferentes cantidades de recursos y tiempos de procesamiento.

9.- Todos los trabajos están disponibles en el tiempo 0. No todos los trabajos están disponibles al inicio del horizonte de estudio. Estos llegan durante todo el horizonte de estudio.

Además, las condiciones de operación para el proceso de manufactura de puertas de acero son diferentes y más sensibles que las configuraciones clásicas donde las condiciones de operación pueden ser irrelevantes, por ejemplo:
a) Reglas de almacenamiento para la producción en proceso en las estaciones de trabajo. Cuando un trabajo finaliza su proceso en alguna máquina, este continúa al siguiente proceso en otras máquinas normalmente, pero algunos procesos necesitan ser hechos por lotes. En tal caso, el trabajo tiene que esperar hasta que un cierto número de trabajos ya han completado el proceso.
Mientras esperan, los operadores apilan los trabajos. Finalmente cuando el proceso puede continuar, los operadores en otra estación toman los trabajos uno por uno del último al primero modificando la secuencia original.
b) Reglas de transferencia para materia prima y producción en proceso. Por lo general, cuando un trabajo está listo para ir a otra estación de trabajo o máquina para iniciar el siguiente proceso, simplemente se va, pero a veces tiene que esperar hasta que una regla de carga sea satisfecha. Esto es común en el proceso real de manufactura, donde la transferencia de las puertas se realiza mediante plataformas o dollies, que no se pueden mover hasta que haya un mínimo de diez puertas.
c) Reglas de carga en el conveyor. Cuando una máquina está disponible, se puede procesar otro trabajo comúnmente. En el caso del proceso de pintura, sin embargo, un transportador elevado transfiere puertas y marcos para el horno de curado, pero no siempre. A pesar de que contiene más de un centenar de ganchos para cargar, no pueden ser totalmente utilizados, debido al tamaño de las puertas o marcos. Los operadores tienen que dejar algunos espacios vacíos con el fin de evitar el contacto entre las puertas o marcos en las curvas del transportador. Esos espacios (ganchos) son entonces máquinas ociosas.
d) Políticas de arranque de turno. Cuando las máquinas se encienden, cualquier trabajo pueda ser procesado. No obstante ciertas herramientas o máquinas necesitan alcanzar algún parámetro crítico, como la temperatura mínima de operación.
e) Capacidades. En muchos casos, el personal de operaciones está disponible cuando se necesita para producir un determinado trabajo, pero el proceso de manufactura puede requerir diferentes capacidades de las máquinas y estaciones de trabajo debido a que el personal programado afecta las capacidades reales del proceso mencionado.

El proceso de manufactura de puertas de acero contiene diversas estaciones de trabajo asociadas con diferentes productos. Cada tipo de puerta pasa a través de una diferente secuencia de etapas de procesamiento; además, el contenido de trabajo varía en cada paso. A diferencia de otros procesos de fabricación, una vez que la producción comienza, la secuencia puede ser cambiada, almacenándose como producción en proceso. Cuando el proceso de manufactura llega a saturarse, las partes deben ser construidas con tiempo extra o maquilando con el fin de satisfacer la demanda extra, lo que resulta en costos más altos y tiempos de entrega tardíos.

Los requerimientos de procesamiento de los trabajos son:
- Estaciones de trabajo específicas debe usarse para cada tipo de trabajo. No obstante, este proceso de manufactura es flexible y algunos trabajos pueden ser procesados en diferentes estaciones de trabajo.
- Cada trabajo involucra un conjunto de operaciones.
- Las precedencias de las operaciones varían de trabajo a trabajo

3. HIPÓTESIS

La relación que existe entre la secuencia de procesamiento de las operaciones, la asignación de las operaciones a las máquinas y la formulación de horarios diferentes asociados a cada máquina determinan los niveles de producción en proceso que se tienen en cada equipo.

4. OBJETIVO GENERAL DE LA INVESTIGACIÓN

Diseñar e implementar un EDA de tipo continuo basado en tres modelos gráficos probabilísticos, que de manera conjunta estimen la distribución de probabilidad de las soluciones representadas por el secuencia de procesamiento de las operaciones, la asignación de las operaciones a las máquinas y el tiempo inactivo de cada máquina para combinarlo con un modelo de simulación y poder evaluar cada solución individual del conjunto de soluciones factibles.

4.1. OBJETIVOS ESPECÍFICOS DE LA INVESTIGACIÓN

Algunos objetivos se derivan del objetivo primario:

- Evaluar el desempeño de un EDA continuo comparándolo con un GA en la programación y secuenciamiento de operaciones del sistema de manufactura bajo estudio.
- Implementar la comunicación requerida y necesaria entre un EDA continuo y un modelo de simulación para evaluar escenarios que optimicen el desempeño del sistema de manufactura mencionado.
- Modelar, verificar y validar un modelo de simulación derivado del sistema de manufactura bajo estudio que permita reflejar soluciones propuestas de programación en el sistema.
- Programar secuencias de trabajos definidas en un sistema de manufactura a través de la estimación de la dependencia condicional de las mismas, utilizando para ello un modelo de simulación apoyado en un EDA.
- Generar programas de secuenciamiento de N diferentes trabajos que requieren ser procesados sobre M estaciones de trabajo, basados en diferentes K turnos de trabajo, en un proceso de manufactura flexible de puertas de acero para obtener el mínimo de producción en proceso a través del EDA. El EDA es usado para guiar el proceso global de búsqueda e identificar los mejores programas de secuenciamiento.

5. ORGANIZACIÓN DE LA INVESTIGACIÓN

Esta tesis está dividida en tres capítulos.
En el capítulo 1 se realiza una descripción de las principales escuelas, enfoques y teorías existentes sobre el tema objeto de estudio, es decir, la programación y secuenciamiento de operaciones. Se muestra el nivel de conocimiento de dicho campo y las principales técnicas empleadas en su solución. Además se precisan los distintos conceptos especializados y relevantes que se utilizan a lo largo de esta tesis.

En el capítulo 2 se describe detalladamente la metodología utilizada en la solución del problema relativo a la programación y secuenciamiento de las operaciones en un sistema de manufactura de puertas de acero. Se establece el procedimiento global para construir un modelo de simulación que representa de manera fiel y creíble el sistema de manufactura mencionado. Además, el planteamiento de un Algoritmo Genético y como fue construido es explicado en este mismo capítulo. Por último, un Algoritmo de Estimación de Distribuciones se desarrolla enfatizando las diferencias y similitudes con otros algoritmos y el porqué de sus ventajas con respecto a otros.

El capítulo 3 detalla los resultados que se obtienen al comparar el desempeño de ambos algoritmos entre sí, combinándolos con el modelo de simulación mencionado. Pruebas estadísticas son realizadas para determinar de manera objetiva las diferencias entre estos. Finalmente se aclara las conclusiones que se tienen derivadas de esta investigación y se visualiza el trabajo futuro a partir de los resultados obtenidos.

MARCO TEORICO

1. PROGRAMACIÓN Y SECUENCIAMIENTO DE OPERACIONES

La programación y secuenciamiento de las operaciones es un proceso de toma de decisiones que es utilizado regularmente en muchas compañías manufactureras e industrias de servicio. En general, se trata de establecer que recursos serán utilizados para realizar ciertas tareas en un determinado periodo de tiempo logrando así optimizar algún objetivo.
Los recursos y tareas en una organización pueden ser concebidos de diferentes formas. Los recursos por ejemplo, pueden ser máquinas en un taller de trabajo, pistas de un aeropuerto, cuadrillas de personal en alguna construcción, etc; las tareas por su parte, pueden ser operaciones en algún proceso de producción, las salidas o llegadas en un aeropuerto, las etapas en un proyecto de construcción, etc. Cada tarea puede tener cierta prioridad de ejecución sobre otras, por ejemplo aquellas que se culminan más pronto o bien aquellas que tiene un vencimiento próximo. El objetivo puede también tomar diferentes formas. Uno de ellos puede ser minimizar el tiempo requerido para completar todas las tareas asignadas y otro minimizar la diferencia entre el tiempo de terminación y la fecha de vencimiento.

La programación y secuenciamiento de las operaciones juega un importante rol en la mayoría de los sistemas de manufactura y de producción, así como en la mayoría de ambientes de procesamiento de información y datos. También es importante en la transportación y distribución de bienes y sin olvidar otros tipos de industrias de servicios. Para mostrar dicho valor se ilustran a continuación algunos ejemplos:

Considere una fábrica que produce bolsas de papel para cemento, carbón, alimento para mascotas, entre otros. La materia prima básica para dichas bolsas son rollos de papel. El proceso de producción consiste en tres etapas: la impresión del logo, el pegado del fondo de la bolsa, y la costura de los extremos de la misma. Cada etapa consiste de un número de máquinas la cuales no son necesariamente idénticas. Las máquinas en cada etapa pueden diferir ligeramente en la velocidad a la cual operan, el número de colores que pueden imprimir, o el tamaño de la bolsa que pueden producir. Cada orden de producción indica un tipo y una cantidad determinada de bolsa que debe producirse y embarcarse de acuerdo a una fecha compromiso. Los tiempos de procesamiento para las diferentes operaciones son proporcionales al tamaño de la orden, es decir, al número de bolsas ordenadas.
Una entrega tardía implica una penalización en la forma de pérdida de credibilidad e imagen y la magnitud de la penalización depende del cliente o de la importancia de la orden. Uno de los objetivos de la programación y secuenciamiento de las operaciones seria minimizar la suma de dichas penalizaciones.

Otro caso es la manufactura de microprocesadores. El proceso de producción usualmente consiste en cuatro fases; la fabricación del componente electrónico, las pruebas al circuito, ensamble y empaque.
La fabricación del componente es la fase más compleja. Las capas de metal son mezcladas con silicio para producir los circuitos. Cada capa requiere un número de operaciones que típicamente incluyen; limpieza, oxidación, metalizado, litografía, ionizado, e inspección. Como los circuitos se componen de capas, cada una de ellas debe someterse a estas operaciones varias veces. Por lo que existe una gran cantidad de recirculación en el proceso. Las capas se envían en lotes de 24. Algunas máquinas requieren ajustes para adecuarse a los trabajos que ingresan al sistema. El tiempo de los ajustes varía

dependiendo del lote que va a salir y del lote que esta por ingresar. La meta principal puede ser minimizar los tiempos muertos por ajustes en máquinas o bien, maximizar la utilización del equipo.

Una terminal aérea donde se encuentran docenas de accesos para abordar y cientos de aviones llegando y partiendo cada día. El tamaño de los accesos para abordar no son idénticos ni los aviones tampoco. Algunos accesos están localizados por su tamaño en lugares con gran cantidad de espacio donde los aviones con fuselaje ancho pueden ser estacionados. Otros accesos se encuentran en lugares donde es difícil llevar a los aviones fácilmente hasta allí, a menos que estos sean jalados.
Los aviones llegan y parten de acuerdo a un programa. Sin embargo, el programa está sujeto a cierta cantidad de aleatoriedad. Durante el tiempo que un avión ocupa un acceso el arribo a este por los pasajeros puede ser retrasado por alguna reparación al mismo y la salida por ende retrasada.
El programa de llegadas y salidas puede verse por las fechas compromiso. Los indicadores de desempeño de la terminal pueden estar relacionadas dichas fechas. Otro caso de aleatoriedad es cuando se sabe de antemano que algún avión no podrá aterrizar en su hora de llegada prevista al siguiente aeropuerto debido a alguna congestión, entonces el avión no despega –política para evitar gasto de turbosina–. Otro más es cuando el avión no puede despegar, los pasajeros deben permanecer en la terminal en lugar del avión. Por último, si el abordaje es pospuesto, un avión debe permanecer en el acceso por un periodo de tiempo adicional.
El programador tiene que asignar aviones a los accesos adecuados que están disponibles en base a los tiempos de llegada de estos. Un objetivo adecuado entonces es minimizar la cantidad de retrasos.

Finalmente, considere el siguiente ambiente de manufactura. Ordenes que ingresen al sistema traducidas a trabajos asociados a una fecha de entrega. Estos trabajos deben ser procesados en máquinas dentro de estaciones de trabajo en un determinado orden o secuencia. El procesamiento de los trabajos puede demorarse si cierta máquina se encuentra ocupada o bien si algún trabajo con una prioridad mayor llega retrasando aún más los trabajos en espera. Cualquier imprevisto en el piso de manufactura, tal como fallas en las máquinas o tiempos de proceso más largos de lo acostumbrado, deben ser considerados en el programa y secuencia de las operaciones. En un entorno como este, un programa detallado de tareas ayuda a mantener eficiencia y control de las operaciones.
El piso de producción no es solo lo que impacta al programa y secuencia de las operaciones. Este también es afectado por el proceso de planeación de producción el cual administra la planeación a mediano y largo plazo. Este último nivel de planeación intenta optimizar la mezcla de productos y asigna los recursos para ello basado en los niveles de inventario, pronósticos sobre la demanda, y requerimientos de operadores. La Figura 1.1 muestra un diagrama del flujo de información en un sistema de manufactura y la posición que tiene el programa y secuencia de las operaciones en el sistema.

Figura 1.1 Flujo de información en sistemas de manufactura.

La programación y secuencia de las operaciones ha sido una de las áreas más ampliamente investigadas en la disciplina de la investigación de operaciones, debido en gran parte a la amplia variedad de problemas que se presentan en este campo. La cantidad de artículos científicos que se pueden encontrar en la red científica con el tópico de *"programación y secuencia de operaciones"* está cerca de 200 publicaciones por año y ascendió a 300 en el 2005, 2006, y 2007. Se puede decir que la programación y secuencia de las operaciones se remonta al inicio del siglo XX con Gantt (1916) al discutir un problema de programación. Sin embargo, fue hasta después de 40 años que un conjunto de publicaciones sobre programación comenzó a aparecer.
La evolución de la programación puede dividirse por secciones claramente explicadas en la contribución de Potts & Strusevich (2009) y que en parte se mencionan a continuación:

1.1. ANÁLISIS COMBINATORIO.

Las contribuciones a la programación y secuencia de las operaciones al inicio fueron acerca de la introducción de modelos y el uso de análisis combinatorio para desarrollar algoritmos. El enfoque fue principalmente basado en la programación de un conjunto de trabajos sobre un conjunto de máquinas.

La aportación de Johnson (1954) ofrece el inicio de la programación y secuencia de las operaciones como área independiente a la investigación de operaciones. Johnson consideró el modelo de producción como un proceso en línea. En general, el proceso en línea está dado por un conjunto de trabajos N = {1, 2, ... ,*n*} y *m* máquinas sucesivas 1, 2, ... *m*, donde $m \geq 2$. Cada trabajo *j* consiste en *m* operaciones, O_{1j}, O_{2j}, ... O_{mj}, donde cada operación O_{ij} se procesa en la máquina *i* con un tiempo p_{ij}. Cada trabajo *j* tiene la misma ruta en el proceso; el orden entonces de sus operaciones está dado por la secuencia (O_{1j}, O_{2j}, ... O_{mj}), o bien, cada trabajo *j* visita a las máquinas en el orden (1, 2, ... *m*).
El principal resultado obtenido por Johnson (1954) es un procedimiento de solución elegante que resuelve el problema de proceso en línea para 2 máquinas y minimiza el plazo que se requiere para completar todos los trabajos en todas las máquinas, conocido en la literatura como *"makespan"*. Johnson demostró que su procedimiento encuentra la solución óptima entre la gran cantidad de programas que pueden existir sin importar el número de trabajos a programar, ya que es el mismo orden para ambas máquinas; nombrando este tipo de programas como *programas de permutaciones*. Además, Johnson (1954) demuestra que existe un programa óptimo en el cual un trabajo *j* es secuenciado antes que algún otro trabajo *k*, si el $\min\{p_{1j}, p_{2k}\} \leq \min\{p_{2j}, p_{1k}\}$. El algoritmo más común está basado en la siguiente propiedad: una secuencia óptima inicia con los trabajos en los cuales $p_{1j} \leq p_{2j}$ ordenados de forma creciente de p_{1j}, seguido por el resto de los trabajos en forma decreciente de p_{2j}.

Para una sola máquina todas las secuencias producen el mismo *makespan*, entonces otros objetivos fueron de interés. Adicional, los primeros resultados sobre programación de una máquina aparecieron inmediatamente después del trabajo de Johnson, y estos ofrecen algoritmos basados en *reglas de prioridad*.

Dado un conjunto de trabajos, el retraso de un trabajo j está definido como $C_j - d_j$, donde C_j es el tiempo que se requiere para completar el trabajo j y d_j es la fecha de vencimiento de dicho trabajo. Jackson (1956) demuestra que la secuencia la cual minimiza el máximo retraso posible esta dado al utilizar la regla de prioridad *EDD*, fecha de vencimiento más próxima, (del inglés *Earliest Due Date*). Donde los trabajos son ordenados de forma creciente acorde a la fecha de vencimiento.

Smith (1956) ofrece un regla de prioridad para minimizar el promedio del tiempo que se requiere para que un trabajo permanezca en el sistema o máquina, al dividir el tiempo de culminación de todos los trabajos entre el número de estos $\sum C_j$. Esta regla es una medida de desempeño utilizada en la Teoría de colas. Una permutación es obtenida por la secuencia de trabajos en orden creciente basado en sus tiempos de procesamiento; dicha regla es conocida como *SPT*, tiempo de procesamiento más corto, (del inglés *Short Processing Time*). Además, Smith (1956) demostró que si cada trabajo j tiene un peso asociado w_j el cual representa su importancia relativa, la suma de los tiempos de terminación ponderada $\sum w_j C_j$ es minimizada por la secuencia de trabajos en orden decreciente de la relación p_j/w_j; esta regla de prioridad es frecuentemente llamada regla de Smith.

McNaughton (1959) estudio problemas donde existen m máquinas paralelas, siendo $m \geq 2$. Para minimizar el *makespan*, un programa se define asignando el tiempo de procesamiento de los trabajos a las máquinas. Dado que ningún trabajo puede ser procesado por varias máquinas a la vez, se puede deducir que el trabajo con mayor duración sirve como límite inferior basado en un trabajo para encontrar el óptimo. Además, para cualquier asignación de trabajos a las máquinas, existe una máquina la cual contiene una carga de al menos

$$\sum_{j \epsilon N} p_j / m \qquad \text{(Ec. 1.1)}$$

unidades de tiempo; este valor, la carga promedio de la máquina, es un límite inferior basado en una máquina para encontrar el óptimo. Sea LI que denota el máximo de los límites inferiores, entonces

$$LI = \max\left\{ \frac{1}{m} \quad \sum_{j \epsilon N} p_j \quad , \max_{j \epsilon N} p_j \right\} \qquad \text{(Ec. 1.2)}$$

Si cada trabajo es procesado sin interrupción, entonces el *makespan* puede ser más grande que el LI, y significará que el LI no está ajustado. Ahora bien, si existe prioridades entre los trabajos, el *makespan* será igual que el LI. En un programa con prioridades, un trabajo en proceso puede ser interrumpido en cualquier momento para ingresar una prioridad y reanudar posteriormente, ya sea en la misma máquina o en otra, bajo la condición de que la duración total de todos los tiempos de procesamiento parciales es igual al tiempo de procesamiento total del trabajo interrumpido. McNaughton (1959) ofreció un simple algoritmo que encuentra el óptimo en un programa con prioridades. Imagine que todos los trabajos son procesados por una máquina ficticia sin tiempo inactivo al inicio o entre los trabajos, y se corta el programa en m segmentos de longitud LI (excepto tal vez el último que puede ser más corto), e interpretar cada uno de estos como una asignación con prioridad de los trabajos a las máquinas paralelas originales.

Al final de los años 50, los problemas de programación de una máquina habían formado una rama independiente de la investigación de operaciones, con sus propios problemas y enfoques de solución. Se clasifica entonces los sistemas de producción en una sola etapa, ya sea con una máquina o bien con máquinas en paralelo, y en multi-etapa con máquinas operando en serie.
Los trabajos procesados en sistemas de una sola etapa consisten de una sola operación que se desarrolla ya sea en una sola máquina o bien en *m* máquinas en paralelo. A su vez, las máquinas en paralelo pueden ser idénticas, uniformes o bien sin relación. Sea p_{ij} el tiempo de procesamiento de cualquier trabajo *j* si este es procesado sobre una máquina *i*, donde $p_{ij} = p_j$ si las máquinas paralelas son idénticas, y $p_{ij} = p_j/s_i$, donde s_i denota la velocidad de la máquina *i* si las máquinas son uniformes. En el caso de máquinas no relacionadas los valores de p_{ij} son arbitrarios.
Por el contrario, los sistemas multi-etapa, cada trabajo requiere de diversas operaciones, cada una realizada por una máquina específica. Una distinción entonces fue hecha entre proceso en línea, donde todos los trabajos tienen una misma ruta de proceso, y proceso por orden, donde cada trabajo tiene una ruta de proceso distinta. Normalmente la función objetivo a minimizar, depende de los tiempos de terminación de los trabajos, pero existen otras de interés además del *makespan*, como el máximo retraso o el número de trabajos retrasados.

Jackson (1956) extendió los resultados de Johnson (1954) para el problema de dos máquinas y así minimizar el *makespan*, siempre y cuando cada trabajo requiera a lo mucho dos operaciones. Jackson (1956) describió un algoritmo que encuentra un programa que minimiza el *makespan* para aquellos trabajos que tienen la misma ruta de procesamiento.
Akers & Friedman (1955) estudiaron el problema de procesamiento por orden con dos trabajos. Aquí, cada uno de los dos trabajos tiene su propia ruta de procesamiento, con posibles visitas repetidas a una misma máquina. Una interpretación elegante para encontrar el mínimo de *makespan* es encontrando la ruta más corta, en rectángulo con obstáculos rectangulares.
En los modelos clásicos de proceso en línea, la restricción principal en el procesamiento es el hecho de que un trabajo cualquiera puede iniciar en una máquina solamente si la operación previa ha sido completada. En diversas aplicaciones en la industria química y metalúrgica es indispensable que la siguiente operación de un trabajo inicie exactamente cuando la operación previa ha concluido. Esta condición es conocida como *"proceso sin espera"*. Piehler (1960) fue el primero en demostrar que el problema de *m* máquinas configuradas en proceso en línea sin espera entre operaciones para minimizar el *makespan* se reduce al problema del agente viajero. Recalcando que el problema del agente viajero en cuando este requiere encontrar el recorrido más corto (recorrido de Hamilton) para visitar cada una de las ciudades programadas exactamente una sola vez. El resultado obtenido por Piehler (1960) fue redescubierto posteriormente en diversas ocasiones. Gilmore & Gomory (1964) mostraron que para dos máquinas configuradas en proceso en línea sin espera se reduce a un problema del agente viajero con una estructura especial y por lo tanto admite un algoritmo más rápido.
Conway *et al* (1967) consideran el problema de minimizar el tiempo total de terminación de todos los trabajos sobre máquinas paralelas idénticas. Demostraron que un programa óptimo puede ser encontrado a través de la regla de prioridad SPT, pero con una adecuación. Específicamente, su procedimiento está basado sobre la idea de lo que ahora se conoce como *"programación de una lista"*. Se forma una lista de trabajos (por la regla SPT) y en el momento de que alguna de las máquinas esté disponible, se asigna el primer trabajo de la lista actual y se asigna a la máquina disponible.
Aunque se ha deseado identificar una técnica general que pueda responder a una gran gama de problemas de programación, se ha reconocido que muchos problemas por su naturaleza requieren la construcción de algoritmos para propósitos específicos. Como ejemplo el problema de minimizar el número de trabajos retrasados para una sola máquina. Parece natural utilizar la regla de prioridad EDD, pero resulta que la secuencia resultante no es necesariamente la óptima. Moore (1968) reenumera los trabajos en el orden de la regla EDD e intenta programar estos uno por uno hasta que algún trabajo *k*

no puede ser programado por su fecha de entrega. Esto implica que cualquier programa al menos tendrá un trabajo retrasado. Si todos los trabajos tienen la misma penalización al convertirse en trabajos retrasados entonces resulta útil retirar del programa aquel trabajo *j* donde $1 \leq j \leq k$, que tiene el tiempo de procesamiento más largo. El hacer esto crea mayor capacidad para programar el resto de los trabajos por sus fechas de vencimiento. El proceso se repite hasta que todos los trabajos están secuenciados.

Una considerable influencia en el desarrollo de la programación puede ser atribuido a dos libros que resumen los logros en programación muy temprano. Uno editado por Muth & Thompson (1963) y una investigación por Conway *et al* (1967). Es importante mencionar que ambos libros fueron traducidos a otros idiomas del original, lo que contribuyó a extender ampliamente la comunidad de investigadores en programación.

Los artículos incluidos en el libro de Muth & Thompson (1963) son principalmente elaborados para encontrar soluciones a problemas prácticos pero sencillos; incluyendo reglas de prioridad, programación entera, Monte Carlo, análisis estocástico, algoritmos enumerativos, entre otros. La aportación de Johnson (1954) es impreso en este libro. Otras contribuciones resolvieron problemas de 10 X 10 X 10 (10 trabajos, 10 máquinas, 10 operaciones por trabajo), motivando el desarrollo de métodos de búsqueda para los siguientes 20 años.
El libro de Conway *et al* (1967) combina tanto la teoría de programación de máquinas con tiempos de procesamiento determinísticos y la teoría de colas.
Es interesante notar que durante la primera década no había un entendimiento completo entre los investigadores respecto al concepto de complejidad computacional. También digno de mención, es el hecho de que todos los algoritmos descritos hasta ahora son lo que ahora llamamos algoritmos de tiempo-polinomial. Por lo general, este tipo de algoritmos fueron desarrollados originalmente para problemas de dimensiones pequeñas (una sola máquina, dos máquinas, dos operaciones por trabajo, o dos trabajos en procesos por orden). Muchos problemas de optimización combinatoria son esencialmente más complejos si algún parámetro numérico (trabajos o máquinas) crece más allá de dos. Parece que los investigadores se vieron intuitivamente conscientes de ese fenómeno.

1.2. RAMIFICACIÓN Y ACOTAMIENTO.

Las investigaciones sobre análisis combinatorio continúan en la década de los setentas, pero con un énfasis en problemas más dinámicos. Los resultados que se obtuvieron son válidos si ciertas condiciones se satisfacen. Se desarrollan algoritmos enumerativos como la programación dinámica y el método de ramificación y acotamiento.
El método de ramificación y acotamiento es un procedimiento de búsqueda enumerativo, para resolver problemas de optimización combinatoria. Este método fue desarrollado de manera independiente al final de los años cincuenta por Eastman (1958) para el problema del agente viajero y Land & Doig (1960) para programación entera.
El algoritmo de ramificación y acotamiento tiene las siguientes características:

a) Una regla de ramificación que indica como las soluciones son partidas en subconjuntos.
b) Una regla de acotamiento o bien de límite inferior para el caso de la minimización o límite superior para el caso de la maximización, que calcula el valor de la función objetivo de cualquier solución dentro del conjunto de soluciones bajo consideración.
c) Reglas de dominancia que ayudan a elegir o eliminar algunos subconjuntos de soluciones.

El desarrollo del método de ramificación y acotamiento es frecuentemente representado como un árbol de expansión, donde cada nodo representa un conjunto de soluciones, y las ramas corresponden a la partición de las soluciones acorde a la regla de ramificación.
Las primeras aplicaciones del método de ramificación y acotamiento a programación y secuenciamiento de operaciones se dan en la mitad de los años sesenta. Muchos de esos estudios fueron hechos para problemas de secuenciación de trabajos a producirse. Una regla de ramificación para programación fue típicamente utilizada. Así, la primera rama considera todos los posibles trabajos que pueden ser secuenciados en la primera posición. La segunda rama considera aquellos trabajos que pueden ocupar la segunda posición, y así sucesivamente. A pesar de ser un enfoque simplista, fue efectivo principalmente para reducir la búsqueda.

Las primeras aplicaciones del método de ramificación y acotamiento a problemas de programación involucran el clásico problema de procesos en línea, conocido como *Flowshop* en el cual existen *m* máquinas y *n* trabajos, donde cada trabajo *j* tiene un tiempo de procesamiento p_{ij} sobre cada máquina *i*. La mayoría de los estudios consideran una versión de permutación donde el mismo orden de procesamiento de los trabajos es utilizado sobre cada máquina, sin existir perdida de generalidad al programar más de una máquina si se busca minimizar el *makespan* (Potts & Strusevich, 2009). Los primeros estudios fueron realizados por Lomnicki (1965) e Ignall & Schrage (1965) quien independientemente desarrollaron algoritmos de ramificación y acotamiento para minimizar el *makespan* en problemas de *flowshop* para tres máquinas. Ignall & Schrage (1965) también ofrecieron un algoritmo para minimizar la suma del tiempo de culminación de los trabajos en problemas de *flowshop* para dos máquinas.
Nabeshima (1967) mejora esto al incluir en su propuesta cualquier tiempo muerto resultante del tiempo de procesamiento de las operaciones en la máquina anterior. El tiempo muerto es evaluado al resolver subproblemas de dos máquinas utilizando el algoritmo de Johnson (1954). Potts (1974) generaliza la propuesta de Nabeshima (1967) al incorporar el tiempo muerto derivado de las operaciones sobre cualquier máquina previa.

Al final de la década de los setenta, el mejor rendimiento del método de ramificación y acotamiento podía resolver problemas de programación hasta de 10 trabajos.

Otro problema que atrajo un significativo esfuerzo de investigación en el contexto del método de ramificación y acotamiento fue el de programar una máquina minimizando tanto la tardanza como la tardanza ponderada. En este problema, cada trabajo *j* tiene establecido una fecha compromiso de entrega d_j, y posiblemente dicho trabajo tiene un peso en importancia asociado w_j. Dado un programa en el cual cada trabajo *j* es completado en el tiempo C_j, la tardanza del trabajo *j* está definida por $T_j = \max \{ C_j - d_j, 0 \}$.
Para los problemas de la tardanza y tardanza ponderada para una máquina la regla de dominancia juega un fuerte papel en la restricción del espacio de búsqueda del método de ramificación y acotamiento. Algunas reglas de dominancia son particularmente útiles en situaciones de no linealidad de la función de la tardanza, reduciendo la búsqueda en el árbol de ramificación.
La mayoría de los métodos de ramificación y acotamiento utilizan la regla de ramificación para programación donde la primera rama fija el primer trabajo en la última posición de la secuencia, la segunda rama fija el segundo trabajo elegido en la segunda posición de la secuencia, y así sucesivamente. Esta regla tiene dos principales ventajas. La primera, es que los trabajos que se culminan más tarde por lo general tienen una mayor tardanza, así que fijar sus posiciones al inicio de la secuencia permite un acotamiento o bien de límite inferior ávido. La segunda, es que Elmaghraby (1968) muestra que si existe un trabajo que tiene cero tardanza donde esta secuenciado, entonces existe una solución óptima en la cual este trabajo es secuenciado en la última posición.

Por otro lado, una regla de acotamiento o límite inferior para minimizar la tardanza ponderada, está dado por el valor mínimo de las tardanzas máximas de los trabajos obtenidos en la secuencia de trabajos en orden EDD multiplicado por la importancia asociada al trabajo más pequeño. Shwimer (1972) utilizaron esta regla de acotamiento en su método propuesto de ramificación y acotamiento.

El desempeño de los métodos de ramificación y acotamiento introducidos durante la década de los años setenta fue dependiente de las características de los ejemplos generados: ejemplos con un rango pequeño de fechas compromiso para entrega. En cuanto a los problemas *flowshop*, ejemplos con menos de diez trabajos pudieron ser rutinariamente resueltos pero mayores a estos quedaron sin resolver en límites razonables de tiempo computacional.

A diferencia de los problemas *flowshop* donde un programa está definido por una secuencia de trabajos, el problema de programación para procesos por orden conocido como *Jobshop* requiere una secuencia de trabajos para cada una de las máquinas. Minimizar el *makespan* ha sido el principal objetivo en el caso *jobshop*. Debido a la estructura compleja del *jobshop*, las reglas de dominancia no juegan en este caso un papel importante para restringir la búsqueda.

Casi todos los problemas prácticos de programación y secuenciamiento de las operaciones pueden ser descritos en términos de la configuración *jobshop*, usualmente como versiones restringidas de este problema de optimización combinatoria. El problema puede ser definido de la manera siguiente: considere un ambiente de manufactura en el cual *n* trabajos deben ser procesados por *m* máquinas. Cada trabajo tendrá un conjunto de restricciones sobre el orden en el cual las máquinas deben ser utilizadas y el tiempo de procesamiento en cada máquina. Los trabajos pueden tener diferente duración y pueden utilizar diferentes subconjuntos de las *m* máquinas. La idea es encontrar la secuencia de trabajos en cada máquina para minimizar una función objetivo establecida.

Una representación útil del problema *jobshop* es un grafo disyuntivo propuesto por Roy & Sussman (1964). Este grafo tiene un nodo para cada operación más un nodo inicial y final con diversos arcos disyuntivos o conjuntivos. Existe un arco conjuntivo del nodo inicial al primer nodo que representa la primera operación de algún trabajo, de cada nodo que representa la primera operación de algún trabajo al siguiente nodo que representa la siguiente operación de algún trabajo, y del último nodo que representa la última operación de algún trabajo al nodo final. Existe un arco disyuntivo entre cada par de nodos que corresponden a las operaciones requeridas en la misma máquina. Una posible solución se obtiene mediante la orientación de cada arco disyuntivo para representar el orden de procesamiento de las operaciones correspondientes en la máquina de referencia, las orientaciones se eligen de tal manera que el gráfico resultante sea acíclico. Al asociar una importancia o ponderación de cero con el nodo inicial y final y una importancia o ponderación igual al tiempo de procesamiento de la operación correspondiente para todos los demás nodos, el *makespan* se puede encontrar mediante el cálculo de una trayectoria de ponderación máxima del nodo inicial al nodo final.
Dos tipos de reglas de ramificación han sido propuestas en la literatura. Németi (1964) introduce el concepto de *grafo disyuntivo ramificado*. Este concepto implica seleccionar un par de nodos conectados en un arco disyuntivo *u* y *v*, donde *u* y *v* corresponden a operaciones requeridas en la misma máquina. Dos ramas son creadas. En la primera, el arco disyuntivo es orientado para que las operaciones en *u* sean procesadas antes que las operaciones en *v*, mientras que la segunda rama realiza el proceso en orden inverso. Brooks & White (1965) proponen la *generación de un programa activo de ramificación*. Un programa activo es aquel donde no hay operación que pueda ser iniciada más pronto sin retrasar otra operación. Bajo esta regla de ramificación, una operación no secuenciada con el tiempo de procesamiento más corto debe ser la primera en elegirse junto con la máquina *i* que esta

requiere. Para cada operación que requiere la máquina *i* y que tiene un tiempo de inicio más pronto que es estrictamente menor que el más pequeño tiempo de procesamiento, una ramificación debe ser creada para secuenciar la operación en la primera posición vacante sobre la máquina *i*.

A pesar de los más sofisticados métodos de ramificación y acotamiento que estuvieron disponibles al final de la década, problemas de tamaño 10 X 10 X 10 (10 trabajos, 10 máquinas, 10 operaciones por trabajo), continuaron sin poder resolverse. Conway *et al* (1967) declararon que *"muchas personas competentes han considerado el problema, pero todos han salido con las manos vacías"*, aceptándose este pensamiento una década después.

Adicional al análisis combinatorio para generar reglas de dominancia para problemas *flowshop*, la actividad de investigación significativa fue dedicada a casos especiales en los cuales ciertas restricciones se colocan sobre los tiempos de procesamiento de las operaciones. Algunos otros contribuyen sustancialmente a esta área con numerosas publicaciones iniciando con Nabeshima (1961) y Szwarc (1968).

Para problemas en los cuales la estructura combinatoria clave es la selección o asignación, la programación dinámica ofrece ser una técnica de solución conveniente. Por ejemplo considerar el problema de la programación de trabajos sobre una máquina buscando minimizar el número ponderado de trabajos tardíos. Es sencillo observar que existe una solución óptima en la cual los trabajos a tiempo son secuenciados primero en el orden de la regla EDD seguido por los trabajos tardíos en un orden arbitrario. Así, el problema se reduce a seleccionar los trabajos que están en tiempo. Lawler & Moore (1969) introdujeron un algoritmo de programación dinámica para resolver este problema. Se define $F_j(t)$ como el número mínimo ponderado de trabajos tardíos para el subproblema involucrando los trabajos 1, 2, ... , *j*, donde el último trabajo en tiempo se finaliza en el tiempo *t*, con valor inicial $F_0(0) = 0$. La siguiente expresión calcula la función a minimizar y representa la ecuación de transición de un estado a otro en el algoritmo de programación dinámica:

$$Fj(t) = \begin{cases} \min\{F_{j-1}(t - p_j), F_{j-1}(t) + w_j\} & \text{para } t = 0, \dots, d_j \\ F_{j-1}(t) + w_j & \text{para } t = d_j + 1, \dots, T \end{cases} \qquad \text{(Ec. 1.3)}$$

Para $j = 1, \dots, n$, donde $T = \min\{ d_n, \sum_{j=1}^{n} p_j \}$ es un límite superior para el tiempo de finalización del último trabajo en tiempo. El número mínimo ponderado de trabajos tardíos es entonces $min_{t=0,\dots,T} \{F_n(t)\}$.
Rothkopf (1966) mostró que la programación dinámica puede ser utilizada para problemas de programación sobre máquinas paralelas idénticas para minimizar el *makespan*, el tiempo total de finalización ponderado y la tardanza máxima de los trabajos. Para este tipo de problemas, una vez que los trabajos han sido asignados a las máquinas, el orden de estos en cada máquina se hace en base en alguna regla de prioridad para el caso de una máquina. Así, estos problemas se reducen al de asignar trabajos a máquinas, y consecuentemente son susceptibles de solución con programación dinámica.

Aún no se lograron producir herramientas que llevaran a cabo un estudio sistemático de problemas de programación mediante el cual se pudieran hacer afirmaciones por ejemplo del tipo *"este problema es sencillo si se tienen 'x' condiciones satisfechas y se vuelve complejo si no las tiene"*. Los investigadores recurrieron a los métodos de ramificación y acotamiento para desarrollar soluciones que no se podían obtener basados en el análisis combinatorio. A pesar de que la intuición por lo general resulto ser correcta sobre los estudios realizados para obtener métodos de ramificación y acotamiento útiles para resolver problemas que ahora conocemos como *NP-hard* (por sus siglas en inglés Non-

deterministic Polynomial-time hard), no existe una justificación formal para proporcionar en este momento. En general el desempeño de los métodos de ramificación y acotamiento fue decepcionante, en ocasiones con ninguna evaluación computacional o bien solo encontrando el óptimo para problemas pequeños. Además el desempeño tiene que ver con la relativa lentitud de los ordenadores utilizados en esos días. Sin embargo, el principal inconveniente fue la falta de características más sofisticadas en el diseño de los algoritmos que efectivamente restringieran la búsqueda. Los límites de acotamiento frecuentemente relajaban las restricciones y usualmente producían simplificaciones del problema careciendo parte de la estructura original.
Por último, se emite el libro de texto de Baker (1974) convirtiéndose en el libro de programación por excelencia, ocupando ese lugar durante muchos años.

1.3. COMPLEJIDAD COMPUTACIONAL.

Este momento fue el más importante en la historia de la programación y secuenciamiento de las operaciones. Esto se debe principalmente al desarrollo de la teoría sobre complejidad computacional y también por el diseño de una clara clasificación de los problemas en programación. Estos dos eventos consolidaron la teoría de programación hasta ahora.

La programación y secuenciamiento de operaciones como parte de la optimización combinatoria, por muchos años ha sido objeto de críticas por aquellos grupos de investigación del área de las *"matemáticas puras"*. Se sabe que una característica típica de la optimización combinatoria es que el conjunto de soluciones es finito, ejemplo permutaciones de elementos de un conjunto finito. Entonces un argumento de critica usual es: *"Las matemáticas, como se les conoce, se interesan por dos principales aspectos; dado un problema determinar donde se encuentra la solución, y si la tiene, como encontrarla. Pero en optimización combinatoria las respuestas no son más que triviales. Considere tres máquinas en configuración flowshop para minimizar el makespan. Esto tiene solución, ya que con alguna permutación de los trabajos se encontrará. Desde esta perspectiva se deben comparar todas las permutaciones ya que el número de éstas es finito. Por lo tanto, la optimización combinatoria y sus ramas, incluyendo programación y secuenciamiento de operaciones, no pueden considerarse como una disciplina matemática"*.

