Escuela Superior Politécnica Del Litoral

Instituto de Ciencias Matemáticas

Ingeniería en Logística y Transporte

“Implementación de un Algoritmo Genético para resolver el Problema de Programación de Proyectos con Recursos Limitados”

Informe del Proyecto de Graduación

Previo a la obtención del Título de:
Ingeniero en Logística y Transporte

Presentado por:
Gabriela Narváez Molina
Ramiro Saltos Atiencia

Guayaquil – Ecuador

Año
2010

Agradecimiento

Agradecemos en primer lugar a Dios porque nos ha permitido alcanzar nuestras metas y nos ha bendecido con unos maravillosos padres y profesores.

Agradecemos de manera especial a todos los profesores que formaron parte de nuestro proceso de formación académica y profesional así como al Mat. Fernando Sandoya por el apoyo brindado en la elaboración de este proyecto y por su excelente trabajo como coordinador de nuestra carrera buscando siempre que se brinden todas las facilidades para que sus estudiantes más destacados puedan seguir adelante en su formación.

Dedicatoria

Dedicamos este trabajo a nuestros padres que siempre nos han brindado su apoyo incondicional y nos han guiado en los momentos difíciles de nuestra vida.

Tribunal de Graduación

Ing. Fabricio Echeverría Briones	Mat. Fernando Sandoya Sánchez
Director del Proyecto de Graduación	Delegado

Declaración Expresa

“La responsabilidad del contenido de este Trabajo Final de Graduación, nos corresponde exclusivamente; y el patrimonio intelectual del mismo a la Escuela Superior Politécnica del Litoral”

Ramiro Javier Saltos Atiencia	Ana Gabriela Narváez Molina

Resumen

Mayoritariamente las empresas de producción industrial y aquellas que brindan servicios de mantenimiento, cotidianamente se enfrentan a un problema de índole operativo que debe ser resuelto por las personas competentes en el menor tiempo posible. Este equipo de trabajo tiene en frente un proyecto que está compuesto por una serie de actividades y procesos que tienen una duración determinada. Las actividades del proyecto consumen los recursos escasos de la compañía y obedecen a una serie de relaciones de precedencia. El equipo de trabajo deberá elaborar una secuencia según la cual se deben ejecutar las diferentes actividades, respetando todas las limitantes consideradas, de tal manera que el tiempo de finalización del proyecto sea el menor posible. Este tipo de problema suele ser conocido en la literatura científica como el Problema de Programación de Proyectos con Recursos Limitados (RCPSP por sus siglas en inglés).

En el primer capítulo daremos a conocer el entorno cambiante sobre el cual se desenvuelven las empresas actualmente, junto con la importancia que tienen los niveles gerenciales en la búsqueda de la mejora continua y en la generación de nuevo conocimiento. Se mencionará qué es un proyecto, cuáles son los elementos que lo componen y se describirá el problema que abordará la presente investigación junto con los objetivos que se esperan alcanzar.

En el segundo capítulo se mencionarán los esquemas algorítmicos y las reglas heurísticas de prioridad usualmente utilizadas para resolver el RCPSP. Luego se abordará la teoría que soporta a los algoritmos genéticos, conoceremos sus ventajas y cuál es su analogía con los problemas de optimización.

En el tercer capítulo se detallan los criterios utilizados para el diseño del algoritmo genético que buscará resolver eficientemente el problema abordado en la presente investigación. Así mismo se menciona otros criterios usualmente utilizados.

En los capítulos siguientes se detalla la codificación del algoritmo genético desarrollado y se realiza un análisis de su eficacia. Finalmente, se presentan las conclusiones y recomendaciones de nuestra investigación.

Índice General

Índice de Tablas

Índice de Figuras

Abreviaturas

JSSP	Problema de Programación de las Órdenes de Trabajo (Job Shop Scheduling Problem)
RCPSP	Problema de Programación de Proyectos con Recursos Limitados (Resource Constrained Project Scheduling Problem)
MTS	Mayor Número de Sucesores (Most Total Succesors)
GRPW	Mayor Ranking por Peso Posicional (Greatest Rank Positional Weight)
LST	Tiempo de Inicio más Tardío (Latest Start Time)
EST	Tiempo de Inicio más Temprano (Earliest Start Time)
LFT	Tiempo de Finalización más Tardío (Late Finish Time)
AG	Algoritmo Genético

1. Antecedentes

1.1. Introducción

En la actualidad, el entorno tanto macro como microeconómico sobre el cual se desenvuelven las empresas de nuestro país está cambiando continuamente y a una gran velocidad. Esto se debe principalmente al auge de las nuevas tecnologías, la revolución de la informática y a la gran facilidad con la que ahora podemos comunicarnos con cualquier parte del mundo. La globalización está obligando a las industrias a mejorar continuamente en todos sus aspectos, los clientes son cada vez más exigentes en cuanto a los productos y servicios que demandan, no solo en el aspecto relacionado a la calidad intrínseca de los mismos, sino también desean que exista una rápida respuesta por parte de los productores en el momento que aparece su necesidad. Esto es, esperan que su proveedor pueda ofrecerles los tiempos de entrega más cortos posibles a un costo razonable y manteniendo la calidad del producto ofrecido.

Para poder cumplir con las nuevas y continuamente crecientes exigencias de los clientes, las empresas deben buscar mantener una eficiente administración en todas sus áreas funcionales, estas áreas básicamente están agrupadas en tres zonas importantes dadas por los niveles gerenciales. Estos niveles gerenciales suelen ser clasificados en tres: Alta Gerencia, Gerencia Media y Gerencia Operativa. Su relación y jerarquía se aprecia en la figura 1.

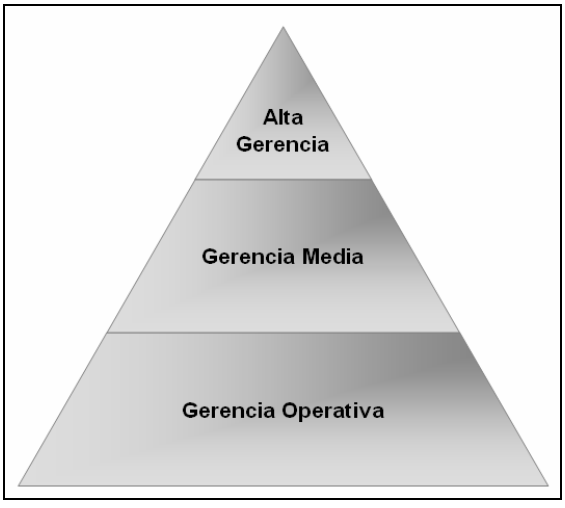

Figura 1: Niveles Gerenciales

La alta gerencia es la responsable de todos los departamentos de la organización, se encarga de fijar los objetivos de la empresa y supervisar la manera cómo los gerentes medios utilizan los recursos de la organización en la ejecución de los planes empresariales. Ellos tienen como prioridad minimizar los impactos que el entorno macroeconómico pueda provocar en la empresa. El éxito o fracaso de una compañía depende en gran medida de este nivel gerencial.

La gerencia media se encarga de supervisar a los gerentes operativos en el cumplimiento diario de sus funciones, así mismo, son responsables de buscar mejores formas de utilizar los recursos de la organización y dárselas a conocer a los gerentes operativos para mejorar su desempeño, por lo cual continuamente deben investigar y desarrollar nuevas estrategias y tácticas para lograr el cumplimiento de los objetivos planteados por la alta gerencia.

Los gerentes operativos suelen ser conocidos también como supervisores y sus principales funciones consisten en poner en marcha los planes elaborados por sus gerentes medios, velar por el adecuado cumplimiento de los mismos y controlar el personal de planta que tienen a su cargo.

En pocas palabras, la alta gerencia tiene a su cargo la planificación estratégica de la empresa por lo que sus decisiones suelen afectar a largo plazo a la compañía, la gerencia media es la encargada de producir el nuevo conocimiento en la organización y este será el que permita la mejora continua de la empresa mientras que los gerentes operativos se encargan de controlar los aspectos técnicos que se presenten diariamente para lo cual suelen disponer de manuales de procedimientos para resolverlos efectivamente.

Basado en lo mencionado con anterioridad, podemos sacar a la luz que probablemente el nivel gerencial más crítico sea el mando medio porque tiene entre sus responsabilidades tomar una gran cantidad de decisiones provocadas por acontecimientos que no son estructurados en la empresa, con esto nos referimos a situaciones completamente nuevas que requieren decisiones únicas y personalizadas acorde con el problema enfrentado. Entendamos en este momento como problema a una situación inédita que puede ser beneficiosa o perjudicial que

se presenta en la vida diaria de la compañía, y no necesariamente como un aspecto estrictamente negativo.

Específicamente, en el sector industrial de la producción y en el sector de servicios de mantenimiento, los gerentes medios deben enfrentar continuamente dos problemas de naturaleza combinatoria: el problema de secuenciación de tareas en varias máquinas conocido como Job Shop Scheduling Problem (JSSP) y el problema de programación de proyectos con recursos limitados comúnmente mencionado en la literatura como Resource Constrained Project Scheduling Problem (RCPSP).

En el desarrollo del presente trabajo trataremos únicamente el RCPSP dando a conocer en primera instancia cuáles son las herramientas que utilizan los gerentes para hacerle frente y además desarrollaremos una solución basada en las técnicas metaheurísticas que nos permita obtener mejores resultados en comparación con las herramientas convencionales.

En este punto conviene realizar algunas definiciones que permitirán comprender de mejor manera el desarrollo de este trabajo.

> ***Problemas de Clase Combinatoria:*** Los problemas de clase combinatoria son aquellos problemas de optimización cuya complejidad computacional se incrementa de manera no polinomial conforme se incrementa el número de variables de decisión y el número de restricciones que forman parte del problema combinatorio.
>
> ***Complejidad Computacional:*** Estudia las necesidades de memoria, tiempo de procesamiento y otros recursos computacionales que los algoritmos necesitan para poder resolver un problema, de esta manera es posible explicar por qué unos problemas son más difíciles de resolver que otros [11].