Es posible que el espacio de soluciones sea finito, pero aun en problemas de programación de un tamaño modesto, por ejemplo 50 trabajos, enumerar todas las posibles soluciones factibles aun con las computadoras más actuales y más rápidas es imposible dentro de un plazo razonable, a menudo imposible en el periodo comprendido desde el *Bing Bang* hasta ahora. Por lo tanto el problema de encontrar la solución debe ser visto desde una perspectiva diferente: se requiere un método que encuentre rápidamente una solución. Pero a su vez, esto plantea nuevas preguntas, que se entiende por rápido, un día, una hora, o bien bajo que dispositivo, con lápiz y papel o en computadora. Para 1965 la mayoría de los investigadores estaban de acuerdo en llamar a aquellos métodos que encontraban soluciones rápidas como *"buenos algoritmos"*.

Edmonds (1965) sostuvo que un buen algoritmo es aquel cuyo tiempo de funcionamiento depende exponencialmente del tamaño del problema. Dado que las computadoras utilizan la representación binaria de los números, el tamaño del problema se obtiene por el producto de la cantidad de parámetros de entrada y la longitud máxima del número binario correspondiente a cualquiera los parámetros de entrada. Por ejemplo, minimizar el tiempo total de procesamiento de n trabajos en m máquinas no relacionadas es dado por $L = nm \log(\max p_{ij})$. Un algoritmo que requiere $O(L^k)$ de tiempo, donde k es una constante que no depende de L, es llamado *algoritmo de tiempo-polinomial* o simplemente *algoritmo polinomial*.

Así, el objetivo es encontrar algoritmos de tiempo-polinomial. Ejemplos de ellos son los algoritmos de Johnson (1954), Jackson (1956), Smith (1956), y Moore (1968) donde cada uno requiere *O(n log n)* tiempo de ejecución, siendo *n* son los trabajos a procesar. Además, el primer algoritmo de tiempo-polinomial basado en programación lineal (herramienta de la investigación operativa) se debe a Khachiyan (1979) que posteriormente fue mejorado por Karmakar (1984). El hecho de que los problemas que utilizan programación lineal se pueden resolver exponencialmente ha dado lugar a una serie de algoritmos para encontrar soluciones exactas o bien aproximadas a problemas de programación y secuenciamiento de operaciones. Desde entonces la meta en programación y en optimización combinatoria fue clara: la búsqueda de algoritmos de tiempo-polinomial. Desafortunadamente los investigadores no encontraron dichos algoritmos para resolver problemas considerados interesantes desde el punto de vista teórico y práctico. El sentimiento general de la comunidad fue que la mayoría de estos problemas tenían alguna *"dificultad intrínseca"* y por consiguiente no admiten una solución en un tiempo polinomial. Y finalmente esta sensación de dificultad encontró una justificación sólida. Cook (1971) define a estos problemas como *"Hard"* o bien *"duros"*, porque no se resuelven en tiempo-polinomial y para ellos es improbable un algoritmo de este tipo. Karp (1972) tradujo mucho de la aportación de Cook (1971) construyendo un lenguaje accesible de esto: los algoritmos estándar o determinísticos son aquellos que procesan todas las instrucciones una a una en secuencia. Mientras que los algoritmos no-determinísticos son aquellos que evalúan cada instrucción y determinan que se ha de procesar. Entonces los problemas que pueden ser resueltos en tiempo-polinomial por algoritmos determinísticos son llamados problemas de *clase P*, mientras que los problemas que pueden ser resueltos en tiempo-polinomial por algoritmos no-determinísticos son llamados problemas de *clase NP*. Aunque no es conocido aun donde estas clases coinciden se asume que $P \neq NP$. Además, existen problemas que son más difíciles de resolver que cualquiera de *clase NP*, estos son conocidos como *NP-hard*.

En este tiempo aparece una nueva configuración multi-etapa en programación y secuenciamiento de operaciones, los conocidos procesos abiertos llamados *Openshop*. Aquí cada trabajo *j* será procesado en una máquina *i* con un tiempo p_{ij}; pero, a diferencia de las configuraciones *flowshop* y *jobshop* el orden de las operaciones es libre, no se establece de antemano, y por ello existen diversas rutas de procesamiento a escoger. A Gonzalez & Sahni (1976) se les atribuye este término y demostraron su alta complejidad tan solo para minimizar el *makespan*. Ofrecieron un algoritmo que crea un programa basado en una configuración *flowshop* donde cambian el orden de procesamiento de un trabajo a la vez. Pinedo & Schrage (1982) desarrollan un algoritmo basado en la operación con el tiempo más largo. De Werra (1989) crea un programa que organiza los trabajos en tres bloques sobre cada máquina y los programa evitando el traslape.

Gonzalez & Sahni (1976) mostraron que con tan solo tres máquinas el problema *openshop* es *NP-hard*, incluso con cuatro máquinas y dos operaciones por trabajo cuando mucho. Graham *et al* (1979) reportaron que si el número de máquinas es variable, la configuración *openshop* es *NP-hard* en sentido estricto.

Para el caso de que sea posible interrumpir los trabajos en cualquier máquina, esto no es ventaja para ningún algoritmo inclusive para aquellos casos de tan solo dos máquinas. Gabow & Kariv (1982) ofrecen una manera de resolver problemas *openshop* con interrupciones a través de gráficos bipartitos.

Muchas técnicas y enfoques dentro de la optimización combinatoria han sido aplicados a problemas *NP-hard* como el problema del *"agente viajero"*. El éxito de alguna técnica en este problema a

menudo inspira a los investigadores a aplicar el mismo enfoque a otros problemas como la programación y secuenciamiento de las operaciones.

El *"enfoque poliédrico"* utiliza la programación lineal entera (otra técnica de la investigación operativa) al planteamiento del problema e introduce desigualdades a la formulación resolviéndose así por el método de ramificación y acotamiento. Este tipo de enfoque fue utilizado para resolver el problema del *"agente viajero"* por investigadores como Miliotis (1976), Miliotis (1978), Crowder & Padberg (1980), Grötschel (1980) y Padberg & Hong (1980) que obtuvieron soluciones óptimas para instancias de más de 100 ciudades. Sorpresivamente, los métodos poliédricos han sido notablemente infructuosos para obtener soluciones óptimas a problemas de programación y secuenciamiento de operaciones y por ello muchos de estos no son reportados. Sin embargo, Queyranne & Wang (1991) ofrecen una postura diferente en teoría poliédrica que ayuda a nuestra comprensión en problemas de programación. Incluyen el uso de la *"relajación langrange"* la cual ha sido muy útil para obtener cotas inferiores para usarse con el método de ramificación y acotamiento. Esto fue originalmente desarrollado por Held & Karp (1971). Al realizar una *"relajación langrange"* sobre las restricciones y estas a su vez son incluidas en la función objetivo por medio de un *"multiplicador de Lagrange"* el problema resultante es más simple para resolver y proporcionar así una cota inferior. Una vez elegido que restricciones a relajar, el problema se deriva en encontrar el valor de los multiplicadores. Held *et al* (1974) ofrecen una técnica llamada *"optimización subgradiente"* para encontrar dichos valores, pero computacionalmente es demasiado tiempo. Y a pesar de que el uso de la *"relajación langrange"* ayuda a resolver el problema en un tiempo-polinomial $O(nP)$, donde P es la suma de los tiempos de procesamiento, y n el número de trabajos, no se han resuelto problemas con más de 50 trabajos satisfactoriamente.

1.4. SOLUCIONES APROXIMADAS.

La mayoría de los problemas de interés en programación y secuenciamiento de operaciones son *NP-hard* y hacen ver que es improbable que haya un algoritmo que pueda obtener una solución óptima en un tiempo-polinomial. En muchas de las ocasiones resulta práctico adoptar una solución aproximada que está relativamente cerca del óptimo. Tradicionalmente los algoritmos que ofrecen soluciones aproximadas a problemas *NP-hard* se clasifican en *algoritmos de aproximación* y en *algoritmos heurísticos*. Generalmente los algoritmos de aproximación son evaluados a través de un análisis del *peor caso*. La terminología y principales conceptos que son utilizados en un análisis del *peor caso* se describen a continuación.

1.4.1. ALGORITMOS DE APROXIMACIÓN.

Para un problema de programación dado que tiene una meta de minimizar alguna función $F(S)$ sobre todos los posibles programas factibles S, permita S^* como el programa óptimo. Un algoritmo de tiempo-polinomial que encuentra una solución factible S_H tal que $F(S_H)$ es mayor p (donde $p \geq 1$) veces del valor óptimo $F(S^*)$ es llamado *algoritmo de aproximación-p*; al valor de p se le conoce como *cota del peor caso*. Si un problema admite un *algoritmo de aproximación-p* entonces se dice que es aproximable en un factor de p. Una familia de *algoritmos de aproximación-p* es nombrada *esquema de aproximación de tiempo-polinomial*, o PTAS (por sus siglas en inglés *Polinomial-Time Approximation Scheme*). Si $p = 1 + \varepsilon$ para cualquier valor de $\varepsilon > 0$; y si el tiempo de ejecución del algoritmo es polinomial con respecto al tamaño del problema y a $1/\varepsilon$, entonces el esquema es llamado *esquema de aproximación de tiempo-polinomial completo* o bien FPTAS (por sus siglas en inglés *Full Polinomial-Time Approximation Scheme*). Schuurman & Woeginger (1999) ofrecen una revisión sobre algoritmos de aproximación aplicados a programación y secuenciamiento de operaciones.

El análisis del *peor caso* fue propuesto por Graham (1966) y Graham (1969). El problema que el abordo es sobre minimizar el *makespan* en *m* máquinas paralelas. Ofreció un algoritmo de programación basado en una lista de trabajos. Suponga que un programa S_L es encontrado por un procedimiento de programación en base a una lista arbitraria de trabajos, y algún trabajo *k* es el último trabajo procesado en ese horario, es decir, formalmente, $C_{max}(S_L) = C_k$. Supóngase que el trabajo *k* inicia en el tiempo *t*, la naturaleza del programa por medio de una lista implica que todas las máquinas están ocupadas hasta el tiempo *t*, de lo contrario el trabajo *k* se puede asignar a alguna máquina disponible. Note además que $t \leq (\sum_{j \in N} p_j - p_k)/m$. Donde *m* es el número de máquinas.
Se obtiene

$$C_{max}(S_L) = t + p_k \leq (\sum_{j \in N} p_j - p_k)/m + p_k = \frac{1}{m}\sum_{j \in N} p_j + \frac{m-1}{m} p_k \qquad \text{(Ec. 1.4)}$$

Al utilizar la cota inferior sobre el óptimo *makespan* y finalmente al derivar se tiene

$$C_{max}(S_L) = LI + \frac{m-1}{m} LI = (2 - \frac{1}{m})LI \leq (2 - \frac{1}{m}) C_{max}(S^*) \qquad \text{(Ec. 1.5)}$$

Entonces para el problema en consideración de Graham (1966) y Graham (1969) el algoritmo que ofrece es un algoritmo de aproximación (2 – 1/*m*). El *peor caso* aquí es (2 – 1/*m*). Esto está garantizado para cualquier secuencia de trabajos de la lista. En un programa óptimo S^*, el trabajo más amplio se asigna a una de las máquinas, mientras que el resto de las *m* - 1 máquinas procesan el resto de los trabajos. Además, si los trabajos son organizados dentro de una lista donde el más amplio se encuentra en la última posición, entonces el algoritmo asignara *n* – 1 trabajos a cada una de las *m* máquinas y cuando inicie el trabajo más amplio dará un *makespan* igual a $C_{max}(S_L) = 2m - 1$. Este algoritmo requiere $O(n^{km})$ veces, y es parte de los PTAS, pero su tiempo de ejecución es impráctico.

Uno de los primeros FPTAS en optimización combinatoria fue desarrollado por Ibarra & Kim (1975) donde utilizan una técnica de redondeo sobre un algoritmo de programación dinámica aplicado al problema de *"la mochila"*. Recordando que la idea central es asignar un conjunto de elementos con cierto peso buscando obtener el total de peso máximo sin exceder un valor límite. Este problema está muy relacionado a problemas de programación y secuenciamiento buscando reducir el *makespan* sobre dos máquinas paralelas idénticas. Sahni (1976) demostró que para casos con *m* máquinas idénticas, un FPTAS existe para cualquier valor de *m*. Horowitz & Sahni (1976) extienden este hecho a máquinas no relacionadas. Hochbaum & Shmoys (1987) interpretaron cada una de *m* máquinas idénticas como *"espacios"* de capacidad de *d* unidades de tiempo, desarrollando así un algoritmo de aproximación que encuentra una asignación de trabajos para un número mínimo de máquinas y donde el último trabajo en cada máquina se complete en el tiempo *d* de la misma.

Para el caso de minimizar el *makespan* sobre *m* máquinas paralelas no relacionadas, diversos algoritmos utilizaron la idea del redondeo al adherir una relajación a las restricciones del problema formulado por programación entera. Sea la variable definida por $x_{ij} = \{0,1\}$, 1 si el trabajo es asignado a la máquina *i*, 0 de otra manera. El *makespan* óptimo es igual al valor más pequeño de la función objetivo de un programa entero:

Minimizar C

sujeto a $\sum_{j=1}^{n} p_{ij} x_{ij} \leq C$, donde *i* = 1, 2, ... *m*,

$\sum_{i=1}^{m} x_{ij} = 1$, donde $j = 1, 2, \ldots n$,
$x_{ij} \in \{0,1\}$, donde $i = 1, 2, \ldots m$, j = 1, 2, ... n.

La relajación lineal es obtenida al reemplazar la restricción de enteros por $0 \leq x_{ij} \leq 1$. El problema de programación lineal obtenido puede ser resuelto en tiempo polinomial. Potts (1985) demostró que el número de variables en fracción en una solución óptima básica es lineal en *m*. Obtuvo un algoritmo de aproximacion-2 para cualquier valor de *m*. Lenstra *et al* (1990) demostraron que una solución en fracción puede ser redondeada en tiempo polinomial para producir un algoritmo de aproximacion-2. Además establecen que la búsqueda de un programa *S* para este problema con $C_{max}(S) \leq 2$ es *NP-hard*. Los problemas de programación y secuenciamiento de operaciones resueltos por algoritmos de aproximación sobre una máquina o máquinas paralelas para minimizar la suma de los tiempos de culminación ponderada $\sum w_j C_j$ llego a ser un tópico de considerable interés a mediados de los noventa. Hall *et al* (1997) ofrecieron un algoritmo para esto considerando fechas de vencimiento y restricciones de precedencia basándose en el uso de relajaciones en el modelo de programación lineal. Afrati *et al* (1999) resumen los esfuerzos de once investigadores por desarrollar PTAS para diversas versiones de programación y secuenciamiento de operaciones en una máquina, máquinas paralelas y máquinas paralelas no relacionadas considerando fechas de vencimiento.

Ahora considere los algoritmos de aproximación para el caso *flowshop* donde Gonzalez & Sahni (1978) trabajaron en una solución aproximada para aquellos casos conocidos como *"programas ocupados"* que se refiere a que en cualquier momento al menos una máquina está procesando alguna operación. Esto significa que para un programa S_B *"flowshop ocupado"* el limite $C_{max}(S_B) / C_{max}(S^*) \leq m$ se mantiene. Una idea natural para generar un algoritmo de aproximación es reemplazar el problema original por uno artificial de tan solo dos máquinas en configuración *flowshop* y convertir su solución (obtenida con el algoritmo de Johnson) en la aproximada para el problema original. Sin embargo, los algoritmos de este tipo conocidos ofrecen un valor *p* de *m*/2. Note que para *m* = 3 se tiene *m*/2 = 2 siendo un algoritmo de aproximacion-2. Chen *et al* (1996) ofreció un algoritmo de aproximación-(5/3) cuando *m* = 3. Hall (1998) ofreció un PTAS para el caso *flowshop* con un número definido de máquinas que se extiende para manejar trabajos con fechas de entrega definidas.

Para el caso general de *m* máquinas en configuración *flowshop* Sevastianov (1994) y Sevastianov (1995) ofrece un algoritmo de aproximación basado en un programa de permutaciones *S* con $C_{max}(S) \leq \Pi_{max} + \varphi(m)p_{max}$, donde Π_{max} es la máxima carga de todas las máquinas, p_{max} es la duración de la operación más larga y $\varphi(m) \leq m(m\text{-}1)$ es una función que depende de *m*, el número de máquinas. Bajo este algoritmo encontrar un programa de permutaciones toma $O(n^2m^2)$ veces.

Sevastianov & Woeginger (1998) ofrecieron el enfoque de etiquetar los trabajos en grandes, pequeños y diminutos para abordar el problema de programación y secuenciamiento de operaciones en configuración *openshop*. Su algoritmo asigna casi de manera óptima los trabajos grandes, y el resto se va anexando al programa. Este enfoque conduce a un PTAS. Kononov & Sviridenko (2002) describen un PTAS considerando fechas de entrega. Además, Hoogeveen *et al* (2001) demostraron que muchos problemas de programación y secuenciamiento de operaciones con un número variable de máquinas intentando minimizar la suma de los tiempos de terminación (incluyendo las configuraciones *flowshop* como *openshop*) son *NP-hard*, y esto implica que para cualquiera de estos problemas la existencia de un PTAS implicaría que *P = NP*.

1.4.2. ALGORITMOS HEURÍSTICOS.

Los principales algoritmos heurísticos que se han utilizado en programación y secuenciamiento de operaciones se engloban en esta sección y están basados en métodos de búsqueda llamados de *"vecindario"*.
Inicialmente la técnica de *Búsqueda Local* se comenzó a utilizar cuando se buscaba mejorar los recorridos (*tours*) para el problema del agente viajero sustituyendo aristas (*ciudades*) de la trayectoria por otras. Croes (1958) propuso intercambiar aristas de dos en dos y Lin (1965) extendió el enfoque a tres aristas. En programación y secuenciamiento de operaciones Nicholson (1967) fue uno de los primeros en proponer el uso de la búsqueda local. El intercambiada un trabajo *j* definido en la secuencia y lo insertaba en alguna otra posición para minimizar el *makespan* en una configuración *flowshop* de tres máquinas. Sin embargo, los métodos de búsqueda local no fueron muy atractivos hasta mucho más tarde. Las razones posiblemente eran: las computadoras no eran lo suficientemente rápidas para una adecuada búsqueda en problemas prácticos, y los métodos eran muy simplistas como para considerarlos como una contribución importante. No obstante, las publicaciones de Kirkpatrick *et al* (1983) y Cerny (1985) que propusieron el *recocido simulado* como técnica de optimización fueron el instrumento para establecer a la búsqueda local como área de investigación prospera. Glover (1986), (1989), (1990) ofreció el estímulo para muchos investigadores para contribuir en el área de la búsqueda local. Paralelamente, el libro de Goldberg (1989) trajo la atención de los investigadores en los algoritmos genéticos, iniciándose así el camino de las heurísticas.
El más simple algoritmo de búsqueda es denominado en *descenso* donde repetidos intentos son hechos para mejorar la solución actual. Los intentos en mejorar involucran perturbaciones a la actual solución, si la perturbación nos lleva a una mejor solución un movimiento es hecho hacia la nueva solución. Un *vecino* define que posibles movimientos son permitidos. La búsqueda continúa hasta que ya no es posible mejorar sobre la actual solución. En una búsqueda con inicio *multi-descenso* se ejecuta el algoritmo en descenso varias veces. Usando un diferente punto de solución puede ayudar a producir mejores soluciones.
Existen estrategias bien conocidas que permiten progresar para evadir solución pobres y atravesar en el espacio para encontrar mejores soluciones. En *Recocido Simulado*, las soluciones pobres son aceptadas con una probabilidad de aceptación de $e^{-\Delta/T}$, donde Δ es la cantidad en la cual la función objetivo empeora con la nueva solución y T es un parámetro llamado temperatura que es reducido durante la ejecución del algoritmo acorde a un programa de enfriamiento. En *Búsqueda Tabu*, un movimiento en el *vecindario* es hecho aun si la solución es pobre y para lograr direccionar la búsqueda en diferentes zonas del espacio de solución, una *lista tabu* almacena propiedades de los puntos de la solución recién visitados. Los posibles movimientos que conducen a soluciones con propiedades ya almacenadas son prohibidos.

Las permutaciones en configuraciones *flowshop* fueron utilizadas por múltiples investigadores en el cual evalúan el potencial de la búsqueda local para generar soluciones de calidad superior a las obtenidas por otros métodos heurísticos. Osman & Potts (1989), y Ogbu & Smith (1990), comparan el *intercambio de vecindad* donde una solución es creada intercambiando un par de trabajos y la *inserción de un vecino* donde un trabajo *j* es removido de su posición y colocándolo en otra, dentro de un algoritmo de recocido simulado. Estos estudios demuestran que la *inserción de un vecino* provee mejores soluciones. Conclusiones similares se tienen en Taillard (1990) utilizando inserción de un vecino que en Widmer & Hertz (1989) que utilizan intercambio de vecindad.

1.4.3. ALGORITMOS GENÉTICOS.

1.4.3.1. IDEA, COMPONENTES Y FUNCIONAMIENTO.

En contraste a los métodos de búsqueda del *vecindario* que se enfocan en mejorar una simple solución, los *algoritmos genéticos* tienen el propósito de mejorar un conjunto de soluciones. Los métodos de búsqueda conocidos como algoritmos genéticos tienen sus raíces en los mecanismos de la evolución y la genética natural. El interés en algoritmos heurísticos basados en procesos físicos y naturales se inicia desde los años setentas cuando Holland (1975) propone los algoritmos genéticos. Estos se inspiran en los procesos naturales de búsqueda y selección que lleve a la supervivencia de los individuos más aptos. Los algoritmos genéticos utilizan mecanismos de búsqueda probabilística dirigidos a disminuir el esfuerzo de la búsqueda, generando para ellos una *población* a través de operadores conocidos como *selección*, *cruza* y *mutación*. La idea central es intercambiar información entre las soluciones representadas por los individuos de la población en la búsqueda de mejores soluciones o bien mejores individuos.

En la naturaleza, los individuos que mejor se adapten en la competencia por recursos escasos sobreviven. Adaptarse a los cambios en el ambiente es esencial en la supervivencia de los individuos de cada especie. Mientras que las características particulares de un individuo determinan su capacidad de supervivencia, estas mismas están determinadas por el contenido genético. Cada característica está determinada por una unidad básica llamada *gen*. Los genes controlan las características *"clave"* de la supervivencia de un individuo en ambiente competitivo.

La evolución de una especie se manifiesta como una sucesión de cambios. Dichos cambios son en esencia realizados en el material genético de la especie y ocurren durante la selección y combinación de material genético realizado en la reproducción.

Solamente los individuos más aptos sobreviven y reproducen, un fenómeno natural llamado *"la supervivencia del más apto"*. Por lo tanto, los genes de los más aptos sobreviven mientras que los genes de los menos aptos mueren. La selección natural conduce a la supervivencia de los más aptos e implica conducir a la supervivencia de sus genes.

El proceso de reproducción genera diversidad en el material genético. La evolución se inicia cuando el material genético de dos padres se combina durante la reproducción. Nuevas combinaciones se generan de los padres, creándose nuevos genes. Específicamente el intercambio de material genético es llamado *cruza*.

Los algoritmos genéticos manejan una población de soluciones potenciales a la búsqueda de la solución. Específicamente ellos operan sobre representaciones *codificadas* de las soluciones, equivalentes al material genético de los individuos en la naturaleza, y no directamente sobre la soluciones en sí mismas. Los algoritmos genéticos codifican las soluciones como *cadenas* ya sea de bits, números, letras, entre otros. Como en la naturaleza, la *selección* provee el mecanismo que conduce a las mejores soluciones a sobrevivir. Cada solución es asociada con un valor de *aptitud* que refleja que tan bueno es comparado con otras soluciones de la población. Los valores más altos en aptitud tienen mayor probabilidad de sobrevivir y reproducirse en la siguiente generación. La combinación de material genético entre los padres es simulada a través de la *cruza* que intercambia partes de las cadenas para producir una nueva. Por último, la *mutación* causa esporádicos y aleatorios cambios en los elementos de la cadena que se produjo, jugando un papel de regeneración de la perdida de material genético que se puede dar con el paso de las generaciones. Por último, el ciclo generacional es repetido hasta que un criterio de paro es alcanzado, por ejemplo, un número definido de generaciones.

El mecanismo de codificación es fundamental en la estructura de cualquier algoritmo genético y permite representar las variables del problema a optimizar. El mecanismo de codificación depende de la naturaleza de las variables del problema, por ejemplo, en un problema de transporte las variables asumen valores continuos, mientras que las variables en un problema de ruteo son binarias representando si se visita o no un lugar antes que otro.

La función de aptitud es la función objetivo del problema, es decir, la función a ser optimizada, provee el mecanismo de evaluación de cada cadena. Sin embargo, el rango de valores varía de problema a problema. Para mantener uniformidad sobre el dominio de varios problemas se utiliza la función de aptitud para normalizar la función objetivo a un rango conveniente de 0 a 1. El valor normalizado de la función objetivo es la aptitud de la cadena, donde el mecanismo de selección la usa para evaluar a las cadenas de la población.

La selección modela la supervivencia del más apto. En un simple algoritmo genético, una cadena con aptitud mayor tiene una alta probabilidad de ser seleccionado para la reproducción y así una alta probabilidad de sobrevivir en la siguiente generación. Las cadenas con una aptitud mayor al promedio pueden ser escogidas más de una vez para reproducirse, no sucediendo así con cadenas con menor aptitud.

La cruza se considera el operador crucial del algoritmo genético, donde pares de cadenas son elegidas previamente en la selección para realizar la combinación de sus elementos. Uno de los más simples enfoques está en la *cruza en un punto*. Se asume que la cadena tiene una longitud definida *l*. Se elige aleatoriamente un punto de cruza entre 1 a *l*-1. Las partes de las dos cadenas a partir del punto de cruza son intercambiadas creando así dos nuevos individuos –cadenas–, véase Figura 1.2.

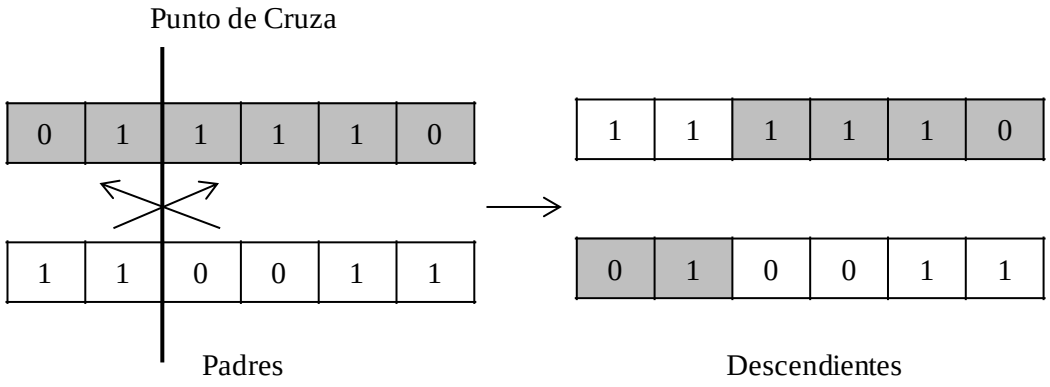


Figura 1.2 Ejemplo de Cruza en un punto.

La cruza no siempre es efectuada, ya que después de haber elegido a un par de cadenas para ser cruzadas, el algoritmo invoca el operador de cruza siempre y cuando un número generado entre 0 y 1 es más grande que la probabilidad de cruza p_c conocida como *tasa de cruza*. De no ser así, las cadenas permanecen sin alteración y pasan a la siguiente generación tal cual están.

La mutación es posterior a la cruza. Aquí algún elemento de la cadena cambia de un valor a otro en el caso de mutación binaria o bien cambia su posición con algún otro elemento de la cadena. Al igual que en la cruza, existe una probabilidad de mutación p_m conocida como *tasa de mutación*, para cada elemento de la cadena, por lo que la mutación de un elemento de la cadena no afecta la probabilidad de mutar el elemento siguiente.
La mutación trata de restaurar la perdida de material genético en las diversas generaciones por lo que su rol es de operador secundario, véase Figura 1.3.

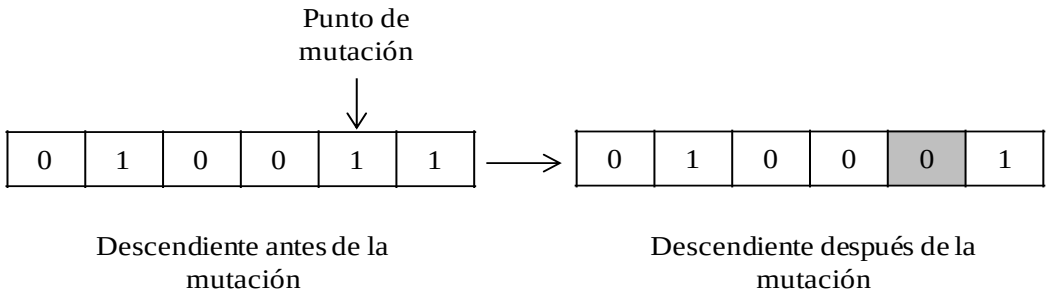


Figura 1.3 Ejemplo de Mutación en un simple gen.

Un criterio de paro establece la forma de concluir el algoritmo genético, por ejemplo, un número definido de generaciones, cuando se localiza un valor de aptitud en alguna cadena, o cuando las cadenas han alcanzado un cierto grado de homogeneidad, es decir, la mayoría de sus elementos tienen idénticos valores en sus posiciones.

1.4.3.2. PARÁMETROS DE CONTROL.

Se puede visualizar el funcionamiento de un algoritmo genético como una combinación balanceada de *exploración* de nuevas regiones en el espacio de búsqueda y *explotación* de las zonas ya visitadas. Este balance el cual controla de manera crítica la ejecución del algoritmo está determinado por la correcta elección de los *parámetros de control*: la tasa de cruza, la tasa de mutación y el tamaño de la población. Escoger el valor óptimo de los parámetros de control ha sido un constante debate entre investigaciones empíricas y analíticas. Algunos planteamientos que destacan son:

- Incrementar la probabilidad de cruza excesivamente interrumpe, trastorna la creación de cadenas prometedoras.
- Incrementar la probabilidad de mutación excesivamente tiene a transformar la búsqueda genética en una búsqueda altamente aleatoria.
- Incrementar el tamaño de la población excesivamente aumenta su diversidad y reduce la posibilidad de que el algoritmo converja prematuramente a un óptimo local, pero también provoca destinar un mayor tiempo para que la población converja a las regiones óptimas en el espacio de búsqueda.

No se puede escoger los parámetros de control sin considerar las interacciones entre los mismos, convirtiéndose el ajuste de parámetros en un problema de optimización no-lineal complejo. Además es evidente que el ajuste de los parámetros de control depende de la naturaleza de la función objetivo (Srinivas & Patnaik, 1993). El ajuste de los parámetros de control sigue siendo un problema aún por resolverse. Diversos investigadores han propuesto establecer un conjunto de parámetros de control que garanticen un buen rendimiento al realizar pruebas cuidadosamente seleccionadas de la función objetivo. Bajo esta premisa, dos enfoques han emergido: el primero es considerar una población pequeña y tasas de cruza y mutación relativamente grandes. Mientras que el segundo es utilizando una población grande pero con tasas de cruza y mutación mucho más pequeñas. Estos enfoques se resumen en:

- Tasa de cruza: 0.60, tasa de mutación: 0.001, tamaño de población 100 (DeJong & Spears, 1990).
- Tasa de cruza: 0.90, tasa de mutación: 0.01, tamaño de población 30 (Grefenstette, 1986).

Estos enfoques implican que cuando la probabilidad de cruza es alta trastornar las cadenas es deseable en poblaciones pequeñas. Además, la probabilidad de mutación juega un rol secundario y menos importante en poblaciones muy grandes.

1.4.3.3. APLICACIONES DEL ALGORITMO GENÉTICO EN PROGRAMACIÓN Y SECUENCIAMIENTO DE OPERACIONES.

A la programación y secuenciamiento de operaciones se le considera un problema práctico sumamente importante. Por ello es una de las áreas más populares para la investigación y aplicación de los Algoritmos Genéticos –GAs– (por sus siglas en inglés Genetic Algorithms). La programación de operaciones ha suscitado mucho interés para el trabajo de quienes utilizan GAs.

Uno de los primeros trabajos publicados sobre la aplicación de GAs fue el de Davis (1985). Su contribución fue desarrollar un esquema altamente simplificado para resolver un problema de programación y secuenciamiento de operaciones en una fábrica de juguetes con configuración *flowshop* (todos los trabajos tienen la misma precedencia de operaciones y máquinas). Davis (1985) señala que muchos problemas de programación prácticos constan de restricciones difíciles de definir y de representar dentro del marco formal exigido por las técnicas clásicas de investigación de operaciones. El éxito entonces de los GAs en problemas prácticos como estos ha sido el desarrollo de una adecuada combinación entre codificación de las variables y operadores genéticos específicos para representar las restricciones. Claramente una codificación en un GA de un programa de secuenciamiento de operaciones seria simplemente una lista especificando el orden y la duración de las operaciones a ser realizadas en cada máquina. Pero los operadores de cruza tradicionales aplicados a este tipo de cadenas casi siempre producirán individuos infactibles, unas operaciones perdidas, otras representadas más de una vez, y la precedencia de las operaciones para los correspondientes trabajos puede ser violada. La solución que Davis (1985) ofreció, fue el de un genotipo diferente para realizar la cruza pero requiere una fase de decodificación para representar la solución como factible. El genotipo mencionado es una lista de listas de preferencias, una para cada máquina. Una lista de preferencias consiste de un entero que especifica el tiempo en que una máquina debería iniciar el proceso, seguido por una permutación de los trabajos disponibles y de los elementos *"espere"* y *"ocioso"*. La rutina de decodificación es entonces una simulación del flujo de las operaciones en el piso de producción. El primer trabajo de la lista de preferencias se elige cada vez que se debe decidir qué operación realizará la máquina siguiente. El elemento *"ocioso"* forzará a la máquina a permanecer ociosa dando preferencia a otras máquinas. El elemento *"espere"* impide el procesamiento de la máquina sobre la lista de preferencias. Los operadores genéticos empleados son: un operador de cruza que intercambia la lista de preferencias para las máquinas seleccionadas; un operador de mezcla que aleatoriamente reordena los miembros de una lista de preferencias; y un operador heurístico que inserta *"espere"* como segundo elemento de la lista de preferencias de una máquina y esta debe esperar más de una hora para que los trabajos se encuentren disponibles. Cada operador fue aplicado probabilísticamente. La función de evaluación utilizada calcula los costos al simular en el piso de producción durante cinco horas a cada individuo. Se adicionan costos de penalización si los trabajos no son completados en este tiempo. Davis (1985) reconoce como su modelo debería ser ampliamente extendido para ser utilizado sobre un problema realista. Él provee una metodología basada en el estudio detallado de heurísticas para problemas de programación de tipo determinísticos al desarrollar operadores y decodificaciones sofisticadas. Sin embargo esto ha sido ampliamente ignorado.

Durante los años ochenta, Fourman (1985) experimento con GAs para resolver problemas relacionados a diseño de layout en plantas industriales. Dichos problemas están estrechamente relacionados a programación. El formuló un problema simbólico de layout en el cual unos bloques rectangulares de

tamaño fijo, conectados por líneas de ancho fijo y sujetas a varias topologías y restricciones geométricas, para minimizar el total de área ocupado. Sus genotipos fueron listas de símbolos que representan las restricciones geométricas sobre la posición de los bloques y las líneas de conexión las cuales fueron utilizadas para determinar un layout apropiado. Un operador de cruza estándar y un operador de mutación basado en la inversión de símbolos fueron fácilmente adaptados a estas listas de símbolos. Los resultados fueron prometedores pero no sobresalientes.
En las mismas fechas Smith & Davis idearon un algoritmo hibrido para un problema de embalaje en *"bins"* (Smith D. , 1985), el cual consiste en empacar un conjunto de cajas dentro de un espacio definido acorde a un criterio de densidad de empaque. Una vez más, el problema está estrechamente relacionado a programación. En particular a la secuenciación de trabajos en una simple máquina. El genotipo utilizado fueron simples listas de enteros determinando el orden en el cual las cajas ingresan al espacio definido. Ambas posturas, tanto Fourman como Smith & Davis obtienen una alta calidad de soluciones que los clásicos métodos de programación dinámica.

El problema del agente viajero, es un problema de secuenciación puro el cual tiene estrecha afinidad a la programación y secuenciamiento de operaciones. El problema del agente viajero es uno de los problemas con mayor aplicación general de los GAs. El objetivo es encontrar la ruta más corta a través de un conjunto de ciudades, visitando cada una de ellas una sola vez y regresando al punto de partida. Un obvio genotipo es una permutación de una lista de enteros representado las ciudades. Usando esta representación, un operador de cruza simple produciría rutas infactibles la mayor parte del tiempo, con algunas ciudades representadas dos veces y algunas ni siquiera figurarían. Los esfuerzos iniciales por Goldberg (1989) y Grefenstette (1986) superan estas dificultades al sustituir las ciudades repetidas por ciudades omitidas o bien eliminadas. Withley *et al* (1990) produjeron mejores resultados al desarrollar una representación y un operador de recombinación que maneja *"aristas"* (vínculos entre ciudades) en vez de las ciudades mismas. Su operador de recombinación de aristas utiliza un *"mapa de aristas"* para construir un individuo que hereda toda la información posible de la estructura de sus padres. Este mapa almacena todas las conexiones de dos padres que entran y salen de una ciudad. Un individuo se inicia al escoger aleatoriamente una de las dos ciudades iniciales de sus padres. Una ruta entonces es construida adicionando una ciudad a la vez mientras se favorece a aquellas ciudades con el menor número de aristas no utilizadas (para evitar que alguna se quede aislada). Las siguientes ciudades se conectarán en base a las conexiones con la ciudad actual de cualquiera de los padres (información que el mapa de aristas contiene). Whitley y sus compañeros de trabajo utilizaron este enfoque para desarrollar un prototipo de sistema de programación para una producción en línea de Hewlett Packard.

Fox & McMahon (1991) introducen una representación de cadena para secuencias basada sobre una matriz booleana que contiene las relaciones de orden entre operaciones. Una secuencia de N elementos es representada por una matriz N x N donde cada fila y cada columna es única e identificada con uno de los elementos. La matriz de elementos [X, Y] contiene un 1 si y solo si el símbolo X ocurre antes del símbolo Y en la secuencia. Ellos introducen dos nuevos operadores de recombinación que trabajan sobre esta representación. Su operador de *"intersección"* fue diseñado para pasar las características comunes de dos padres al hijo. El hijo es formado de la siguiente manera: primero se crea una matriz con las relaciones lógicas entre las matrices de dos padres. Esta matriz contiene las relaciones sucesoras y predecesoras; segundo, se adhiere a esta matriz un subconjunto de números 1 que son únicos a uno de los padres; finalmente se convierte esta matriz sin restricciones a una secuencia real a través de un análisis de las sumas de filas y columnas. Todas estas operaciones son fácilmente realizadas a través de la representación matricial. Ahora el operador *"unión"* el cual es un operador de cruza tradicional pero modificado para trabajar eficientemente sobre matrices. Fox & McMahon (1991) compararon estos operadores con el operador de recombinación de aristas de Withley *et al* (1990) entre otros. Sus operadores se desempeñaron de manera muy similar en términos de la calidad

de la solución, pero ofrecen mejores soluciones en mucho menos generaciones. Sin embargo, sus aplicaciones no fueron probadas en ambientes industriales.

Cleveland & Smith (1989) investigaron el uso de GAs en programación de líneas de flujo con multi-etapas con características no estándares. Estudiaron tres modelos básicos: una versión de secuenciación pura, la cual asume que todos los trabajos están disponibles para ser procesados al inicio del horizonte de planeación y que el costo de la producción en proceso es despreciable; otro modelo que considera los tiempos reales de liberación de los trabajos; y un tercer modelo que incluye costos de la producción en proceso.
En el primer modelo un análisis comparativo del uso de varios operadores de recombinación indico que muchos de ellos fueron capaces de encontrar buenas soluciones en 100 generaciones. Demostraron que el Algoritmo Genético –GA– fue capaz de manejar una versión no determinística del problema sin aparente pérdida en el rendimiento. Pero, para el segundo modelo los GAs probados no se ejecutaron de manera correcta. Sin embargo, cuando se utilizó la función objetivo más realista los GAs con el enfoque de secuenciación pura se ejecutaron de forma incorrecta mientras que los GAs que emplean representaciones basadas en programación -como las listas de preferencias de Whitley *et al* (1990)- encontraron significativamente mejores soluciones. Estos resultados para un problema complejo en el ámbito industrial son prometedores y atractivos.

Wren & Wren (1990) realizaron interesantes aplicaciones de los GAs a un problema complejo de programación de conductores de autobuses equivalente a programar y secuenciar operaciones de diversos trabajos en diversas máquinas. Utilizando una representación genética sencilla del problema y con un operador de recombinación ingenioso fueron capaces de obtener tan buenas soluciones como las mejores técnicas de investigación de operaciones. Ellos señalan que los GAs han sido en gran parte ignorados por la comunidad en investigación de operaciones, a pesar de los resultados obtenidos hasta esos días.

Dorndorf & Pesch (1992) utilizaron un algoritmo hibridizado donde por un lado un algoritmo de búsqueda utiliza un AG para obtener secuencias en problemas de programación y secuenciamiento con estructura *jobshop*. Sus resultados encuentran *makespan* más cortos y más rápidamente que Balas *et al* (1988) considerando que estos últimos han ofrecido una de las mejores técnicas disponibles.

Nakano & Yamada (1991) abordo el problema de la programación con configuración *jobshop* con una codificación genética similar al de Fox & McMahon (1991) descrita anteriormente. Utilizan un operador de cruza simple pero con un operador de reparación genética interesante para producir descendientes factibles. Compararon sus resultados contra los que se obtienen al utilizar técnicas de ramificación y acotamiento. Sin embargo, no ofrecen resultados comparativos de los recursos computacionales necesarios para lograr esto. Además, el método de reparación es computacionalmente costoso.

Finalmente, los GAs han sido ampliamente utilizados principalmente en ambientes donde las condiciones determinísticas de la función objetivo pueden ser aceptables. Sin embargo, si el problema es incierto y contiene aspectos difíciles de formalizar o bien no son prácticos los enfoques tradicionales, entonces los GAs pueden ser utilizados pero se debe tener sutileza, visión y muy posiblemente trabajarlos en conjunto con otras técnicas para alcanzar buenos resultados (Husbands, 1994).

1.4.4. ALGORITMOS DE ESTIMACIÓN DE DISTRIBUCIONES.

1.4.4.1. DE LA RECOMBINACION A LA DEPENDENCIA CONDICIONAL.

En el proceso de evolución de cualquier GA puede ser visto como la combinación de dos procesos; la selección y la variación. La selección se encarga de dirigir la evolución hacia mejores soluciones, mientras que los operadores de cruza y mutación conforman el proceso de variación el cual ayuda a explorar el espacio de soluciones.
En un GA, el operador de cruza *implícitamente* recombina soluciones parciales de los individuos seleccionados a cruzarse. El operador de mutación *implícitamente* causa esporádicos y aleatorios cambios en los individuos que se produjeron. Por lo tanto, la contribución clave de la variación es producir mejores individuos y ayudar a mantener la diversidad en la población. Los procesos de la selección y la variación juntos forman la base para una mejor evolución.

En muchos problemas de optimización, las variables en la solución pueden o no interactuar entre ellas para tener un efecto positivo en la aptitud de la solución. Por ejemplo en un GA, una solución $x = \{x_1, x_2, \dots, x_n\}$ es codificada como un conjunto de valores, x_i. La cadena de valores es conocida como cromosoma. Dependiendo del problema a tratar, un bit, un número real o entero puede ser usado para los elementos del cromosoma, es decir, los genes. Ahora bien, el valor de aptitud es calculado bajo un criterio de optimización que es modelado en la forma de una función conocida llamada función de aptitud *f(x)*. La Figura 1.4 muestra un cromosoma con valores en bits. Aquí la función objetivo es simplemente la suma de todos los bits en el cromosoma.

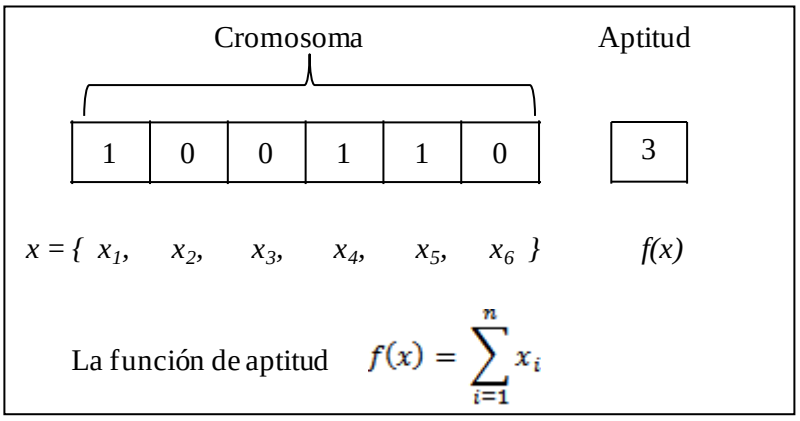


Figura 1.4 Cromosoma de longitud 6 y su función de aptitud.

Sin embargo, cada gen del cromosoma contribuye individualmente e independientemente a la función de aptitud, lo que significa que no existe una interacción entre las variables de este ejemplo.
Aunque la interacción puede o no estar presente, para la mayoría de los problemas de optimización y en particular de programación y secuenciamiento de operaciones esta información no es conocida de antemano. Por lo tanto, para resolver problemas de programación y secuenciamiento de operaciones es deseable tener el conocimiento de la interacción entre las variables. Según Shakya *et al* (2006) esto puede ser usado para dirigir la búsqueda más eficientemente.

Retomando la parte de la variación de un GA, esta intenta recombinar soluciones prometedoras de la población a través de los operadores de cruza y mutación, asumiendo que estos producirás mejores individuos. Sin embargo, ni el operador de cruza ni el de mutación, intentan aprender o aprovechar la interacción entre variables que pudiese existir entre ellas. En vez de eso, estos operadores aleatoriamente escogen el momento de cruza y mutación definidos por la tasa de cruza y mutación

respectivamente. Esta naturaleza aleatoria de los operadores puede algunas veces trastornar los valores que ofrece la interacción entre las variables evitando así obtener efectos positivos en la aptitud del individuo. Además, puede ocasionar que se requiere un mayor tiempo computacional para converger a una buena solución.

Al notar este hecho, los investigadores han estado enfocados en descubrir y aprovechar la interacción entre variables asociadas al problema en estudio. Esta tarea ha originado dos enfoques principalmente.

El primer enfoque está basado en el cambio de la representación del problema. La idea es manipular la representación de la cadena de soluciones para prevenir trastornos en los valores de las variables que interactúan. Los algoritmos genéticos de Goldberg *et al* (1989), Kargupta (1996), y Harik (1997) caen en esta categoría.

El segundo enfoque está basado en cambiar el proceso de variación. La idea es aprender y aprovechar la interacción entre variables mediante la estimación de una distribución de la población y muestrear a partir de dicha distribución los descendientes. Entonces los operadores de cruza y mutación involucrados en el proceso de variación son remplazados por la estimación y el muestreo de una distribución de probabilidad, véase Figura 1.5.

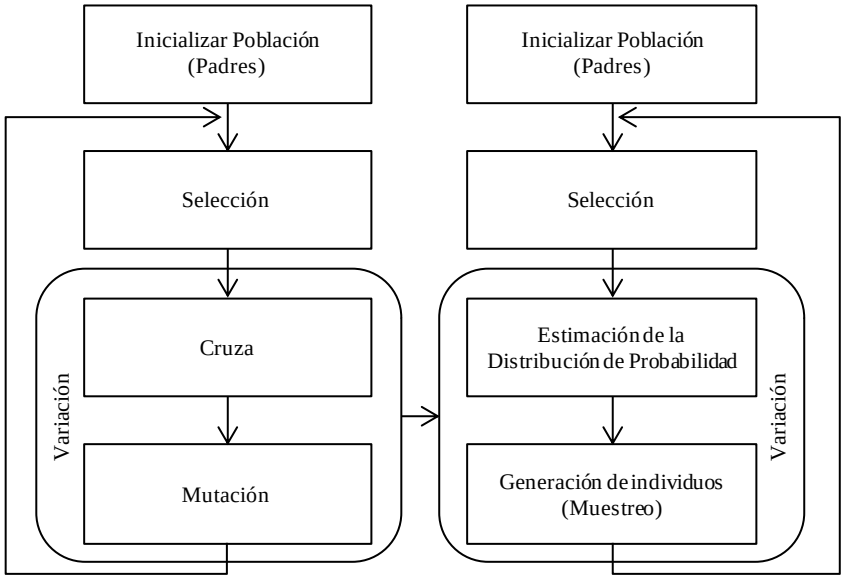


Figura 1.5 Diferencia del Enfoque de Variación entre un AG y un EDA.

En estadística, la distribución de un dato o conjunto de datos es la probabilidad de que se observe o se presente dicho dato o conjunto de datos en el universo de datos de donde se extrae. Por lo tanto, generalmente, la distribución se conoce como distribución de probabilidad. La estimación de la distribución es la tarea de calcular la distribución de probabilidad del dato o conjunto de datos en el universo de datos. Cuando se conoce la distribución de probabilidad asociada a los datos entonces se utiliza esta directamente. De lo contrario, la aproximación de la distribución de probabilidad puede realizarse con algún subconjunto de la población que se construyó, generalmente con los mejores individuos.

Se requiere introducir algunas notaciones para hacer más claro el hecho de que a través de una distribución de probabilidad se puede generar nuevos individuos en el proceso evolutivo. Para ello se adopta el enfoque planteado por Larrañaga *et al* (1999).

Sea X_i una variable aleatoria, y x_i uno de sus posibles valores. Usamos $p(X_i = x_i)$ para representar la función de densidad de probabilidad sobre un punto x_i. Ahora sea $X = \{X_1, X_2, ..., X_n\}$ un vector de n variables aleatorias y $x = \{x_1,x_2,...,x_n\}$ ser un vector de valores que toma cada variable del vector X, entonces $p(X = x)$ representa la función de densidad de probabilidad conjunta de X. Similarmente la función de densidad de probabilidad condicional de la variable X_i que toma el valor x_i dado el valor x_j de la variable X_j, será representado como $p(X_i = x_i \mid X_j = x_j)$.

Si el problema es de dominio discreto, por ejemplo, si cada X_i es una variable aleatoria con un conjunto finito de valores, entonces $p(X_i = x_i)$ es la distribución marginal univariante de la variable X_i. Si todas las variables en X son de tipo discreto entonces $p(X = x)$ será la distribución de probabilidad conjunta. De igual forma, la probabilidad condicional que X_i tomará el valor x_i dado el valor x_j de la variable X_j, y puede ser representado por $p(X_i = x_i \mid X_j = x_j)$.

Sea ahora X_s un subvector de X, y sea x_s un posible conjunto de valores que puede tener X_s, entonces $p(X_s = x_s)$ es la distribución marginal del conjunto X_s.
Ahora bien sean X_a y X_b vectores disjuntos de X, y sean x_a y x_b los posibles subconjuntos de valores que puede tener X_a y X_b respectivamente, entonces $p(X_a = x_a \mid X_b = x_b)$ denota la probabilidad condicional de $X_a = x_a$ dado $X_b = x_b$ y puede ser definido como

$$p(x_a|x_b) = \frac{p(x_a,x_b)}{p(x_b)} \qquad \text{(Ec. 1.6)}$$

Aquí, $p(x_a,x_b)$ es la probabilidad conjunta de los subconjuntos $X_a = x_a$ y $X_b = x_b$.

La factorización de la distribución de probabilidad conjunta, $p(X = x)$ será

$$p(X = x) = p(x_1|x_2,...x_n)p(x_2|x_3,...x_n)...p(x_{n-1}|x_n)p(x_n) \qquad \text{(Ec. 1.7)}$$

conocida también como regla de la cadena.

Considerar una solución $x=\{x_1,x_2,...,x_n\}$ como un conjunto de valores que toman un conjunto de variables aleatorias $X=\{X_1, X_2, ..., X_n\}$ donde $x_i \in \{0,1\}$. Entonces, la estimación de la distribución es la aproximación de la distribución de probabilidad conjunta $p(X = x) = p(x_1,x_2,...,x_n)$ de una población P de soluciones. En general, el cálculo de $p(X = x)$ consiste en el cálculo de la probabilidad de todas las 2^n combinaciones de –x– por lo tanto no es un enfoque factible. Sin embargo, dependiendo de las interacciones entre las variables, $p(X = x)$ puede ser factorizada en términos de la probabilidad marginal (o condicional) calculada con las variables que interactúan. Por lo tanto, para estimar $p(X = x)$ se requiere a) aprender la interacción entre variables y b) estimar la probabilidad marginal (o condicional) de las variables que interactúan.

De acuerdo al ejemplo previo, véase Figura 1.4, se establece que la interacción entre las variables es conocida de antemano, donde cada variable, $X_i \in X$, solamente interactúa con sí misma, y no interactúa con otras variables. Debido a esa falta de interacción entre variables la distribución de la probabilidad conjunta $p(X = x)$ puede ser factorizada en términos de probabilidades marginales univariantes de las variables individuales de X, es decir, $p(X_i = x_i)$ como

$$p(X = x) = \prod_{i=1}^{n} p(X_i = x_i) \qquad \text{(Ec. 1.8)}$$

Ahora que ya se tiene un modelo factorizado para *p(X = x)*, el siguiente paso es estimar las probabilidades de las variables que interactúan. Existen diversas formas de hacerlo. Un manera sencilla para el ejemplo en cuestión es, dado un conjunto de soluciones, *D*, de la población *P*, la probabilidad marginal univariante de *x*, siendo esta 1, $p(x_i = 1)$, puede ser estimada al dividir el número de soluciones en *D* con 1 en la *i*-ésima posición por el número total de soluciones *N* en *D*.

$$p(x_i = 1) = \frac{1}{N}\sum_{x \in D, x_i = 1} 1 \qquad \text{(Ec. 1.9)}$$

$p(x_i = 0)$ puede ser calculado de manera similar.

Una vez que se calcula *p(X = x)*, (en este caso todas las probabilidades marginales ya que se estableció de antemano que no existe interacción entre las mismas variables) se pueden generar muestras para x_i que representa un descendiente de la siguiente manera

$$x_i = \begin{cases} 1, \text{ si un número generado con distribución uniforme entre 0 y 1} \leq p(x_i = 1) \\ 0, \text{ de otra manera} \end{cases}$$

Repitiendo este proceso para cada $i \in \{1,2,..., n\}$ resultará un nuevo descendiente. La misma idea se utiliza para construir la nueva generación de individuos, es decir, una población completa, la cual puede ser utilizada para remplazar la población padre *P*. la Figura 1.6 muestra el ejemplo descrito previamente.

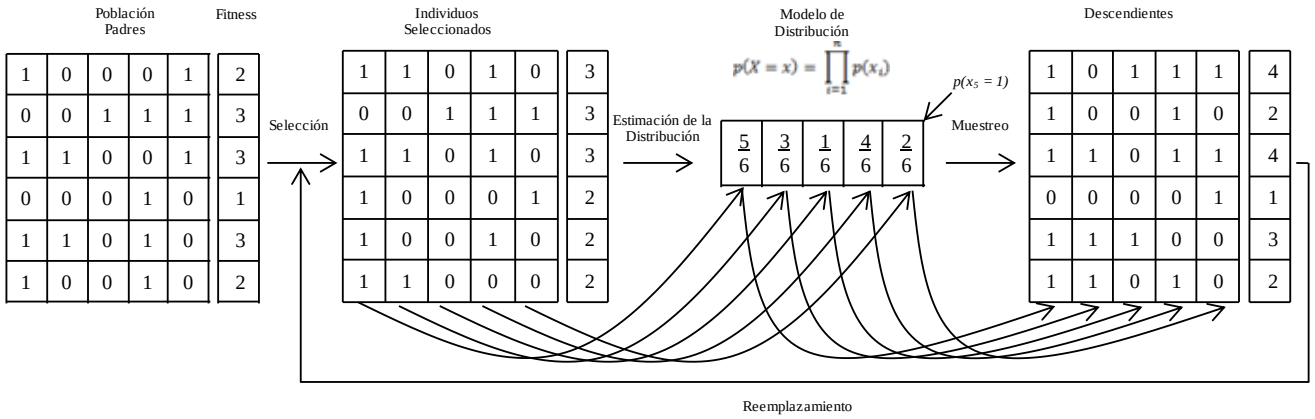


Figura 1.6 Simulación de la variación por medio de la estimación de la distribución y la generación de descendientes a través del muestro.

Aunque fue un ejemplo muy simple es suficiente para demostrar que ya no es requerido los operadores de cruza y mutación definidos en un GA.

1.4.4.2. MOTIVACIÓN, FUNCIONAMIENTO y TIPOS.

La clase de algoritmos que estiman la distribución de probabilidad y realizan muestras a partir de ella para realizar el proceso de variación se les conoce como Algoritmos de Estimación de Distribuciones

EDAs (por sus siglas en inglés Estimation of Distribution Algorithms). Introducidos inicialmente por Mühlenbein & Paaß (1996), estos algoritmos son un área de desarrollo en el campo del cómputo evolutivo.
Se han propuesto una amplia variedad de EDAs que utilizan diferentes técnicas para estimar una distribución de probabilidad y muestrear sobre ella y son objeto de investigación activa por la comunidad que desarrolla algoritmos evolutivos.

La motivación de su creación principalmente se debe a intentar identificar y aprovechar la interacción entre las variables que participan en la solución de un problema para asistir en la evolución del algoritmo. Sin embargo, existen dos factores más que han motivado a los investigadores hacia un enfoque probabilístico de la evolución (Larrañaga & Lozano, 2002). El primero es que la ejecución de un GA depende de elegir adecuadamente los parámetros de control al menos tradicionales tales como tasa de cruza, tasa de mutación y tamaño de población. Pero esto ha llegado a ser un problema de optimización en sí mismo (Grefenstette, 1986). La motivación entonces está en minimizar los parámetros del algoritmo. El segundo factor es el hecho de que el análisis teórico de un GA es una tarea difícil. Diversas teorías han sido propuestas para explicar la evolución de un GA sin embargo una teoría convincente aún no se ha desarrollado. La motivación en este punto es lograr un mejor análisis y un mayor rigor teórico sobre el proceso de la evolución (Larrañaga & Lozano, 2002).

La forma de operar un EDA inicialmente es igual que en un GA, generando una población inicial *P*, que consiste de *M* soluciones. El subconjunto *D* consiste de *N* soluciones que son seleccionadas de *P* acorde a un criterio preestablecido. La estimación de la distribución se hace a partir del conjunto *D* y se utiliza para generar nuevos descendientes que sustituyen a *P*. este proceso se repite hasta que un criterio de paro es alcanzado. La Figura 1.7 muestra el pseudo-código de un EDA.

Algoritmo de Estimación de Distribuciones

1. Generar población inicial *P* (padres) de tamaño *M*
2. Seleccionar un subconjunto *D* de *P* que consiste de *N* soluciones, donde $N \leq M$
3. Estimar la distribución de probabilidad $p(X=x)$ a partir del subconjunto *D*
4. Muestrear $p(X=x)$ para generar descendientes
5. Ir al paso 2 hasta que el criterio de paro es alcanzado

Figura 1.7 Pseudo-código de un EDA básico.

De este pseudo-código se puede observar que el corazón de cualquier EDA es la estimación de la distribución y la generación de descendientes con ella. La tarea de estimar la distribución dependen de dos factores adicionales: 1) de la precisión de la interacción y/o relación entre las variables, y 2) de la precisión de la estimación de las probabilidades en el modelo de distribución. Cualquier EDA puede ser visto como la combinación de estos tres factores:

- Estimar la relación entre las variables
- Estimar las probabilidades
- Generar muestras (descendientes) a partir de la distribución

Los EDAs se pueden distinguir y clasificar por el tipo de variables con las que tratan; discretos o continuos. Aunque el objetivo real es clasificarlos según tipo de vínculo utilizado por su modelo de

distribución. Dependiendo del tipo de conexión utilizado en su modelo de distribución, los EDAs puede clasificarse en univariados, bivariados y multivariados.

EDAs Univariados.

Los EDAs de esta categoría asumen que entre las variables del problema existe independencia, es decir, no consideran interacciones entre las variables. Por lo tanto, la distribución de probabilidad $p(X = x)$ llega a ser simplemente el producto de las probabilidades marginales univariadas de todas las variables de la solución representada en Ec. 1.8.
La Figura 1.8 muestra la representación de la ausencia de interacción utilizada por estos algoritmos donde cada nodo representa una variable en la solución. El ejemplo de la Figura 1.4 cae en esta categoría.
Debido a la simplicidad del modelo de distribución utilizado, los algoritmos en esta categoría son computacionalmente económicos y se ejecutan muy bien en problemas donde la interacción entre las variables es insignificante. Algoritmos muy conocidos en esta categoría son el PBIL (por sus siglas en inglés Population Based Incremental Learning) desarrollado por Baluja (1994), el UMDA (por sus siglas en inglés Univariate Marginal Distribution Algorithm) propuesto por Mühlenbein & Paaß (1996), y cGA (por sus siglas en inglés Compact Genetic Algorithm) de Harik *et al* (1999).

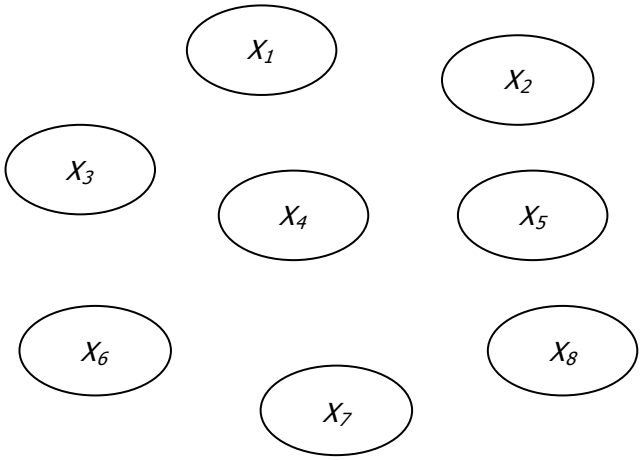


Figura 1.8 Representación gráfica de la ausencia de interacción entre variables.

EDAs Bivariados.

Los algoritmos de esta categoría consideran las interacciones entre pares de variables que intervienen en la solución. Por lo tanto, y en contraste con el caso univariado, el modelo de probabilidad contiene factores relacionados con la probabilidad condicional de pares variables que interactúan. Obviamente, este tipo de algoritmos en comparación con los EDAs univariados se desempeñan mejor en problemas donde la interacción entre pares de variables existe. Algunos de estos algoritmos de esta categoría son el MIMIC (por sus siglas en inglés Mutual Information Maximization for Input Clustering) propuesto por de Bonet *et al* (1997), el COMIT (por sus siglas en inglés Combining Optimizers with Mutual Information Trees) ofrecido por Baluja & Davies (1997a) y el BMDA (por sus siglas en inglés Bivariate Marginal Distribution Algorithm) desarrollado por Pelikan & Mühlenbein (1999).
Una representación gráfica de la relación e interacción (y por ende de la dependencia) entre las variables de estos algoritmos puede verse en la Figura 1.9

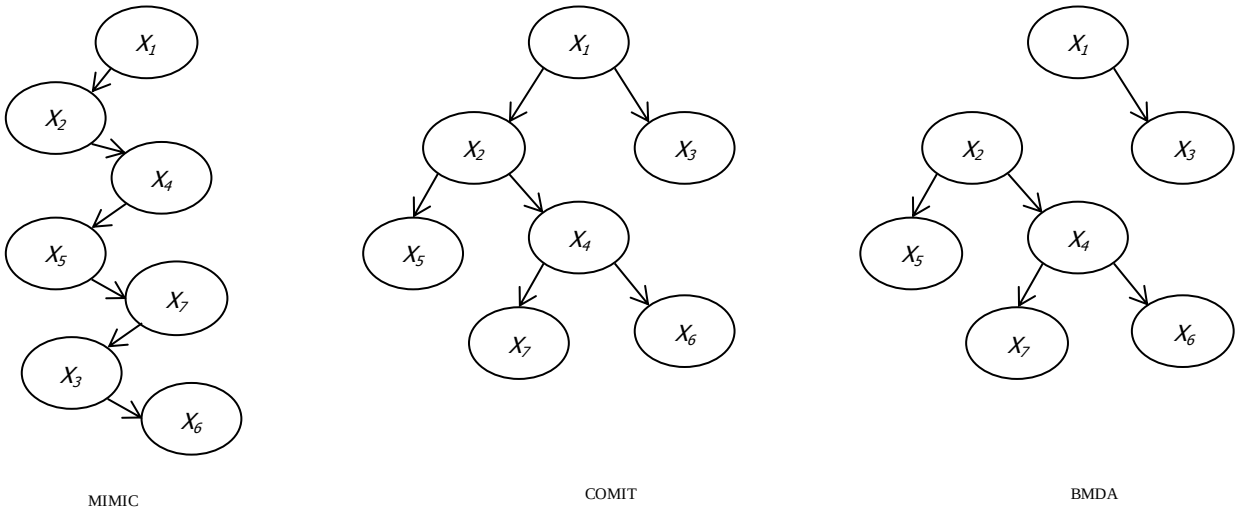


Figura 1.9 Representación gráfica de la interacción entre pares de variables.

EDAs Multivariados.

Cualquier algoritmo que considera las interacciones entre variables de orden mayor a dos puede ser clasificado en esta categoría. El modelo de distribución de probabilidad obviamente llega a ser más complejo que alguno que se haya utilizado en las anteriores clasificaciones.
La complejidad de estos modelos se incrementa exponencialmente por las posibles combinaciones de las interacciones haciendo infáctible buscar por todas los posibles modelos. Algoritmos conocidos son el ECGA (por sus siglas en inglés Extended Compact Genetic Algorithm) de Harik (1999), el FDA (por sus siglas en inglés Factorised Distribution Algorithm) de Mühlenbein *et al* (1999), el BOA (por sus siglas en inglés Bayesian Optimization Algorithm) de Pelikan *et al* (1999), el LFDA (por sus siglas en inglés Learning Factorised Distribution Algorithm) propuesto por Mühlenbein & Mahnig (1999), y el EBNA (por sus siglas en inglés Estimation of Bayesian Network Algorithm) desarrollado por Etxeberria & Larrañaga (1999).
Una representación gráfica de la relación e interacción (y por ende de la dependencia) entre las variables de estos algoritmos puede verse en la Figura 1.10

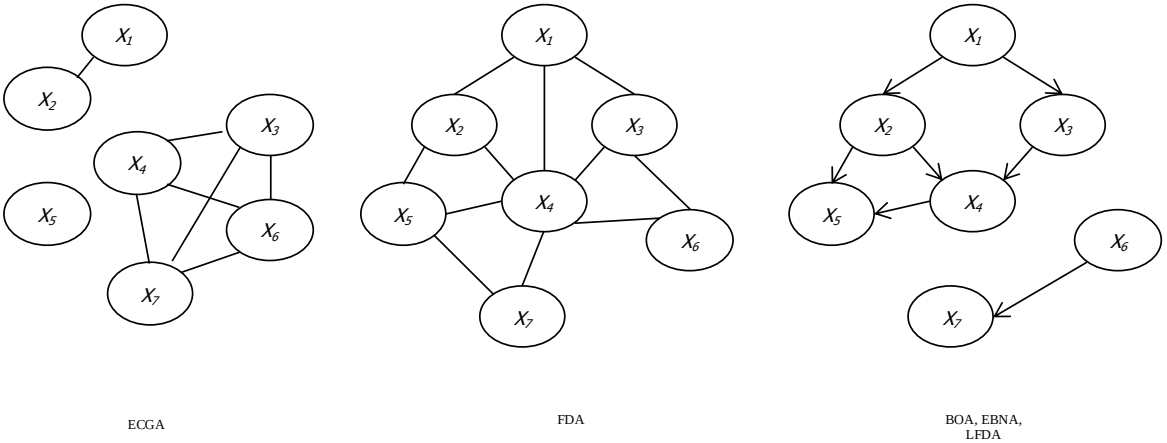


Figura 1.10 Representación gráfica de la interacción entre múltiples variables.

1.4.4.3. MODELOS GRÁFICOS PROBABILÍSTICOS.

El éxito o fracaso de un EDA depende de que tan bien estime la distribución de probabilidad conjunta $p(X = x)$ de las soluciones en la población. En la sección 1.4.4.2 se ha mostrado como puede ser estimada dicha distribución por la factorización de esta en términos de la probabilidad marginal (o condicional) de las variables que interactúan en la solución del problema.
Una de las más efectivas formas para representar la factorización de la distribución de probabilidad conjunta es con un Modelo Gráfico Probabilístico (Whittaker, 1990). De hecho muchos EDAs utilizan el concepto de un modelo gráfico probabilístico para estimar dicha distribución. Por lo tanto es esencial entender el modelo gráfico probabilístico en el contexto de un EDA.

Un Modelo Gráfico Probabilístico PGM (por sus siglas en inglés Probabilistic Graphical Model) puede considerarse como la fusión de dos disciplinas, la teoría de probabilidad y la teoría de grafos (Jordan, 1998). Estos contienen dos componentes: estructura y parámetros. La estructura de un PGM representa la interacción entre variables aleatorias en la forma de un grafo. En el contexto de un EDA, la estructura de un PGM es el vínculo entre las variables. Cada nodo del grafo representa una variable de la solución. La presencia de una arista entre dos nodos representa la existencia de una interacción entre las variables.
Los parámetros de un PGM son colecciones de funciones potenciales asociadas con un nodo o conjunto de nodos en la estructura de un modelo (Jordan, 1998). La función potencial representa la fuerza de la interacción entre las variables. En el contexto de un EDA los parámetros de un PGM son colecciones de probabilidades marginales (o condicionales).

Acorde al tipo de estructura de un PGM, estos pueden ser categorizados en:

- Modelos Dirigidos (Redes Bayesianas)
- Modelos No-Dirigidos (Redes de Markov)

En la literatura existente sobre EDAs, los modelos dirigidos (y en específico las redes Bayesianas), se han aplicado con frecuencia y se establecen como un enfoque útil para el modelado de la distribución.

Una red Bayesiana puede ser considerada como un par (B, θ), donde B es la estructura del modelo y θ es el conjunto de parámetros en el modelo. La estructura B es un Grafo Acíclico Dirigido DAG (por sus siglas en inglés Directed Acyclic Graph), en el cual cada arista que une dos nodos es una arista dirigida y no hay ciclos en el grafo. Además en dicho grafo cada nodo corresponde a una variable y cada arista corresponde a una *dependencia condicional*. Adicionalmente, un conjunto de nodos Π_i se dice que son los padres de X_i si hay aristas de cada variable *i que apunta* a X_i.
El parámetro $\theta = \{p(x_1|\Pi_1)p(x_2|\Pi_2)...p(x_n|\Pi_n)\}$ del modelo es el conjunto de probabilidades condicionales, donde cada $p(x_i|\Pi_i)$ es el conjunto de probabilidades asociadas con una variable $X_i=x_i$ dada la diferente configuración de las variables padres Π_i.

La Figura 1.11 muestra un ejemplo de la estructura y los parámetros de una red Bayesiana, donde cada variable X_i es binaria $x_i \in \{0,1\}$.

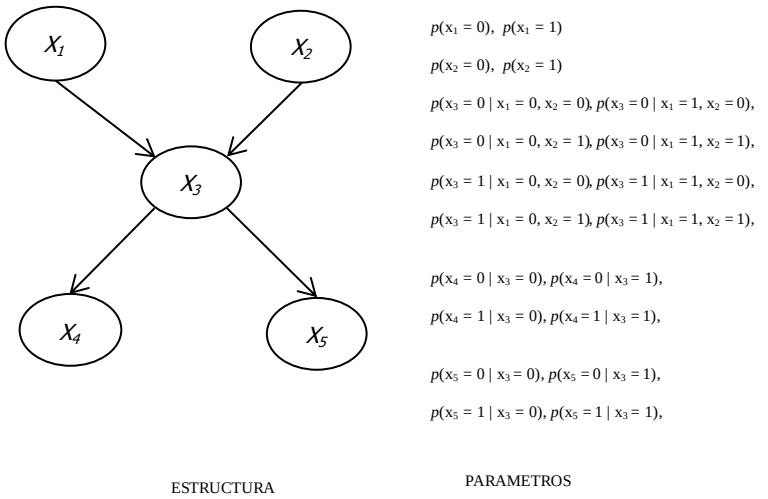


Figura 1.11 Red Bayesiana con 5 variables binarias aleatorias.

Por la regla de la cadena, la distribución de probabilidad conjunta de la red presentada en la Figura 1.11 es

$$p(x_1,x_2,x_3,x_4,x_5)=p(x_1)p(x_2|x_1)p(x_3|x_1,x_2)p(x_4|x_1,x_2,x_3)p(x_5|x_1,x_2,x_3,x_4) \quad \text{(Ec. 1.10)}$$

Sin embargo, las relaciones de independencia condicional codificadas en la red Bayesiana nos dicen que una variable X_i es independiente de las demás variables dados sus padres Π_i. En la Figura 1.11 la variable X_5 es independiente de X_1, X_2 y de X_4 dado X_3. Por lo tanto, para esta red Bayesiana, se puede decir que $p(x_5|x_1,x_2,x_3,x_4) = p(x_5|x_3)$. Similarmente, se puede decir que $p(x_4|x_1,x_2,x_3) = p(x_4|x_3)$ y $p(x_2|x_1) = p(x_2)$. Esto da una factorización compacta para la distribución de probabilidad conjunta como sigue

$$p(x_1,x_2,x_3,x_4,x_5)=p(x_1)p(x_2)p(x_3|x_1,x_2)p(x_4|x_3)p(x_5|x_3) \quad \text{(Ec. 1.11)}$$

En general, dado un conjunto de variables $X = \{X_1, X_2,..., X_n\}$ su distribución de probabilidad conjunta $p(X = x)$ para cualquier red Bayesiana es

$$p(X = x) = \prod_{i=1}^{n} p(\mathrm{x}_i | \Pi_i) \quad \text{(Ec. 1.12)}$$

Por último, en años recientes las redes Bayesianas han sido objeto de un creciente interés en la comunidad de Inteligencia Artificial (Lauritzen, 1996).

1.5. SIMULACIÓN DE EVENTOS DISCRETOS.

1.5.1. SIMULACIÓN Y SISTEMAS DE MANUFACTURA.

Sobre los sistemas de manufactura pueden ser clasificados como aquellos que procesan partes discretas y aquellos que procesan partes continuas. La manufactura de partes discretas es caracterizada por partes individuales que son claramente distinguibles tales como los circuitos integrados o partes automotrices y la manufactura de partes continuas opera sobre productos que se encuentran sobre un flujo continuo en su transformación, tales como las refinerías de petróleo o los compuestos químicos.

Esta investigación se ubica sobre la perspectiva de la manufactura de partes discretas. La manufactura de partes discretas está principalmente relacionada con el control de materiales, asignación de operadores a equipos o tareas y a la programación y secuenciamiento de las operaciones.

Las operaciones en manufactura son generalmente para fabricación o ensamble. La fabricación se refiere a transformar la forma de la materia prima que origine una utilidad práctica. La inyección de plástico, la extrusión de aluminio son ejemplos de ello. El ensamble se refiere a la combinación de diferentes partes para producir una pieza útil. Una mesa, el chasis de un auto, una bicicleta son ejemplos claros.
Los sistemas de manufactura también pueden ser clasificados de acuerdo a la configuración del proceso de producción. Las configuraciones más comunes son por producto, por proceso, por celda y posición fija. La diferencia entre ellos es fácilmente vista en el flujo del material. La Figura 1.12 muestra el flujo de material para configuraciones por producto, por proceso y por celda. La configuración por posición fija es ampliamente utilizada en productos grandes como barcos, aviones y edificios, debido a que el tamaño del producto hace impráctico moverlo. En las configuraciones por producto, por proceso y por celda, el producto se mueve por cada uno de los procesos.

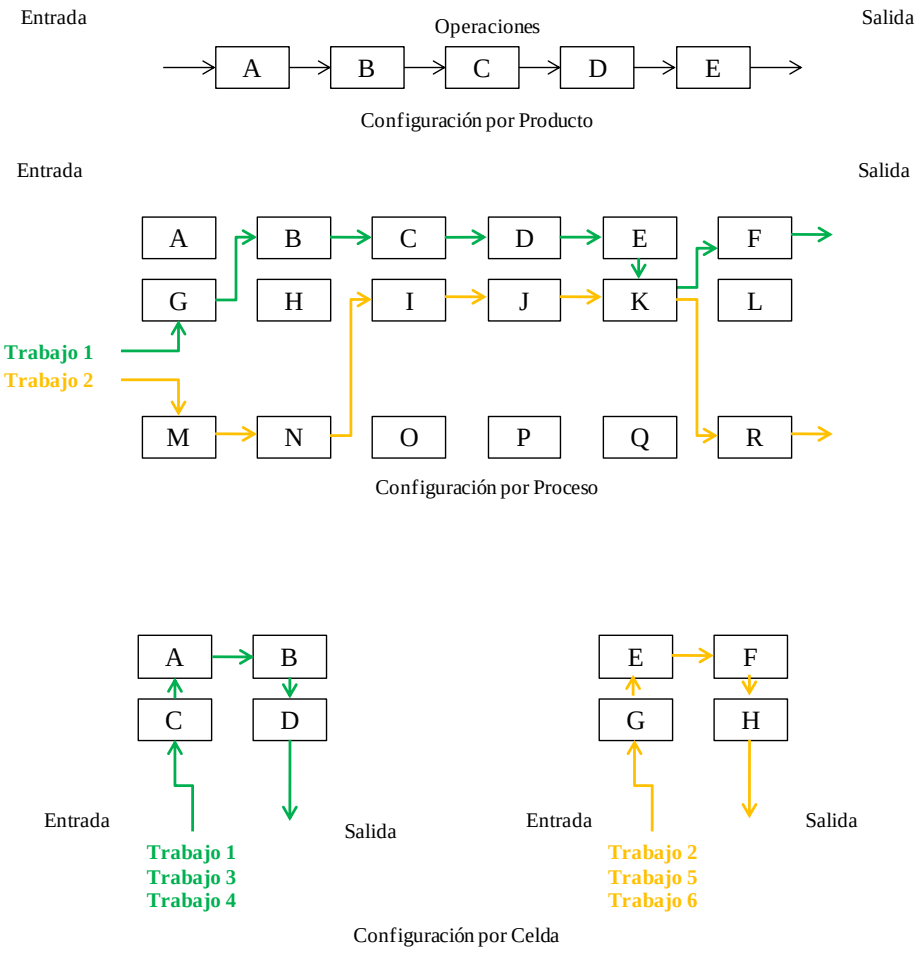


Figura 1.12 Representación gráfica de configuraciones en sistemas de manufactura.

La distribución de un taller o planta industrial para un proceso de manufactura con configuración por producto es diseñada para un producto en específico, conocida como producción en línea o en serie, porque las máquinas o estaciones de trabajo están orientadas de acuerdo al flujo del producto. El proceso culmina en la última máquina y las partes tienen la misma secuencia de manufactura, conocido esto como *flowshop*. Esta configuración es la más efectiva cuando el volumen lo justifica. Tiene las ventajas de que ofrece un tiempo corto entre la producción de producto a producto y niveles bajos de producción en proceso. La producción en proceso WIP (por sus siglas en inglés Work In Process) son lotes de partes y materiales que han sido ingresados al piso de producción para ser manufacturados pero aún no han sido terminados (Askin & Standridge, 1993). Aunado al costo del inventario del WIP, se incurre en costos por almacenar, mover, dañar y mantener registros del mismo. Esta configuración es más efectiva para disminuir dichos costos.

La distribución de un taller o planta industrial para un proceso de manufactura con configuración por proceso es útil cuando los productos no tienen la suficiente demanda para justificar una línea. Las máquinas que se configuran en línea no son tan fácilmente adaptables a otros productos. Esto no es económicamente factible a menos que el producto tenga suficiente volumen, sin embargo se estima que más del 75% de la manufactura de productos ocurre en lotes con menos de 50 artículos. En tales ambientes las máquinas deben ser capaces de ejecutar una variedad de operaciones sobre una variedad de partes. La respuesta más común es utilizar una distribución por procesos o bien llamada *jobshop*. En esta configuración las estaciones de trabajo están compuestas de máquinas con capacidades similares para ejecutar funciones similares. Lotes sucesivos que se asignan a un centro de trabajo pueden requerir diferentes herramientas y tiempos distintos de ajuste. Por lo tanto operadores con diferentes habilidades son típicos de esta configuración. A diferencia de la configuración por producto ésta se caracteriza por tiempos largos entre la producción de producto a producto y un alto WIP, pero al agrupar máquinas similares nos permite una alta utilización de las mismas.