Metaheurísticas: En su definición original, son los métodos de solución que orquestan una interacción entre los procedimientos de mejora local y estrategias de alto nivel para crear un proceso capaz de escapar de óptimos locales y realizar una búsqueda robusta en el espacio de soluciones factibles del problema [10].

1.2. Los Proyectos

Los proyectos han existido desde el inicio de la humanidad pero solo hasta después de la segunda guerra mundial fueron reconocidos como tal y organizados dentro de una disciplina conocida como la Administración de Proyectos [7]. Los proyectos suelen ser muy diversos en cuanto a su naturaleza, tamaño y alcance, siendo ejemplos de los mismos:

- Lanzamiento del trasbordador NASA.
- Construir un bote.
- Construir un hospital.
- Remodelación de un edificio.
- Planear una fiesta o una boda.
- Organizar olimpiadas.
- Desarrollar un nuevo software.
- Obtener un título en Logística y Transporte.
- Escribir la tesis de grado.

1.2.1. ¿Qué es un proyecto?

Un proyecto es una secuencia bien definida de eventos con un principio y un final identificados y se centra en alcanzar un objetivo claro, y es responsabilidad del director del proyecto llevar a éste hasta la meta basándose en unos parámetros establecidos, tales como tiempo, costo y recursos, manteniendo siempre el nivel de calidad especificado [8].

1.2.2. Las Tareas o Actividades

Todo proyecto puede ser dividido en una serie de tareas bien definidas. Cada tarea tiene un cierto tiempo para ser completada. Algunas tareas pueden ser realizadas simultáneamente, mientras que otras necesitan ser ejecutadas en una secuencia especifica. También puede necesitar definir algunos hitos, u objetivos intermedios, que pueden ser utilizados para controlar el progreso del proyecto antes de que finalice. Además, cada tarea requiere de la disponibilidad de los recursos adecuados, tales como personas, herramientas e instalaciones [8].

Las tareas, también llamadas pasos, requeridas para completar un proyecto definen el ámbito del objetivo del proyecto. La identificación de las tareas es un paso muy importante en la planificación de un proyecto. Teniendo presente el objetivo del proyecto, se comienza por identificar los elementos o fases principales del proyecto. Una vez identificados, se comienza a descomponer cada elemento o fase. Algunas tareas se ejecutan en forma secuencial, mientras otras se pueden realizar simultáneamente. Por ejemplo, los cimientos de un edificio tienen que construirse antes que las paredes, así que la tarea de cimentación se lista antes que la tarea paredes. Por otra parte, la instalación de las conducciones de agua y electricidad pueden hacerse probablemente al mismo tiempo, por lo que no importa cual de estas tareas se liste primero [8].

El tiempo necesario para realizar una tarea es su duración. Al crear la lista de tareas, también hay que identificar las duraciones. Generalmente, para la estimación de duración de tareas se recurre a cuatro fuentes:

- Información histórica.
- Participación.
- Intuición
- Indeterminación.

La duración más fiable se obtiene a partir de datos históricos recogidos por su empresa en una consultoría reconocida, la siguiente duración más fiable se obtiene mediante la participación de alguien que ha realizado tareas similares anteriormente [8].

1.2.3. Los Recursos

Los recursos, según la teoría de la Administración de Proyectos, son los factores productivos necesarios para realizar una actividad, no se consumen con su uso ni se incorporan al producto resultante de la actividad. Al término de la tarea quedan disponibles para ser utilizados por otras tareas o en otros proyectos. Algunos ejemplos de recursos son la mano de obra, los equipos y las herramientas [9].

1.3. Descripción del Problema

En el presente trabajo abordaremos el problema particular de una empresa del sector industrial metalmecánico de la ciudad de Guayaquil. Esta empresa se dedica a la planificación, construcción e instalación, así como también al mantenimiento de una gran variedad de estructuras metálicas y equipos conforme lo deseen sus clientes. Entre los principales productos que ofrece están:

- Fabricación y montaje de equipos industriales y agropecuarios como lo son bandas transportadoras, calderos, etc.
- Fabricación y montaje de estructuras metálicas arquitectónicas como lo son estructuras para edificios, cerramientos, etc.
- Mantenimiento y reparación de maquinaria y equipos industriales como lo son sopladores, compresores de aire industrial, etc.

El problema de la empresa aparece en los altos costos de producción que tienen al momento de ejecutar una orden de trabajo y a la imposibilidad de cumplir con el tiempo de entrega prometido a sus clientes. Estos problemas ocasionan que la compañía sea incapaz de ofertar al mercado precios y fechas de entrega competitivos provocando que los márgenes de ganancia de la organización no sean muy altos, tener pocos clientes y como consecuencia final, grandes periodos en donde no tienen trabajo por lo cual lo ocupan dando mantenimiento a sus propias maquinarias para no tener el personal sin trabajo.

Esta situación se da por el bajo nivel de eficiencia que tienen las planificaciones que se realizan de los proyectos u órdenes de trabajo en el Departamento de Proyectos de la empresa. Los encargados de este departamento no disponen de las herramientas efectivas para poder elaborar planificaciones adecuadas y en un corto

periodo de tiempo. El departamento actualmente dispone de dos computadores y el software Microsoft Project así como los conocimientos adquiridos en su formación profesional para la administración de proyectos.

La manera en que se efectúa la planificación de la orden de trabajo empieza con la llegada de la misma al departamento de proyectos, la cual no necesariamente ha sido contratada por el cliente, sino que él ha convocado una licitación para seleccionar la empresa que se hará cargo del proyecto y por consiguiente nuestra empresa debe elaborar una oferta en un periodo determinado de tiempo de manera que pueda competir con las ofertas que presenten las demás industrias del sector. La mayoría de las órdenes de trabajo que llegan a la empresa requieren pasar primero por una licitación antes de ser finalmente adjudicadas a la misma, mientras que una minoría es directamente contratada sin licitación previa.

Las órdenes de trabajo que usualmente requieren licitación son las construcciones de estructuras por su largo periodo de duración y por el costo implicado en la elaboración de la misma. Las órdenes de trabajo para mantenimiento y reparación de equipos y maquinarias no suelen requerir tal licitación y el cliente contrata directamente a la empresa que sea de su agrado. Principalmente, el mayor margen de ganancias está en ganar las licitaciones para las construcciones por su alto valor de mercado por lo cual el gerente de proyectos debe ser capaz de disponer de herramientas eficaces para poder crear el conocimiento que le permita a nuestra empresa adjudicarse dichas obras. Con esto se recalca una vez más la importancia que tienen los gerentes de nivel medio en la elaboración de conocimiento y nuevas formas de administrar eficazmente los recursos de la empresa.

Una vez que la orden de trabajo ha llegado al departamento de proyectos con las especificaciones del producto solicitado por el cliente, se procede a elaborar la planificación del proyecto, siendo el proyecto en este caso la elaboración del producto solicitado. El personal del departamento empieza con la primera etapa de la planificación desmenuzando el proyecto en las tareas respectivas, estimando sus duraciones y la cantidad de recursos que se necesitarán para la realización de las mismas. Esta etapa depende en gran medida de los conocimientos conceptuales de los planificadores y en su capacidad de abstracción.

La siguiente etapa es la programación del proyecto la cual consiste en determinar en qué orden se ejecutarán las tareas del proyecto respetando las precedencias que se establecieron en la primera etapa y cómo asignaremos los recursos a las mismas considerando que tenemos una cantidad máxima determinada para cada recurso. La programación del proyecto termina cuando tenemos el calendario de ejecución de las tareas y las asignaciones de recursos realizadas. Este calendario es conocido como Diagrama de Gantt, un ejemplo típico se muestra en la figura 2.

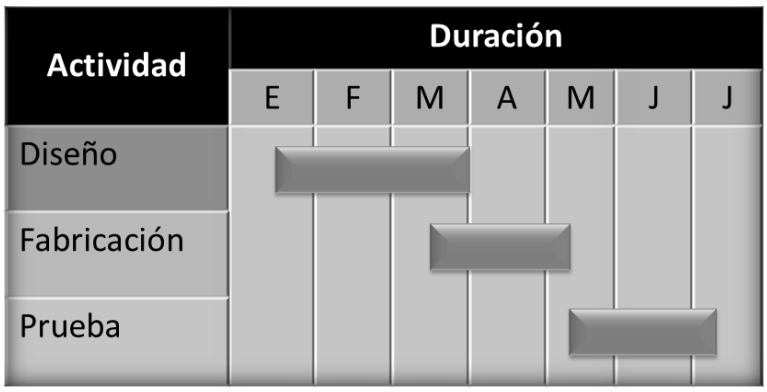

Figura 2: Diagrama de Gantt

La buena o mala elaboración de este calendario repercutirá sustancialmente en la duración total del proyecto y por consiguiente en el tiempo de entrega del producto final al cliente. En este punto es donde entra en juego el software Microsoft Project y la solución que desarrollaremos más adelante.

Microsoft Project es un software bajo licencia producido por la Corporación Microsoft siendo su principal función brindar soporte en la gestión y control de proyectos. Este programa permite al planificador ingresar las tareas que forman parte del proyecto, sus duraciones, asignar los recursos que consumen y los costos de los mismos. Elabora el diagrama de Gantt basado en las relaciones de precedencia entre las tareas y permite determinar en primera instancia una cota inferior para la fecha de terminación del proyecto.

Microsoft Project no es un software de optimización por lo que la programación que en él se efectúe no es necesariamente una solución óptima del problema. Además no considera las limitantes impuestas por la escasez de los recursos por lo que el planificador deberá revisar cuidadosamente que la programación no rebase la

capacidad de los recursos disponibles. El rebasar estas capacidades se conoce como sobreasignación.

Las sobreasignaciones de los recursos pueden ser detectadas usando el software pero será el planificador quien deberá tomar las decisiones de cómo las va a eliminar. Cuando existe una sobreasignación es muy probable que la duración del proyecto se extienda. La teoría de administración de proyectos ofrece las siguientes recomendaciones para eliminar las sobreasignaciones:

- Cambiar las asignaciones de tarea del recurso sobreasignado.
- Asignar el recurso sobreasignado para que en algunas tareas trabaje a tiempo parcial.
- Reducir la extensión de la tarea, de manera que sea posible reducir el número de horas de trabajo necesarias para completarla.
- Permitir que el recurso sobreasignado trabaje más horas aumentando las horas de trabajo del recurso.
- Resolver los conflictos de recursos automáticamente, o bien manualmente, redistribuyendo la programación.

La problemática principal de estas recomendaciones radica en que no siempre es factible aplicarlas ya sea por limitaciones de presupuesto, por las leyes vigentes en el país, por los costos implicados, etc.

La opción de redistribuir la programación es la más indicada en la mayoría de los casos, pero como podemos notar en el enunciado, el software ofrecido por Microsoft redistribuye esta planificación atrasando la ejecución de las tareas que provocan la sobreasignación del recurso. Cambiar el tiempo de inicio de las tareas que componen el proyecto sin un procedimiento adecuado no necesariamente determina la mejor solución al problema.

En el manual de usuario de Project se menciona claramente que el planificador debe volver a revisar detalladamente la programación una vez se ha efectuado la redistribución de la misma para asegurase que la nueva planificación es la deseada o es aceptable para la empresa. Así también permite que sea el planificador quien manualmente retrase las tareas pero miles de estudio en el campo de la

Investigación Operativa han demostrado que las decisiones tomadas usando el sentido común o como se suele decir, las decisiones tomadas usando papel y lápiz no siempre son las mejores.

En el presente trabajo se desarrollará una aplicación que cubra la debilidad que presenta Microsoft Project, elaborando un calendario de ejecución para el proyecto que considere todas las limitantes presentes en el problema como lo son las restricciones de precedencia y la escasez de los recursos obteniendo la fecha de entrega del proyecto terminado más próxima posible. Así mismo esta aplicación deberá ser sencilla de tal manera que el usuario solo deba ingresar todos los datos referentes al proyecto y el programa determine cuál es la mejor programación para el mismo sin necesidad de la intervención humana en el proceso de optimización.

Al mejorarse el proceso de planificación del proyecto, también se puede atender a una mayor cantidad de ofertas y obtener mejores fechas de entrega, reducir los costos tanto operativos como de recursos, mejorando la imagen y competitividad de la empresa en el mercado.

1.4. Objetivos

Los objetivos que se persiguen con la elaboración del presente proyecto son los siguientes:

- Conocer el marco que engloba la planificación y programación de proyectos en la vida de una empresa. Esto es mencionar cuáles son las situaciones que derivan en la elaboración de un proyecto.
- Diseñar un algoritmo genético que permita elaborar el calendario de ejecución de un proyecto considerando todas las restricciones del problema minimizando el tiempo de duración del proyecto.
- Que la solución aportada por el algoritmo genético diseñado si bien no es la óptima, sea la mejor posible en cuanto a la complejidad del proyecto.
- Dar a conocer la necesidad que se tiene de desarrollar programas de optimización para así mejorar la productividad de las empresas de la ciudad de Guayaquil.

2. Marco Teórico

2.1. Introducción

El problema considerado en este trabajo es el de secuenciación de un proyecto con limitación de recursos y sin posibilidad de interrupción del proceso de las actividades que lo componen una vez iniciadas, y las relaciones de precedencia entre las tareas son del tipo fin – inicio, que quiere decir que una actividad no puede iniciar mientras todas sus predecesoras no hayan concluido. También suponemos que la duración de las actividades, los requerimientos de recursos y la disponibilidad de cada tipo de recurso son cantidades no negativas, conocidas y constantes a lo largo del tiempo en el que se realiza el proyecto. El objetivo es minimizar el tiempo total de ejecución [1].

2.2. Definición Matemática del RCPSP

El Problema de Secuenciación de Proyectos (RCPSP) puede ser definido matemáticamente de la siguiente manera:

Un proyecto consiste en un conjunto $A=\{a_0,a_1,a_2,...,a_{n+1}\}$ de actividades y un conjunto $R=\{r_1,r_2,...,r_m\}$ de recursos renovables limitados. Las actividades a_0 y a_{n+1} son ficticias, tienen una duración de cero unidades de tiempo, no consumen ningún recurso y representan el inicio y fin del proyecto. Luego cada actividad a_i para $i=1,2,...,n$ tienen una duración $d_i \geq 0$ y consumen una cantidad $q_{ij} \geq 0$ de los recursos, siendo $Q_j \geq 0$ la disponibilidad máxima de cada recurso $r_j \in R$ en cada instante de tiempo. Adicionalmente existen restricciones de precedencia para las actividades del proyecto de tal manera que cada actividad a_i no puede ser iniciada mientras todas sus actividades predecesoras P_i no hayan sido finalizadas. De la misma manera se suele denotar S_i como el conjunto de actividades sucesoras de la actividad a_i para $i=0,1,...,n+1$. El objetivo consiste en encontrar un conjunto $T=\{t_0,t_1,t_2,...,t_{n+1}\}$ de tiempos de inicio de cada actividad que cumpla

con las restricciones de precedencia y disponibilidad de los recursos, y minimice el tiempo total de duración del proyecto.

2.3. Modelo Matemático del RCPSP

Sea A_t el conjunto de actividades en proceso en el tiempo t, entonces el RCPSP puede ser modelado matemáticamente de la siguiente manera:

$$Min\ z = t_{n+1} \qquad (1)$$

$$St.$$

$$t_i > t_p + d_p \quad \forall a_i \in A, \forall p \in P_i \qquad (2)$$

$$\sum_{a_i \in A_t} q_{ij} \leq Q_j \quad \forall r_j \in R, \forall t \geq 0 \qquad (3)$$

La función objetivo (1) busca minimizar el tiempo de inicio de la actividad ficticia que representa el fin del proyecto, la ecuación (2) garantiza que se cumpla las restricciones de precedencia de las actividades y la última (3) asegura que no se sobrepase la disponibilidad máxima de los recursos en cada instante de tiempo.

2.4. Grafo de Precedencia del RCPSP

Se define el grafo de precedencia $G(A,E)$ donde el conjunto de nodos A son las actividades del proyecto. El conjunto de aristas E posee un arco (a_i, a_j) si existe una relación de precedencia entre la actividad a_i y la actividad a_j para $i \neq j$. El costo de cada arco es la duración de la actividad representada en el vértice inicial [1].

El grafo G es dirigido y acíclico, y puede ser utilizado para calcular el camino más largo entre cualquier par de vértices [1].

2.5. Ejemplo del RCPSP

A continuación presentamos un ejemplo del RCPSP: Se tiene un proyecto con 12 tareas y 2 recursos r_1 y r_2 con disponibilidades de 5 y 6 unidades respectivamente. En la tabla 1 se detallan los datos relevantes a la duración de las tareas y a la cantidad de recursos consumidos por cada una de ellas mientras que en la figura 3, se muestran las relaciones de precedencia entre las actividades.

a_i	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}
d_i	0	3	4	4	2	1	3	1	3	2	2	2	1	0
q_{i1}	0	3	2	1	0	3	3	2	4	4	2	3	2	0
q_{i2}	0	2	4	4	3	2	0	3	3	0	2	0	4	0

Tabla 1: Un Proyecto con 12 Tareas y 2 Recursos

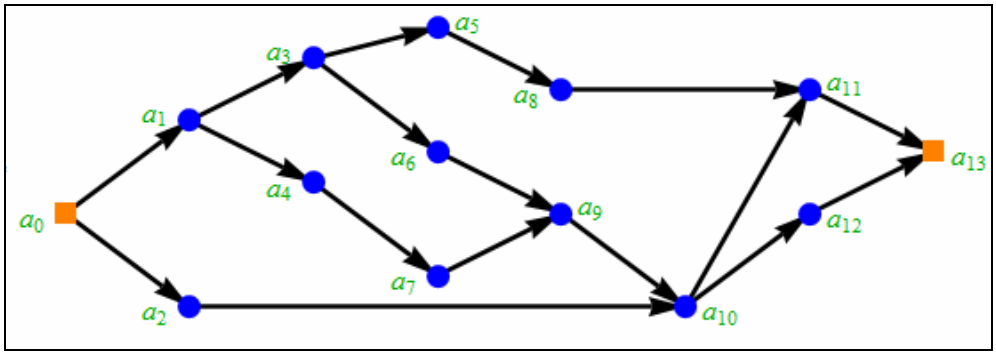

Figura 3: Diagrama de Precedencia

Una posible solución para este ejemplo está dada en la tabla 2, en la cual se indican los tiempos de inicio de cada actividad obteniéndose un tiempo total de ejecución de 24 días.

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}
0	3	3	11	11	13	13	16	19	21	23	23

Tabla 2: Tiempos de Inicio de las Actividades del Proyecto

2.6. Complejidad Computacional

De acuerdo con la teoría de la complejidad computacional, el RCPSP es uno de los más difíciles de resolver entre los problemas de optimización combinatoria. El RCPSP pertenece a la clase de problemas NP - *hard* en sentido estricto [2].

2.7. Métodos Heurísticos utilizados para la Resolución del RCPSP

2.7.1. Esquemas Algorítmicos

Kelley y Walker distinguen dos formas básicas de generar una secuencia posible, que denominan *en serie y en paralelo.* En ambos procedimientos, una vez que una actividad ha sido introducida en la secuencia, nunca es resecuenciada [1].

La secuenciación en serie comienza numerando los vértices de forma que para cada arco del grafo de precedencia, su vértice inicial tiene un número menor que su vértice final. Este esquema de numeración tiene la propiedad de que si se secuencian las actividades en el orden indicado por su número, entonces ninguna actividad aparecerá antes que ninguna de sus predecesoras. Se puede construir una secuencia posible considerando las actividades en este orden y secuenciando cada una de ellas tan pronto como las relaciones de precedencia y las restricciones sobre los recursos lo permitan. El procedimiento de numeración no es, en general, único. El orden entre las mismas puede obtenerse de forma aleatoria o mediante una función de prioridad tal como el requerimiento de recursos, duración de las actividades, etc. Cada regla de prioridad define un algoritmo en serie diferente [1].