Por último, la configuración por celdas, la cual es una mezcla entre las dos anteriores, donde partes similares se agrupa en suficientes cantidades para justificar el uso de un determinado número y tipo de máquinas. Así la celda se dispone a producir solo ese conjunto de partes. Este tipo de configuración es útil y usual por ejemplo en celdas robóticas. El uso de los equipos en un área de un taller o planta industrial designada únicamente a la producción de un especifico conjunto de partes facilita la programación de las operaciones en dichos equipos y reduce sustancialmente el tiempo de ajuste, el manejo de material, el WIP y el tiempo entre la producción de producto a producto.

Los artículos publicados sobre programación y secuenciamiento de las operaciones de sistemas de manufactura inician a finales de los años setenta. Pero gracias al desarrollo de las tecnologías de computación, los sistemas de manufactura han recibido una mayor atención en los últimos 20 años. Los sistemas de manufactura son el resultado del crecimiento en la demanda de productos en cantidad y de calidad. Los sistemas de manufactura están diseñados para combinar la eficiencia que se tiene en una línea de alta producción y la flexibilidad de una estructura *jobshop* donde la producción por lotes es de volumen y variedad bajo-medio (Sarin & Chen, 1987). Además Kaltwasser *et al* (1986) establecen que los sistemas de manufactura son sistemas de producción capaces de producir una gran variedad de diferentes partes utilizando el mismo equipo.

Ahora bien, tradicionalmente los problemas de programación han sido resueltos por métodos analíticos. Estos métodos frecuentemente ofrecen soluciones óptimas para problemas de programación pequeños bajo supuestos que los simplifican. Pero uno de los métodos más comúnmente utilizados en años recientes es la simulación por computadora. Los modelos de simulación que se construyen para resolver problemas de programación y secuenciamiento de operaciones ayudan a comprender el

entorno dinámico de un sistema de manufactura así como el de asistir en la evaluación de diferentes estrategias operativas sobre el sistema (Chan & Koh, 1994). La simulación es una herramienta altamente flexible que puede ser utilizada efectivamente para analizar sistemas de manufactura complejos. Nos permite modelar el sistema a detalle, de modo que las diferentes estrategias operativas pueden ser aplicadas en un ambiente más realista. La simulación también permite manejar problemas estocásticos, donde los modelos analíticos han demostrado ser intratables si no se tienen simplificaciones grandes del problema. A la simulación se le conoce como una herramienta de diseño, pero en recientes años, un número creciente de investigadores la han estado utilizando para el desarrollo de estrategias en la operación y el control de la manufactura. Las aplicaciones de la simulación en programación y secuenciamiento de operaciones, ha sido también un tópico de investigación popular (Tunali, 1997).

Podemos definir a la simulación por computadora como una técnica de modelado *descriptivo* que es utilizada para evaluar la programación y secuenciamiento de las operaciones a través de una serie de experimentos utilizando un equipo de cómputo. La simulación además, ha demostrado ser una excelente herramienta para programar las operaciones dinámicamente.
Se ha demostrado por Garey & Johnson (1979) que la programación y secuenciamiento de operaciones de manera dinámica es un problema *NP-hard*, con un número inmenso de posibilidades de secuenciar las operaciones de tan solo un trabajo. Por lo tanto, la simulación de la programación y secuenciamiento de operaciones de manera dinámica no ayuda a generar un análisis matemático de la solución, especialmente en sistemas de manufactura de escala realista. La naturaleza dinámica de dichos sistemas demanda un procedimiento de programación, el cual debe ser reactivo y sensible a las condiciones del sistema en vez de uno predictivo. Los modelos de programación matemática que son predictivos, se basan en supuestos simplificados del problema, se crean para algún sistema de manufactura en específico y requieren un alto grado de certidumbre en los datos utilizados. Sin embargo en un sistema de manufactura el problema de programación requiere también sincronizar recursos, en ocasiones por medio de un subsistema de manejo de materiales, para producir una variedad de partes en un cierto periodo de tiempo. Las reglas de programación son utilizadas para seleccionar la siguiente parte a ser procesada de un conjunto de partes esperando algún servicio. Estas mismas reglas se pueden utilizar para introducir nuevas partes dentro del sistema; e inclusive asignar partes a un sistema de manejo de materiales y sus correspondientes facilidades tales como estaciones de trabajo, unidades móviles, sistemas de cargas, entre otros.

Debido a la complejidad de un sistema de manufactura, no es útil buscar una solución óptima dentro de un contexto industrial debido a que los cambios en los procesos y en las reglas mismas de operación se hacen demasiado rápido (Chan & Chan, 2004). Por lo tanto, no es deseable y económico un programa óptimo, pero en vez de eso, desarrollar un modelo de programación flexible que asista a monitorear el sistema y a tomar decisiones sobre él es recomendable. Entonces así, los investigadores en Inteligencia Artificial encontraron en este contexto una oportunidad de aplicación ilimitada y que junto con un modelo de simulación pueden ayudar a imitar la programación y secuenciamiento de las operaciones de un sistema de manufactura y así poderlo describir (Baid & Nagarur, 1994). En años recientes, los GAs recibieron significativa atención por muchos investigadores por su mecanismo especial evolutivo. Los cuales también han sido utilizados para resolver problemas de programación y secuenciamiento de operaciones en sistemas de manufactura. Un ejemplo se encuentra en el trabajo de Greenwood *et al* (2005).

1.5.2. MODELOS DE SIMULACIÓN DE EVENTOS DISCRETOS.

Esta investigación tuvo como objetivo general, desarrollar y utilizar el uso de un modelo de simulación para programar y secuenciar las operaciones de una planta industrial. La expresión modelo significa representar un sistema de manufactura por otro medio, usualmente de forma simplificada. Por ello los modelos pueden ser abstracciones físicas o analíticas de la realidad.
Los modelos físicos han sido ampliamente utilizados por muchos años. Un ejemplo es el desarrollado por un arquitecto para mostrar la construcción de un edificio. Un modelo de este tipo provee una ayuda visual que permite determinar los detalles para su construcción, logrando expresar en una imagen lo que en palabras puede producir ambigüedad o imprecisión.
Los modelos físicos pueden ser de dos o tres dimensiones. De dos dimensiones se encuentran por ejemplo los planos de la distribución de una planta, o bien, los dibujos de piezas o partes específicas. De tres dimensiones comienzan hacerse más populares para lograr representaciones más agudas, visualmente hablando sobre los detalles que desean ser evaluados. Generalmente son aplicados en diseño de celdas de trabajo o bien en la determinación correcta de los espacios.

Los modelos matemáticos por su parte pueden encontrarse tanto en una computadora o bien en una hoja de papel. No obstante, ellos comparten en común ecuaciones matemáticas o relaciones lógicas para describir el sistema de manufactura que pretende representar. Los parámetros de los modelos tales como tiempo de producción estándar, tiempo entre fallas, tamaño de lotes son generalmente estimados.
Los modelos matemáticos a diferencia de un modelo físico utilizan variables de decisión, las cuales son las variables que pretendemos controlar al usar el modelo. La clave para construir modelos útiles es seleccionar las apropiadas variables de decisión y está íntimamente relacionado con la definición del problema a resolver. Una manera breve de determinar las variables de decisión es preguntándose: ¿que se está tratando de resolver? Ejemplos de variables de decisión puede ser la cantidad de máquinas necesarias o la cantidad de tareas asignadas a una máquina.
Los modelos matemáticos se pueden clasificar en descriptivos y prescriptivos de acuerdo a la información que ofrezcan en la salida del mismo. Los modelos basados en programación matemática como la programación lineal son prescriptivos, ya que a partir de un conjunto de valores establecidos para las variables de decisión, los valores obtenidos en el modelo como salida del mismo son los valores que debe tener el conjunto de variables de decisión. En ellos una función objetivo existe y el modelo está sujeto a determinadas restricciones.
Por su parte en los modelos descriptivos, un conjunto de valores establecidos para las variables de decisión son evaluados por el modelo y como salida del mismo, los valores de las variables llegan a hacer una estimación de lo que sucede en el sistema de manufactura.
Aunque posiblemente se pueden construir modelos prescriptivos con un alto grado de detalle, dichos modelos crecen en tamaño y llegan a contener no linealidades al incorporarles más detalles, haciendo virtualmente imposible resolver de una manera óptima el modelo. De manera que la habilidad para prescribir es menos importante que la habilidad para describir lo que sucede en una gran gama de variables de decisión o bien parámetros de entrada en el modelo. Además los modelos prescriptivos no necesariamente pueden ofrecer una solución óptima en un tiempo apropiado.

Los modelos matemáticos también pueden ser clasificados como analíticos y experimentales. Los modelos analíticos representan una abstracción del sistema de manufactura. Son generalmente ecuaciones que resumen la ejecución de un sistema pero no describe detalladamente los eventos que en el ocurren. Ejemplos de estos, son la teoría de juegos y la programación matemática. Los modelos experimentales imitan que ocurre en el sistema, permitiendo la experimentación con parámetros de operación en su entrada o bien con lógica de control. Ejemplos de estos son los modelos de simulación.
El propósito de un modelo de simulación es determinar que puede suceder cuando ciertos eventos ocurren y/o las condiciones preestablecidas en el modelo cambian. Los modelos experimentales tales como los modelos de simulación son naturalmente desarrollados para esto por que obtiene alguna

medida o parámetro de lo que sucederá en el sistema como salida del modelo, al evaluar las diferentes opciones que se requieran.

Construir un modelo es un arte. La ciencia juega un papel más importante en la solución del modelo que en su construcción. La construcción de un modelo itera entre el uso del razonamiento deductivo e inductivo, pero no pude haber un modelo único para representar un sistema. Se recomienda construir los modelos más simples posibles que puedan describir un sistema. Si es así, serán más fáciles de usar, mantener y utilizar. La verificación y validación del modelo son los siguientes pasos. La verificación asegura que el modelo representado en papel es correctamente implementado en la computadora y que son equivalentes (Sargent, 1998). La validación asegura que el modelo computarizado produce resultados válidos para tomar decisiones.
Por último aspectos importantes relacionados a la construcción de modelos son:

- Los modelos no deben ser construidos para probar un particular punto de vista.
- El modelo no es el fin sino un medio.

Debido a la complejidad de programar y secuenciar las operaciones en sistemas de manufactura, una serie de supuestos que simplifican el problema siempre han sido utilizados en las aplicaciones prácticas de la programación y secuenciamiento de operaciones. Estos supuestos están ahora implícitos y se han convertido en estándar al formular los problemas, pero en algunas circunstancias son mucho más restrictivos de lo necesario. De hecho los supuestos más comunes son: todos los tiempos de procesamiento son conocidos y no hay restricciones para que los trabajos puedan iniciar. Pero en muchos problemas de programación dichos supuestos no se cumplen o no están bien definidos. Considere un ambiente de manufactura altamente dinámico, con trabajos de prioridades distintas ingresando al sistema en intervalos impredecibles, con máquinas fallando o en reparación, con personal ausente o en proceso de entrenamiento, así como los tiempos de procesamiento inciertos. La importancia del dinamismo y los aspectos estocásticos del problema variarán en cada caso, pero en muchos su consideración es crucial (Husbands, 1994).

Una gran parte de las investigaciones en programación y secuenciamiento de operaciones en configuraciones *jobshop* están limitadas a casos determinísticos; la parte estocástica y dinámica es difícil de manejar. Este es un aspecto importante donde la simulación de eventos discretos tiene algo que ofrecer. Simular eventos discretos consiste en relacionar los diferentes eventos que pueden cambiar el estado de un sistema bajo estudio por medio de distribuciones de probabilidad y condiciones lógicas del problema que se esté analizando. Entonces, el objetivo de un modelo de simulación de eventos discretos consiste, precisamente, en comprender, analizar y mejorar las condiciones de operación relevantes del sistema. Un evento puede ser definido como un cambio en el estado actual del sistema; por ejemplo, la entrada o salida de una parte, la finalización de un proceso, la interrupción o reactivación de una operación. El estado del sistema se puede definir como la condición que guarda el sistema bajo estudio en un momento determinado; es decir, lo que pasa en el sistema en un cierto instante. Una de las principales ventajas de utilizar la simulación de eventos discretos es la mejora el conocimiento del proceso actual al permitir al analista ver la descripción del sistema mediante el modelo y notar cómo se comporta bajo diferentes escenarios y además ayuda a conocer el impacto que se tiene en algún cambio en los procesos propios del sistema bajo estudio sin necesidad de llevarlos a cabo en el sistema realmente.

La realización de un modelo de simulación de eventos discretos requiere el análisis y la ejecución de una serie de actividades que permitan obtener un modelo fiel y creíble del sistema que se pretende modelar. Garcia *et al* (2006) presentan algunas de las siguientes:

- Definición del sistema bajo estudio: en esta etapa es necesario conocer el sistema a modelar. Para ello se requiere saber que origina el estudio de simulación y establecer los supuestos del modelo: es conveniente definir con claridad las variables de decisión del modelo, determinar las interacciones entre estas y establecer con precisión los alcances y limitaciones que aquel podría llegar a tener.
- Generación del modelo de simulación básica: una vez que se ha definido el sistema en términos de un modelo conceptual, la siguiente etapa del estudio consiste en la generación de un modelo de simulación base. No es preciso que este modelo sea demasiado detallado pues se requiere mucha más información estadística sobre el comportamiento de las variables de decisión del sistema. La generación de este modelo es el primer reto para el programador de la simulación toda vez que se debe traducir a un lenguaje de simulación la información que se obtuvo en la etapa de definición del sistema, incluyendo las interacciones de todos los posibles subsistemas que existen en el problema a modelar. En caso de que se requiera una animación, este también es un buen momento para definir que grafico puede representar mejor el sistema que se modela.
- Recolección y análisis de los datos: de manera paralela a la generación del modelo base es posible comenzar la recopilación de la información estadística de las variables aleatorias del modelo. En esta etapa se debe determinar qué información es útil para la determinación de las distribuciones de probabilidad asociadas a cada una de las variables aleatorias necesarias para la simulación.
- Generación del modelo: en esta etapa se integra la información obtenida a partir del análisis de los datos, los supuestos del modelo y todos los datos que se requieran para tener un modelo lo más cercano posible a la realidad del problema bajo estudio.
- Verificación del modelo: se deben realizar las comparaciones necesarias del modelo con el proceso bajo estudio. Se busca comprobar que las acciones y parámetros en el modelo funcionen correctamente, de lo contrario, los ajustes en programación deben ser realizados para garantizar que el modelo representa fielmente el proceso bajo estudio.
- Validación del modelo: se trata de evaluar y analizar los resultados que arroja el modelo de simulación bajo ciertos parámetros de entrada. Los parámetros de entrada pueden ser construidos como escenarios para evaluar que el comportamiento del modelo sea acorde a las expectativas en base a la experiencia de los administradores de los procesos.
- Experimentación: al realizar réplicas de los escenarios que se pretenden evaluar se puede garantizar que los resultados entre las diversas muestras son independientes entre sí, originando así una credibilidad en el modelo ya que describe con suficiente certeza los resultados en el proceso bajo estudio.
- Análisis de sensibilidad: una vez que se obtienen los resultados de los diferentes escenarios es importante realizar pruebas estadísticas que permitan comparar los escenarios con los mejores resultados finales construyendo intervalos de confianza para demostrar su precisión.

1.5.3. OPTIMIZACIÓN DE LAS SIMULACIONES.

Una de las principales razones que subyace la importancia de la optimización de las simulaciones es el hecho de que muchos problemas prácticos en optimización son muy complejos para ser manejados a través de formulaciones únicamente matemáticas. Las relaciones de no-linealidad, combinatorias y la incertidumbre frecuentemente hace inaccesible modelar los problemas a menos que se realice por medio de la simulación. Un resultado que plantea graves dificultades para los métodos de optimización clásica.

La comunidad de investigadores en el área de optimización y en el área de simulación han logrado establecer una metodología para guiar a una serie de simulaciones en busca de soluciones de alta calidad en ausencia de estructuras matemáticas tratables. Las soluciones a los problemas tienen implícito el objetivo a encontrar, por ejemplo:

- La mejor configuración de los equipos en programación de la producción
- El mejor layout de una planta industrial
- La mejor utilización de operadores para planear la fuerza de trabajo

Cuando se intenta optimizar modelos de simulación se trabaja con la situación en la cual al analista le gustaría encontrar de un conjunto de posibilidades las especificaciones idóneas (por ejemplo de parámetros de entrada y/o condiciones de operación en el modelo), buscando así optimizar alguna variable de interés. En el área de diseño de experimentos, los parámetros de entrada y/o condiciones de operación asociadas con el modelo de simulación son llamados *"factores"*. La variable de salida es llamada *"respuesta"*. Por ejemplo, un modelo simulación de una celda de manufactura puede incluir factores tales como el número de máquinas de cada tipo, número de trabajadores, tiempos de proceso, entre otras. Las respuestas pueden ser el porcentaje de utilización, el tiempo muerto, la producción en proceso.
En el contexto de la optimización, los factores llegaran a ser variables de decisión y las respuestas son utilizadas para modelar un función objetivo y restricciones. Mientras que el objetivo de un diseño de experimentos es encontrar cuales factores tienen el mayor efecto sobre la variable de respuesta, la optimización busca la combinación de factores que minimiza o maximiza una respuesta.

En el contexto de la optimización de simulaciones, un modelo de simulación puede ser considerado como un mecanismo que *"convierte"* a los parámetros de entrada en las medidas de rendimiento a la salida del modelo (Law & Kelton, 1991). En otras palabras, un modelo de simulación es una función donde su forma analítica explicita es desconocida, pero que evalúa un conjunto de especificaciones, típicamente representadas por un conjunto de valores.

Fu (2002) identifico cuatro enfoques para optimizar simulaciones:

- Aproximación estocástica (enfoque basado en el gradiente)
- Secuencial (basado en la metodología de superficie de respuesta)
- Búsqueda aleatoria
- Optimización por trayectorias

Los algoritmos de aproximación estocástica intentan imitar el método de búsqueda del gradiente utilizado en optimización determinística. La aproximación estocástica estima el gradiente de la función objetivo para determinar una dirección de búsqueda. La aproximación estocástica tiene como objetivo tratar de resolver problemas de variable continua debido a su estrecha relación con la búsqueda de gradiente en descenso. Sin embargo, este enfoque ha sido aplicado a problemas discretos (Gerencsér, 1999).

El enfoque secuencial está basado en la metodología de superficie de respuesta. La superficie de respuesta *"local"* es utilizada para determinar una estrategia de búsqueda (ejemplo moviendo la dirección del gradiente estimado) y así el proceso se repite. En otras palabras, no se intenta caracterizar la función objetivo en el espacio de solución completa sino que se concentra en el área local donde la búsqueda se está ejecutando.

El método de búsqueda aleatoria se mueve a través del espacio de solución de manera aleatoria al seleccionar un punto *"vecino"* del punto actual. La búsqueda aleatoria ha sido aplicada principalmente a problemas discretos y su propuesta está basada en la existencia de la convergencia. Desafortunadamente, basarse en la convergencia puede no ser muy útil en la práctica (como en los sistemas de manufactura) donde es más importante encontrar una solución de alta calidad en un tiempo razonable que garantizar la convergencia a el óptimo en un tiempo excesivo.

La optimización por trayectorias es una metodología que explota el conocimiento y experiencia desarrollada para resolver problemas de optimización continua determinística. La idea es optimizar una función determinística que es basada en *n* variables aleatorias, donde *n* es el tamaño de la trayectoria. En el contexto de simulación, el método de generación de números aleatorios se utiliza para proporcionar una trayectoria muestra y así calcular la respuesta de acuerdo a los diferentes valores de los factores de entrada. Este enfoque debe su nombre a que la solución óptima *"estimada"* que se obtiene, está basada en una función determinística construida por la trayectoria obtenida con un modelo de simulación. Generalmente *n* debe ser grande para acercar lo más posible la función determinística aproximada a la función determinística original (Andradóttir, 1998).

Los cuatro enfoques mencionados representan la mayor parte de la literatura en la optimización de simulaciones. Sin embargo, no se han utilizado dentro de lenguajes de simulación comerciales. Fu (2002) encontró un solo caso (Optimz de Simul8®), con un procedimiento similar a la metodología de superficie de respuesta. Pero a partir de que el artículo de Fu fue publicado, Simul8® dejo de utilizar Optimz, llevando a cero el número de aplicaciones prácticas de estos enfoques.
A pesar que en años recientes la optimización de simulaciones ha recibido una importante atención de la comunidad de investigadores, estos enfoques generalmente requieren una considerable cantidad de habilidades técnicas por parte del usuario, además de requerir un tiempo computacional excesivo (Andradóttir, 1998).

Los líderes actuales en lenguajes de simulación comerciales utilizan la Inteligencia Artificial para proporcionar herramientas de optimización a sus usuarios. Hoy en día casi todos los lenguajes de simulación comerciales contienen un módulo de optimización que realiza algún tipo de búsqueda de los mejores valores para los parámetros de entrada y no solamente la estimación estadística tradicional. Este es un cambio significativo respecto a 1990 cuando ninguno de los paquetes incluía esta funcionalidad.

De acuerdo con April *et al* (2009), el enfoque de la Inteligencia Artificial aplicado a la optimización de simulaciones está basado en ver al modelo de simulación como una *"caja negra"* que evalúa la función a optimizar. Bajo este enfoque la técnica utilizada para optimizar escoge un conjunto de valores para los parámetros de entrada (por ejemplo factores o variables de decisión) y utiliza la respuesta generada por el modelo de simulación para tomar decisiones sobre la selección del siguiente conjunto de valores para los parámetros de entrada a probar.
La mayoría de las técnicas utilizadas e integradas en los lenguajes de simulación comerciales están basadas en estrategias evolutivas propias del campo de la Inteligencia Artificial. Dichas estrategias buscan en el espacio de solución mediante la construcción y evolución de una población de soluciones. La evolución se logra a través de la recombinación de dos o más soluciones de la actual población. Las estrategias evolutivas utilizadas en lenguajes de simulación comercial son mostradas en la Tabla 1.1.

Tabla 1.1 Enfoques evolutivos para optimización de simulaciones.

Optimizador	Tecnología	Software de Simulación
OptQuest	Búsqueda Tabú	Any Logic, Arena, Crystal Ball, CSIM19, Enterprise Dynamics, Micro Saint, ProModel, Quest, SimFlex, SIMPROCESS, Simul8, TERAS
Evolutionary Optimizer	Algoritmos Genéticos	Extend
Evolver	Algoritmos Genéticos	@Risk
AutoStat	Estrategias Evolutivas	AutoMod

1.6. ALGORITMOS EDAs PARA SECUENCIAMIENTO.

Una discusión sobre las investigaciones más actuales sobre problemas de programación y secuenciamiento utilizando EDAs se describe a continuación.
Chen *et al* (2010a) proponen guías para desarrollar EDAs efectivos para solucionar el problema de secuenciamiento de una máquina simple, particularmente la minimización del costo total promedio de la tardanza. En general, ellos utilizan un EDA con un operador al cual llaman "mutación guiada" para generar descendientes efectivos. Al inicio de su algoritmo, este produce nuevas soluciones únicamente por los operadores genéticos principalmente. Después de esto, ellos utilizan el modelo probabilístico para generar mejores individuos cuando el proceso de búsqueda alcanza un estado más estable. Por lo tanto, muestrear nuevos individuos periódicamente es la característica que lo hace diferente con otros EDAs porque la mayoría de los EDAs generan nuevas soluciones integralmente.

Se han realizado algunos intentos para combinar EDAs con los tradicionales operadores de cruza y mutación de los AGs recientemente (Peña, *et al*., 2004). Chen *et al* (2012b) utilizan este enfoque. Ellos emplean un modelo probabilístico aproximado para estimar la calidad de las soluciones que son candidatos y permiten a los operadores de cruza y mutación generar más soluciones prometedoras. Ellos trabajan en el problema de secuenciamiento en configuración flowshop PFSP (por sus siglas en inglés Permutation Flowshop Scheduling Problem). Este es uno de los problemas más conocidos como "NP-hard". El modelo probabilístico utilizado no es una fuente para generar nuevas soluciones, pero actúa como un predictor de la aptitud del individuo para guiar a los operadores de cruza y mutación para generar mejores soluciones.

Chen *et al* (2012c) trabajan sobre el PFSP también. Ellos emplean dos modelos gráficos probabilísticos, mientras que la mayoría de los EDAs no aplican más de un modelo. El primer modelo representa el número de veces que cualquier trabajo aparece antes o en una posición especifica en la secuencia. Este modelo muestra la importancia de los trabajos en la secuencia, y fue utilizado en el trabajo de Jarboui *et al* (2009) también. El segundo modelo indica si cualquier trabajo esta inmediatamente después de otro trabajo específico en las secuencias, es decir, este modelo indica el número de veces que cualquier trabajo está inmediatamente después de otro trabajo específico. Además, es importante notar que al combinar el enfoque de un operador genético con un modelo probabilístico, los autores fueron capaces de rechazar la perdida de diversidad que los AEDs a menudo muestran.

Pan & Ruiz (2012) ofrecen un EDA para problemas de secuenciamiento por lotes en configuración flowshop con tiempos de preparación. Como ellos explican, en una configuración tradicional flowshop, cada trabajo se asume indivisible y por tanto, este no puede ser transferido a la máquina siguiente hasta que se termine la operación en la máquina actual. Sin embargo, en muchos ambientes prácticos donde un trabajo o lote consiste de muchos elementos idénticos no es el caso. Una contribución real es como Pan y Ruiz manejan el tiempo de preparación de máquinas en su investigación.

Wang *et al* (2012) trabajan en el problema de secuenciamiento en configuración jobshop flexible FJSP (por sus siglas en inglés Flexible Jobshop Scheduling Problem). Los autores proponen un EDA basado en una doble población, el algoritmo es llamado BEDA para resolver el FJSP con el criterio de minimizar tiempo máximo de realización de los trabajos. En el BEDA, la población puede ser dividida en dos subpoblaciones con un criterio de separación, y las dos subpoblaciones pueden ser combinadas como una población entera, con un criterio de combinación para conseguir una calidad satisfactoria de búsqueda.

Todo este trabajo de investigación actual usa EDAs discretos. En este tipo de EDAs, cada individuo muestra su información sobre la secuencia de trabajos a ser procesados explícitamente. La hibridación entre cualquier EDA discreto y cualquier método heurístico permite obtener soluciones prometedoras. Los modelos probabilísticos utilizados en todo este trabajo son actualizados cada vez que un trabajo es asignado a la secuencia. Esta actualización elimina la posibilidad de escoger un trabajo previo; aunque los autores de toda esta investigación casi nunca mencionan que una modificación en el proceso de muestreo tiene que ser llevado a cabo explícitamente. Por ejemplo, Shim *et al* (2011) utilizan EDAs para resolver el problema del agente viajero multiobjetivo. Como ellos explican el mecanismo de muestreo no considera que ciudad ha sido o no incluida en la ruta. Para obtener soluciones factibles, un operador de refinamiento es propuesto para hacer frente al inconveniente de la representación basada en permutaciones.

Finalmente, aunque algunos resultados prometedores han sido reportados al utilizar interacciones de orden superior en los modelos probabilísticos utilizados en los EDAs, no necesariamente superan a los modelos simples para hacer frente a algunos problemas de la vida real debido a que estos modelos complicados solo puede considerar un porcentaje muy pequeño de las interacciones de las variables en un problema complejo (Chen *et al* 2012c). Debido a esto, el EDA propuesto es una buena alternativa para hacer frente a esta situación. Las principales diferencias y similitudes entre todo este trabajo de investigación actual es mostrado en la Tabla 1.2.

Tabla 1.2 EDAs para Secuenciamiento.

Elementos	Algoritmos						
	Chen *et al* (2010a) EA/G	Chen *et al* (2012b) GA - autoguiado	Chen et al., (2012c) eACGA	Jarboui et al., (2009) JEDA	Pan and Ruiz (2012)	Wang et al., (2012) BEDA	EDA Propuesto
Configuración	Máquina simple	PFSP	PFSP	PFSP	PFSP por lotes	FJSP	FJSP
Tiempos Proceso	Fijo	Fijo	Fijo	Fijo	Fijo	Fijo	Variable
Supuestos	Requeridos	Requeridos	Requeridos	Requeridos	Requeridos parcialmente	Requeridos	No Requeridos
Modelo Probabilístico	Univariado	Univariado	Uni/Bivariado	Univariado	Uni/Bivariado	Bivariado	Bivariado
Tipo de EDA	Discreto	Discreto	Discreto	Discreto	Discreto	Discreto	Discreto y Continuo
Hibridación	EDA-GA	EDA-GA	EDA-GA	EDA-VNS	EDA-VNS	EDA-Local Search	EDA
Población	Una	Una	Una	Una	Una	Dos	Una
Objetivo	Costo total promedio de tardanza	Makespan	Makespan	Tiempo total de flujo	Makespan	Makespan	Work in Process
Enfoque	Académico	Académico	Académico	Académico	Industrial, secuencia dependiente del tiempo de preparación	Académico	Industrial, secuencia dependiente de las horas fuera de servicio

PFSP por sus siglas en inglés Permutation Flowshop Scheduling Problem
FJSP por sus siglas en inglés Flexible Jobshop Scheduling Problem
VNS por sus siglas en inglés Variable Neighborhood Search

METODOLOGÍA

2.1. MODELADO POR SIMULACIÓN.

Para realizar modelos por simulación, es ampliamente recomendable establecer una técnica que permita generar un modelo en particular y que garantice ser viable y creíble del sistema a modelar. En base al trabajo de investigación propuesto por Sargent (2007), no existe una teoría para elegir una apropiada técnica de modelado por simulación, ya que los sistemas a estudiar pueden ser tan diferentes entre sí. Por lo tanto, se utiliza un proceso básico de modelado descrito en la Figura 2.1

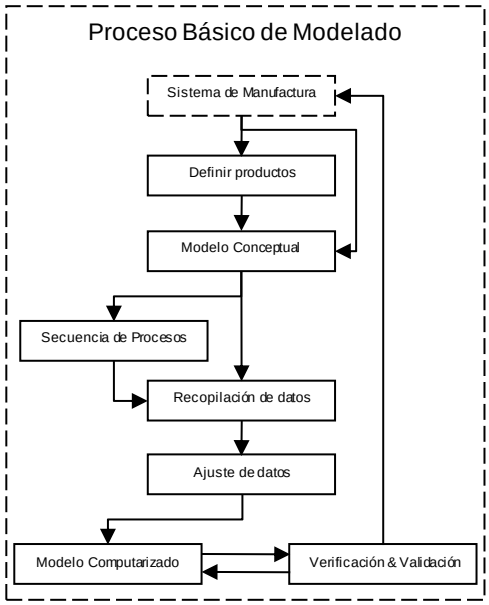


Figura 2.1 Proceso Básico de Modelado por Simulación.

2.1.1. DEFINIR PRODUCTOS.

La primera tarea para desarrollar de este modelo fue determinar los principales productos que ofrece la compañía a sus clientes. Esto con el fin de acotar el alcance del modelo y de la investigación.
A través de un diagrama de Pareto se pudo establecer que son ocho los principales modelos de puertas de acero que la compañía tiene posicionado en el mercado, mismos que se pueden apreciar en la Figura 2.2. Además, la compañía estableció la estrategia de lanzar al mercado una variante de los modelos actuales, lo que origina un total de doce modelos a simular, véase Figura 2.3

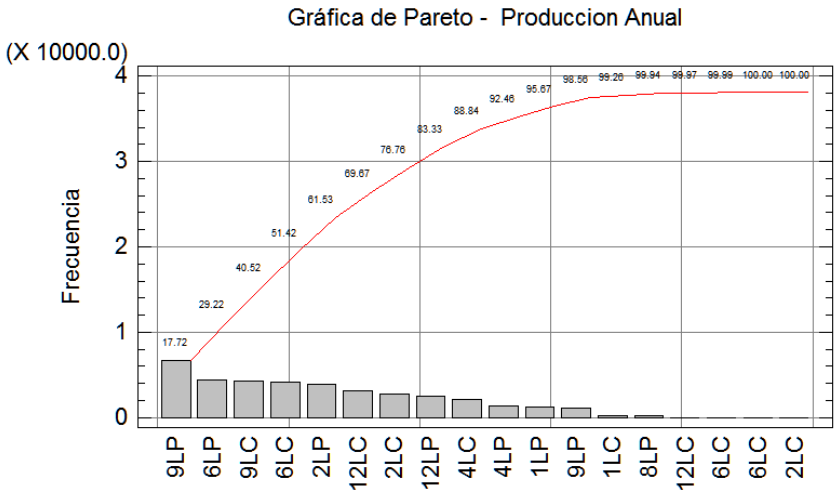


Figura 2.2 Diagrama de Pareto de los principales productos posicionados en el mercado.

	Modelo
Puertas	12LP - 12 Luces / pasador
	12LC - 12 Luces / chapa
	12LB - 12 Luces / chapa de bola
	2LP - 2 Luces / pasador
	2LC - 2 Luces / chapa
	2LB - 2 Luces / chapa de bola
	6LP - 6 Luces / pasador
	6LC - 6 Luces / chapa
	6LB - 6 Luces / chapa de bola
	9LP - 9 Luces / pasador
	9LC - 9 Luces / chapa
	9LB - 9 Luces / chapa de bola

Figura 2.3 Modelos de puertas de acero definidos a simular.

2.1.2. MODELO CONCEPTUAL.

Se elaboró un modelo conceptual del sistema de manufactura bajo estudio, a un nivel de detalle apropiado para ser considerado razonable. El modelo adoptado es un modelo gráfico descriptivo que contiene todo tipo de condiciones de operación idénticas a las que se encuentran en el sistema mencionado.

El modelo conceptual propuesto establece las relaciones que existen entre los diversos procesos de manufactura, así como las entidades de mayor impacto e importancia. Además, se enuncia en el mismo todas aquellas condiciones críticas de operación que no pueden ni deben ser simplificadas a fin de evitar supuestos que no podrían se justificados al momento de validar el modelo de simulación.

La Figura 2.4 muestra parte del modelo conceptual descrito en las distintas áreas de manufactura propias del sistema modelado.

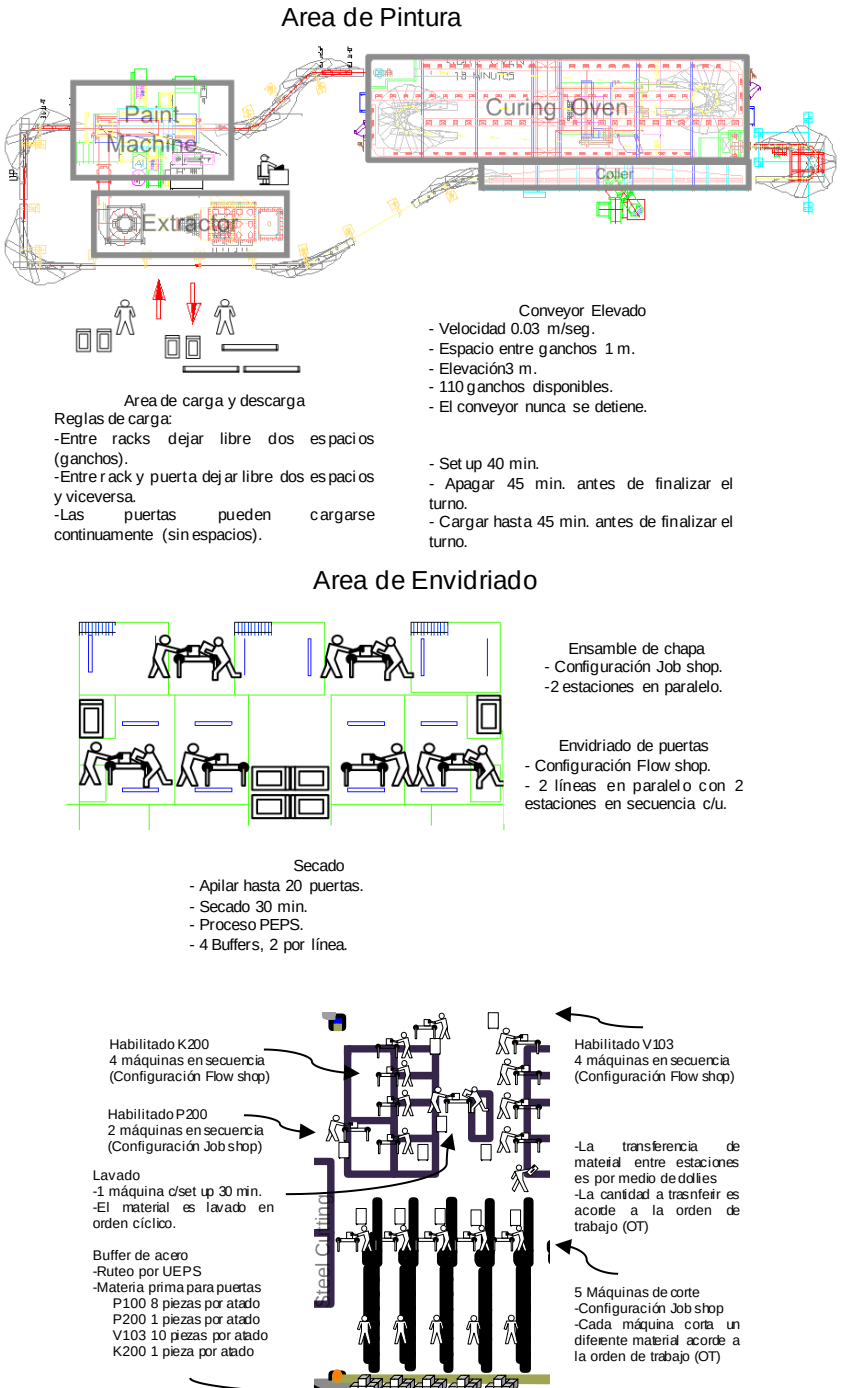


Figura 2.4 Modelo conceptual definido en diversas áreas del sistema.

2.1.3. SECUENCIA DE PROCESOS.

Un levantamiento de la secuencia de los procesos requeridos para construir cada modelo de puerta es elaborado con el fin de satisfacer dos aspectos elementales: tener claro el flujo de las operaciones que se tienen al producir cada modelo de puerta para poder integrar este en el lenguaje de simulación a utilizar y determinar cuáles serán las operaciones en las que se deben recopilar los datos necesarios para determinar sus tiempos de procesamiento.
Para esto, se enlistan todas las operaciones que se requieren ejecutar en cada modelo de puerta a simular, generando una matriz de secuencia de procesos donde se puede visualizar la configuración *jobshop* que tiene la manufactura de puertas de acero. La Figura 2.5 describe parte de la matriz mencionada.

Codigo de Operaciones	Operaciones	12LP
	HERRERIA	
	CORTE PERFILES	
H1	Corte de bastidor de hoja	x
H1.1	Habilitado para chapa bola	
H2	Corte de bastidor de ventila	
H2.1	Habilitado para bisagra en ventila	
H3	Ranura para chapa	
H4	Ranura de chapa de bola	
H5	Corte de intermedio	x
H5.1	Habilitado para chapa bola	
H6	Resaque de las laterales de intermedio	x
H7	Corte de resaque frontales de intermedio	x
H8	Quitar resaque	x
H9	Corte de reticula	x
H10	Resaque de laterales de reticula	x
H11	Corte de resaques frontales de reticula	x
H12	Quitar rebaba	x
H13	Transporte Dolli	x
H14	Lavado de piernas	x
H15	Lavado de cabezal	x
H16	Lavado de retícula e intermedio	x
H17	Transporte de perfiles lavados	x
AN1	Corte de antepecho	
AN2	Habilitado de antepecho	
AN3	Transporte de antepechos a lavado	
	CORTE DE TABLEROS	
H18	Transporte de lamina galvanizada	
H19	Corte de tableros	
H20	Traslado a doblez	
H21	Doblez del tablero	
H22	Traslado a soldadura	
	SOLDADURA	
H23	Armado y punteado de ventila	
H23.1	Soldado de ventila	
H23.2	Pulir ventila	
H24	Armado y soldado de bastidor	x
H25	Colocación y punteado de retícula e intermedio	x
H26	Soldar cara 1	x
H27	Soldar cara 2	x
H28	Colocar y soldar tablero	
H29	Pulir cara 1	x
H30	Pulir cara 2	x
H31	Habilitado(Perforaciones) y revisión	x

Figura 2.5 Matriz de secuencia de procesos para una puerta 12LP en el área de Ensamble.

2.1.4. RECOPILACIÓN DE DATOS.

Una vez que se tiene definido el flujo de las operaciones a partir de la matriz de secuencia de procesos, el siguiente paso es recopilar los tiempos de procesamiento de cada operación en tiempo real, directamente de las distintas estaciones de trabajo que el sistema de manufactura tiene. La información entonces es enviada a través de la red inalámbrica que la compañía posee a una base de datos que el simulador lee directamente. Esto es posible a través del uso de dispositivos electrónicos tales como *"handhelds"* donde se incorpora un software especial para dicha tarea. Así la recopilación de los tiempos es posible sin importar los espacios restringidos de la estaciones de trabajo, las altas temperaturas existentes en algunas áreas, o bien las operaciones con alto riesgo como el sueldado de perfiles. La Figura 2.6 muestra el dispositivo electrónico mencionado.

Figura 2.6 Handheld.

2.1.5. AJUSTE DE DATOS.

El ajuste de los datos para determinar cuál es la mejor distribución de probabilidad asociada al tiempo de cada operación definida por la matriz de secuencia de procesos es realizado a través del software estadístico especializado Statgraphics® utilizando la prueba de bondad de ajuste Kolmogorov-Smirnov. Una vez que se tiene definido la distribución a emplear, los parámetros que aplique, como la media y la varianza, son utilizados en la construcción del modelo. La Figura 2.7 presenta el ajuste a los datos en una operación del área de ensamble.

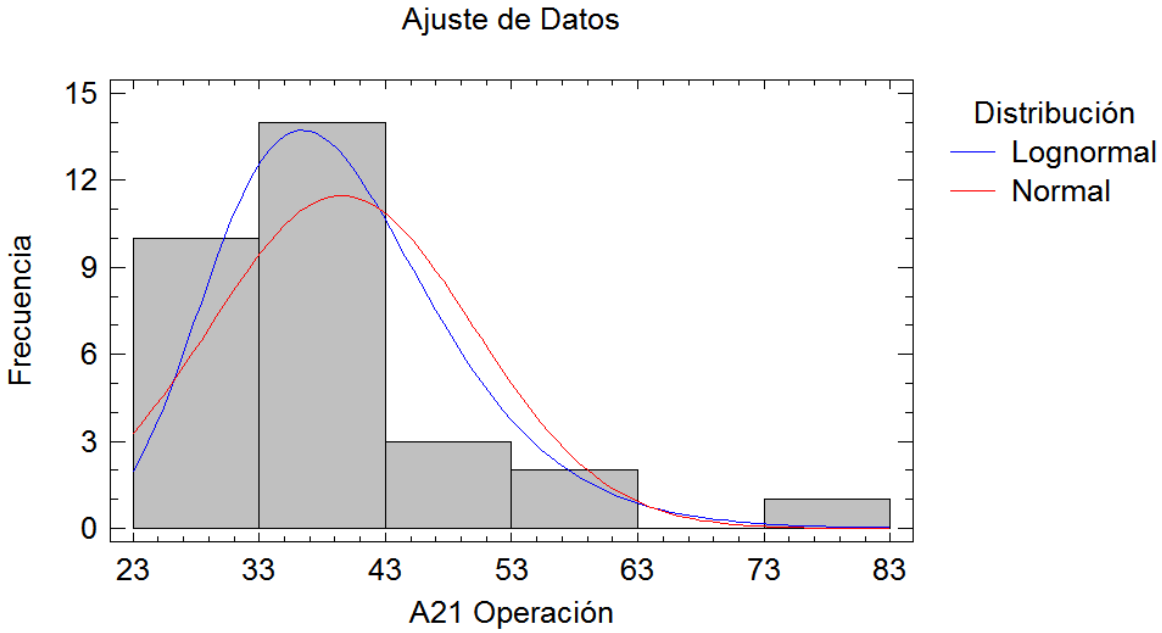


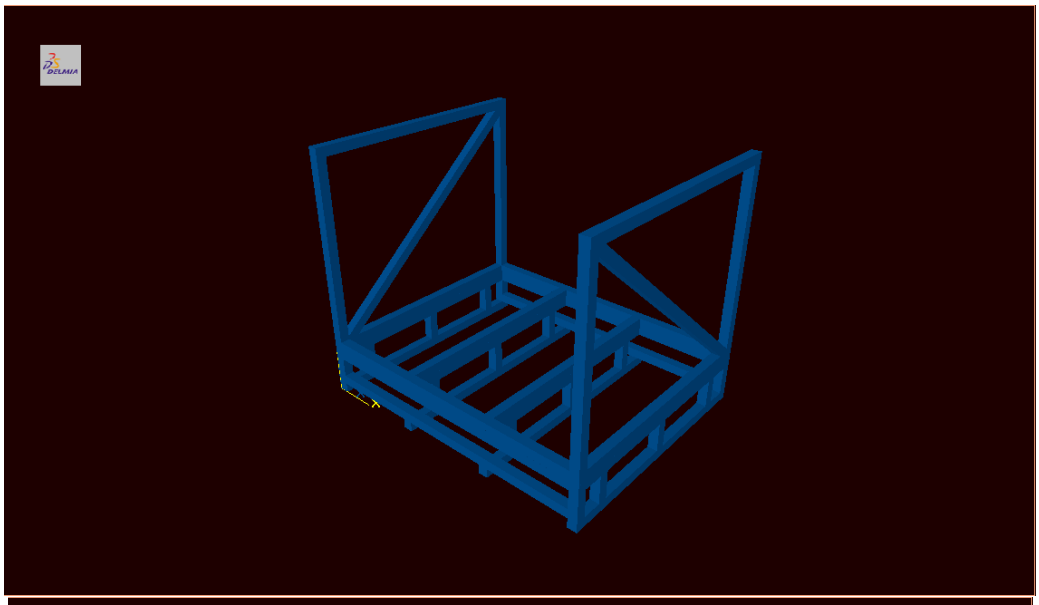
Figura 2.7 Ajuste de datos mediante la prueba Kolmogorov-Smirnov.

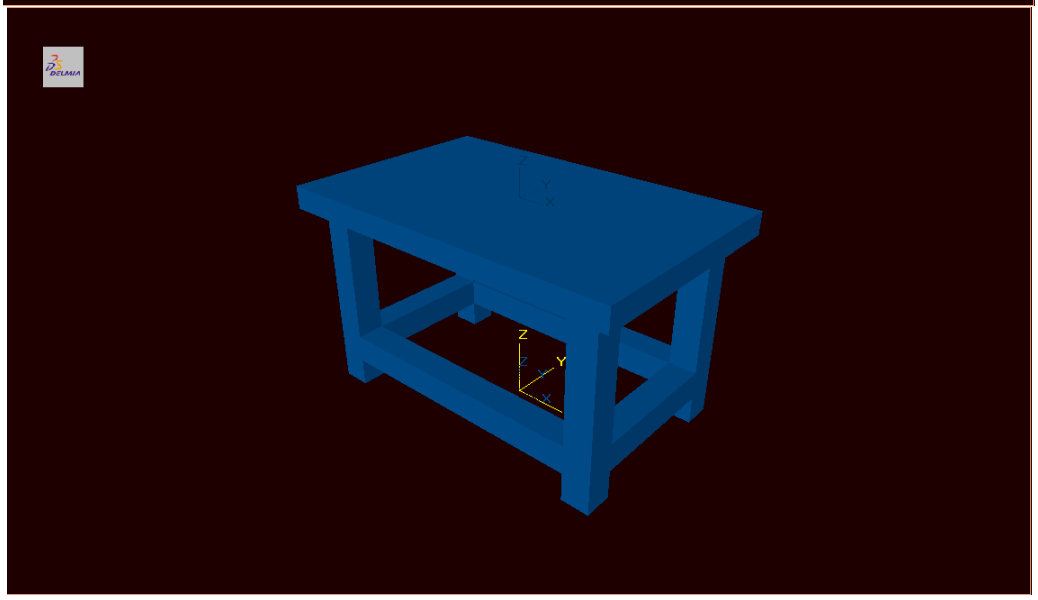
2.1.6. MODELO COMPUTARIZADO.

El uso de un software especializado para realizar modelos de simulación por computadora incide en la posibilidad de tener un modelo más preciso. Un software especializado por lo tanto, contribuye de manera significativa en asegurar que la implementación del modelo conceptual previamente definido sea la correcta. A esto se le conoce como verificación del modelo de simulación (Sargent, 2007). En esta investigación se utiliza Delmia Quest®, un lenguaje de simulación que permite reducir errores en la implementación y tiempo en la construcción del modelo significativamente.

2.1.6.1. CONSTRUCCIÓN DE ELEMENTOS.

Todos los componentes del modelo de simulación han sido diseñados en la herramienta CAD (por sus siglas en inglés Computer Aided Design) que ofrece Quest®.
Las partes, producción en proceso, productos, máquinas, almacenes, buffers, conveyors, polipastos, grúas móviles, estaciones de trabajo, herramientas de trabajo, racks, facilidades y en general todo elemento de la planta de manufactura muestran un realismo importante que ayuda a garantizar la verificación del modelo. La Figura 2.8 muestra algunos de los elementos diseñados para el modelo de simulación.




Z
Y
X
Z
Y

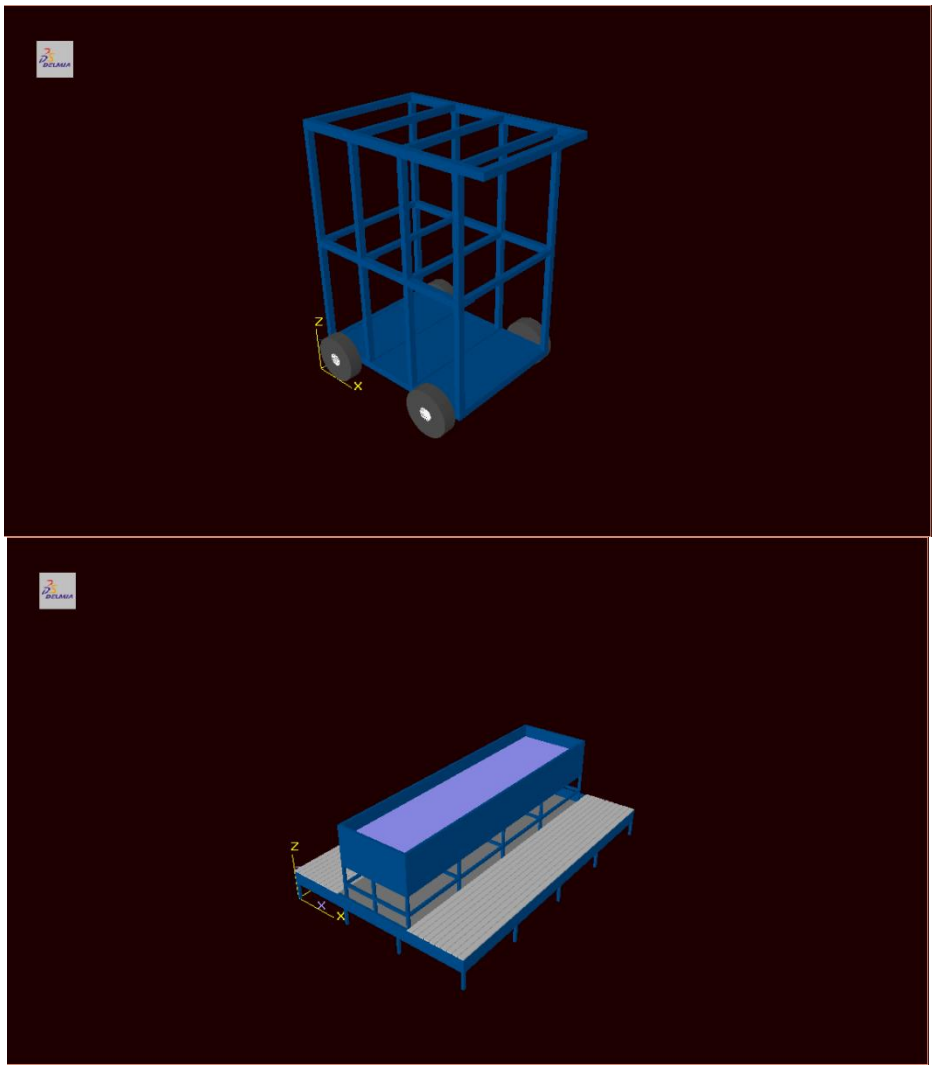


Figura 2.8 Facilidades dibujadas del modelo de simulación.

2.1.6.2. IMPORTACIÓN DE LAYOUT.

Así mismo, la validación correcta de los espacios de cada elemento en el modelo computarizado es posible al importar el layout de la planta de manufactura diseñado previamente para determinar la veracidad de las dimensiones y ubicaciones y en si del flujo tanto de materiales como de personas. La Figura 2.9 muestra el layout de la planta diseñado en Autocad®.

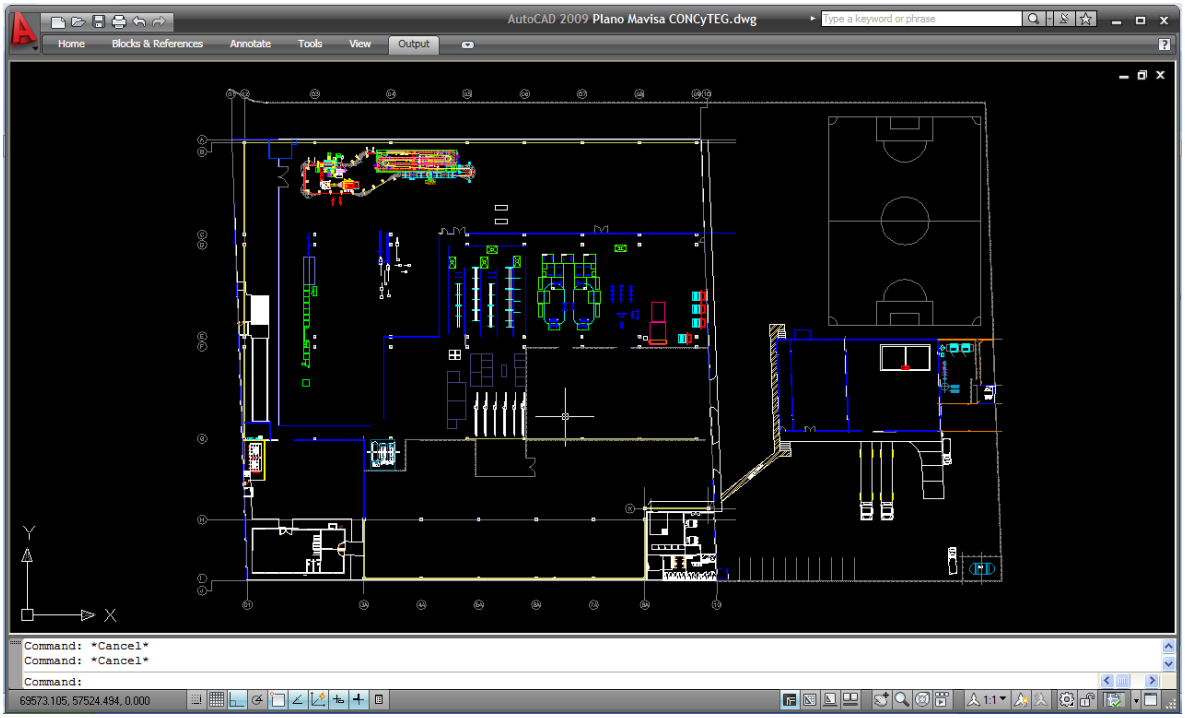


Figura 2.9 Layout.

2.1.6.3. CONSTRUCCIÓN DE PARTES.

Las partes en Quest® representan las piezas, tanto materia prima como producción en proceso, que serán utilizadas en el modelo de simulación y representan los componentes que se requieren para producir las puertas de acero como producto terminado. De hecho las puertas de acero como producto terminado también se declaran como partes. La Figura 2.10 presenta la ventana de alta de partes. La Figura 2.11 presenta los comandos requeridos para relacionar la parte a un dibujo construido de acuerdo a la sección 2.1.6.1.

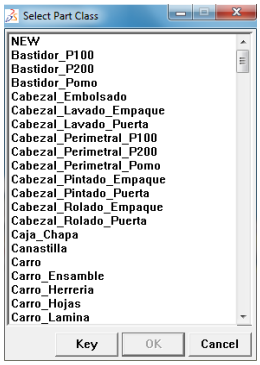


Figura 2.10 Ventana de alta de partes.

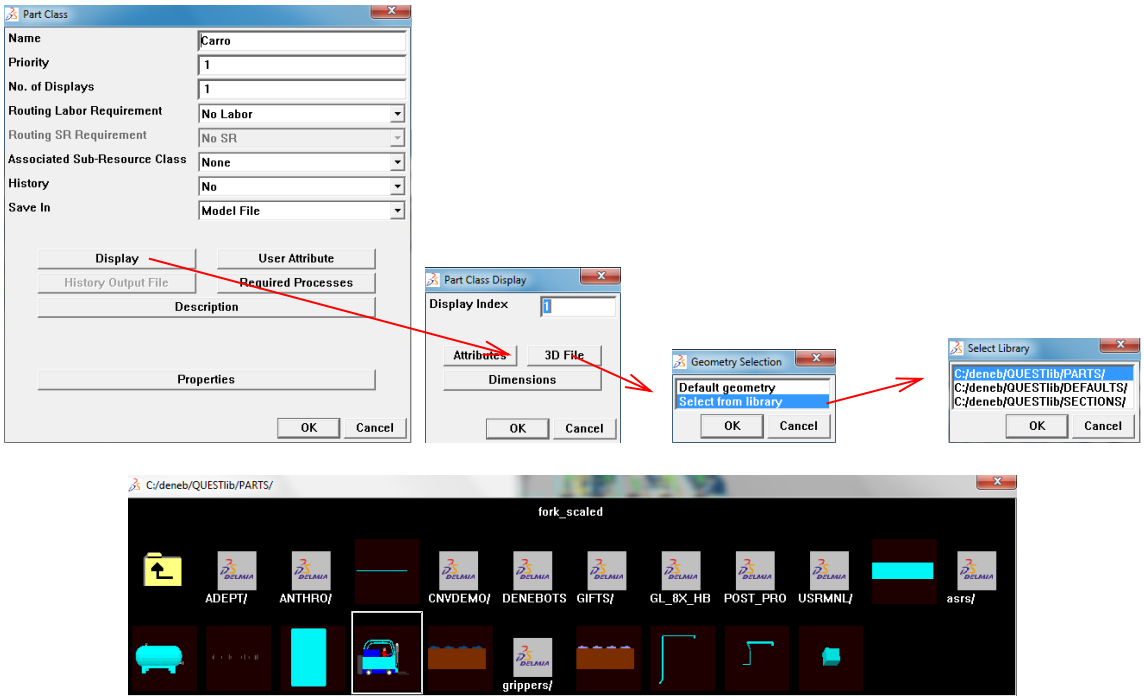


Figura 2.11 Comandos de la ventana de partes.

2.1.6.4. CONSTRUCCIÓN DE EQUIPOS.

Los equipos se utilizan para manufacturar las partes y procesarlas para convertirlas en producción en proceso y finalmente en producto terminado. Los equipos principales en este modelo de simulación son máquinas, estaciones de trabajo, y herramientas. Todos los equipos a definir son construidos por la opción 'Machine' en el menú de construcción de Quest®. La Figura 2.12 presenta la ventana de alta de equipos. La Figura 2.13 presenta sus principales comandos.

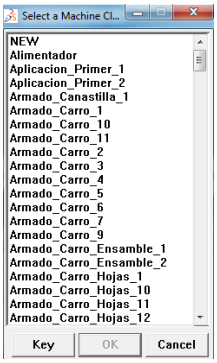


Figura 2.12 Ventana de alta de equipos.

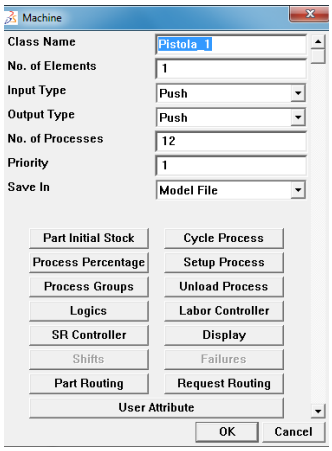


Figura 2.13 Principales comandos de la ventana de equipos.

2.1.6.5. DECLARACIÓN DE PROCESOS.

Los procesos representan las tareas u operaciones que un equipo debe desempeñar con el fin de realizar la manufactura de las partes y procesarlas. Cada proceso que se requiere para construir un producto terminado (puerta) debe ser definido. Todos los procesos son declarados en el menú 'Process' en el menú de construcción de Quest®. La Figura 2.14 muestra la ventana de alta de procesos.

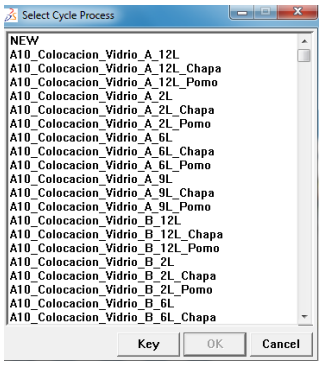


Figura 2.14 Ventana de alta de procesos.

2.1.6.6. INICIALIZACIÓN DE PROCESOS.

Una vez que se ha definido un proceso, es muy común que se distinga de otros procesos. Para ello, tres aspectos clave deben ser revisados al inicializar cada proceso. La parte o las partes requeridas para desarrollar el proceso, el tiempo de la operación involucrada y la parte o partes a salir que se obtienen con el proceso. La Figura 2.15 visualiza el comando utilizado para declarar las partes requeridas en el proceso. La Figura 2.16 visualiza el comando utilizado para declarar el tiempo de la operación y la Figura 2.17 visualiza el comando utilizado para declarar las partes a salir.

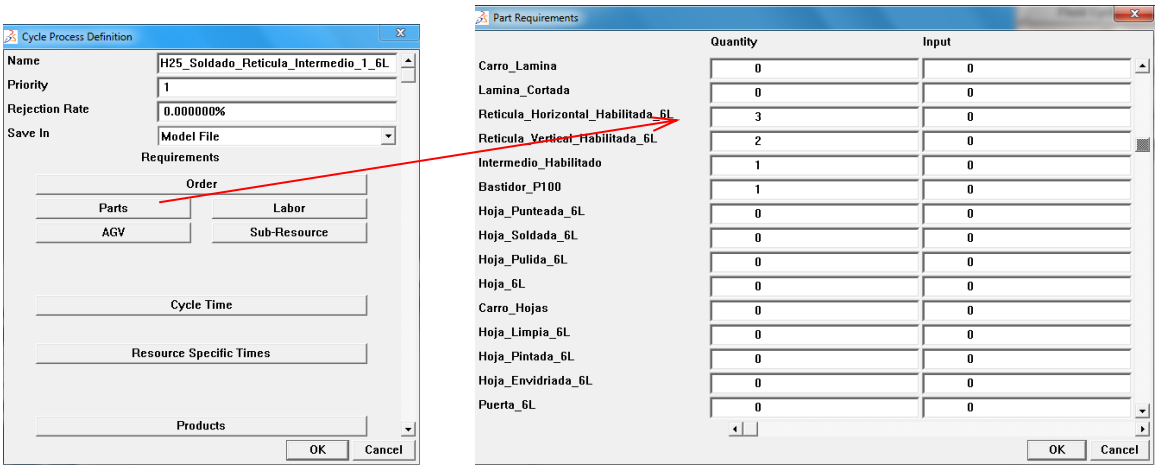


Figura 2.15 Declaración de partes requeridas.

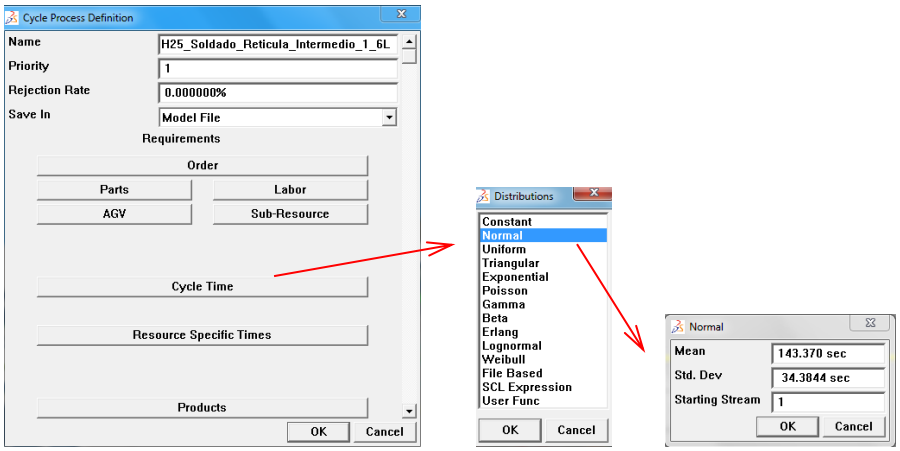


Figura 2.16 Declaración del tiempo de operación.

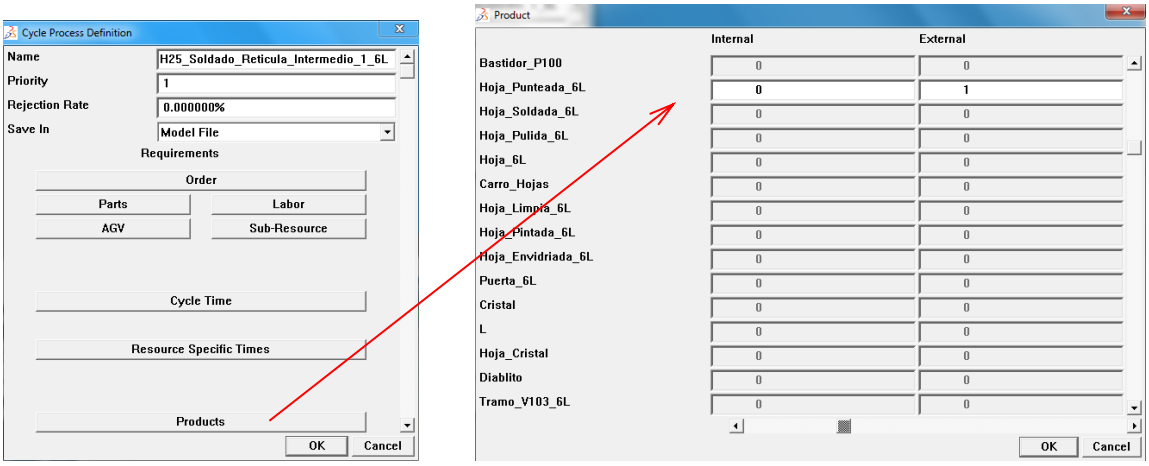


Figura 2.17 Declaración de partes a salir.

2.1.6.7. SECUENCIAMIENTO DE PROCESOS.

Una vez que se han definido los procesos, estos deben ser integrados en los equipos que los requieran. La Figura 2.18 presenta la ventana de relación proceso-equipo. Así mismo, debe declararse los procesos precedentes entre ellos. La Figura 2.19 presenta los comandos para definir procesos procedentes.

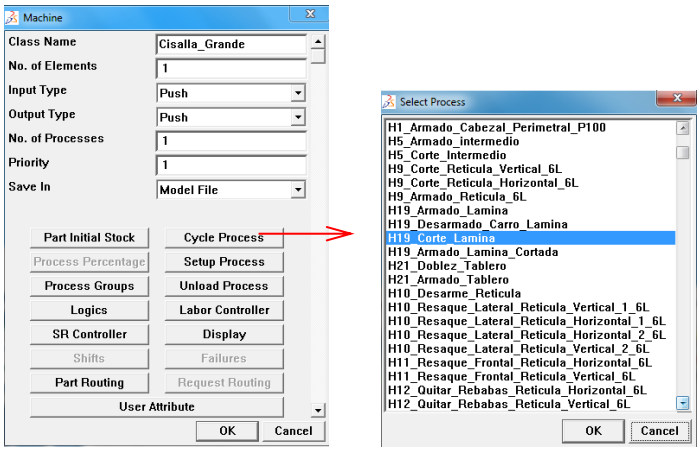


Figura 2.18 Ventana de relación proceso-equipo.

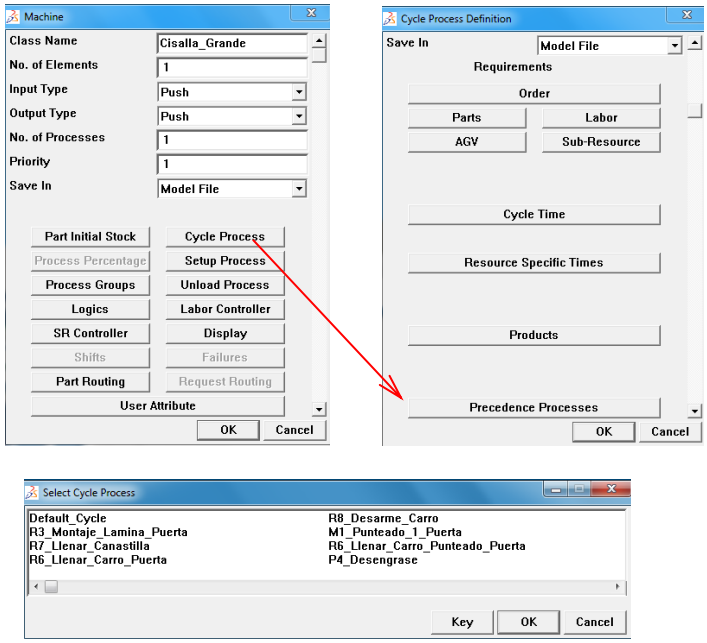


Figura 2.19 Relación de procesos precedentes.

2.1.6.8. CONSTRUCCIÓN DE FUENTES.

A pesar de que se han construido las partes, estas no se presentarán en el modelo hasta que un elemento las genere. Las fuentes son elementos que producen las partes (al menos las de inicio como la materia prima) para el modelo de simulación. Es común que al menos una fuente se encuentre en cada modelo. La Figura 2.20 visualiza la ventana de construcción de una fuente. La Figura 2.21 visualiza los comandos para determinar que partes producirá y en qué tiempo lo hará.

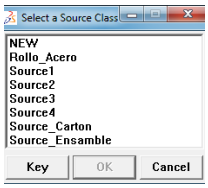


Figura 2.20 Ventana de construcción de una fuente.

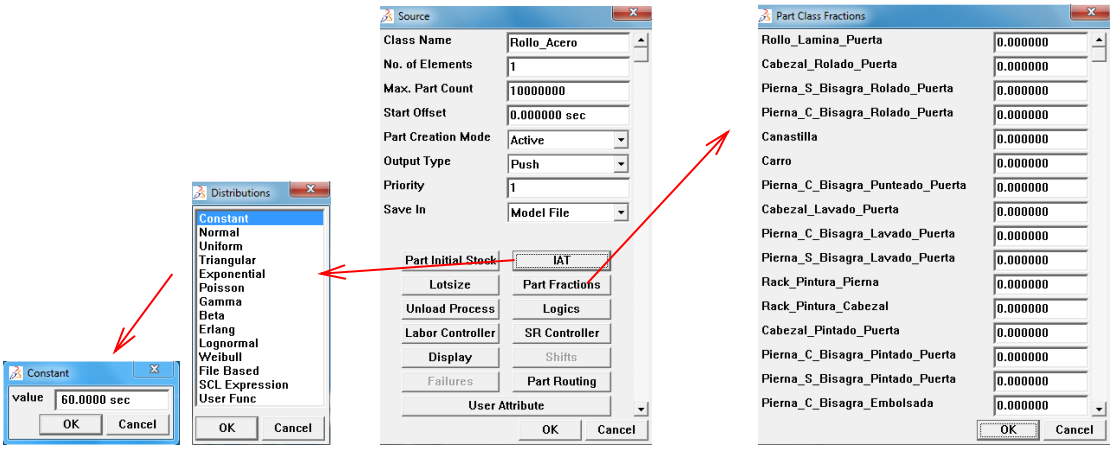


Figura 2.21 Comandos de una fuente.

2.1.6.9. CONSTRUCCIÓN DE SUMIDEROS.

Contrario a las fuentes, los sumideros son elementos en el modelo que destruyen las partes ya previamente procesadas. Es la manera que cualquier parte puede eliminarse del modelo a su salida. También sirve para disminuir el uso de la memoria en la computadora. La Figura 2.22 muestra la ventana de construcción de un sumidero.

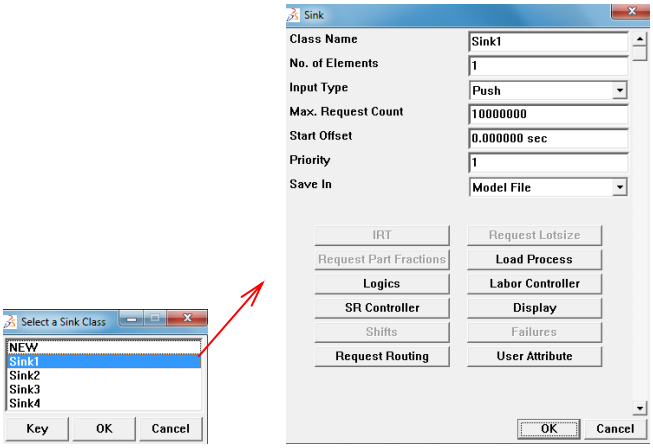


Figura 2.22 Ventana de construcción de un sumidero.

2.1.6.10. CONEXIÓN DE ELEMENTOS.

El flujo de las partes en el modelo de simulación puede ser hecho a través de la conexión de los elementos involucrados. Esto es posible en Quest® a través de los comandos mostrados en la Figura 2.23

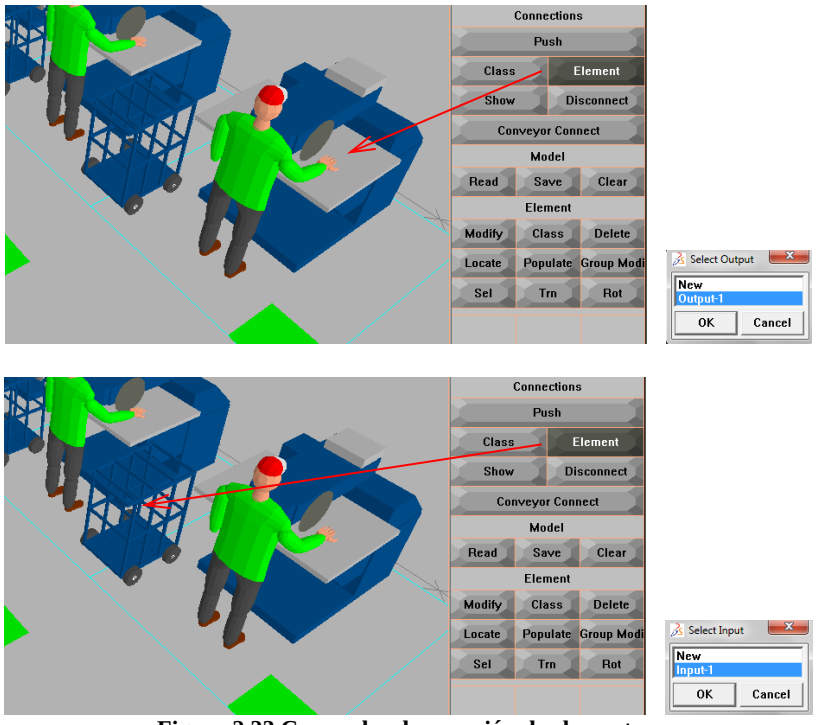


Figura 2.23 Comandos de conexión de elementos.

2.1.6.11. MODIFICACIÓN DE CONEXIONES (PRIMERA PARTE).

Sin embargo, la conexión solo garantiza el flujo de todas las partes sin restringir las mismas. Si se requiere establecer que partes pasarán al siguiente elemento, las conexiones deben sufrir una modificación. La Figura 2.24 presenta los comandos requeridos para esto.

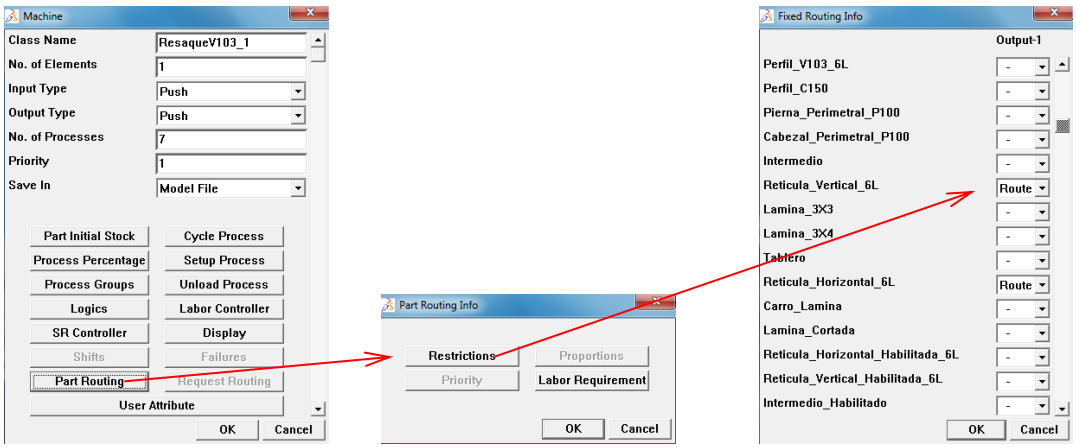


Figura 2.24 Modificación de flujo de partes entre elementos.

2.1.6.12. CONSTRUCCIÓN DE OPERADORES.

Los operadores son parte imprescindible en el proceso de manufactura de puertas de acero. Ellos no solo realizan una importante cantidad de operaciones de manufactura, sino también, son responsables de la transferencia de materiales entre estaciones y del manejo de los equipos.
La construcción de operadores en el modelo comienza por declarar un supervisor que controla las acciones de los operadores para garantizar la ejecución correcta del modelo. La Figura 2.25 detalla la ventana de construcción del supervisor. La Figura 2.26 detalla la ventana de construcción de operadores.

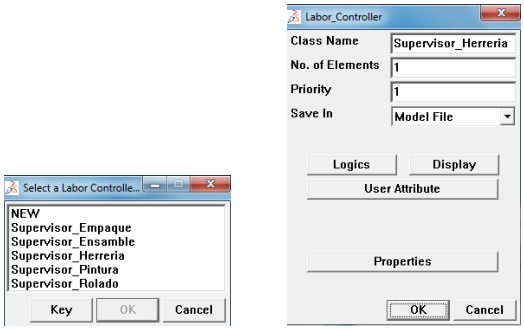


Figura 2.25 Ventana de construcción de un supervisor.

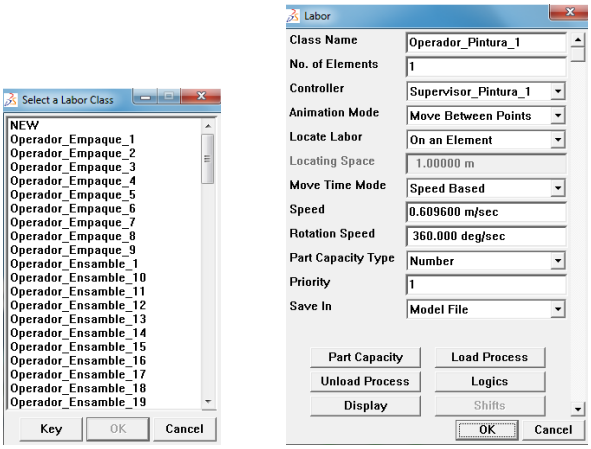


Figura 2.26 Ventana de construcción de operadores.

2.1.6.13. RELACIÓN DE OPERADORES A PROCESOS.

El hecho de construir operadores no significa que estos ya están integrados a los procesos previamente definidos en la sección 2.1.6.6. Para ello se requiere relacionar al operador encargado de la operación en el proceso mismo. La Figura 2.27 muestra los comandos de relación operador-proceso.