En la secuenciación en paralelo se construye una secuencia posible procediendo hacia delante en el tiempo. En cada momento durante la construcción, se determina el conjunto de actividades que pueden ser secuenciadas de acuerdo con las restricciones de precedencia y de recursos. Este conjunto se ordena mediante una regla de prioridad y las actividades se secuencian en ese orden mientras no se supere la capacidad de los recursos y así sucesivamente. Si las actividades reciben la prioridad independientemente de la secuencia ya existente, el algoritmo se denomina estático. Caso contrario se denomina dinámico [1].

Una forma completamente diferente de obtener una secuencia posible es el *método de muestreo*. Los métodos de muestreo forman un conjunto de secuencias posibles usando técnicas de aleatorización y eligen la mejor secuencia obtenida [1]. La desventaja de utilizar este método radica en que no se dirige la búsqueda, simplemente se toma el mejor individuo de un vecindario y obtenemos óptimos locales que no necesariamente están cerca del óptimo global.

2.7.2. Reglas Heurísticas de Prioridad

Según un estudio realizado por Valdés y Tamarit en 1989, acerca de la eficiencia relativa de los principales algoritmos conocidos, se obtuvo una clasificación sustancialmente homogénea de seis reglas que pueden considerarse las más eficientes. Estas reglas tienen en común usar de forma correcta información global acerca del proyecto y difieren en el tipo de información utilizada acerca de la estructura del grafo, de las características de las actividades y de los recursos [1].

En este trabajo mencionaremos de forma breve, las cuatro mejores reglas encontradas:

2.7.2.1. MTS (Most Total Succesors)

Esta heurística elige primero aquella actividad con mayor número de sucesores, inmediatos o no, ya que el retraso de dicha actividad los retrasa a todos ellos [1].

Basados en el ejemplo anterior, la actividad con el mayor número de sucesores sería a_1 con un total de 11 sucesores de los cuales son inmediatos a_3 y a_4.

2.7.2.2. GRPW (Greatest Rank Positional Weight)

Esta regla selecciona las actividades de acuerdo con su peso posicional, obtenido sumando a la duración de la actividad, la duración de todos sus sucesores [1].

Basados en el ejemplo ya mencionado con anterioridad, el peso posicional de la actividad a_1 sería 21 mientras que el peso posicional de la actividad a_2 sería 9.

2.7.2.3. LST (Latest Start Time)

Esta regla secuencia en primer lugar aquella actividad que tiene un menor tiempo máximo de inicio (LST), obtenido mediante el método del Camino Crítico. El LST da una medida de la urgencia de empezar la actividad debido a que si ésta se secuencia en un tiempo posterior a su LST se producirá un retraso en tiempo total de ejecución del proyecto, respecto a la solución dada por el método del Camino más Largo. La máxima prioridad se dará a la actividad que produce más retraso [1].

Para poder dar un ejemplo de esta regla de prioridad primero tomaremos en consideración la información dada en la tabla 3.

a_i	d_i	EST	LST	Hol
a_0	0	0	0	0
a_1	3	0	0	0
a_2	4	0	8	8
a_3	4	3	3	0
a_4	2	3	7	4
a_5	1	7	10	3
a_6	3	7	7	0
a_7	1	5	9	4
a_8	3	8	11	3
a_9	2	10	10	0
a_{10}	2	12	12	0
a_{11}	2	14	14	0
a_{12}	1	14	15	1
a_{13}	0	16	16	0

Tabla 3: Datos Adicionales del Proyecto

En esta tabla se dan a conocer los datos referentes al tiempo de inicio más temprano (EST), el tiempo de inicio más tardío (LST) junto con la holgura de cada actividad. Esta holgura puede ser encontrada restando el EST del LST.

Definición: El Camino Crítico o Ruta Crítica de un proyecto está conformado por aquellas actividades que tienen holgura igual a cero.

Definición: El método del Camino más Largo consiste en hallar el camino más largo que exista entre los vértices que representan a las actividades ficticias de inicio y fin del proyecto.

Entonces si tuviésemos que aplicar la regla de prioridad LST entre las tareas a_5, a_6 y a_7, seleccionaríamos la actividad a_6 por tener el LST de menor valor entre las tres.

2.7.2.4. LFT (Late Finish Time)

Es similar a la regla anterior, pero se tiene en cuenta la duración de la actividad [1]. Se selecciona la actividad que tiene el menor tiempo de finalización. Para calcular el LFT de cada tarea se suma el LST con la duración de la misma.

En nuestro ejemplo, el LFT de las tareas a_5, a_6 y a_7 sería de 11, 10 y 10 respectivamente. Seleccionamos de manera indiferente entre las actividades a_6 y a_7 por tener el valor del LFT más bajo.

2.8. Algoritmos Genéticos

2.8.1. Filosofía de los Algoritmos Genéticos

Los algoritmos genéticos son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización. Están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza acorde con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Charles Darwin (1859), un esquema simplificado se muestra en la figura 4. Por imitación de este proceso, los algoritmos genéticos son capaces de ir creando soluciones para problemas del mundo real. La evolución de dichas soluciones hacia valores óptimos del problema depende en buena medida de una adecuada codificación de las mismas [3].

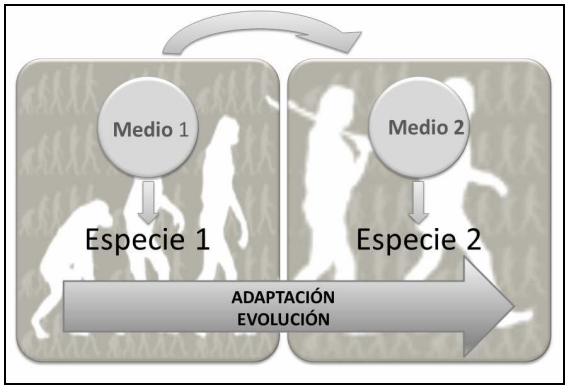

Figura 4: Proceso Evolutivo de las Especies

En la naturaleza, los individuos de una población compiten entre si en la búsqueda de recursos tales como comida, agua y refugio. Incluso los miembros de una misma especie compiten a menudo en la búsqueda de un compañero. Aquellos individuos que tienen más éxito en sobrevivir y en atraer compañeros tienen mayor probabilidad de generar un gran número de descendientes. Por el contrario, individuos poco dotados producirán un menor número de descendientes. Esto significa que los genes de los individuos mejor adaptados se propagaran en sucesivas generaciones hacia un número de individuos creciente. La combinación de buenas características provenientes de diferentes ancestros, puede a veces producir descendientes "superindividuos", cuya adaptación es mucho mayor que la de cualquiera de sus ancestros. De esta manera, las especies evolucionan logrando características cada vez mejor adaptadas al entorno en el que viven [3].

El poder de los algoritmos genéticos proviene del hecho de que se trata de una técnica robusta, y pueden tratar con éxito una gran variedad de problemas provenientes de diferentes áreas, incluyendo aquellos en los que otros métodos encuentran dificultades. Si bien no se garantiza que el algoritmo genético encuentre la solución óptima del problema, existe evidencia empírica de que se encuentran soluciones de un nivel aceptable, en un tiempo competitivo con el resto de algoritmos de optimización combinatoria [3].

2.8.2. Ventajas de un Algoritmo Genético

Las principales ventajas que poseen los algoritmos genéticos son:

- Son intrínsecamente paralelos. La mayoría de los otros algoritmos son en serie y sólo pueden explorar el espacio de soluciones en una dirección al mismo tiempo, y si la solución que descubren resulta subóptima, no se puede hacer otra cosa que abandonar todo el trabajo hecho y empezar de nuevo. Sin embargo, ya que los algoritmos genéticos tienen descendencia múltiple, pueden explorar el espacio de soluciones en múltiples direcciones a la vez. Si un camino resulta ser un callejón sin salida, pueden eliminarlo fácilmente y continuar el trabajo en avenidas más prometedoras, dándoles una mayor probabilidad en cada ejecución de encontrar la solución [3].
- Debido al paralelismo que les permite evaluar implícitamente muchos esquemas a la vez, los algoritmos genéticos funcionan particularmente bien resolviendo problemas cuyo espacio de soluciones potenciales es realmente grande – demasiado vasto para hacer una búsqueda exhaustiva en un tiempo razonable [3].
- Los algoritmos evolutivos han demostrado su efectividad al escapar de los óptimos locales y descubrir el óptimo global incluso en paisajes adaptativos muy escabrosos y complejos [3].

2.8.3. Esquema General de los Algoritmos Genéticos

Los algoritmos genéticos tienen el siguiente esquema:

- Existe una población inicial de individuos con determinadas características.
- Los individuos se cruzan entre sí dando origen a los habitantes de la siguiente generación.
- Estos habitantes heredarán las características de sus padres y si éstas son beneficiosas se irán propagando a través de las generaciones futuras, caso contrario desaparecerán por el proceso de selección natural que indica que sólo las especies más fuertes sobreviven.
- Durante el proceso de cruce puede darse la mutación del nuevo individuo, si esta mutación es mala seguramente el nuevo hijo morirá al nacer, pero si es beneficiosa se irá transmitiendo en las generaciones futuras. La mutación es un proceso mediante el cual se altera de forma significativa el código

genético del individuo otorgándole nuevas características que nadie posee en la población, pero al mismo tiempo ocurre con muy poca frecuencia. Posiblemente en una especie determinada uno de cada millón de habitantes ha mutado, el proceso se representa en la figura 5 y 6.