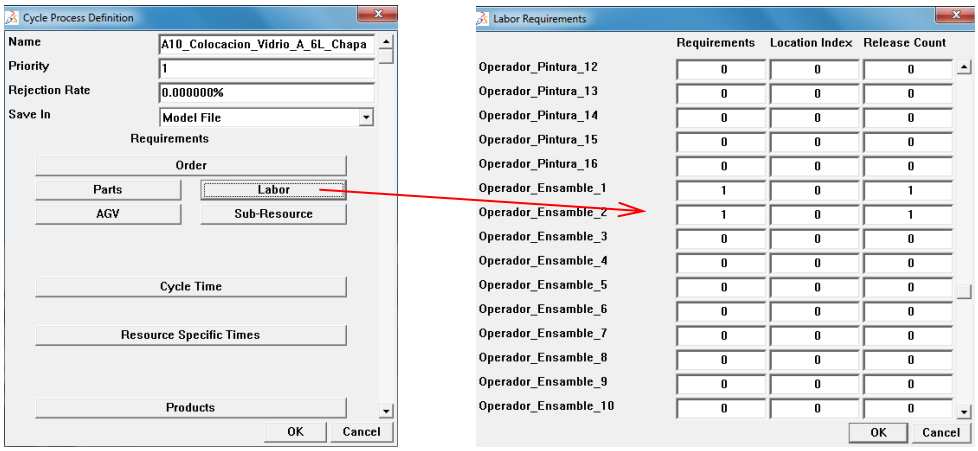


Figura 2.27 Comandos de relación operador-proceso.

2.1.6.14. MODIFICACIÓN DE CONEXIONES (SEGUNDA PARTE).

Si dentro del flujo de partes se requiere que un operador transporte las mismas dentro de los equipos, es necesario modificar las conexiones ya preestablecidas en la sección 2.1.6.11. La Figura 2.28 presenta los comandos para relacionar al operador con el flujo de las partes a transportar.

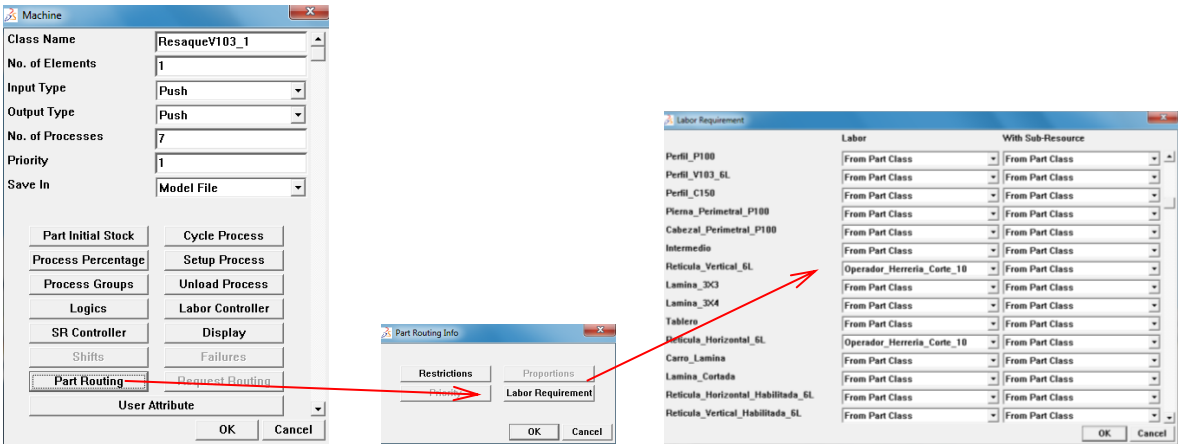


Figura 2.28 Comandos de relación operador-flujo de partes.

2.1.6.15. VERIFICACIÓN DE TRAYECTORIAS.

Una vez que los elementos, componentes, procesos, operadores, partes y conexiones han sido hechos, se requiere realizar una serie de corridas de simulación para confirmar que se está realizando de manera correcta la interpretación del modelo conceptual sobre el modelo de simulación. La Figura 2.29 describe los comandos para ejecutar corridas de simulación.

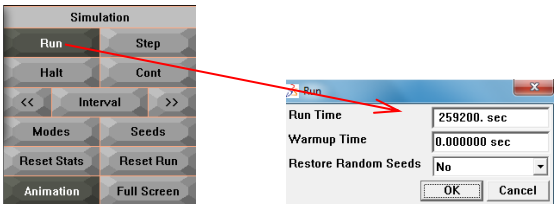


Figura 2.29 Ejecución de corridas de simulación.

2.1.6.16. MODIFICACIÓN DE PUNTOS DE LOCALIZACIÓN (POINTS) Y TRAYECTORIAS (PATHS).

De manera predeterminada Quest® construye para cualquier modelo de simulación, una serie puntos que localizan las partes, elementos, componentes y operadores, así como las trayectorias que realizarán estos últimos en el modelo. Sin embargo, estos puntos y las trayectorias pueden no estar en la ubicación, posición y dirección que se tiene en el sistema de manufactura en realidad. Para ello, se debe modificar tantos los puntos como las trayectorias que genera de manera predeterminada el software para poder recrear las mismas condiciones del sistema bajo estudio.

2.1.6.17. MODIFICACIÓN DE PUNTOS DE APILAMIENTO (STACKPOINTS).

Las partes al ser procesadas o transferidas entre elementos del modelo son ubicadas en puntos denominados StackPoints. La Figura 2.30 describe los comandos requeridos para: visualizar el punto sobre el modelo. Además se puede modificar su ubicación, su orientación y ocultar el punto.

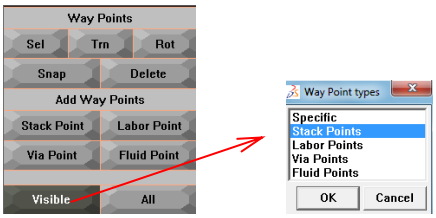


Figura 2.30 Visualización de un punto localizador de partes.

2.1.6.18. MODIFICACIÓN DE PUNTOS DE POSICIÓN PARA OPERADORES (LABORPOINTS).

Los operadores al realizar sus tareas de acuerdo al modelo construido son ubicados en puntos denominados LaborPoints. Es en estos puntos donde los operadores parten y arriban a otros elementos. La Figura 2.31 describe los comandos requeridos para: visualizar el punto sobre el modelo. Además se puede modificar su ubicación, su orientación y ocultar el punto.

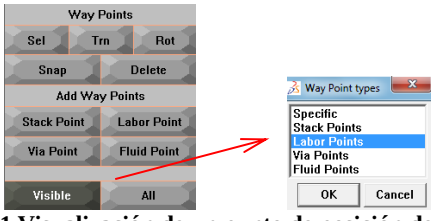


Figura 2.31 Visualización de un punto de posición de operador.

2.1.6.19. MODIFICACIÓN DE PUNTOS DE TRAYECTORIA DE OPERADORES (VIAPOINTS).

Los operadores al realizar sus tareas de acuerdo al modelo construido son desplazados a través de una trayectoria que tiene como punto inicial y final puntos denominados ViaPoints. Es en estos puntos donde los operadores realizan sus recorridos entre elementos. La Figura 2.32 describe los comandos requeridos para: visualizar el punto sobre el modelo. Además se puede modificar su ubicación, su orientación y ocultar el punto.

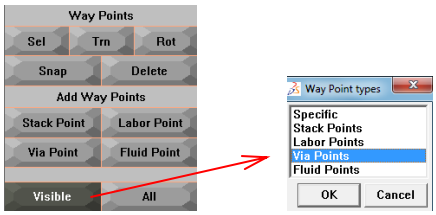


Figura 2.32 Visualización de un punto de trayectoria de operador.

2.1.7. VERIFICACIÓN Y VALIDACIÓN DEL MODELO.

La verificación del modelo de simulación es esencial para asegurar que el modelo conceptual elaborado previamente y su implementación es correcto (Sargent, 2007). Pero aunque el lenguaje utilizado Quest® es un lenguaje desarrollado para reducir errores y tiempo de programación significativamente al construir modelos de simulación, su flexibilidad puede no ser suficiente. Algunas lógicas predeterminadas que el lenguaje ofrece pueden llegar a no ser las mejores para diferentes y diversos sistemas de manufactura. Además, los ejemplos ofrecidos por el manual de referencia que se tiene con el lenguaje no son lo suficientemente robustos para apoyar la construcción de modelos complejos. Por estas razones, la verificación del modelo computarizado tiene una incidencia directa en esta investigación.

Las pruebas y herramientas utilizadas para verificar el modelo son: la prueba de animación, la prueba de valor fijo, la prueba de condición extrema, y los gráficos operacionales.

2.1.7.1. PRUEBA DE ANIMACIÓN.

Para asegurar la correcta emulación de los procesos, las capacidades gráficas del lenguaje son apropiadas para lograr una mejor visualización de los elementos en el modelo computarizado, debido a que todo se construye en modo 3D. La validación de los espacios y de los elementos se realiza fácilmente al ser construido en modo 3D. La visualización del modelo en acción fue presentada al equipo gerencial de la compañía para definir y evaluar claramente los procesos establecidos en el modelo. La Figura 2.33 despliega una parte del modelo en animación.

Figura 2.33 Prueba de animación.

2.1.7.2. PRUEBA DE VALOR FIJO.

El modelo de simulación se somete a condiciones diferentes para determinar si la programación y construcción del modelo computarizado y su implementación son correctas (Whitner & Balci, 1989). Se define como condiciones diferentes a cada uno de los diferentes tipos de puertas definidos en el modelo. Se establece verificar la tasa de producción por tipo de puerta obtenida como resultado del modelo. Estos resultados fueron verificados contra los valores calculados previamente por métodos analíticos. La Figura 2.34 despliega la descripción previa en tres departamentos diferentes de producción.

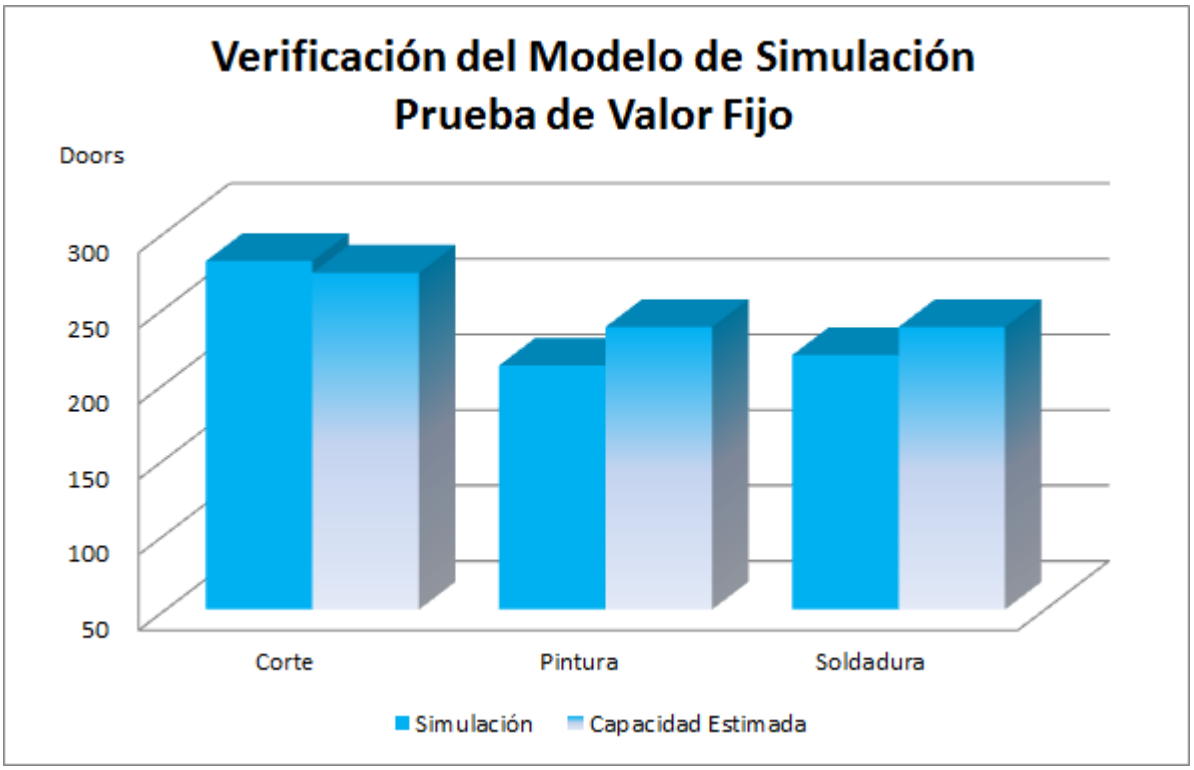


Figura 2.34 Prueba de valor fijo.

2.1.7.3. PRUEBA DE CONDICIÓN EXTREMA.

Para esta prueba el modelo de simulación se somete a producir únicamente un determinado producto. En este caso 'Marcos', omitiendo la producción de 'Puertas'. Se analiza el tiempo de ocupación obtenido para algunos operadores del área de empaque de marcos y el tiempo de ocupación de operadores que ensamblan puertas. La Figura 2.35 despliega los resultados obtenidos utilizando esta prueba.

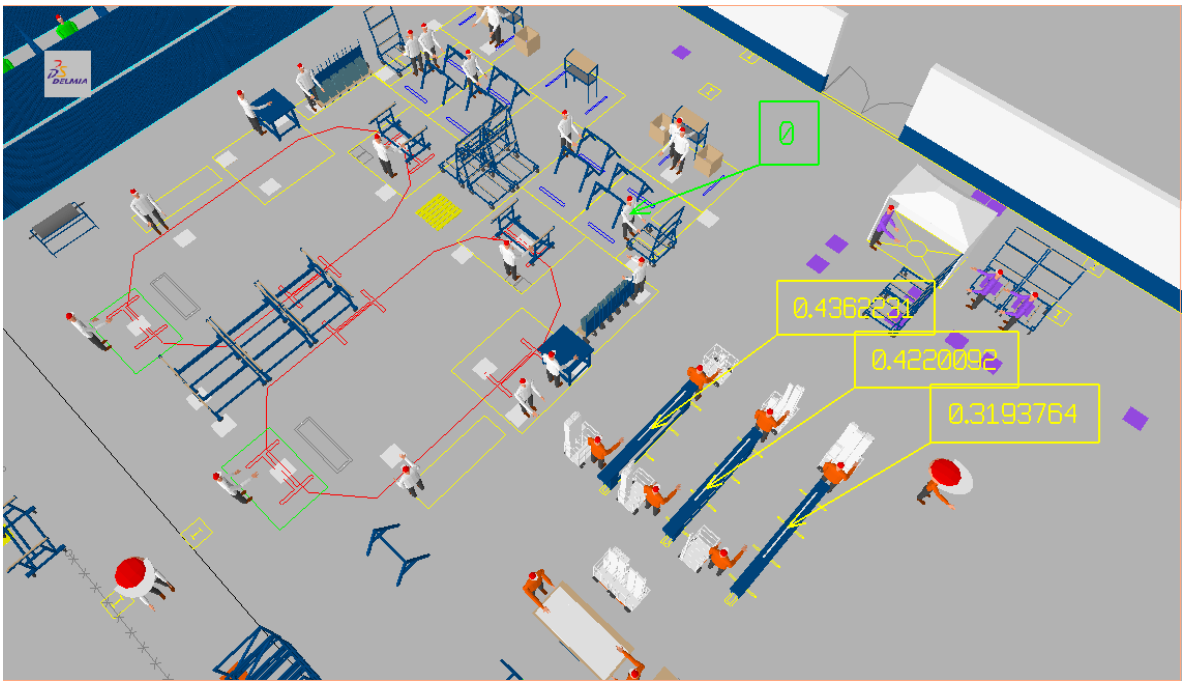


Figura 2.35 Prueba de condición extrema.

2.1.7.4. GRÁFICOS OPERACIONALES.

Los gráficos operacionales son considerados una prueba de verificación. En este caso, se evalúa el tiempo de operación efectivo de un operador realizando sus tareas en un determinado turno de trabajo. La Figura 2.36 despliega un gráfico construido en el lenguaje Quest®.

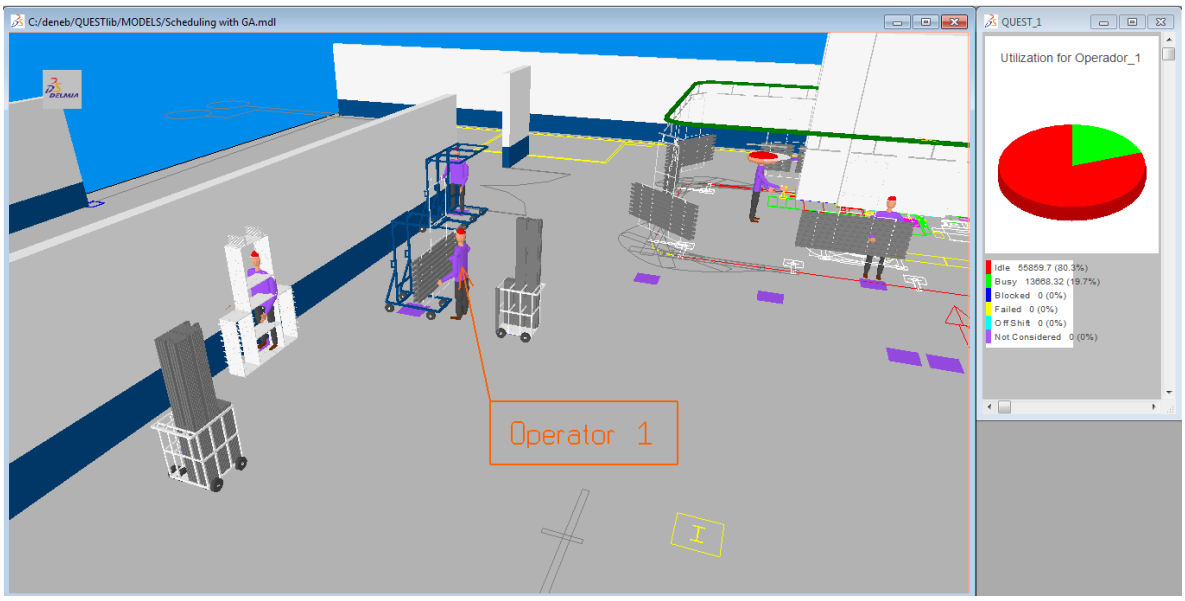


Figura 2.36 Prueba de gráficos operacionales.

2.1.7.5. VALIDACIÓN OPERACIONAL DEL MODELO.

La validación del modelo es utilizada para determinar si las salidas que arroja el modelo tienen la suficiente precisión que se requiere sobre la aplicación para la cual fue construido (Sargent, 2007). La validación en este caso, se realizó estadísticamente, es decir, se comparan los resultados obtenidos por el modelo de simulación con los que se tienen en el proceso de manufactura sujetos a las mismas condiciones iniciales de operación. Esto es posible al utilizar la información relativa al trimestre de mayor producción en el año. Estadísticamente, no hay diferencia significativa a un nivel de significancia de 0.05, entre la diferencia de medias de la producción y la obtenida por el modelo de simulación. Los intervalos obtenidos con la prueba t-Student son: [1252, 2236] para la producción y [1169, 2126] para el modelo de simulación. La diferencia entre ambos intervalos va de -553 a 746, que contiene el valor de cero. Además, el valor-P obtenido por la prueba de hipótesis es menor que el nivel de significancia establecido para la prueba de 0.05, por lo tanto, la hipótesis nula de igualdad de medias o bien que la diferencia tiende a cero no puede ser rechazada.
Estos resultados asumen que las varianzas de ambas muestras son iguales. Se realiza una prueba F para determinar la igualdad de varianzas. El intervalo obtenido es de [0.32, 3.467] el cual contiene al uno, por lo que no hay diferencia significativa entre varianzas de las muestras.
Estos resultados asumen que los datos obtenidos por las muestras provienen de poblaciones con distribución normal o que pueden ser originados a partir de esta. Se calcula el sesgo y se obtiene -0.67 para la producción y -0.72 para el modelo de simulación. Ambos están dentro del intervalo de [-2, 2] porque no se puede rechazar la hipótesis de que los datos provienen de una distribución normal. La Figura 2.37 describe los resultados mencionados en un diagrama de caja y bigotes.

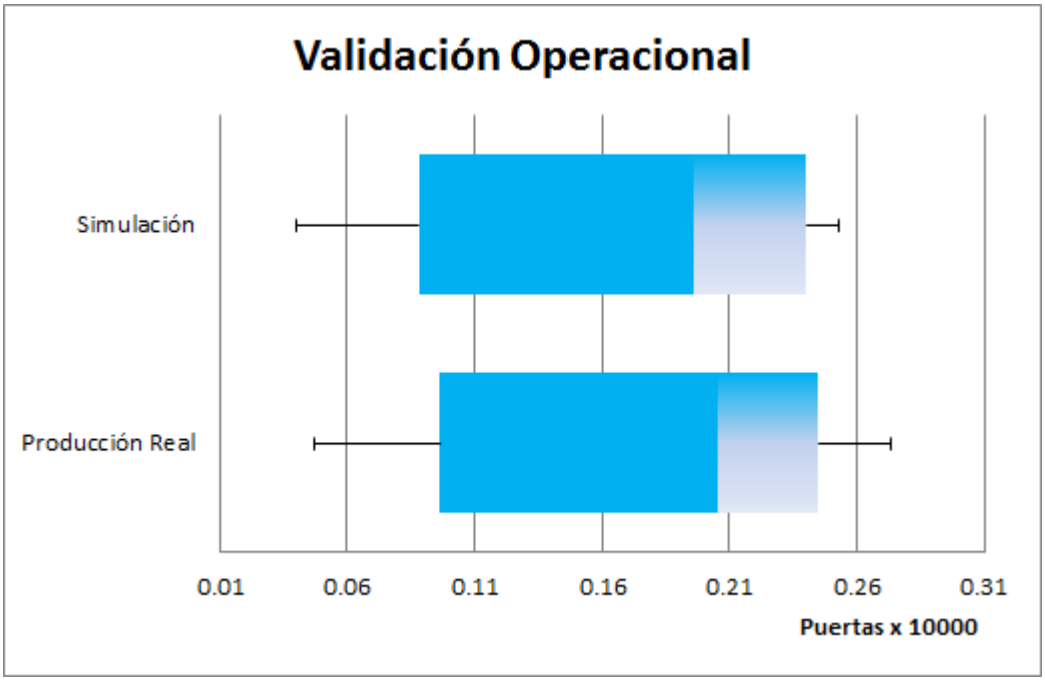


Figura 2.37 Validación operacional del modelo.

2.1.7.6. VALIDACIÓN DEL MODELO.

El método de validación desarrollado para este estudio de caso clasifica las variables de entrada en función de cómo afectan las variables de salida. Las variables de entrada seleccionadas son la velocidad del transportador en la pintura, el tiempo de proceso del sistema de sellado en el montaje, el tiempo de proceso del sistema de secado en el lavado, las secuencias y los turnos de trabajo. Las variables de salida están en la media de utilización de todos los empleados, el tiempo promedio de proceso efectivo de todos los empleados, el trabajo en proceso y el rendimiento. La Tabla 2.1 muestra los principales resultados.

Tabla 2.1 Método de Validación.

Variables de Salida	Variables de Entrada		
	Sistema real 0.036 m/s	Velocidad de conveyor 0.03 m/s	Velocidad de conveyor 0.05 m/s
Utilización promedio (%)	17.39	17.97	18.25
Tiempo promedio de proceso (segundos)	40922	39975	40964
Producción en proceso (puertas)	84	91	68
Tasa de producción (puertas / turno)	916	909	932
	Sistema real 30 min	Sellado de silicón 25 min	Sellado de silicón 35 min
Utilización promedio (%)	17.39	18.07	17.91
Tiempo promedio de proceso (segundos)	40922	***35749***	38815
Producción en proceso (puertas)	84	113	118
Tasa de producción (puertas / turno)	916	886	881
	Sistema real 30 min	Secado en el Lavado 25 min	Secado en el Lavado 35 min
Utilización promedio (%)	17.39	18.22	18.15
Tiempo promedio de proceso (segundos)	40922	40938	40871
Producción en proceso (puertas)	84	68	62
Tasa de producción (puertas / turno)	916	931	930
	Sistema real (Regla PEPS)	Secuencias de Trabajo (por complejidad)	Secuencias de Trabajo (por simplicidad)
Utilización promedio (%)	17.39	***15.23***	***19.63***
Tiempo promedio de proceso (segundos)	40922	***35516***	40724
Producción en proceso (puertas)	84	***268***	82
Tasa de producción (puertas / turno)	916	***731***	917
	Sistema real (sin tiempo libre)	Turnos (8 horas libres)	Turnos (4 horas libres)
Utilización promedio (%)	17.39	***16.97***	***20.30***
Tiempo promedio de proceso (segundos)	40922	***33515***	41127
Producción en proceso (puertas)	84	***270***	***68***
Tasa de producción (puertas / turno)	916	***730***	***931***

De los resultados mostrados anteriormente, las secuencias de trabajo y las horas fuera de servicio (horas antes de iniciar el turno de trabajo) son una oportunidad clave para mejorar el proceso de fabricación. Esto es principalmente porque la cantidad de órdenes de producción no es constante o suficiente para mantener el proceso de fabricación ocupado por un largo tiempo, causando tiempos de inactividad intermitentes. Esta característica es sensible en el proceso de fabricación real. Un buen equilibrio de horas fuera de servicio ayuda a mejorar los valores de las variables de salida y se muestran en el método de validación descrito anteriormente e incluye el WIP. Un enfoque combinado utilizando secuencias de trabajo y horas fuera de servicio fue el propósito de esta investigación.

2.2. ALGORITMO GENÉTICO.

2.2.1. REPRESENTACIÓN DE INDIVIDUOS.

En la revisión de la literatura se pudo notar que los problemas de programación y secuenciamiento de operaciones son NP-hard. Por lo que no hay algoritmos conocidos hasta el momento que garanticen ofrecer una solución óptima en un tiempo polinomial.

Los Algoritmos Genéticos son un enfoque alternativo para resolver problemas complejos de programación y secuenciamiento de operaciones ya que son técnicas que emulan la evolución natural. Específicamente los Algoritmos Genéticos modelan el proceso de evolución de las especies como una secuencia de cambios frecuentes en una población de individuos. El espacio de solución es explorado aplicando una serie de transformaciones a las soluciones que son candidatas. La Figura 2.38 describe la forma de operar de un Algoritmo Genético básico.

```
Comenzar
t=0
Inicializar P(t)
Evaluar P(t)
Mientras el criterio de paro no se alcance
        t = t + 1
        Seleccionar P´(t) de P(t-1)
        Obtener P´´(t) a partir de la cruza de P´(t)
        Obtener P´´´(t) a partir de la mutación de P´´(t)
        Reemplazar P(t) por (P(t-1), P´´´(t))
        Evaluar P(t)
Fin mientras
Fin
```

Figura 2.38 Algoritmo Genético.

Una vez que ha sido construido el modelo de simulación, un simple pero efectivo Algoritmo Genético se propone para lograr una comunicación con el modelo de simulación y así obtener las más adecuadas secuencias de producción y turnos de trabajo que incidan en la optimización de la medida de desempeño en el sistema de manufactura modelado.
El propósito es crear un proceso capaz de producir individuos prometedores, donde cada secuencia de trabajo y turno definido es un individuo.
Para codificar a cada individuo dentro del Algoritmo Genético, se utiliza tres representaciones de cadena, por un lado una cadena contiene información codificada relativa a la programación de las operaciones, otra a la asignación de máquinas y otra a las horas fuera de servicio antes de iniciar el turno de trabajo.
Para la cadena de secuencia de operaciones, el número de elementos es igual al número total de operaciones, donde cada elemento contiene un valor entero que representa el número de trabajo a programar y este número se repite tantas operaciones haya que realizar para que se considere un trabajo terminado. La Figura 2.39 describe un ejemplo de una secuencia a manufacturar.

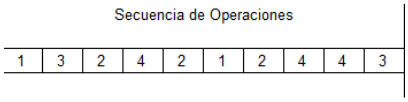


Figura 2.39 Representación de un individuo (Secuencia de Producción).

Para la cadena de asignación de máquinas, cada elemento representa la máquina correspondiente seleccionada para cada operación. Se omite la figura por simplicidad.

Para la última cadena, esta contiene una longitud de ocho elementos que representan los departamentos de producción principales dentro del sistema de manufactura y así mismo en el modelo de simulación. Cada elemento de la cadena muestra las horas fuera de servicio que estarán los operadores y los equipos de dicho departamento de producción antes de iniciar sus actividades, ejemplo: si el elemento número uno de la cadena contiene el número cuatro significa que durante las primeras cuatros no habrá servicio en el área uno, por lo que las operaciones se iniciarán cuatros horas más tarde. La figura 2.40 describe un ejemplo de esta representación.

Departamento							
Rolado	Lavado	Corte	Soldadura	Pintura	Corte de Vidrio	Ensamble	Empaque
3	1	0	1	2	6	4	5
Horas libres antes de iniciar producción							

Decodificación

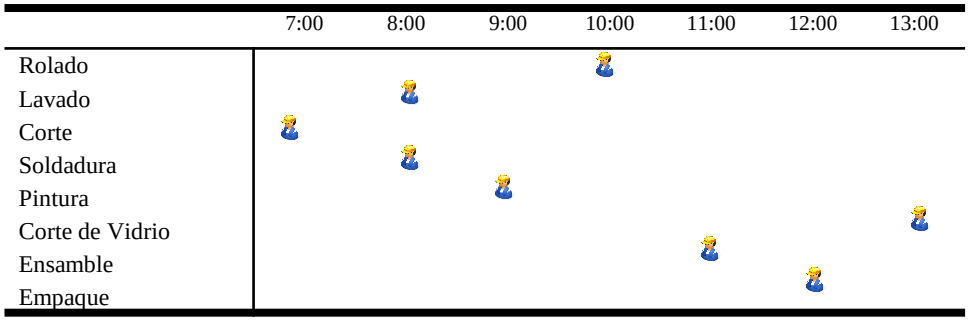


Figura 2.40 Representación de un individuo (Calendario Laboral).

De acuerdo con Greenwood *et al* (2005) los miembros de la población inicial son generados aleatoriamente para obtener un amplio rango de soluciones. Esto es posible debido a que en primera instancia se tiene la cantidad de trabajos a procesar en un determinado horizonte de tiempo. Con estos trabajos se genera un determinado número de secuencias, definiendo el límite de estas por el tamaño de la población. Además, los números que representan las horas fuera de servicio antes de comenzar la fabricación para cada departamento de producción son también generados aleatoriamente.

Una vez que la primera generación ha sido construida, esta debe ser decodificada para producir por una parte materiales o partes dentro del modelo de simulación, las cuales ya tendrán una determinada ruta de manufactura en función al trabajo que se procesa y las operaciones que este requiere.
Además, se establecen las instrucciones para crear un calendario laboral diario y semanal y así asociarlo con los elementos involucrados que pertenecen al modelo de simulación.

Después de esto, los miembros de la población son evaluados por el modelo de simulación utilizando la variable de respuesta definida.

2.2.2. COMUNICACIÓN MODELO – ALGORITMO.

El Algoritmo Genético es programado en DevC++® y el modelo de simulación en Delmia Quest® R20. Al ser plataformas diferentes, un mecanismo de comunicación debe ser implementado que permita evaluar a cada individuo de manera sistematizada por el simulador y retornar la ejecución al Algoritmo Genético cada vez que finaliza dicha evaluación. Esto es posible a través de un archivo por lotes, este es llamado desde el programa ejecutable del Algoritmo Genético cediendo así la ejecución a una ventana de comandos la cual invoca al archivo ejecutable que inicializa a Quest® en modo de apertura por lotes y así Quest® inmediatamente invoca un archivo especifico que pertenece al lenguaje de control de procesos por lotes BCL (por sus siglas en inglés Batch Control Language). Este último contiene una serie de comandos que apertura e inicializa el modelo de simulación a utilizar, se definen los parámetros de entrada y condiciones iniciales establecidos por el Algoritmo Genético y se realizan la corrida de simulación requerida. Una vez hecho esto se recopila las estadísticas relativas a la medida de desempeño utilizada. La Figura 2.41 describe los comandos utilizados para realizar este proceso de comunicación.

```
Comenzar
t=0
Aperturar archivo con carga de trabajo
Crear turnos de trabajo y secuencias de producción
Inicializar P(t)
Llamar a Función de Evaluación
Mientras el criterio de paro no se alcance
        t = t + 1
        Seleccionar P´(t) de P(t-1)
        Obtener P´´(t) a partir de la cruza de P´(t)
        Obtener P´´´(t) a partir de la mutación de P´´(t)
        Reemplazar P(t) por (P(t-1), P´´´(t))
        Llamar a función de Evaluación P(t)
        Encontrar mejor turno de trabajo y secuencia de producción
Fin mientras
Fin

Función de Evaluación
Comenzar
Para cada Individuo
        Calcular materiales requeridos
        Establecer turnos de trabajo en el modelo
        BCL Aperturar el modelo
        BCL Correr el modelo
        Almacenar estadísticas
        Asignar aptitud al individuo
        BCL Resetear
        BCL Salir
Fin para cada individuo
Fin
```

Figura 2.41 Comandos para comunicar Modelo – Algoritmo.

Se puede notar a través de los comandos descritos en la Figura 2.41 que el simulador al concluir la ejecución de la corrida de simulación mencionada, este cede la ejecución a la ventana de comandos que lo invoco. La ventana de comandos retorna la ejecución del programa ejecutable del Algoritmo Genético. La Figura 2.42 ilustra la importancia de la comunicación para lograr la evaluación de los individuos en cada generación.

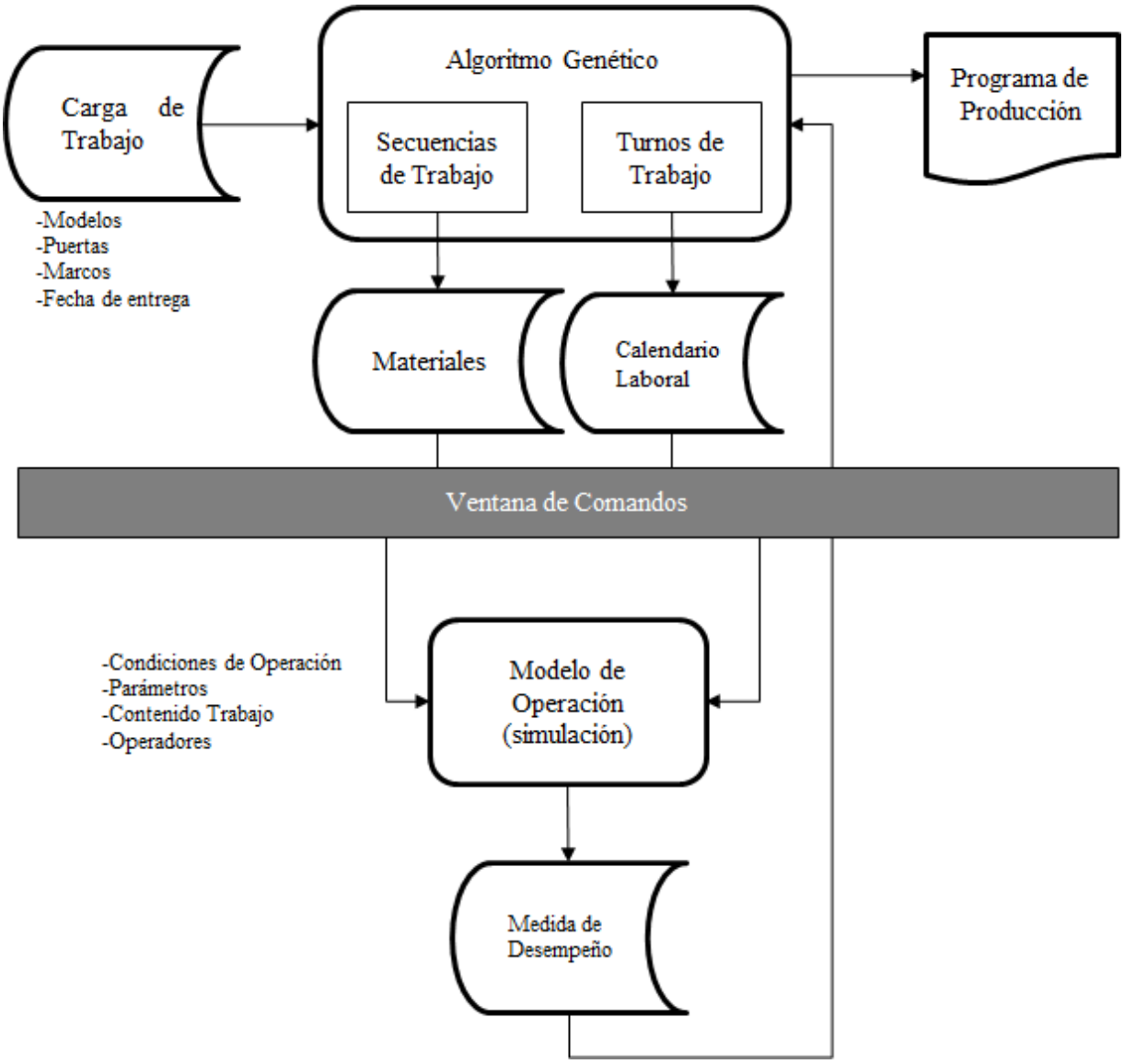


Figura 2.42 Interfaz de comunicación.

2.2.3. SELECCIÓN.

El objetivo es encontrar las mejores secuencias de producción y el mejor calendario laboral para el sistema de manufactura mencionado. Las "mejores" secuencias y el "mejor" calendario laboral son determinados a partir de la variable de respuesta definida para este caso por la producción en proceso WIP (por sus siglas en inglés Work In Process). Obtener el menor WIP posible dado las restricciones

físicas, de operación y de programación que se tengan es el fin. Todos los individuos tendrán asociados un valor de WIP que determina su aptitud, es decir, que tan bueno o que tan malo es el individuo en términos de lo que se busca.
En este Algoritmo Genético se utiliza la selección por torneo. Esto significa que dos individuos elegidos aleatoriamente del conjunto poblacional inicial son comparados en su aptitud para determinar quién será candidato a la cruza.
Se escoge la selección por torneo de solamente dos individuos ya que esto evita una convergencia prematura del algoritmo o bien el incurrir en soluciones que no son las mejores que se buscan. Aumentar el tamaño del torneo tan solo incrementa la presión de la selección (Miller & Goldberg, 1995).

2.2.4. CRUZA.

El operador de cruza utilizado es "combinación de extremos", el cual es en sí mismo un algoritmo. En base a Whitley *et al* (1990) este es capaz de proveer soluciones de alta calidad para problemas se secuenciamiento u ordenación. Este operador utiliza "información de los extremos" de la cadenas que representan a los padres y así se hereda la mayor información posible de las estructuras de estos a los hijos. Por ello es que se necesitan a dos padres elegidos previamente en la selección para crear una descendencia a partir de la recombinación de los extremos. Esto asegura la generación de descendientes en la secuencia. En resumen, se construye una lista para cada trabajo con sus correspondientes operaciones a secuenciar y esta contendrá los trabajos que se encuentran como vecinos de este en ambos padres. El primer trabajo/operación a secuenciar se elige aleatoriamente. Se elimina la operación elegida de las listas mencionadas anteriormente. Después, se elige de la lista del trabajo secuenciado aquel trabajo que en su lista contiene menos trabajos adyacentes. Se repite el proceso hasta secuenciar todos los trabajos. La Figura 2.43 presenta un ejemplo del operador de cruza utilizado.

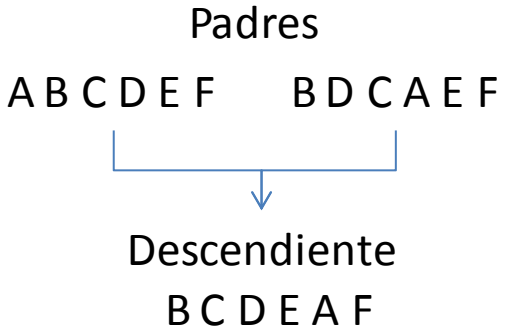


Figura 2.43 Ejemplo de Operador de Cruza.

2.2.5. MUTACIÓN.

El operador de mutación utilizado es "por intercambio reciproco". La idea central es seleccionar dos trabajos aleatoriamente del individuo elegido para mutar e intercambiar sus posiciones. La Figura 2.44 ejemplifica esta descripción.

Figura 2.44 Ejemplo de Operador de Mutación.

2.2.6. REEMPLAZO GENERACIONAL.

Se elige realizar una competencia entre padres y descendientes producidos por los operadores anteriores mencionados a fin de establecer quienes estarán presentes en la próxima generación. Schwefel (1997) ha recomendado la relación de 1 a 5 ó 1 a 6 para realizar el reemplazo generacional. Esto significa que aproximadamente el 20% de los individuos deben pertenecer a la generación de los padres y 80% a los descendientes. Hay un óptimo en la relación padres/hijos que obtiene el compromiso más razonable entre el esfuerzo computacional que se invirtió y el progreso que se obtiene en la búsqueda de la solución (Bäck, 1996).