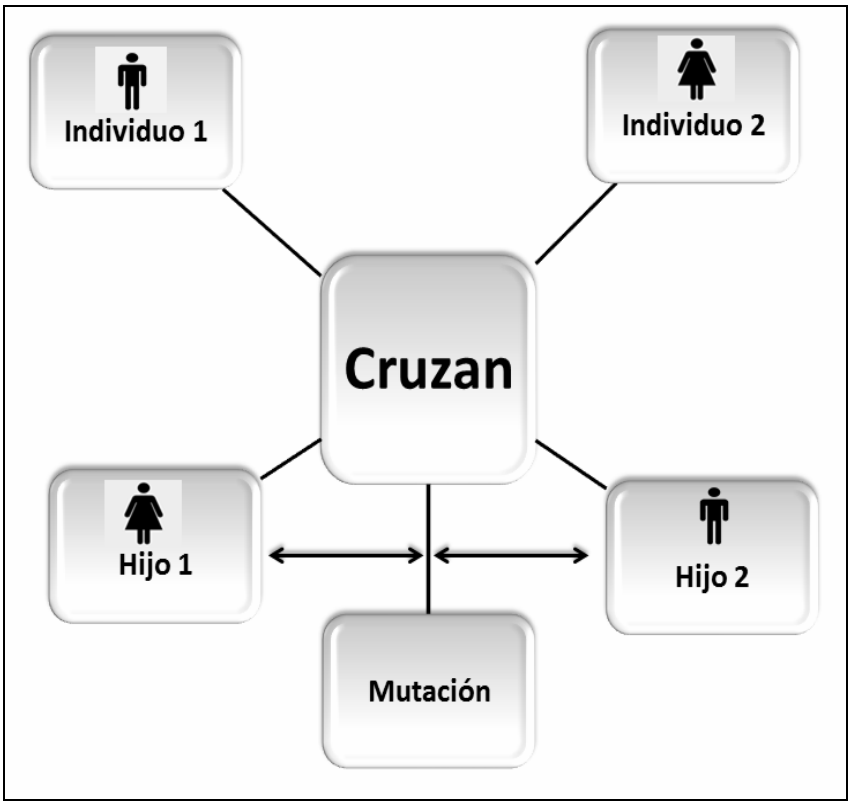

Figura 5: Esquema de un Algoritmo Genético

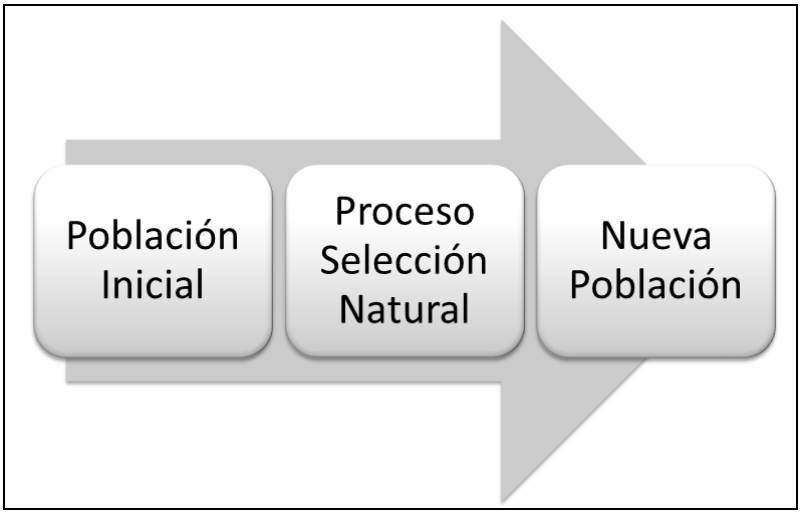

Figura 6: Proceso del Algoritmo Genético

2.8.4. Analogía con los Problemas de Optimización

Dado un problema de optimización, los algoritmos genéticos tienen la siguiente secuencia:

1. Codificar el individuo $(x_1,...,x_n) \in \Omega$ representándolo como un cromosoma, en donde Ω es el espacio de soluciones factibles del problema.
2. Seleccionar el tamaño de la población inicial. Este es un parámetro clave en el funcionamiento del algoritmo por lo que debe ser calibrado adecuadamente.
3. Crear un conjunto de individuos como población inicial conforme al parámetro establecido en el punto 2.
4. Aplicar el proceso de selección natural que consiste en, mediante algún criterio formar las parejas de individuos con las cuales se realizarán los cruces. Se pueden escoger las parejas para la reproducción estocásticamente.
5. Crear los criterios para realizar el cruce. El más usado es el punto de cruce. Mientras mayor sea el número de puntos de cruce, mayor será el tiempo requerido para la resolución del problema.
6. Cruzar las parejas de acuerdo al criterio seleccionado dando origen a la nueva población.
7. Delimitar la nueva población al mismo tamaño de la población inicial escogiendo los mejores nuevos individuos. En ocasiones se suele permitir pertenecer a la nueva población individuos con ciertas debilidades para que en la próxima generación se puedan aprovechar fortalezas que tal vez los individuos mejor adaptados no poseen.
8. Aplicar la mutación. La mutación clásica es el cambio de alelos. Se debe aplicar a menos del 1% de la población. También se puede seleccionar estocásticamente si se muta o no a cada individuo.
9. Actualizar Población Inicial = Nueva Generación.
10. Si se cumple el criterio de parada, finalizar y presentar la mejor solución encontrada. Caso contrario volver al paso 4. El criterio de parada es libre.

3. Diseño del Algoritmo Genético para el RCPSP

La primera etapa en la elaboración de un algoritmo es crear un diseño global de su estructura y funcionamiento. Por este motivo el diseño de nuestro algoritmo consta de las siguientes partes:

3.1. Codificación

La codificación de los individuos (cromosomas) de la población es un vector $(a_0, a_1, ..., a_{n+1})$ que indica el orden en el cual se van a ubicar las tareas dentro del calendario. Esta codificación nos permite dar simplicidad al algoritmo.

3.2. Elaboración del Cromosoma

Para elaborar el cromosoma utilizamos la secuenciación en serie considerando los siguientes aspectos:

1. Se secuencia una sola tarea en cada iteración.
2. Se tienen dos conjuntos disjuntos en cada iteración: el conjunto de tareas secuenciadas y las tareas elegibles. El conjunto de tareas secuenciadas contiene las actividades ya pertenecientes al cromosoma hasta dicha etapa mientras que el conjunto de tareas elegibles posee todas las posibles tareas a ser secuenciadas en la siguiente etapa. Una tarea es elegible si todos sus predecesores ya han sido secuenciados.
3. Se selecciona una actividad del conjunto de tareas elegibles según el criterio de prioridad establecido. En este trabajo se estableció que la selección de esta actividad será realizada de forma aleatoria para así poder crear una gran diversidad de individuos. Adicionalmente, en este punto se puede implementar cualquiera de las reglas de prioridad mencionadas en los apartados anteriores pero la desventaja de usarlas radica en que siempre generan la misma secuenciación dificultando el trabajo de creación de la población inicial.

Al finalizar este procedimiento se obtiene una secuenciación que garantiza el cumplimiento de todas las restricciones de precedencia. Resumiendo obtenemos el siguiente algoritmo:

$$
\begin{aligned}
&CrearCromosoma(.)\\
&\quad Secd = \{a_0\}\\
&\quad Mientras(|Secd| < n+1)\\
&\quad\quad Eleg = \{a_i \in A / P_i \subseteq Secd\}\\
&\quad\quad Seleccionar\ aleatoriamente\ una\ actividad\ a \in Eleg\\
&\quad\quad Secd = Secd \cup \{a\}\\
&\quad FinMientras\\
&FinCrearCromosoma
\end{aligned}
$$

3.3. Creación de la Población Inicial

Una parte fundamental de los algoritmos genéticos es la generación de la población inicial, a partir de la cual se originarán las siguientes generaciones. Una labor importante es calibrar el tamaño de la población debido a que éste influye muy significativamente en el tiempo de ejecución del algoritmo. Para crear la población inicial se ejecuta el siguiente procedimiento:

$$
\begin{aligned}
&CrearPoblacionInicial(./tamaño)\\
&\quad PobIni = \varnothing\\
&\quad Mientras(|PobIni| < tamaño)\\
&\quad\quad Individuo = CrearCromosoma\\
&\quad\quad PobIni = PobIni \cup \{Individuo\}\\
&\quad FinMientras\\
&FinCrearPoblacionInicial
\end{aligned}
$$

Donde el tamaño de la población es un parámetro que debe ser ingresado ya sea por el usuario o por el programador.

3.4. Fitness

Un aspecto importante en los algoritmos genéticos es la determinación del fitness o fortaleza de los individuos en la población, esta función evalúa la calidad de las soluciones. Dada la naturaleza de nuestro problema el fitness de los individuos será el tiempo total de ejecución del proyecto y mientras menor sea este valor más fuerte será el individuo.

3.5. Selección Natural

La selección natural es el proceso que permite al algoritmo escoger los mejores individuos de la población y cruzarlos de tal manera que los hijos hereden las mejores características de cada padre. Los criterios que se utilizan para este proceso suelen ser muy diversos y entre los más conocidos tenemos:

- Ordenar los individuos de mayor a menor según su fitness y formar las parejas tomando los dos primeros individuos, luego los dos siguientes y así sucesivamente. Este criterio de selección permite que individuos fuertes se reproduzcan con individuos fuertes y los débiles con los débiles.
- Ordenar los individuos de mayor a menor según su fitness y formar las parejas tomando el primero con el último, luego el segundo con el penúltimo y así sucesivamente. La ventaja de este cruce radica en que permite aprovechar ciertas características beneficiosas que seguramente tiene el individuo más débil y que los fuertes no han de poseer, permitiendo que se repita en las próximas generaciones.
- Ordenar los individuos de mayor a menor según su fitness y formar las parejas siendo el padre el primero de la lista y la madre seleccionada al azar entre los habitantes restantes. La siguiente pareja se formará tomando el segundo en el ranking (siempre y cuando no haya sido parte de la primera pareja, caso contrario se escogerá el siguiente) y su pareja al azar entre los individuos restantes y así sucesivamente.
- Formar las parejas de manera aleatoria, lo cual no es muy recomendable ya que se pierde el direccionamiento de la búsqueda y en esencia lo que constituye a la metaheurística.

Nuestro algoritmo genético está enfocado en formar las parejas utilizando el criterio mencionado en el primer punto, el cual ha sido resumido en el siguiente pseudocódigo:

$SeleccionNatural\left(PobIni\right)$
$\quad Calcular\ el\ \ fitness\ de\ los\ individuos\ en\ PobIni$
$\quad Ordenar\ PobIni\ de\ mayor\ a\ menor\ según\ el\ \ fitness$
$\quad Parejas = \left\{(1,2),(3,4),...,(p-1,p)\right\}$
$FinSeleccionNatural$

3.6. Operador de Cruce

El operador de cruce es aquella función mediante la cual se crean los nuevos individuos. Esta función recibe una pareja de cromosomas y la cruza generando los hijos de la misma. Estos hijos deben garantizar el cumplimiento de las restricciones de precedencia y para ello utilizaremos el siguiente procedimiento:

1. Se genera un número entero aleatorio u entre 1 y n el cual representará el punto de cruce entre los cromosomas.
2. El primer hijo hereda los genes del padre de izquierda a derecha hasta u, y completa su secuencia genética heredando los genes de la madre, de izquierda a derecha, que aún no formen parte del nuevo individuo.
3. De manera similar, el segundo hijo hereda los genes de la madre de izquierda a derecha hasta u, y completa su secuencia genética heredando los genes del padre, de izquierda a derecha, que aún no formen parte de la misma.

Con este procedimiento se garantiza la creación de una pareja de hijos por cada pareja de padres que cumplen todas las restricciones de precedencia.

Una variante a este cruce es la propuesta por Hartman, consistente en:

1. Generar dos números enteros aleatorios u_1 y u_2 entre 1 y n los cuales representarán el punto de cruce entre los cromosomas.

2. El primer hijo hereda los genes del padre de izquierda a derecha hasta $Min\{u_1, u_2\}$, y los de la madre, de izquierda a derecha, que aún no formen parte de la secuencia genética del hijo.
3. De manera similar, el segundo hijo hereda los genes de la madre de izquierda a derecha hasta $Max\{u_1, u_2\}$, y los del padre, de izquierda a derecha, que aún no formen parte de la secuencia genética del individuo.

Algorítmicamente el operador de cruce utilizado en nuestra implementación queda establecido de la siguiente manera:

$Cruzar(padre, madre)$
$u = aleatorioentero(1, n)$
$Para\ i = 1, u$
$hijo1(i) = padre(i)$
$hijo2(i) = madre(i)$
$FinPara$
$m = u;$
$Para\ i = 1, n$
$Si\ madre(i) \notin hijo1$
$u = u + 1$
$hijo1(u) = madre(i)$
$FinSi$
$Si\ padre(i) \notin hijo2$
$m = m + 1$
$hijo2(m) = padre(i)$
$FinSi$
$FinPara$
$FinCruzar$

3.7. Mutación

La mutación es un mecanismo que juega un papel fundamental en cualquier algoritmo genético debido a que brinda la posibilidad de salir de un entorno de búsqueda monótono y pasar a otro que probablemente tenga entre sus habitantes

al individuo óptimo. Dada la analogía que existe entre los algoritmos genéticos con la naturaleza, este factor de mutación se aplica muy rara vez siendo la cantidad de habitantes de la nueva generación que han sufrido alguna mutación menor al 1%.

Cuando apliquemos la mutación a alguno de los nuevos habitantes, debemos asegurarnos que el individuo mutado aún pertenezca a la misma especie que los demás habitantes, esto es, que aún preserve las restricciones de precedencia que forman parte del problema de optimización.

Para garantizar que esto se mantenga, elaboramos un procedimiento de mutación que consiste en intercambiar un par de alelos del nuevo individuo, siempre y cuando estos alelos no tengan relaciones de dependencia. Los pasos a seguir para lograr esto son:

1. Generar un número aleatorio u entre 1 y $n-1$.
2. Si la actividad ubicada en el puesto u de la secuencia genética no es predecesora de la actividad situada en la posición $u+1$ de la misma secuencia, entonces intercambiar ambas actividades. Caso contrario volver al paso 1.

Hay que recalcar que esta no es la única forma posible de efectuar la mutación, intercambios dobles y no consecutivos también son alternativas interesantes de analizar. El procedimiento algorítmico de los pasos arriba mencionados es:

$$
\begin{aligned}
&Mutacion\left(individuo\right)\\
&\quad u = aleatorioentero\left(1, n-1\right)\\
&\quad indmutado = individuo\\
&\quad Si\, individuo\left(u\right) \notin P_{individuo(u+1)}\\
&\qquad indmutado\left(u+1\right) = individuo\left(u\right)\\
&\qquad indmutado\left(u\right) = individuo\left(u+1\right)\\
&\quad FinSi\\
&FinMutacion
\end{aligned}
$$

3.8. Criterio de Parada

La finalización del algoritmo genético generalmente viene determinada por criterios a priori sencillos como suelen ser:

- Haber alcanzado un número máximo de generaciones.
- Haber alcanzado el tiempo máximo de resolución.
- Se ha encontrado una solución que satisface condiciones mínimas deseadas.
- Se ha llegado a una solución de tal manera que las siguientes generaciones no producen una mejora sustancial en comparación con el esfuerzo necesitado para alcanzar dicha mejora.
- Combinación de dos o más de las estrategias mencionadas anteriormente.

Para el diseño de nuestro algoritmo se ha decidido que el criterio de parada sea el alcance de un número determinado de generaciones, este número puede ser calibrado por la persona que programa el algoritmo en base a su experiencia en el tema o realizando experimentos hasta obtener un valor adecuado para este criterio. Igualmente se puede permitir al usuario establecer el número de generaciones que se van a crear. Este criterio se estableció para poder evaluar de manera simplificada la calidad de las soluciones encontradas y brindarle al usuario la posibilidad de poner fin a la ejecución del algoritmo.

La solución que devolverá el algoritmo genético, después de alcanzar el criterio de parada, será aquel individuo que durante todo el proceso evolutivo haya obtenido el fitness más bajo en concordancia con el enfoque del problema, esto es minimizar el tiempo total de ejecución del proyecto.

3.9. Creación del Calendario de Ejecución

Hasta ahora, todo los procedimientos realizados únicamente se centran en la elaboración y creación de las secuenciaciones de las tareas de tal manera que se respeten las restricciones de precedencia, pero en ningún momento se ha considerado la limitante impuesta por la escasez de los recursos, que es la característica que hace interesante y le da relevancia a la presente investigación.

El enfoque que hemos aplicado al algoritmo genético no consiste en determinar los tiempos de inicio de cada actividad sino únicamente elaborar la secuenciación de las tareas y dada esa secuenciación proceder con la asignación de los tiempos de inicio de cada tarea en ella.

Para ubicar las tareas en el calendario final o diagrama de Gantt se procede de la siguiente manera:

1. Seleccionar la actividad que corresponda según el orden establecido en la secuenciación.
2. Ubicar el tiempo de inicio de la actividad seleccionada en 1 en el primer día donde se satisfaga las restricciones de disponibilidad de recursos y todos los predecesores de dicha actividad hayan concluido.
3. Eliminar la actividad seleccionada de la secuenciación.
4. Si ya se han ubicado todas las actividades en el calendario, finalizar el procedimiento y el fitness de esta secuenciación será el tiempo de inicio de la última actividad secuenciada, que dado nuestro enfoque será la ficticia FIN, caso contrario regresar al paso 1.

Este procedimiento no afecta la estructura del algoritmo genético diseñado debido a que únicamente es utilizado para calcular el fitness de cada individuo generado, en pocas palabras, es la función objetivo del problema.

4. Implementación del Algoritmo Genético para el RCPSP

4.1. Selección del Software de Programación

En nuestro medio existen varios lenguajes de programación de carácter general como Visual Basic, C, C++ y C#. Estos lenguajes son conocidos como de bajo nivel por no ser dedicados para resolver un determinado tipo de problema. Tienen la ventaja de tener un tiempo de ejecución bastante bajo pero así mismo por su carácter general no poseen los beneficios que otro lenguaje más especializado brindaría. En efecto, al programar en estos lenguajes de bajo nivel, el programador deberá escribir cada una de las funciones que vaya a utilizar desde la más elemental como ordenar los elementos de un vector hasta la más compleja como elaborar un gráfico de un determinado problema. Adicionalmente, algunos de estos lenguajes son muy sensibles a los cambios aleatorios provocando que, si el programador no tiene cuidado al momento de generar números aleatorios, se creen resultados correlacionados o muy apartados de la realidad.

Por nuestra experiencia empírica, al programar en C++ se tuvieron problemas con la creación de los números aleatorios dado que siempre escoge la misma semilla y genera los mismos números. Para poder corregir este error era necesario incluir dentro los parámetros iniciales de programación el uso de un comando que era parte de una de las librerías de C++. Otro aspecto importante a considerar fue la necesidad de números aleatorios enteros positivos, los cuales no son generados directamente por el programa sino que se debe declarar la variable donde se guarda el número como *integer* de tal manera que al crearse el número aleatorio y guardarse en la misma se trunque por no aceptarse la parte decimal. Este procedimiento lo consideramos incorrecto porque, si por ejemplo se desea números enteros entre 0 y 2, C++ puede generar varias veces números menores a 1 y al truncarlos se estarían produciendo demasiados ceros.

Para la codificación del algoritmo genético se decidió usar el software *Mathematica 7.0* elaborado por la empresa Wolfram Research Inc. *Mathematica 7.0* es un software especializado y general al mismo tiempo, basado en el lenguaje C. En él se han incluido una gran diversidad de funciones desde las más simples como

calcular una raíz cuadrada hasta las más complejas como elaborar animaciones, interfaces de usuario, demostraciones y resolver ecuaciones diferenciales.

La potencia de *Mathematica 7.0* no sólo radica en la gran librería de funciones que posee, sino también en la simplicidad de su interfaz de usuario siendo esta una ventana en blanco simulando una hoja de papel sobre la cual el usuario puede, utilizando las paletas que incluye el programa, escribir simbólicamente sobre la misma como si se tratase de un cuaderno. Por este motivo los archivos creados por *Mathematica 7.0* son considerados *notebooks* y guardados con extensión *.nb*.

Adicionalmente, la sintaxis de programación es muy sencilla y fácil de aprender permitiendo que un usuario ordenado pueda adaptarse rápidamente al programa y preferir desde ese momento en adelante siempre crear sus programas en este software.