2.2.7. AJUSTE DE PARÁMETROS.

Se utilizó un Diseño de Experimentos de Diseño Central Compuesto, con el fin de encontrar el mejor valor de cada uno de los parámetros del Algoritmo Genético. Se trata entonces de determinar los mejores valores para la tasa de mutación, cruza, reemplazo generacional y tamaño de la población. Para ello, se evaluaron dos variables de respuesta en busca de la mejor solución en base a la velocidad de búsqueda y a la calidad de la solución. Se ha utilizado las ideas de Bäck (1996) donde la velocidad se expresa por el número de generaciones necesarias para lograr la convergencia y para la calidad de la solución se analiza el promedio de la aptitud obtenida por el algoritmo en cada generación. Se busca entonces el menor número posible de generaciones para la primera variable de respuesta y el promedio más bajo posible de la aptitud para la segunda variable de respuesta. La Tabla 2.1 detalla los niveles utilizados para los factores (parámetros del algoritmo) en el Diseño de Experimentos.

Tabla 2.2 Niveles utilizados para encontrar los mejores valores de los parámetros.

Parametros	Niveles		
	-1	0	1
Tasa de cruza	64%	75%	88%
Tasa de mutación	5%	10%	15%
Relación Padres/Hijos	10%	15%	20%
Tamaño de Población	50	75	100

2.3. ALGORITMO DE ESTIMACIÓN DE DISTRIBUCIONES.

Parte de las ideas construidas en la sección anterior son utilizadas en esta sección, es decir, es posible usar parte del Algoritmo Genético para construir un Algoritmo de Estimación de Distribuciones. El enfoque ahora está en generar una población inicial de manera aleatoria como en el Algoritmo Genético. Esta población inicial es evaluada de la misma forma como se hace con los individuos del Algoritmo Genético, es decir, a través del modelo de simulación. Además, se adopta el mismo procedimiento para seleccionar a los individuos, es decir, selección por torneo descrito en la sección anterior.

La construcción de tres modelos gráficos probabilísticos hace la principal diferencia entre ambos algoritmos. Cada modelo gráfico probabilístico construido trabaja para un diferente objetivo. Esta es una importante diferencia entre cualquier Algoritmo de Estimación de Distribuciones propuesto previamente.

2.3.1. MODELADO DE LA DEPENDENCIA EN SECUENCIAS DE PRODUCCIÓN.

2.3.1.1. REPRESENTACIÓN DE LA SOLUCIÓN.

Cualquier solución del proceso de manufactura mencionado debe ser una combinación de la decisión de programación de las operaciones, asignación de máquinas y horas fuera de servicio antes de iniciar el turno de trabajo. Por lo tanto, una solución puede ser expresada por la secuencia de procesamiento de las operaciones sobre las máquinas, la asignación de las operaciones sobre las máquinas y las horas fuera de servicio que estarán los operadores antes de iniciar sus actividades. En esta investigación, una solución está representada por tres vectores (vector de secuencia de operaciones, vector de asignación de máquinas y vector de horas fuera de servicio).
Para el vector de secuencia de operaciones, el número de elementos es igual al número total de operaciones, donde cada elemento contiene un valor aleatorio con distribución U[0,1], una sensible diferencia entre nuestro enfoque y el trabajo de Wang *et al* (2012). Para el vector de asignación de máquinas, cada elemento representa la máquina correspondiente seleccionada para cada operación. Para el vector de horas fuera de servicio, cada elemento muestra las horas fuera de servicio que estarán los operadores antes de iniciar sus actividades en los más importantes departamentos de manufactura del taller. Para explicar la representación, se presenta un ejemplo al considerar un problema con 4 trabajos, 4 máquinas, y diferentes horas fuera de servicio posibles como se muestra en la Figura 2.45. En la Figura 2.46, se ilustra la representación de un individuo.

Operación	Factibilidad			
(trabajo, precedencia)	M_1	M_2	M_3	M_4
$O_{1,1}$	o	o	o	o
$O_{1,2}$	o	o	x	o
$O_{2,1}$	o	x	x	x
$O_{2,2}$	o	x	o	o
$O_{2,3}$	x	o	o	o
$O_{3,1}$	o	x	x	x
$O_{3,2}$	x	o	o	o
$O_{4,1}$	o	x	x	x
$O_{4,2}$	x	o	o	x
$O_{4,3}$	o	o	o	o
Horas fuera de servicio	0	2	4	2

Figura 2.45 Factibilidad Operacional.

Secuencia de Operaciones - Valores Reales U[0,1]										Asignación de Máquinas								Horas s/servicio			
0.86	0.92	0.25	0.73	0.95	0.58	0.52	0.22	0.58	0.55	2	1	2	3	4	4	3	2	3	4	1	2	0	2
										Job 1		Job 2			Job 3		Job 4			Máquina			
																				1	2	3	4

Figura 2.46 Representación de un individuo.

Para mostrar una comparativa entre este enfoque y el trabajo de Wang *et al* (2012) se proporciona un ejemplo. La Figura 2.47 detalla el tiempo de procesamiento fijo para cada trabajo en cada máquina y una secuencia factible. La Figura 2.48 ilustra la gráfica de Gantt de esta solución trabajando al mismo tiempo todas las máquinas. En la Figura 2.49 se muestra otra secuencia factible pero esta incluye horas fuera de servicio y su impacto sobre el WIP y el tiempo muerto se puede observar en la Figura 2.50. Sin embargo, en el ambiente industrial mencionado el tiempo de procesamiento no es fijo, así que la representación de Wang *et al* (2012) no debe ser aplicado directamente. Los resultados podrían ser diferentes totalmente. Por ello se utilizan los tiempos más reales de las operaciones de los trabajos a secuenciar. Esto hace sensible el resultado que se obtiene en la variable de respuesta, el WIP.

Operación	Tiempos de Procesamiento Fijos (hrs)			
(trabajo, precedencia)	M_1	M_2	M_3	M_4
$O_{1,1}$	2	4	3	5
$O_{1,2}$	4	4	-	3
$O_{2,1}$	3	-	-	-
$O_{2,2}$	3	-	3	3
$O_{2,3}$	-	5	6	2
$O_{3,1}$	2	-	-	-
$O_{3,2}$	-	2	3	3
$O_{4,1}$	1	-	-	-
$O_{4,2}$	-	3	3	-
$O_{4,3}$	6	4	5	5
Horas fuera de servicio	0	1	3	3

Secuencia de Operaciones										Asignación de Máquinas								Horas s/servicio			
1	3	2	4	2	1	2	4	4	3	1	1	1	3	3	1	4	1	4	4	0	0	0	0
										Job 1		Job 2			Job 3		Job 4			Máquina			
																				1	2	3	4

Figura 2.47 Datos del tiempo de procesamiento y una secuencia factible.

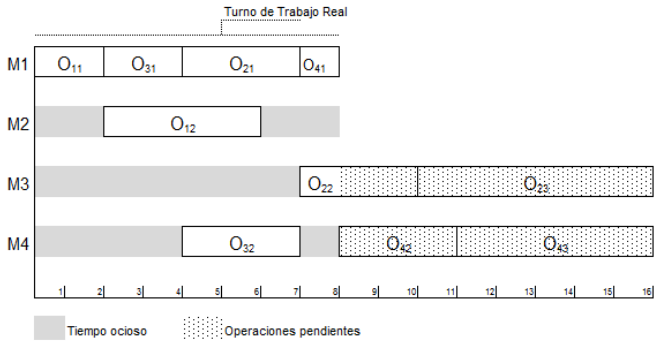


Figura 2.48 Gráfica de Gantt de la solución factible de la Figura 2.47.

Secuencia de Operaciones										Asignación de Máquinas								Horas s/servicio			
4	1	3	2	4	1	3	2	4	2	1	2	1	3	4	1	3	1	2	4	0	1	3	3
										Job 1		Job 2			Job 3		Job 4			Máquina			
																				1	2	3	4

Figura 2.49 Solución factible con horas fuera de servicio.

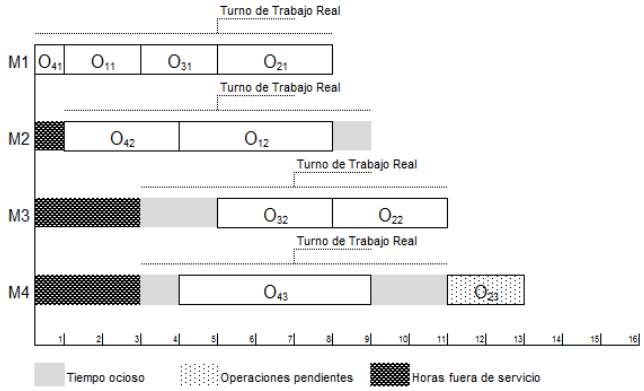


Figura 2.50 Gráfica de Gantt de la solución factible de la Figura 2.49.

La codificación utilizada es una clásica de secuencias de trabajo y es una representación de los individuos seleccionados útil, ya que cada individuo es factible y satisface todas las restricciones al problema determinado, tal que pueda ser considerado como un punto de solución en el espacio de búsqueda. No obstante, al realizar el muestreo por cualquier Algoritmo de Estimación de Distribuciones de tipo discreto, se tiene el inconveniente de que se pueden generar secuencias incorrectas con uno o más trabajos omitidos o bien operaciones duplicadas en la secuencia. Este hecho se debe al estimar la distribución de probabilidad a partir de un conjunto de secuencias. Esto no es posible cuando se utilizan valores discretos en el proceso. Los Algoritmos de Estimación de Distribuciones pueden aprender a generar una distribución de probabilidad sobre un conjunto $\Omega = \Omega_1 \text{ x } \Omega_2 \text{ x } \dots \text{ x } \Omega_n$, donde $\Omega_i = \{ 1, 2, \dots r_i \}$ y $r_i \in N$, pero el muestreo no puede proveer secuencias individuales a partir del conjunto Ω (Larrañaga & Lozano, 2002). Para resolver este inconveniente, se han construido algunos algoritmos, como el PLS (por sus siglas en inglés Probabilistic Logic Sampling) y el ATM (por sus siglas en inglés All Time Modification) para obtener individuos basados en secuencias utilizando valores discretos. Entonces unas modificaciones en el proceso de muestreo deben realizarse de manera que durante el muestreo de cada individuo, las probabilidades aprendidas por la Red Bayesiana sean modificadas. En estos algoritmos, el individuo es generado elemento por elemento de la cadena, es decir, variable por variable siguiendo el orden ancestral entre ellas y de acuerdo a la Red Bayesiana utilizada. Además, las probabilidades obtenidas al principio son *modificadas* de ser necesario para asegurar individuos factibles. Este hecho de alterar las probabilidades en el proceso de muestreo, cualquiera que sea la forma de hacerlo, implica que el resultado del algoritmo es también modificado de la misma forma.

Para evitar los inconvenientes descritos previamente, se adopta un procedimiento de optimización continua en vez de uno discreto para resolver problemas combinatorios como el de secuenciamiento. Los trabajos de Rudolph (1991) y Bean & Norman (1993) caen en esta categoría y pueden ser consultados.
En un Algoritmo de Estimación de Distribuciones de tipo continuo la estimación de la distribución de probabilidad es obtenida al representar a los individuos por vectores de números reales. El aprendizaje

se logra por un modelo gráfico probabilístico conocido como Red Gaussiana que realiza los cálculos para valores continuos, es decir, con números reales. Cada individuo es obtenido muestreando a partir de una *n*-dimensional distribución Gaussiana, y por lo tanto puede tomar cualquier valor de R^n. Con esta nueva representación, los individuos no tienen un significado directo de la solución que representan, los valores de cada una de las variables no contienen valores parecidos entre nodos de cualquier gráfico. Este tipo de representación es considerada como una manera de cambiar la búsqueda del mundo discreto al mundo continuo, donde las técnicas que se aplican para la estimación de densidades de probabilidad son completamente diferentes. A continuación se describe este hecho.

Un caso particular de modelos gráficos probabilísticos se da cuando cada variable $X_i \in \boldsymbol{X}$ es continua y la función de densidad puede obtenerse por el modelo de regresión lineal:

$$f(x_i \mid pa_i^s, \theta_i) \equiv N(x_i;\ m_i + \sum_{x_j \in pa_i} b_{ji}\left(x_j - m_j\right), v_i) \qquad \text{(Ec. 2.1)}$$

donde $N(x; \mu, \sigma^2)$ es una distribución normal univariante con media μ y varianza σ^2. Los parámetros están dados por $\theta_i = (m_i, b_i, v_i)$, donde $b_i = (b_{1i}, b_{1i}, \dots, b_{i-1i})^t$ es un vector columna. Se le llama al modelo grafico probabilístico construido con estas funciones de densidad para cada variable de una Red Gaussiana (Shachter & Kenley, 1989).
La interpretación de los componentes de los parámetros es el siguiente: m_i es la media de X_i no condicionada, v_i es la varianza condicional de X_i dado Pa_i, y b_{ji} es un coeficiente lineal que refleja la fortaleza de la relación entre X_j y X_i. Con esta forma, un arco no existente de X_j a X_i implica que $b_{ji} = 0$ en el modelo de regresión lineal.

Para observar la relación entre la Red Gaussiana y las funciones de densidad normal multivariante, se considera que la función de densidad de probabilidad conjunta de la variable continua *n*-dimensional $\boldsymbol{X}$ es una distribución normal multivariante si y solo si:

$$f\ (x) \equiv N(x;\ \mu, \sum)\ \equiv (2\pi)^{-\frac{n}{2}}\ |\sum|^{-\frac{1}{2}}\, e^{-\frac{(x-\mu)^t \sum^{-1}(x-\mu)}{2}} \qquad \text{(Ec. 2.2)}$$

donde μ es el vector de medias, $\sum$ es una matriz de covarianzas *n* x *n* y $|\sum|$ el determinante de $\sum$. La inversa de la matriz de covarianzas $W = \sum^{-1}$, cuyos elementos son denotados por w_{ij}, se conoce como la matriz de precisión.
La función de densidad anterior puede escribirse como el producto de *n* densidades condicionales a saber:

$$f\ (x) = \prod_{i=1}^{n} f(x_i | x_1, \dots, x_{i-1}) = \prod_{i=1}^{n} N(x_i;\ \mu_i + \sum_{j=1}^{i-1} b_{ji}(x_j - \mu_j), v_i) \qquad \text{(Ec. 2.3)}$$

donde μ_i es la media no condicionada de X_i, v_i es la varianza de X_i, dado X_1, X_2, ... , X_{i-1}, y b_{ij} es un coeficiente lineal que refleja la fortaleza de la relación entre variables X_j y X_i (DeGroot, 1970). Esta notación ofrece la posibilidad de interpretar una distribución normal multivariante como una Red Gaussiana donde existe un arco de X_j a X_i, cuando $b_{ij} \neq 0$ con $j < i$.

2.3.1.2. GENERACIÓN DE LA POBLACIÓN INICIAL.

Los miembros de la población inicial son generados aleatoriamente para con el fin de permitir una amplia gama de soluciones (Greenwood *et al* 2005).

2.3.1.3. MODELO DE PROBABILIDAD.

En esta investigación se utiliza el algoritmo MIMIC_C^G para construir el primer modelo gráfico probabilístico. Este fue introducido por Larrañaga *et al* (2000), el cual es una adaptación del algoritmo MIMIC presentado por De Bonet *et al* (1997) a dominios de tipo continuo.

En el algoritmo MIMIC_C^G el modelo de probabilidad subyacente para cada par de variables se asume que es una variable Gaussiana bivariada. Se establece 'Ω' como el conjunto de posiciones definidas en cada secuencia que representan las variables Gaussianas mencionadas. La idea, al igual que en el algoritmo MIMIC para optimización combinatoria, es estimar la verdadera función de densidad conjunta, utilizando una función marginal univariada y *n-1* pares de funciones de densidad condicional. Para lograr esto, el resultado siguiente es utilizado:

Teorema 1 (Whittaker, 1990) pp 167. Sea **X** una función de densidad normal *n*-dimensional, donde **X** $\equiv N(x;\ \mu, \sum)$, entonces la entropía de **X** es:

$$h(X) = \frac{n(1+\log 2\pi)}{2} + \frac{\log|\sum|}{2} \qquad \text{(Ec. 2.4)}$$

Aplicando este resultado a funciones de densidad normal univariante y bivariante para definir el algoritmo MIMIC_C^G, se obtiene:

$$h(X) = \frac{(1+\log 2\pi)}{2} + \log \sigma x \qquad \text{(Ec. 2.5)}$$

$$h(X|Y) = \frac{1}{2}\left[(1 + \log 2\pi) + \log(\frac{\sigma_X^2\sigma_Y^2 - \sigma_{XY}^2}{\sigma_Y^2})\right] \qquad \text{(Ec. 2.6)}$$

donde $\sigma_X^2\sigma_Y^2$ denota la varianza de la variable univariante X(Y) y σ_{XY}^2 denota la covarianza entre las variables X y Y.

La estructura de aprendizaje del algoritmo MIMIC_C^G asigna en un primer paso como variable univariante a la variable con la varianza más pequeña de la muestra que llamaremos Y. El segundo paso, si la variable X, cuya estimación de $\frac{\sigma_X^2\sigma_Y^2 - \sigma_{XY}^2}{\sigma_Y^2}$ con respecto a la variable anterior elegida, Y, es la más pequeña, es escogida y ligada a Y.

Una vez que se han generado individuos a partir del algoritmo MIMIC_C^G, estos deben ser decodificados para que puedan ser representados como secuencias validas, es decir, como secuencias de producción. Para ello se ordenan los valores continuos del individuo y se establecen sus valores discretos correspondientes asignando a cada $x_i \in \{1, 2, \ldots, n\}$ donde *n* es el tamaño del vector, el orden correspondiente. La Figura 2.51 detalla el pseudocódigo a utilizar. La Figura 2.52 describe un ejemplo de un vector de números reales y su decodificación.

n = tamaño del individuo, es decir, el número de nodos en la red Gaussiana (la secuencia)
$x^C = (x_1^C, x_2^C, \ldots, x_n^C)$: individuos que contienen valores continuos
$x^D = (x_1^D, x_2^D, \ldots, x_n^D)$: individuos que contienen valores discretos en orden creciente
$x_i^D \in \{1, 2, \ldots, n\}$: valor de la i-ésima variable en el individuo

- *Asignar cada valor x_i^D, $1 \le i \le n$, a cada operación de cada trabajo*

- *Asociar cada valor x_i^D, $1 \leq i \leq n$, con cada trabajo j que pertenece al programa*
- *Ordenar los valores $x_1^C, x_2^C, \ldots, x_n^C$ del individuo x^C*
- *Sea K_i la posición en la cual cada valor x_i^C, $1 \leq i \leq n$, ocupa después de ordenar todos los valores*
- *Los valores del individuo x^D se establecerá de la siguiente manera: $\forall\ i = 1, 2, \ldots, n$, $x_i^D = K_i$*
- *Relacionar cada valor x_i^D, $1 \leq i \leq n$, con cada trabajo j que pertenece al programa*

Figura 2.51 Pseudocódigo utilizado para decodificar individuos con valores continuos en Rn, en individuos con valores discretos para formar una secuencia valida.

Operación	$O_{1,1}$	$O_{1,2}$	$O_{2,1}$	$O_{2,2}$	$O_{2,3}$	$O_{3,1}$	$O_{3,2}$	$O_{4,1}$	$O_{4,2}$	$O_{4,3}$
Asignar número fijo	1	2	3	4	5	6	7	8	9	10
Relacionar trabajo	1	1	2	2	2	3	3	4	4	4

Descendientes - Valores Reales

0.24	0.16	0.37	0.77	0.82	0.66	0.81	0.55	0.83	0.96

Ordenando	2	1	3	6	8	5	7	4	9	10
Asignando trabajo	1	1	2	3	4	2	3	2	4	4

Secuencia de Operaciones - Valores discretos

Figura 2.52 Decodificación de un individuo a una secuencia de operaciones valida.

El segundo gráfico probabilístico tiene por objetivo determinar una estimación del modelo de distribución para generar descendientes (asignación de máquinas) utilizando un subconjunto de 'm' asignaciones seleccionadas (individuos). Para obtener la estimación utilizamos el algoritmo COMIT introducido por Baluja & Davies (1997b). En este modelo, la estimación de la distribución conjunta es a través de individuos seleccionados de la población utilizando el modelo estimado de la generación anterior. Se establece 'θ' como el conjunto de departamentos de producción definidos en cada calendario laboral. Estos representan ahora las variables para este modelo grafico probabilístico.
La estimación de la distribución de probabilidad se realiza mediante una estructura de árbol (Red Bayesiana) que se desarrolla y/o construye a través del algoritmo propuesto por Chow & Liu (1968). La Figura 2.53 representa un ejemplo de un árbol que puede resultar al estimar la distribución.

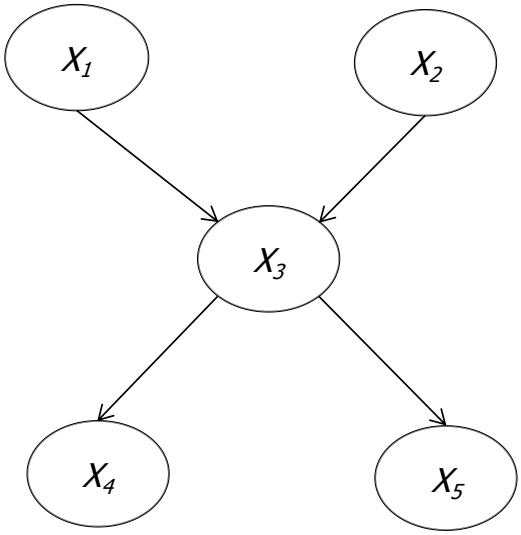


Figura 2.53 Ejemplo de un árbol que puede resultar al estimar la distribución.

El algoritmo COMIT que se construye a partir de las distribuciones de probabilidad de segundo orden obtenidas por un subconjunto de individuos los cuales son considerados los mejores de su generación y que utiliza un árbol de dependencias como modelo gráfico probabilístico para obtener el óptimo. Hibrida el enfoque de los Algoritmos de Estimación de Distribuciones con métodos locales de optimización.
Una vez que la distribución de probabilidad es estimada a partir de los mejores individuos, los descendientes son generados a partir de esta distribución. Los mejores de estos individuos son considerados puntos iniciales para un procedimiento de búsqueda más rápido. Y algunos de estos individuos obtenidos en el muestreo son añadidos a los previos individuos seleccionados que se utilizaron para generar la distribución, creando así la nueva población de individuos.
La Figura 2.54 detalla el pseudocódigo del algoritmo mencionado.

Comenzar
t=0
Generar individuos aleatoriamente D_t
Mientras el criterio de paro no se alcance
    t = t + 1
    Seleccionar N≤M individuos de D_{t-1} para S_{t-1}
    Estimar la Distribución de Probabilidad
    $p_t(x) = p(x|S_{t-1})$
    Muestrear M individuos de $p_t(x)$
Fin mientras
Fin

Figura 2.54 Pseudo-código del algoritmo COMIT.

2.3.2. MODELADO DE LA DEPENDENCIA EN TURNOS DE PRODUCCIÓN.

El tercer modelo grafico probabilístico tiene por objeto determinar una estimación de la distribución de probabilidad para generar así nuevos individuos, los cuales ahora son representaciones del calendario laboral que los operadores y los equipos deben ejecutar.
Aunque la cantidad de horas a laborar está definida para cualquier turno de trabajo en base a los lineamientos legales vigentes, no así la hora de inicio de la jornada, por lo tanto el interés es generar individuos que representen las horas fuera de servicio que estarán los operadores antes de iniciar sus actividades. Esto es aplicable a cada uno de los departamentos de producción involucrados en el proceso de manufactura modelado.

Para lograr estimar la distribución de probabilidad del calendario laboral, nuevamente se utiliza el algoritmo COMIT introducido por Baluja & Davies (1997b).

El reemplazamiento es la última fase del Algoritmo de Estimación de Distribuciones. Según Jarboui *et al* (2009) consiste en actualizar la población. En este caso se utiliza el mismo procedimiento de reemplazamiento que en el Algoritmo Genético descrito en la sección anterior.
Así mismo para el criterio de paro, se sigue las ideas de Beasley *et al* (1993).

2.3.3. ALGORITMO EDA.

El WIP es utilizado por el EDA para proporcionar información sobre los avances de la búsqueda de la mejor solución. A continuación se detalla el núcleo del EDA.

```
PROCEDIMIENTO EDA ()
COMENZAR
ESTABLECER Número de Generaciones
ESTABLECER Número de Individuos

GENERAR_POBLACIÓN ()
MIENTRAS (el criterio de paro no es alcanzado)
        SELECCIONAR ()
        EDA ()
                MIMIC_SECUENCIAS ()
                COMIT_MÁQUINAS ()
                COMIT_TURNOS()
        MUESTREAR ()
        PARA CADA Individuo
                EVALUAR ()
        FINPARA CADA
        REEMPLAZAR ()
FINMIENTRAS
IMPRIMIR Secuencia_Final
FINCOMENZAR

SUBPROCEDIMIENTO GENERAR_POBLACIÓN ()
COMENZAR
LEER ARCHIVO Carga_de_Trabajo
        Trabajos ← Órdenes_de_Producción
        Operaciones ← Precedencia
        Máquinas ← Factibilidad Operacional
CERRAR ARCHIVO Carga_de_Trabajo
PARA CADA Individuo
        Vector de Secuencia de Operaciones ←ASIGNAR  Trabajos
        Vector de Asignación de Máquinas ←ASIGNAR  Máquinas
        Vector de Horas Fuera de Servicio ←CREAR horas fuera de servicio
        EVALUAR ()
FINPARA CADA
FINCOMENZAR
```

ANÁLISIS DE RESULTADOS Y CONCLUSIONES

Se utilizó un equipo Dell Vostro® 3500, con procesador Intel® Core™ i3, 2.6 GHz, 4 GB de RAM, Windows® 7 de 64 bits para ejecutar cada algoritmo. Se programó ambos algoritmos en DevC++®, y un modelo de simulación en Delmia Quest® R20.

Para tener en cuenta la naturaleza estocástica del taller de manufactura, se utilizó 25 ensayos para ambos algoritmos. Cada ensayo consta de 11 generaciones. 75 individuos pertenecen a cada generación. El modelo de simulación necesitó a lo más 4 minutos para evaluar a cada individuo.

Se estableció una carga de trabajo para evaluar y encontrar el mejor programa de producción. Los experimentos fueron diseñados sobre la base de producción de 1,000 puertas. La carga de trabajo mencionada contiene diferentes órdenes, fechas de entrega y tipo de puertas para producir en una jornada laboral semanal, replicando así al sistema de fabricación.

Como variable de respuesta para el experimento, medimos el aumento del porcentaje relativo RPI (por sus siglas en inglés Relative Percentage Increase)

$$RPI\ (ci)\ =\ (ci - c*)\ /c*\ x\ 100 \qquad \text{(Ec. 3.1)}$$

donde ci es el WIP obtenido en la i-esima replica, y $c*$ es el mejor valor obtenido de WIP. Nótese que para este problema, no hay técnicas exactas eficaces y comparar contra una solución óptima no es posible precisamente por los supuestos que no se satisfacen.

En la Tabla 3.1 se detalla los resultados experimentales para cada ensayo, es decir, el promedio obtenido en cada ensayo denotado por μ. Se analiza si existe diferencia estadísticamente significativa entre promedios entre ambos algoritmos.

Tabla 3.1 Comparación de resultados para cada promedio.

Ensayo	GA $c*$	EDA $c*$	GA μ_1	EDA μ_2	$\mu_1 - \mu_2$	$H_0: \mu_1 = \mu_2$ $\lvert Z_c \rvert \leq 1.645 \ \ \alpha = 0.10$
1	2.66	3.50	4.97	6.37	-1.40	16.64*
2	3.21	2.56	5.30	5.35	-0.05	0.58
3	2.93	0.91	5.29	5.63	-0.34	3.37*
4	3.31	0.00	5.23	4.14	1.09	11.47*
5	2.66	0.03	5.01	4.09	0.92	9.175*
6	2.66	0.00	4.95	4.15	0.80	8.160*
7	2.66	0.00	4.90	4.16	0.74	7.580*
8	2.66	0.00	5.03	4.19	0.85	8.450*
9	2.66	0.39	5.01	4.15	0.87	8.785*
10	2.66	0.03	4.91	4.09	0.81	8.381*
11	2.66	0.00	4.98	4.20	0.78	7.931*
12	2.66	0.03	4.98	4.05	0.93	9.447*
13	2.66	0.03	5.08	4.14	0.94	9.402*
14	2.66	0.00	5.02	4.14	0.87	8.906*
15	2.66	0.03	4.96	4.11	0.85	8.994*
16	2.66	0.00	4.98	4.18	0.80	8.430*
17	2.66	0.00	5.05	4.06	0.99	9.730*
18	2.66	0.21	5.09	4.10	0.99	9.946*
19	2.66	0.00	5.02	4.14	0.88	8.927*
20	2.66	0.00	4.98	4.24	0.74	7.620*
21	2.66	0.03	5.07	4.32	0.76	7.390*
22	2.66	0.00	4.89	4.17	0.72	7.603*
23	2.66	0.00	4.97	4.17	0.80	8.248*
24	2.90	0.00	4.89	4.31	0.58	5.943*

25	2.66	0.03	5.02	4.21	0.81	8.089*

De acuerdo a los resultados obtenidos en la Tabla 3.1 existe diferencia estadísticamente significativa de ambos algoritmos. El desempeño del EDA fue superior en 24 de 25 ensayos con un alfa $\alpha = 0.10$ de nivel de significancia.

En la tabla 3.2 se muestran los resultados de la varianza (denotada por σ^2) obtenida por ambos algoritmos en cada ensayo. Igualmente se analiza si existe diferencia estadísticamente significativa entre las varianzas de ambos algoritmos.

Tabla 3.2 Comparación de resultados para cada varianza.

Ensayo	GA $c*$	EDA $c*$	GA σ_1^2	EDA σ_2^2	σ_1^2/σ_2^2	H_0: $\sigma_1^2 = \sigma_2^2$ $\alpha = 0.01$ $F_c < 0.545$ or $F_c > 1.832$
1	2.66	3.50	2.926	2.903	1.0079	1.0079*
2	3.21	2.56	3.124	2.661	1.1738	1.1738*
3	2.93	0.91	2.763	5.553	0.4976	0.4976
4	3.31	0.00	2.377	5.028	0.4727	0.4727
5	2.66	0.03	3.307	4.933	0.6704	0.6704*
6	2.66	0.00	2.848	5.050	0.5640	0.5639*
7	2.66	0.00	2.482	5.284	0.4698	0.4698
8	2.66	0.00	3.116	5.150	0.6050	0.6050*
9	2.66	0.39	2.977	5.053	0.5891	0.5891*
10	2.66	0.03	2.932	4.828	0.6072	0.6072*
11	2.66	0.00	2.803	5.149	0.5443	0.5443
12	2.66	0.03	3.055	4.881	0.6258	0.6258*
13	2.66	0.03	3.267	5.004	0.6529	0.6529*
14	2.66	0.00	2.969	4.990	0.5950	0.5950*
15	2.66	0.03	2.743	4.601	0.5962	0.5962*
16	2.66	0.00	2.897	4.541	0.6380	0.6380*
17	2.66	0.00	3.223	5.323	0.6055	0.6055*
18	2.66	0.21	3.214	4.936	0.6511	0.6511*
19	2.66	0.00	3.049	4.935	0.6178	0.6178*
20	2.66	0.00	2.759	5.036	0.5478	0.5478*
21	2.66	0.03	3.003	5.662	0.5303	0.5303
22	2.66	0.00	2.581	4.919	0.5246	0.5246
23	2.66	0.00	2.913	4.891	0.5955	0.5955*
24	2.90	0.00	2.704	5.152	0.5248	0.5248
25	2.66	0.03	3.006	5.176	0.5807	0.5807*

Como se puede apreciar en la Tabla 3.2 no existe diferencia estadísticamente significativa entre varianzas de ambos algoritmos. El desempeño es prácticamente el mismo en 18 de 25 ensayos con un alfa $\alpha = 0.01$ de nivel de significancia. Se considera entonces que la estabilidad de ambos algoritmos es ciertamente el mismo en 72% de las veces que se realizó el experimento.

La Tabla 3.3 detalla la confiabilidad de ambos algoritmos. Se analiza la cantidad de veces que ambos algoritmos obtienen el mejor valor en cada ensayo.

Tabla 3.3 Comparación de resultados para cada algoritmo.

Ensayo	GA Mejor valor	EDA Mejor valor	GA Peor valor	EDA Peor valor	GA Confiabilidad	EDA Confiabilidad
1	2.66	3.50	11.05	11.28	1	3
2	3.21	2.56	11.19	11.18	1	1
3	2.93	0.91	11.28	11.04	1	1
4	3.31	0.00	10.85	11.19	1	2
5	2.66	0.03	11.05	10.91	1	4

6	2.66	0.00	11.05	11.19	1	4
7	2.66	0.00	11.05	11.19	1	3
8	2.66	0.00	11.05	11.19	1	2
9	2.66	0.39	10.96	11.19	3	5
10	2.66	0.03	11.05	11.19	3	2
11	2.66	0.00	11.05	11.19	1	2
12	2.66	0.03	11.05	11.05	1	7
13	2.66	0.03	11.05	11.19	1	6
14	2.66	0.00	11.05	11.19	1	3
15	2.66	0.03	10.89	10.91	1	2
16	2.66	0.00	10.96	11.19	2	2
17	2.66	0.00	11.05	11.05	2	2
18	2.66	0.21	11.05	11.19	2	3
19	2.66	0.00	11.05	10.43	1	2
20	2.66	0.00	10.96	11.19	1	2
21	2.66	0.03	11.05	11.19	2	1
22	2.66	0.00	10.89	11.19	1	2
23	2.66	0.00	11.05	11.19	1	3
24	2.90	0.00	10.96	11.05	1	4
25	2.66	0.03	11.05	11.05	2	4
					34 / 25	72 / 25

Se aprecia que la confiabilidad del algoritmo EDA es superior. Obteniendo 38 veces más el mejor valor contra el algoritmo genético.

En la Figura 3.1 se muestran los individuos generados por el algoritmo AG en algunos ensayos a través de una gráfica de percentiles.

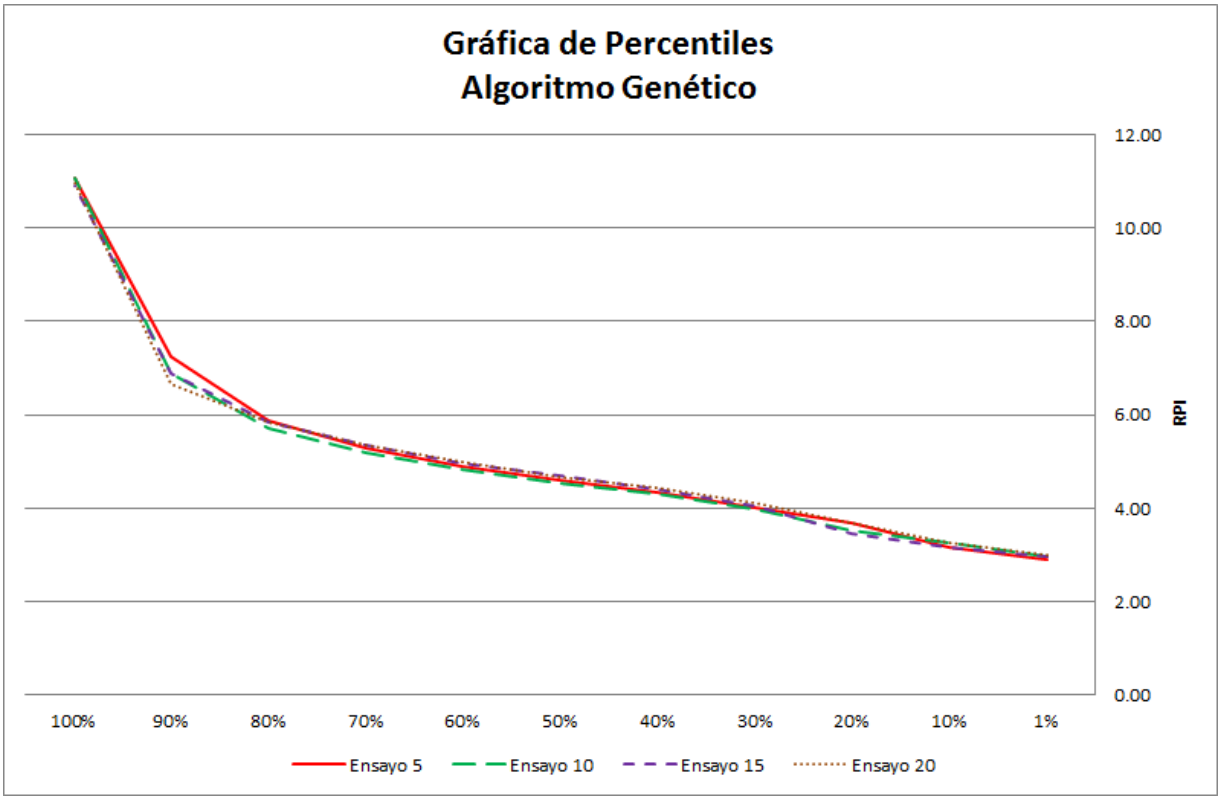


Figura 3.1 Gráfica de Percentiles, Ensayos-AG.

Para todos los ensayos el AG converge y obtiene su mejor valor para la variable de respuesta RPI en alrededor de 3.02

En la Figura 3.2 se muestran los individuos generados por el EDA en algunos ensayos a través de una gráfica de percentiles.

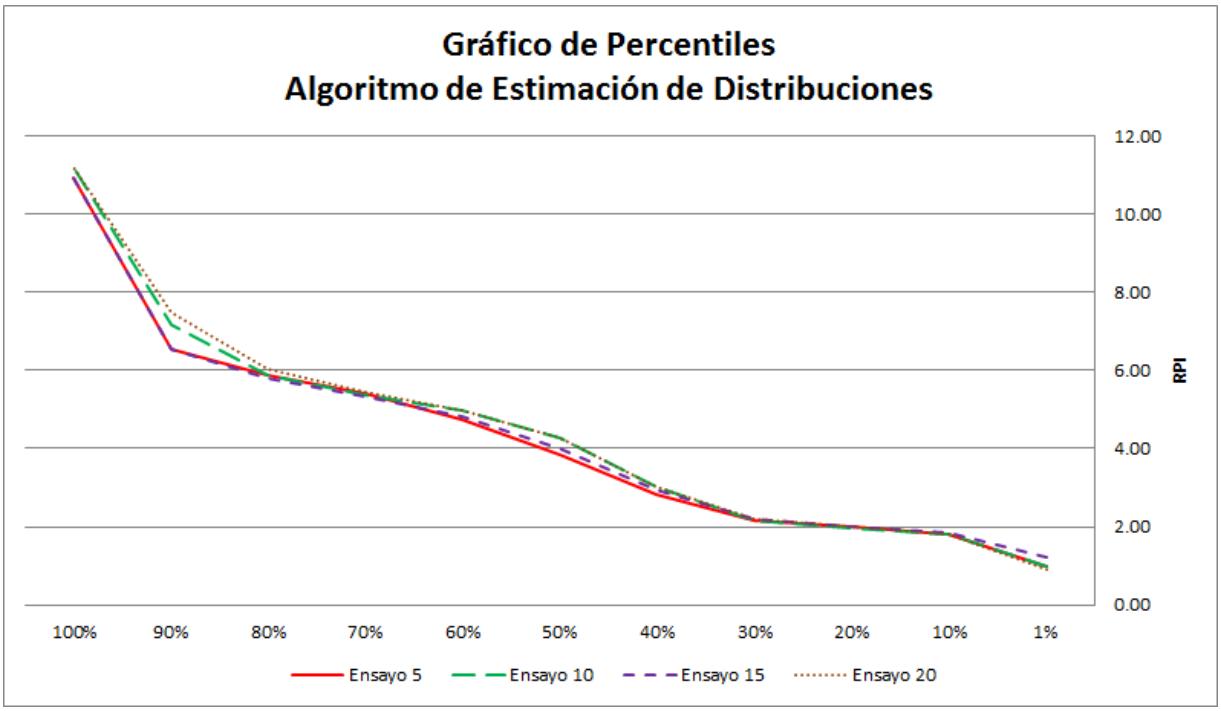


Figura 3.2 Gráfica de Percentiles, Ensayos-EDA.