Finalmente, por los motivos arriba mencionados y a su gran rapidez en tiempo de ejecución (muy insignificante frente a los lenguajes de bajo nivel) implementamos nuestro algoritmo genético en este software.

4.2. Estructura de los Datos

Una de las capacidades de *Mathematica 7.0* es que permite la importación de archivos externos, de tal manera que el usuario no necesita ingresarlos manualmente al programa. Puede tenerlos guardados en una base de datos, en un archivo en Excel o en documentos planos (*.txt*), y nuestro software los importará para ser utilizados en la ejecución de la solución.

Para ello, es muy importante definir una estructura adecuada de los datos que el programa importará con el fin de evitar errores y establecer un estándar para que los usuarios de nuestra solución puedan manejarla sin problemas.

En la tabla 4 se muestra la arquitectura diseñada para el ingreso de los datos, basados en la misma, el programa extraerá los datos y ejecutará el algoritmo genético.

a_i	d_i	q_{i1}	...	q_{im}	$\#suc_i$	suc_{i1}	...
a_0	0	0	...	0	$\#suc_0$	suc_{01}	...
a_1	d_1	q_{11}	...	q_{1m}	$\#suc_1$	suc_{11}	...
⋮	⋮	⋮	...	⋮	⋮	⋮	...
a_{n+1}	0	0	...	0	0		

Tabla 4: Estructura de Ingreso de los Datos

En la columna 1 se mencionan las tareas que forman parte del proyecto incluyendo las ficticias “Inicio” y “Fin”. En la segunda columna se hace referencia a la duración de cada una de las tareas. Desde la columna 3 hasta la columna $3+m$ se ingresan los valores correspondientes a la cantidad de recurso consumido por cada tarea. En la columna $m+4$ se ingresa el número de sucesores que tiene cada actividad del proyecto y finalmente, desde la columna $m+5$ en adelante, se ingresan las tareas sucesoras de cada actividad. Si alguna actividad no tiene sucesoras se deja el espacio vacío. Recordemos que m es la cantidad de recursos renovables limitados que se utilizan en el proyecto.

La disponibilidad máxima por cada instante de tiempo de los recursos del proyecto será almacenada en otro archivo externo (*.txt*) siguiendo el esquema presentado en la tabla 5.

r_1	...	r_m
Q_1	...	Q_m

Tabla 5: Esquema de la Disponibilidad de los Recursos

4.3. Organización de la Solución

En este apartado mencionaremos como está organizada la programación de nuestro algoritmo. Para la escritura del mismo hemos seguido el formato que se utiliza cuando se programa en el lenguaje general C++, es decir, primero hemos declarado y escrito todas las funciones que serán utilizadas en la ejecución del algoritmo y después se ha programado el cuerpo de la solución, el programa principal.

4.3.1. Funciones

En todo código de programación las funciones juegan un papel importante debido a que permiten al programador evitar reescribir secciones del código que tienen una estructura similar y son utilizadas varias veces en distintos puntos del programa.

Nuestra solución está compuesta por las funciones que detallaremos a continuación.

4.3.1.1. Función Sucesores

La función sucesores está diseñada para recibir dos argumentos: el primero es el archivo donde están los datos relevantes a las características de las actividades y el segundo es la actividad de la cual deseamos conocer sus sucesores. Su estructura es la siguiente:

$$Sucesores\left[data, act\right]$$

La función devuelve una lista (conjunto) con los sucesores de la actividad ingresada.

4.3.1.2. Función Matriz de Precedencia

Esta función nos permite crear la matriz de precedencia de las actividades. La matriz de precedencia es una matriz binaria que pertenece al espacio de las matrices de orden $(n+2)\times(n+2)$ en donde el valor de 1 en la posición (i, j) de la matriz indica que la actividad j es predecesora de la actividad i. Su estructura es la siguiente:

$$CrearMatrizPrecedencia\left[data\right]$$

La función recibe como argumento los datos del proyecto y devuelve la matriz de precedencia de las actividades.

4.3.1.3. Función Predecesores

La función predecesores es una de las más importantes en nuestro algoritmo debido a que nos permitirá verificar que se cumplan las restricciones de precedencia del problema. Ésta recibe como argumentos la matriz de precedencia y la actividad de la cual deseamos conocer sus predecesores. Su estructura es:

$$Predecesores[matriz, act]$$

La función devuelve, de la misma forma que la función sucesores, un conjunto con las actividades predecesoras de la tarea ingresada como argumento.

4.3.1.4. Función de Selección de la Actividad

Con esta función escogemos, entre el conjunto de tareas candidatas a ser secuenciadas, la que ingresará a la secuenciación parcial existente hasta dicho punto.

En esta parte del programa es donde el investigador curioso puede implementar las reglas de prioridad mencionadas en el capítulo 4 y sacar sus propias conclusiones acerca de la eficiencia de las mismas.

Para efecto de nuestro trabajo, el criterio utilizado en esta función es de escoger aleatoriamente cualquier actividad que cumpla con las restricciones de precedencia del problema. La estructura de la misma es:

$$SeleccionActividad[lista]$$

Recibe como argumento el conjunto de tareas elegibles y devuelve la actividad que será insertada en la secuenciación existente.

4.3.1.5. Función de Actividades Elegibles

La función de actividades elegibles recibe como argumento el conjunto de actividades secuenciadas hasta la etapa en que se invoca la función y la matriz de precedencia. Su principal objetivo es el de crear el conjunto de actividades

candidatas a ser seleccionadas en la iteración actual y enviar esta lista a la función de selección de actividades. La sintaxis de la misma es:

$$ActividadesElegibles\left[lista, matriz\right]$$

La función devuelve un conjunto con las actividades elegibles que cumplen todas las restricciones de precedencia del problema.

4.3.1.6. Función Crear Cromosoma

El objetivo de esta función es el de crear una secuencia en serie de las actividades, esto es, determinar el orden según el cual serán ingresadas en el calendario de ejecución o diagrama de Gantt.

Para poder realizar esta tarea, la función recibe como argumento la matriz de precedencia y llama a las funciones encargadas de seleccionar las actividades y de crear el conjunto de tareas elegibles. Su estructura es la siguiente:

$$CrearCromosoma\left[matriz\right]$$

La función devuelve un vector $X \in R^{n+2}$ en el cual se detalla el orden según el cual se realizarán las tareas. El vector tiene esta forma para el caso de un proyecto con 5 tareas:

$$\left(a_0, a_3, a_1, a_5, a_4, a_2, a_6\right)$$

4.3.1.7. Función de Creación de la Población Inicial

Con esta función creamos un conjunto de cromosomas que conformarán los habitantes del primer vecindario de búsqueda a explorar. La función recibe como argumentos la matriz de precedencia y la cantidad de habitantes que tendrá la población. Su sintaxis es como sigue:

$$CrearPoblacionInicial\left[matriz, tamaño\right]$$

La función retorna un conjunto de la forma:

$$\{Cr_1, Cr_2, ..., Cr_{tamaño}\}$$

En donde cada elemento es una secuencia creada por la función *CrearCromosoma*

4.3.1.8. Función de Cruce

La función de cruce está diseñada para realizar el apareamiento entre dos individuos de la población siguiendo el esquema mencionado en el capítulo 5 de este trabajo. Recibe como argumento al padre y a la madre devolviendo una pareja de hijos que siguen respetando las restricciones de precedencia del problema. Su sintaxis es como sigue:

$$CruzarPadres[\,padre, madre]$$

4.3.1.9. Función de Mutación

Como se mencionó con anterioridad, la mutación juega un papel fundamental en los algoritmos genéticos al permitir saltarnos de un espacio de búsqueda a otro con facilidad. Nuestra función para realizar su tarea recibe como argumentos un individuo, el cual será mutado, y la matriz de precedencia. Su estructura es la siguiente:

$$Mutacion[cromosoma, matriz]$$

La función de mutación devuelve un individuo que ha sufrido un intercambio en la posición de un par de sus genes, al mismo tiempo que garantiza que este cambio no violó las restricciones de precedencia.

4.3.1.10. Función de Fitness

La función de fitness nos permite calcular la fortaleza de cada individuo en la población. Para ello recibe como argumento el calendario de ejecución de las tareas del proyecto y devuelve un valor que indica el tiempo total de ejecución del mismo. Su sintaxis es la siguiente:

$$Fitness[scheduling]$$

4.3.1.11. Función de Consumo de Recursos

Esta función permite al algoritmo extraer de los datos la información relevante a la cantidad de recursos consumidos por una determinada actividad. Su sintaxis es:

$$ConsumoRecursos[data, act]$$

La función devuelve una lista con los valores que requiere de cada recurso la tarea ingresada como argumento.

4.3.1.12. Función de Selección Natural

Recibe como argumentos la población o generación y las fortalezas de los habitantes que la componen. Su principal objetivo es ordenar los elementos de la población de mayor a menor según su fitness para así proceder con la formación de las parejas. Su estructura es:

$$SeleccionNatural[poblacion, fortalezas]$$

La función retorna la población con sus individuos ordenados desde el más fuerte al más débil.

4.3.1.13. Función de Scheduling

Es una de las más importantes funciones del programa, sin ella no podemos conocer el fitness de los cromosomas generados ni conocer cuando dar a inicio a las actividades del proyecto. La función recibe como argumentos los datos relativos

al proyecto, la lista con las disponibilidades máximas de los recursos y el cromosoma a ser planificado. La sintaxis de la misma es como sigue:

$$Scheduling\left[data, lista, cromosoma\right]$$

La función determina los tiempos de inicio de cada actividad en el orden especificado por el cromosoma y en esta ocasión abarca tanto las restricciones de precedencia como las de disponibilidad de los recursos.

Finalmente, devuelve una tabla con la siguiente estructura:

a_i	t_i	d_i

Tabla 6: Estructura de la Respuesta

En donde en la primera columna se mencionan las tareas del proyecto, en la segunda se indican los tiempos de inicio de las actividades y en la tercera la duración de las mismas. El tiempo total de ejecución del proyecto estará dado por el tiempo de inicio de la última actividad del proyecto.