Los ensayos del EDA fueron diferentes; estos convergen y obtienen su mejor valor debajo de 3.00 para la variable de respuesta RPI, tres veces en los ensayos.

Los resultados experimentales fueron analizados por el método ANOVA (por sus siglas en inglés ANalysis Of VAriance). En el experimento, los principales supuestos fueron corroborados y aceptados. En la Tabla 3.4 se detalla que hay diferencia estadísticamente significativa entre ambos algoritmos.

Tabla 3.4 Análisis de Varianza.

ANOVA						
Fuente de variación	**Suma Cuadrados**	**gl**	**Promedio Cuadrados**	**F**	**Valor p**	**F crítico**
Muestras	50.3280	1	50.328	19.0555	1.28934E-05	3.84288
Generaciones	4846.39	43	112.706	42.6738	0	1.38108
Interacción	2660.61	43	61.874	23.4273	5.274E-169	1.38108
Entre grupos	17199.0074	6512	2.641			
Total	24756.344	6599				

En la Figura 3.3 se muestra el desempeño global entre ambos algoritmos.

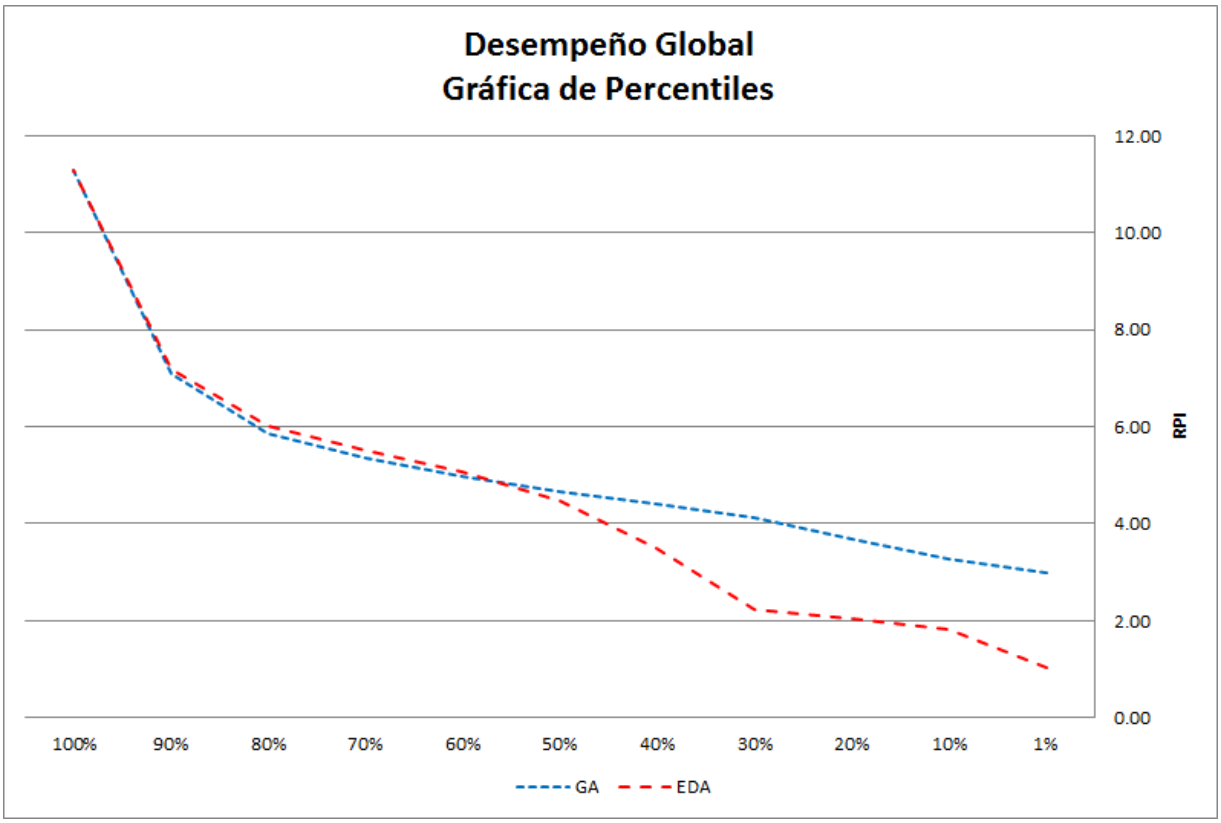


Figura 3.3 Desempeño Global para ambos algoritmos.

Aunque el problema de programación tradicional ha sido resuelto desde una perspectiva académica, su aplicación en entornos industriales ha sido una tarea difícil porque hay muchos supuestos que satisfacer. Basándose en los resultados experimentales mostrados, se confirmó que un modelado adecuado de las variables más importantes que afectan al rendimiento del proceso debe ser considerado en la solución propuesta. Se llega a la conclusión de que el funcionamiento de la tienda se puede mejorar si las diferentes máquinas son asignadas a diferentes horarios. Aunque algunos supuestos del FJSP no podían ser satisfechos en el proceso de fabricación estudiado, debido a que un algunos de los supuestos no son consistentes, el modelo de simulación fue capaz de hacer frente a esta situación mediante la incorporación de las condiciones de operación reales del proceso de fabricación. La razón es que el proceso de manufactura estudiado es más sensible que las configuraciones clásicas. El método de validación del modelo de simulación permitió la identificación de una oportunidad clave para mejorar el proceso de fabricación, es decir, secuencias de trabajo y el tiempo fuera de servicio para cada máquina.

El uso de un EDA continuo no requiere hacer ninguna modificación en el proceso de muestreo para el procesamiento de las operaciones en las máquinas, ya que generalmente esto es requerido por otros algoritmos similares. Se obtuvo una mejor confianza en los datos frente a los obtenidos por el AG al considerar el uso de tres modelos gráficos, mientras que la mayoría de EDAs no aplican más de un modelo. Se permitió el manejo de las variables más importantes del proceso de fabricación estudiado en el mecanismo de muestreo y esta estrategia fue diseñada para evitar la pérdida de la diversidad en el progreso evolutivo del algoritmo. Se logra la conclusión de que la optimización de simulaciones puede ser un mecanismo eficiente para manejar diferentes condiciones de fabricación en los que hay diversas interacciones de variables tales como las que se tienen en el FJSP.

Por último, esta investigación aporta el uso de un EDA como un método de optimización para cualquier lenguaje de simulación.

BIBLIOGRAFÍA

Afrati, F., Bampis, E., Chekuri, C., Karger, D., Kenyon, C., Khanna, S., et al. (1999). Approximation schemes for minimizing average weighted completion time with release dates. *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science.* (pp. 32-43). Los Alamitos, CA.: IEEE Computer Society Press.

Akers Jr., S., & Friedman, J. (1955). A non-numerical approach to production scheduling problems. *Operations Research., 3*, 429-442.

Andradóttir, S. (1998). A Review of Simulation Optimization Techniques. In D. Medeiros, E. Watson, & J. Carson (Ed.), *Proceedings of the 1998 Winter Simulation Conference.*, (pp. 151-158).

April, J., Glover, F., Kelly, J., & Laguna, M. (2009). *The exploding domain of simulation optimization.* Retrieved 2010, from http://leeds-faculty.colorado.edu/glover/opttek%20-20exploding%20domain%20sim%20opt%204-24-06.pdf

Askin, R., & Standridge, C. (1993). *Modeling and Analysis of Manufacturing Systems.* John Wiley and Sons, Inc.

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice.* Oxford, New York.: Oxford University Press.

Baid, N., & Nagarur, N. (1994). An integrated decision support system for FMS: using intelligent simulation. *International Journal of Production Research., 32*(4), 951-965.

Baker, K. (1974). *Introduction to Sequencing & Scheduling.* New York.: Wiley.

Balas, E., Adams, J., & Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science., 34*(3), 391-401.

Baluja, S. (1994). *Population-based incremental learning: A method for integrating genetic search based function optimization & competitive learning.* Technical Report CMU-CS-94-163., Pittsburgh, PA.

Baluja, S., & Davies, S. (1997a). Combining multiple optimization runs with optimal dependency trees. *Technical Report CMU-CS-97-157.* Carnegie Mellon University.

Baluja, S., & Davies, S. (1997b). Using optimal dependence-trees for combinatorial optimization: Learning the structure of the search space. *Technical Report CMU-CS-97-107.* Carnegie Mellon University.

Bean, J., & Norman, B. (1993). Random keys for job shop scheduling problem. *Technical Report TR 93-7.* The University of Michigan.

Beasley, D., Bull, D., & Martin, R. (1993). *An Overview of Genetic Algorithms, Part 1 Fundamentals.* Retrieved 2011, from http://www.ie.metu.edu.tr/~ie505/CourseMaterial/Beasley,%20Bull%20and%20Martin%20(1993a).pdf

Bierwirth, C., & Mattfeld, D. (1999). Production scheduling & rescheduling with genetic algorithms. *Evolutionary Computation., 7*(1), 1-17.

Brooks, G., & White, C. (1965). An algorithm for finding optimal or near optimal solutions to the production scheduling problem. *Journal Industrial Engineering., 16*, 34-40.

Cerny, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory & Applications., 45*, 41-51.

Chan, F., & Chan, H. (2004). A comprehensive survey and future trend of simulation study on FMS scheduling. *Journal of Intelligent Manufacturing., 15*, 87-102.

Chan, S., & Koh, P. (1994). A simulation model for planning and scheduling a complex warehousing operation. *Proceedings of International Conference WORKSIMS'94.*, (pp. 267-272). Thailand.

Chen, B., Glass, C., Potts, C., & Strusevich, V. (1996). A new heuristic for three-machine flow shop scheduling. *Operations Research., 44*, 891-898.

Chen, S.-H., Chang, P.-C., Cheng, T., & Zhang, Q. (2012b). A Self-guided Genetic Algorithm for permutation flowshop scheduling problems. *Computers and Operations Research, 39*, 1450-1457.

Chen, S.-H., Chen, M.-C., Chang, P.-C., Zhang, Q., & Chen, Y.-M. (2010a). Guidelines for developing effective Estimation of Distribution Algorithms in solving single machine scheduling problems. *Expert Systems with Applications, 37*, 6441-6451.

Chen, Y.-M., Chen, M.-C., Chang, P.-C., & Chen, S.-H. (2012c). Extended artificial chromosomes genetic algorithm for permutation flowshop scheduling problems. *Computers & Industrial Engineering, 62*, 536-545.

Chow, C., & Liu, C. (1968). Approximation discrete probability distributions with dependence trees. *IEEE Transactions & Information Theory., 14*, 462-467.

Cleveland, G., & Smith, S. (1989). Using genetic algorithms to schedule flow shop releases. In J. Schaffer (Ed.), *Proceedings 3rd International Conference on GAs.* (pp. 160-169). Morgan Kauffman.

Conway, R., Maxwell, W., & Miller, L. (1967). *Theory of Scheduling.* Massachusetts.: Addison Wesley Publishing Company.

Cook, S. (1971). The complexity of theorem-proving procedures. *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing.* (pp. 151-158). New York.: The Association for Computing Machinery.

Croes, G. (1958). A method for solving traveling salesman problems. *Operations Research., 6*, 791-812.

Crowder, H., & Padberg, M. (1980). Solving large-scale symmetric travelling salesman problems to optimality. *Management Science., 26*, 495-509.

Davis, L. (1985). Job-shop scheduling problem with genetic algorithms. In J. Grefenstette (Ed.), *Proceedings International Conference on GAs*, (pp. 136-140). Lawrence Erlbaum.

De Bonet, J., Isbell, C., & Viola, P. (1997). MIMIC: Finding Optima by Estimation Probability Densities. *Advances in Neural Information Processing Systems., 9*.

De Werra, D. (1989). Graph-theoretical models for preemptive scheduling. In R. Slowinski, & J. Weglarz (Ed.), *Advances in Project Scheduling.* (pp. 171-185). Amsterdam.: Elsevier.

DeGroot, M. (1970). *Optimal Statistical Decisions.* New York.: Mcgraw-Hill.

DeJong, K., & Spears, W. (1990). An Analysis of the Interacting Roles of Population Size and Crossover in Genetic Algorithms. *Proceedings First Workshop Parallel Problem Solving From Nature.* (pp. 38-47). Berlin.: Springer-Verlag.

Dorndorf, U., & Pesch, E. (1992). *Evolution based learning in a job-shop scheduling environment.* Technical Report RM-92-019., Limburg University., Faculty of Economics.

Eastman, W. (1958). A solution to the traveling-salesman problem. *Presentation at the American Summer Meeting of the Econometric Society.* Cambridge, MA.

Edmonds, J. (1965). Paths, trees, & flowers. *Canadian Journal of Mathematics., 17*, 449-467.

Elmaghraby, S. (1968). The one-machine sequencing problem with delay costs. *Journal Industrial Engineering., 19*, 105-108.

Etxeberría, R., & Larrañaga, P. (1999). Optimization with bayesian networks. *Proceedings of the Second Symposium on Artificial Intelligence. Adaptive Systems.* Cuba.: CIMAF 99, 332339.

Fogel, L. (1962). Autonomous automata. *Industrial Research., 4*, 14-19.

Fourman, M. (1985). M. Fourman, Compaction of symbolic layout using genetic algorithms. In J. Grefenstette (Ed.), *Proceedings International Conference on GAs*, (pp. 136-141). Lawrence Erlbaum.

Fox, B., & McMahon, M. (1991). Genetic operators for sequencing problems. In G. Rawlins (Ed.), *Proceedings International Conference on GAs. Foundations of Genetic Algorithms.* Morgan Kauffman.

Fu, M. (2002). Optimization for Simulation: Theory and Practice. *INFORMS Journal on Computing., 14*(3), 192-215.

Gabow, H., & Kariv, O. (1982). Algorithms for edge coloring bipartite graphs & multigraphs. *SIAM Journal Computing., 11*, 117-129.

Gantt, H. (1916). *Work, Wages & Profits.* (2nd Edition. ed.). New York.: Engineering Magazine Co.

García, D., García, R., & Cárdenas, B. (2006). *Simulación y análisis de sistemas con Promodel®.* México: Prentice-Hall.

Garey, M., & Johnson, D. (1979). *Computers and intractability: a guide to the theory of NP-Completeness.* W.H. Freeman.

Gerencsér, L. (1999). Rate of convergence of moments for a simultaneous perturbation stochastic approximation method for function minimization. *IEEE Transactions Automatic Control., 44*, 894-906.

Gilmore, P., & Gomory, R. (1964). Sequencing a one-state variable machine: a solvable case of the traveling salesman problem. *Operations Research., 12*, 665-679.

Glover, F. (1986). Future paths for integer programming & links to artificial intelligence. *Computers & Operations Research., 13*, 533–549.

Glover, F. (1989). Tabu search: Part I. *ORSA Journal on Computing., 1*, 190-206.

Glover, F. (1990). Tabu search: Part II. *ORSA Journal on Computing., 2*, 4-32.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading., MA.: Adison-Wesley.

Goldberg, D., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis and first results. *Complex Systems., 3*, 493–530.

Gonzalez, T., & Sahni, S. (1976). Open shop scheduling to minimize finish time. *Journal Association for Computing Machinery., 23*, 665-679.

Gonzalez, T., & Sahni, S. (1978). Flow shop & job shop schedules: complexity & approximation. *Operations Research., 26*, 36-52.

Graham, R. (1966). Bounds for certain mutiprocessing anomalies. *Bell Systems Technical Journal., 45*, 1563-1581.

Graham, R. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics., 17*, 416-429.

Graham, R., Lawler, E., Lenstra, J., & Rinnooy, K. A. (1979). Optimization & approximation in deterministic sequencing & scheduling: a survey. *Annual Discrete Mathematics., 5*, 287-326.

Greenwood, A., Vanguri, S., Eksioglu, B., Jain, P., Hill, T., Miller, J., et al. (2005). Simulation Optimization Decision Support System for Ship Panel Shop Operations. In M. E. Kuhl, N. M. Steiger, F. Armstrong, & J. Joines (Ed.), *Proceedings of the 2005 Winter Simulation Conference*, (pp. 2078-2086).

Grefenstette, J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, & Cybernetics., 16*, 122–128.

Grötschel, M. (1980). On the symmetric travelling salesman problem: solution of a 120-city problem. *Mathematical Program Study., 12*, 61-77.

Hall, L. (1998). Approximability of flow shop scheduling. *Mathematical Programming., 82*, 175-190.

Hall, L., Schulz, A., Shmoys, D., & Wein, J. (1997). Scheduling to minimize average completion time: off-line & on-line approximation algorithms. *Mathematics of Operations Research., 22*, 513-544.

Harik, G. (1997). Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms. *PhD Thesis.* University of Michigan.

Harik, G. (1999). *Linkage learning via probabilistic modeling in the ECGA.* Technical Report IlliGAL Report No. 99010., University of Illinois., Urbana-Champaign.

Harik, G., Lobo, F., & Goldberg, D. (1999). The compact genetic algorithm. *IEEE-EC, 3*, 287.

Held, M., & Karp, R. (1971). the traveling-salesman problem & minimum spanning trees: Part II. *Mathematical Programming., 1*, 6-25.

Held, M., Wolfe, P., & Crowder, H. (1974). Validation of subgradient optimization. *Mathematical Programming., 6*, 62-88.

Hochbaum, D., & Shmoys, D. (1987). Using dual approximation algorithms for scheduling problems: Theoretical & practical results. *Journal of Association for Computing Machinery., 34*, 144-162.

Holland, J. (1975). *Adaptation in Natural & Artificial Systems.* Arbor, MI.: University of Michigan Press.

Hoogeveen, H., Schuurman, P., & Woeginger, G. (2001). Non-approximability results for scheduling problems with minsum criteria. *INFORMS Journal Computing., 13*, 157-168.

Horowitz, H., & Sahni, S. (1976). Exact & approximate algorithms for scheduling non identical processors. *Journal of Association for Computing Machinery., 23*, 317-327.

Husbands, P. (1994). Retrieved 2011, from http://www.sussex.ac.uk/Users/philh/pubs/aisb.sched.pdf

Ibarra, O., & Kim, C. (1975). Fast approximation algoritms for the knapsack & sum of subset problems. *Journal of Association for Computing Machinery., 22*, 463-468.

Ignall, E., & Schrage, L. (1965). Application of the branch-and-bound technique to some flow-shop scheduling problems. *Operations Research., 13*, 400-412.

Jackson, J. (1956). An extension of Johnson's result on job lo scheduling. *Naval Research Logistics., Quart 3.*, 201-203.

Jarboui, V., Eddaly, M., & Siarry, P. (2009). An Estimation of Distribution Algorithm for minimizing the total flow time in permutation flowshop scheduling problems, Computers and Operations Research, 36 (2009), 2638-2646. *Computers and Operations Research., 36*, 2638-2646.

Johnson, S. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics., Quart 1.*, 61-68.

Jordan, M. (1998). *Learning in Graphical Models.* Dordrecht.: NATO Science Series, Kluwer Academic Publishers.

Kaltwasser, J., Hercht, A., & Lang, R. (1986). Hierarchical control of flexible manufacturing systems. *Proceedings of IFAC Information Control Problems in Manufacturing Technology.*, (pp. 37-45). Suzdal, URSS.

Kargupta, H. (1996). The gene expression messy genetic algorithm. *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation.*, (pp. 631–636).

Karmakar, N. (1984). A new polynomial-time algorithm for linear programming. *Proceedings of the 16th Annual ACM Symposium on Theory of Computing.* (pp. 302-311). New York.: The Association for Computing Machinery.

Karp, R. (1972). Reducibility among combinatorial problems. In R. Miller, & J. Thatcher (Ed.), *Complexity of Computer Computations.* (pp. 85-103). New York.: Plenum Press.

Khachiyan, L. (1979). A polynomial algorithm in linear programming. *Soviet Mathematics Doklady., 20*, 191-194.

Kirkpatrick, S., Gelatt Jr., C., & Vecchi, M. (1983). Optimization by simulated annealing. *Science., 220*, 671-680.

Kononov, A., & Sviridenko, M. (2002). A linear time approximation scheme for makespan minimization in an open shop with release dates. *Operations Research Letters., 30*, 276-280.

Land, A., & Doig, A. (1960). An automatic method of solving discrete programming problems. *Econometrica., 28,* 497-520.

Larrañaga, P., & Lozano, J. (2002). *Estimation of Distribution Algorithms: a new tool for evolutionary computation.* Boston/Dordrecht/London.: Kluwer Academic Publishers.

Larrañaga, P., Etxeberría, R., Lozano, J., & Peña, J. (1999). *Optimization by learning & simulation of bayesian & gaussian networks.* Technical Report EHU-KZAA-IK-4/99., University of the Basque Country.

Larrañaga, P., Exteberria, R., Lozano, J., & Peña, J. (2000). Optimization in continuous domains by learning & simulation of Gaussian networks. In A. Wu (Ed.), *Proceedings of the 2000 Genetic & Evolutionary Computation Conference Workshop Program.*, (pp. 201-204).

Lauritzen, S. (1996). *Graphical Models.* Oxford University Press.

Law, A., & Kelton, W. (1991). *Simulation Modeling & Analysis.* New York.: McGraw-Hill.

Lawler, E., & Moore, J. (1969). A functional equation & its application to resource allocation & sequencing problems. *Management Science., 16,* 77-84.

Lenstra, J., Shmoys, D., & Tardos, É. (1990). Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming., 46,* 259-271.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal., 44*, 2245-2269.

Liu, H., Gao, L., & Pan, Q. (2011). A hybrid particle swarm optimization with estimation of distribution algorithm for solving permutation flowshop scheduling problem. *Experts Systems with Applications*(38), 4348-4360.

Lomnicki, Z. (1965). A branch-and-bound algorithm for the exact solution of the three machine scheduling problem. *Operational Research., Quart 16.*, 89-100.

MacCarthy, B., & Liu, J. (1993). Addressing the gap in scheduling search: a review of optimization & heuristic methods in production scheduling. *International Journal of Production Research., 31*(1), 59-79.

McNaughton, R. (1959). Scheduling with deadlines & loss functions. *Management Science., 6,* 1-12.

Miliotis, P. (1976). Integer programming approaches to the travelling salesman problem. *Mathematical Programming., 10,* 367-378.

Miliotis, P. (1978). Using cutting planes to solve the symmetric travelling salesman problem. *Mathematical Programming., 15,* 177-188.

Miller, B., & Goldberg, D. (1995). *Genetic algorithms, tournament selection, and the effects of noise.* Retrieved 2011, from http://www.dai.ed.ac.uk/groups/evalg/Local_Copies_of_Papers/Miller.Goldberg.GAs_Tournament_Selection_and_the_Effects_of_the_Noise.ps.gz

Moore, J. (1968). An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science., 15*, 102-109.

Mühlenbein, H., & Mahnig, T. (1999). FDA-A scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation., 7*, 353-376.

Mühlenbein, H., & Paaß, G. (1996). From recombination of genes to the estimation of distributions: I. binary parameters. In H. Voigt, W. Ebeling, I. Rechenberg, & H. Schwefel (Ed.), *Parallel Problem Solving from Nature PPSN IV.* (pp. 178–187). Berlin.: Springer.

Mühlenbein, H., Mahnig, T., & Ochoa, A. (1999). Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics., 5*, 215–247.

Muth, J., & Thompson, G. (1963). *Industrial Scheduling.* Englewood Cliffs., NJ: Prentice Hall.

Nabeshima, I. (1961). The order of n items processed on m machines. *International Journal Operations Research Society Japan., 3*, 170-175.

Nabeshima, I. (1967). On the bound of makespans & its application in M machine scheduling problem. *Journal Operations Research Society Japan., 9*, 98-136.

Nakano, R., & Yamada, T. (1991). Conventional genetic algorithms for job-shop problems. In R. Belew, & L. Booker (Ed.), *Proceedings of the Fourth International Conference on Genetic Algorithms ICGA-91* (pp. 474-479). Morgan Kaufmann.

Németi, L. (1964). Das Reinenfolgeproblem in der Fertigungsprogrammierung und Linearplanung mit Logischen Bedingungen. *Mathematica., (Cluj)*, 87-89.

Nicholson, T. (1967). A sequential method for discrete optimization problems & its application to the assignment, travelling salesman, & three scheduling problems. *Journal of the Institute of Mathematics and Its Applications., 3*, 362-375.

Ogbu, F., & Smith, D. (1990). The application of the simulated annealing algorithm to the solution of the n|m|Cmax flowshop problem. *Computers & Operations Research., 17*, 243-253.

Osman, I., & Potts, C. (1989). Simulated annealing for permutation flow-shop scheduling. *Omega., 17*, 551-557.

Padberg, M., & Hong, S. (1980). On the symmetric travelling salesman problem: a computational study. *Mathematical Program Study., 12*, 78-107.

Pan, Q.-K., & Ruiz, R. (2012). An estimation of distribution algorithm for lot-streaming flow shop problems with setup times. *Omega, 40*, 166-180.

Pelikan, M., & Mühlenbein, H. (1999). The bivariate marginal distribution algorithm. In R. Roy, T. Furuhashi, & P. Chawdhry (Ed.), *Advances in Soft Computing – Engineering Design and Manufacturing.* (pp. 521–535). London.: Springer-Verlag.

Pelikan, M., Goldberg, D., & Cantú-Paz, E. (1999). BOA: The Bayesian Optimization Algorithm. In W. e. Banzhaf (Ed.), *Proceedings of the Genetic and Evolutionary Computation Conference GECCO99. I*, pp. 525-532. San Fransisco, CA.: Morgan Kaufmann Publishers.

Peña, J., Peña, Robles, V., Larrañaga, P., Herves, V., Rosales, F., et al. (2004). Ga-eda: hybrid evolutionary algorithm using genetic and estimation of distribution algorithms. In B. Orchard, C. Yang, & M. Ali (Ed.), *Innovations in applied artificial intelligence. 3029*, pp. 361-371. Berlin/Heidelberg: Lecture notes in computer science.

Piehler, J. (1960). Ein Beitrag zum Reihenfolge problem. *Unternehmensforschung., 4*, 138-142.

Pinedo, M., & Schrage, L. (1982). Stochastic shop scheduling: A survey. In M. Dempster, J. Lenstra, & K. A. Rinnooy (Ed.), *Deterministic & Stochastics Scheduling.* (pp. 181-196). Riedel.: Dordrecht.

Potts, C. (1974). The Job-Machine Scheduling Problem. *PhD Thesis.* UK.: University of Birmingham.

Potts, C. (1985). Analysis of a linear programming heuristic for scheduling unrelated parallel machines. *Discrete Applied Mathematics., 10*, 155-164.

Potts, C., & Strusevich, V. (2009). Fifty years of Scheduling. A survey of milestones. *Journal of the Operational Research Society., 60*, 541-568.

Queyranne, M., & Wang, Y. (1991). Single machine scheduling polyhedra with precedence constraints. *Mathematical Operations Research., 16*, 1-20.

Rechenberg, I. (1973). *Evolution strategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Stuttgart: Formman-Holzboog.

Rothkopf, M. (1966). Scheduling independent tasks on independent processors. *Management Science., 12*, 437-447.

Roy, B., & Sussman, B. (1964). *Les problem'es d'ordonnancement avec contraintes disjonctives.* SEMA, Montrouge.: Note DS No. 9 bis.

Rudolph, G. (1991). Global optimization by means of distributed evolution strategies. In H. Schwefel, & R. Manner (Ed.), *Parallel Problem Solving from Nature PPSN I, Lectures Notes in Computer Science. 496*, pp. 209-213. Springer-Verlag.

Sahni, S. (1976). Algorithms for scheduling independent tasks. *Journal of Association for Computing Machinery., 23*, 116-127.

Sargent, R. (1998). A Tutorial on Validation & Verification of Simulation Models. In M. Abrams, P. Haigh, & J. Comfort (Ed.), *Proceedings of the 1988 Winter Simulation Conference.* (pp. 33-39). San Francisco.: IEEE.

Sargent, R. (2007). Verification & Validation of Simulation Models. In S. Henderson, B. Biller, M. Hsieh, J. Shortle, & R. Barton (Ed.), *Proceedings of the 2007 Winter Simulation Conference.*, (pp. 124-137).

Sarin, S., & Chen, C. (1987). The machine loading & toll allocation problem in a flexible manufacturing system. *International Journal of Production Research., 25*, 1081-1094.

Schuurman, P., & Woeginger, G. (1999). Polynomial time approximation alorithms for machine scheduling: Ten open problems. *Journal of Scheduling., 2*, 203-213.

Schwefel, H. P. (1997). Collective Phenomena in Evolutionary Systems. *Preprints of the 31st Annual Meeting of the International Society for General System Research.*, *2*, pp. 1025–1033. Budapest.

Sevastianov, S. (1994). On some geometric methods in scheduling theory: a survey. *Discrete Applied Mathematics., 55*, 59-82.

Sevastianov, S. (1995). Vector summation in Banach space & polynomial time algorithms for flow shops and open shops. *Operations Research., 20*, 90-103.

Sevastianov, S., & Woeginger, G. (1998). Makespan minimization in open shops: a polynomial time approximation scheme. *Mathematical Programming., 82*, 191-198.

Shachter, R., & Kenley, C. (1989). Gaussian influence diagrams. *Management Science., 35*, 527-550.

Shakya, S., McCall, J., & Brown, D. (2006). Solving the ising spin glass problem using a bivariate EDA based on Markov random fields. *Proceedings of IEEE Congress on Evolutionary Computation.* Vancouver, Canada.: IEEE CEC 2006, IEEE press.

Shim, V. A., Chen Tan, K., Yong Chia, J., & Kiat Chong, J. (2011). Evolutionary algorithms for solving multi-objective travelling salesman problem. *Flexible Service and Manufacturing Journal*(23), 207-241.

Shwimer, J. (1972). On the N-job, one-machine, sequence-independent scheduling problem with tardiness penalties: A branch-bound solution. *Management Science., 18*, 301-313.

Smith, D. (1985). Bin packing with adaptive search. In J. Grefenstette (Ed.), *Proceedings International Conference on GAs.*, (pp. 202-207). Lawrence Erlbaum.

Smith, W. (1956). Various optimizers for single-stage production. *Naval Research Logistics., Quart 3.*, 59-66.

Srinivas, M., & Patnaik, L. (1993). Binomially Distributed Populations for Modeling Genetic Algorithms. In K. Morgan (Ed.), *Proceedings Fifth International Conference on Genetic Algorithms.*, (pp. 138-145). San Mateo, CA.

Szwarc, W. (1968). On some sequencing problems. *Naval Research Logistics., Quart 15.*, 127-155.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research., 47*, 65-74.

Tunali, S. (1997). Evaluation of alternate routing policies in scheduling a job-shop type FMS. *Computers in Industrial Engineering., 32*(2), 243-250.

Wang, L., Wang, S., Xu, Y., Zhou, G., & Liu, M. (2012). A bi-population based estimation of distribution algorithm for the flexible job-shop scheduling problem. *Computers and Industrial Engineering, 62*, 917-926.

Whitley, D., Starweather, T., & Shaner, D. (1990). The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination. In L. Davis (Ed.), *Handbook of Genetic Algorithms.* (pp. 350-372). New York.: Van Nostrand Reinhold.

Whitner, R., & Balci, O. (1989). Guidelines for Selecting and Using Simulation Model Verification Techniques. *Proceedings of the 1989 Winter Simulation Conference*, (pp. 248-252). Washington, DC.

Whittaker, J. (1990). *Graphical models in applied multivariate statistics.* John Wiley & Sons.

Widmer, M., & Hertz, A. (1989). A new heuristic method for the flow shop sequencing problem. *European Journal of Operational Research, 41*, 186–193.

Wren, A., & Wren, D. (1990). *Genetics, structures & covers – an application to scheduling.* Technical Report 90.23., University of Leeds., School of Computers Science.

Zhang, Y., & Li, X. (2011). Estimation of distribution algorithm for permutation flow shops with total flowtime minimization. *Computers & Industrial Engineering*(60), 706-718.

ANEXOS

Productos Académicos

Advanced Modeling of Manufacturing Systems using Quest®.
Ricardo Pérez Rodríguez
CONACYT 11ª Feria Nacional de Posgrados 2010.
México, D.F., Mayo 2010.

Best Practices for modeling the manufacture of steel doors using Quest®.
Ricardo Pérez-Rodríguez, J. Sánchez
IIE-IERC Annual Conference & Expo 2010
Cancún, México, 5-9 Junio, 2010, pp. 140

Aplicación de la Simulación de Eventos Discretos en un Taller de Puertas de Acero en Irapuato, Gto.
Ricardo Pérez, Jöns Sánchez, Hector Castañeda Infante
Sexto Congreso Nacional y Primer Congreso Internacional "La Investigación en el Posgrado"
Aguascalientes, México, 21-22 Octubre, 2010.

Tutorial de Simulación Básica utilizando Quest®.
Ricardo Pérez Rodríguez, S. Jöns, Arturo Hernández Aguirre, Darwin Young Vázquez
Conciencia Tecnológica 41, 2011, pp. 28-34
ISSN 1405-5597
In this paper we show the basic elements used to build simulation models on Quest® version R19. The aim of this paper is to facilitate learning to build simulation models, on a 3D environment, of manufacturing processes and material handling on Quest® software. This platform has been designed especially for professionals that are working in the industrial sector, where Quest® has been positioned by its simplicity in defining the workflow and a quick integration of new elements in a simulation model. The model to build is about the construction of parts (entities) from one source, the processing of these on a machine (location), culminating in its output (sink) and its respective analysis. A plus of this tutorial is that users can quickly start the building of their own simulation models without squandering resources on expensive training courses. Finally, 10 engineering students tested the effectiveness of the document, at least 50% of them spent between 33 and 37 minutes to successfully complete the construction of the proposed model.

Discrete-event simulation using estimation of distribution algorithm.
Ricardo Pérez, Alberto Ochoa-Zezzatti, Jöns Sánchez, Arturo Hernández
Difuciencia 5(2), 2011, pp. 34-39.
ISSN 2007-3585
The science of decision making or operations research (OR) is present in all levels and in all industries. The Industrial Optimization can carry out the decision-making through analysis of the operation of any system, formulation and use of models to achieve the proposed goals and targets, with proper utilization of available resources. Its scope is very broad, applying to problems of manufacturing, transportation, construction, telecommunications, planning, financial management, health sciences, and services, among others. In this preliminary work, we present a hybrid proposal between discrete event simulation and a distribution estimation algorithm; our goal is to contribute to knowledge in the area. This work is being developed in a manufacturing plant where steel doors and frames are parallel activities.

Study of a Queue Model Using an Estimation of Distribution Algorithm.
Ricardo Pérez, S. Jöns, Arturo Hernández
International Journal of Soft Computing and Engineering 3(5), 2013, pp. 63-65
ISSN 2231-2307
An analysis of a queue model M/M/1 for an outpatient clinic was done considering no-shows from the patients. In order to detect how no-shows affect the performance measure, i.e., the doctor's idle time on the patient-doctor system we consider analyzing the behavior of the patients when they have an appointment with a previous diagnostic successfully. The alternative approach was validated using a simulation model that was built through Delmia Quest® R20 and an Estimation of Distribution Algorithm to model the workflow in a small health clinic in México.

Simulación de Eventos Discretos utilizando Delmia-Quest®. Un caso aplicado.
Ricardo Pérez Rodríguez, Jöns Sánchez

Editorial Académica Española,
1ª Edición, 2013
ISBN 978-3-659-08447-8
En este libro se presentan los elementos básicos utilizados para desarrollar modelos de simulación en Delmia Quest® versión R20. El objetivo es facilitar el aprendizaje para construir modelos de simulación sobre una integración visual 3D de procesos de manufactura y con características de manejo de materiales que la plataforma Delmia Quest® ofrece. Dicha plataforma ha sido diseñada especialmente para profesionistas que laboran en el sector industrial, donde Quest® ha podido posicionarse por su sencillez al definir el flujo de trabajo, y su rapidez en la integración de elementos. El modelo a construir se refiere al proceso de corte y doblez de lámina de acero galvanizado característico de la industria metalmecánica nacional. Con el presente tutorial, los usuarios podrán iniciar rápidamente la construcción de sus propios modelos de simulación sin dilapidar recursos en costosos cursos de capacitación. Finalmente, 10 estudiantes de ingeniería probaron la efectividad del documento, al menos 50% de ellos consumieron entre 88 y 100 minutos para terminar exitosamente la construcción del modelo propuesto.

Uso de DevC++® con Delmia Quest® para Optimizar Simulaciones.
Ricardo Pérez Rodríguez, S. Jöns, Arturo Hernández Aguirre
Conciencia Tecnológica 47, 2014, aceptado para publicación
ISSN 1405-5597
This paper shows some elements used to achieve communication between any optimization algorithm programmed in DevC++® to optimize any simulation model built on Delmia Quest® platform. The aim of this article is to facilitate the learning process to communicate both platforms, i.e., the goal is to systematize the evaluation of solutions from a practical approach. The contribution of this paper is the development and validation of a tutorial to demonstrate the process of communication between Delmia Quest® and DevC++®. The global idea refers to build a procedure that uses the instructions of the optimization algorithm programmed in DevC++® and these are executed by the simulation language Delmia Quest®, which returns a relevant output parameter to the algorithm optimization. The procedure is repeated iteratively according to the specifications of the optimization algorithm. With this tutorial, the users can quickly program their own procedures of communication even in other programming languages such as Java® without squandering resources on expensive training courses. Finally, 10 engineering students tested the effectiveness of the document, at least 50% of them consumed between 12 and 17 minutes to successfully complete the proposed programming tutorial where traditionally as barrier factor may require hours and even days because there is no bibliography accessible and clear in Spanish language.

Un Algoritmo de Estimación de Distribuciones para resolver un problema real de programación de tareas en configuración jobshop. Un enfoque alternativo para la programación de tareas.
Ricardo Pérez, S. Jöns, Arturo Hernández, Carlos Ochoa
Revista Komputer Sapiens, 2014, aceptado para publicación
ISSN 2007-0691
En este trabajo se describe un enfoque alternativo para hacer frente a problemas de programación de tareas a través de simulación de eventos discretos enlazado con un algoritmo de estimación de distribuciones como método de optimización. Proponemos estimar la dependencia condicional entre secuencias de trabajo y la programación de herramientas para incrementar el número de trabajos terminados en un sistema de manufactura de partes automotrices. Este enfoque de optimización de simulaciones intenta encontrar una relación entre las variables de entrada a través de la estimación de la dependencia condicional entre ellas, para optimizar las variables sensibles de salida del sistema de manufactura mencionado. Dos modelos gráficos serán construidos con el fin de evitar los inconvenientes que algunos métodos heurísticos tienen para la programación de tareas. Además, el método de optimización que se utilizará será de tipo continuo con el fin de eliminar el inconveniente que se tiene con la programación de tareas de tipo discreto.

Simulation Optimization for a Flexible Jobshop Scheduling Problem Using an Estimation of Distribution Algorithm.
Ricardo Perez, Jöns Sánchez, Arturo Hernandez, Carlos Alberto Ochoa
The International Journal of Advanced Manufacturing Technology, 2014, DOI 10.1007_s00170-014-5759-x
ISSN 0268-3768
The flexible jobshop scheduling problem permits the operation of each job to be processed by more than one machine. The idea is to assign the processing sequence of operations on the machines and the assignment of operations on machines such that the system objectives can be optimized. The assignment mentioned is a difficult task to implement on real manufacturing environments because there are many assumptions to satisfy, especially when the amount of work is not constant or sufficient to keep the manufacturing process busy for a long time, causing intermittent idle times. An estimation of distribution algorithm-based approach coupled with a simulation model is developed to solve the problem and implement the solution. Using the proposed approach, the shop performance can be noticeably improved when different machines are assigned to different schedules.

An Estimation of Distribution Algorithm for solving the Knapsack problem.
Ricardo Pérez, S. Jöns, Arturo Hernández, Carlos Ochoa
International Journal of Soft Computing and Software Engineering, 2014, accepted for publishing.
ISSN 2251-7545
The knapsack problem, a NP-hard problem, has been solved by different ways during many years. However, its combinatorial nature is still

interesting for many academics. In this paper, an Estimation of Distribution Algorithm is applied for solving the Knapsack problem. KEDA for simplicity is called. It contains a probabilistic model of type chain for sampling new offsprings to solve the problem. In addition, we use a Greedy Algorithm and a Genetic Algorithm to compare the performance of the KEDA algorithm. According to the experiments, the genetic algorithm and the KEDA provide good solutions, but the performance from this evolutionary algorithm was able to give better results.