4.3.1.14. Función Graficar Calendario

Esta simplemente es una función que cumple un papel estético en el trabajo, la cual recibe la tabla generada por la función de *scheduling* y nos devuelve el diagrama de Gantt del proyecto. Su sintaxis es:

$$GraficarCalendario\left[scheduling\right]$$

El diagrama de Gantt generado por esta función, para un proyecto con 30 tareas se puede apreciar en la figura 7.

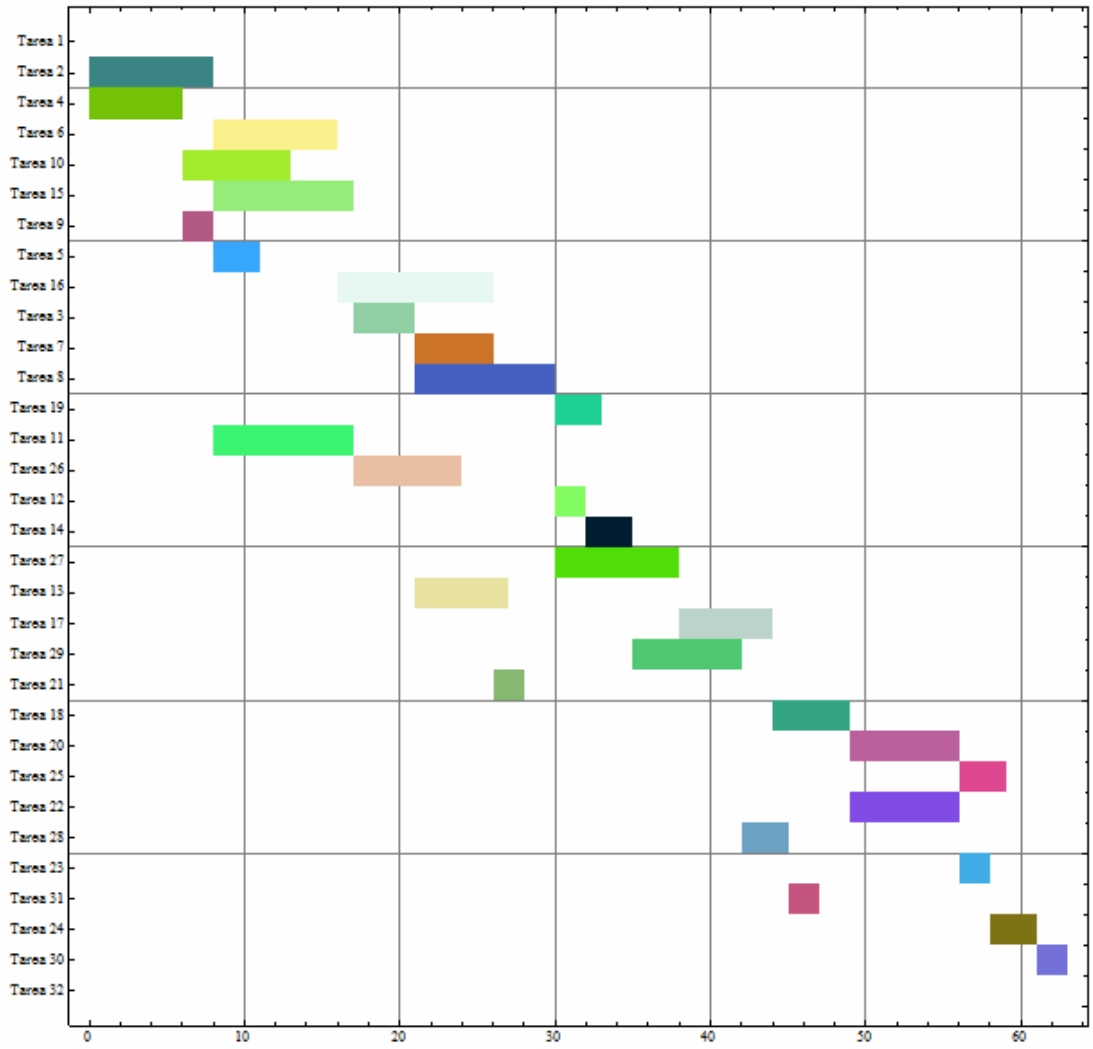

Figura 7: Diagrama de Gantt de un Proyecto con 30 Tareas

4.3.2. Programa Principal

El código principal del programa ejecuta el algoritmo genético en esencia, para ello sigue la siguiente secuencia:

1. Importa los datos desde los archivos externos.
2. Crea la matriz de precedencia de las actividades.
3. Inicializa la población inicial.
4. Se inicia el proceso de selección natural según el criterio del más fuerte.
5. Se guarda el mejor habitante.
6. Se inicia con el proceso evolutivo.
7. Se ejecuta el operador de cruce.

8. Se ejecuta la mutación.
9. Se calculan las fortalezas de la nueva generación y se guarda el mejor habitante.
10. Si el nuevo mejor habitante es mejor que el anterior, se reemplaza el anterior por el nuevo.
11. Se actualiza la población inicial con la nueva generación.
12. Se repite el proceso evolutivo hasta cumplir con el número de generaciones deseadas.

El pseudocódigo del algoritmo genético es el siguiente:

```
Inicio
    Importar(datos)
    MatrizPrecedencia = CrearMatrizPrecedencia
    PobIni = CrearPoblacionInicial
    PobIni = SeleccionNatural(PobIni)
    MejorIndividuo = GuardarMejorIndividuo(PobIni)
    Para i = 1 hasta #generaciones
        Hijos = Cruzar(padre,madre)
        Generacion = Generacion ∪ Hijos
        Mutacion(Habitante)
        Generacion = SeleccionNatural(Generacion)
        NMejorIndividuo = GuardarNMejorIndividuo(Generacion)
        Si NMejorIndividuo es mejor que MejorIndividuo
            MejorIndividuo = NMejorIndividuo
        FinSi
        PobIni = Generacion
    FinPara
    Mostrar MejorIndividuo
    Graficar(MejorIndividuo)
Fin
```

5. Análisis del Algoritmo y Resultados

Todo algoritmo que se diseña para resolver un problema en particular debe pasar por una serie de experimentos que permitan evaluar la calidad de los resultados que genera. Para realizar estos experimentos se suelen utilizar instancias particulares del problema que se desea resolver. Una instancia es un ejemplo específico del problema a tratar.

Para evaluar nuestro algoritmo genético obtuvimos las instancias de *PSPLib*, la cual es una librería que cuenta con una gran variedad de problemas para diferentes variantes del RCPSP. Estos problemas particulares o instancias de prueba fueron generados utilizando el software *ProGen*. Esta aplicación computacional fue desarrollada por Kolish, Sprecher y Drexl en los años 90 del siglo pasado. Actualmente la librería está en constante actualización y los usuarios pueden crear sus propias instancias utilizando la aplicación arriba mencionada. Esta información se puede encontrar con mayores detalles en [5].

Para calibrar los parámetros del algoritmo genético se corrieron dos instancias de prueba obtenidas de la librería mencionada en [5]. Desde la tabla 7 hasta la 12 se muestran los resultados obtenidos relevantes a la solución encontrada, el gap relativo y el tiempo de ejecución para diferentes parámetros referentes al tamaño de la población inicial y al número de generaciones.

n = 30		**Tamaño de la Población Inicial**				
	Sol	**20**	**40**	**60**	**80**	**100**
Generaciones	**25**	51	45	43	45	43
	50	45	46	45	45	45
	75	51	50	47	45	43
	100	53	46	46	43	45

Tabla 7: Resultados para la Instancia 1

n = 30		Tamaño de la Población Inicial				
	Gap	20	40	60	80	100
Generaciones	25	0,186	0,0465	0	0,0465	0
	50	0,0465	0,0698	0,0465	0,0465	0,0465
	75	0,186	0,1628	0,093	0,0465	0
	100	0,2326	0,0698	0,0698	0	0,0465

Tabla 8: GAP Relativo de la Instancia 1

n = 30		Tamaño de la Población Inicial				
	Time	20	40	60	80	100
Generaciones	25	38,51	75,7	101,76	149,29	196,64
	50	62,47	127,73	140,71	200,8	222,74
	75	84,4	176,45	253,17	190,89	320,75
	100	69,12	151,18	286,93	339,77	629,32

Tabla 9: Tiempo de Ejecución de la Instancia 1 en Segundos

n = 30		Tamaño de la Población Inicial				
	Sol	20	40	60	80	100
Generaciones	25	48	47	50	50	47
	50	50	48	47	47	48
	75	53	48	47	48	47
	100	47	48	47	47	48

Tabla 10: Resultados para la Instancia 2

n = 30		Tamaño de la Población Inicial				
	Gap	20	40	60	80	100
Generaciones	25	0,0213	0	0,0638	0,0638	0
	50	0,0638	0,0213	0	0	0,0213
	75	0,1277	0,0213	0	0,0213	0
	100	0	0,0213	0	0	0,0213

Tabla 11: Gap Relativo para la Instancia 2

n = 30		Tamaño de la Población Inicial				
	Time	20	40	60	80	100
Generaciones	25	31,54	39,34	84,14	103,57	183,43
	50	51,25	116,5	179,98	202,05	259,24
	75	74,63	145,37	224,18	305,56	454,02
	100	84,63	182,39	291,51	497,03	556,95

Tabla 12: Tiempo de Ejecución para la Instancia 2 en Segundos

Podemos observar que para instancias de 30 tareas el algoritmo converge sin mayores inconvenientes al óptimo calibrando 50 generaciones y 60 habitantes para la población inicial.

Los resultados obtenidos tras ejecutar diez instancias con 60 tareas y parámetros de 50 habitantes en la población inicial y 50 generaciones como criterio de parada se muestran en la tabla 13.

Instancia	Sol AG	Opt	R-Gap	Time
1	77	77	0	660,55
2	75	68	0,102941	676,78
3	73	68	0,073529	736,16
4	93	91	0,021978	674,46
5	78	73	0,068493	723,82
6	66	66	0	782,22
7	75	72	0,041667	644,92
8	82	75	0,093333	687,92
9	85	85	0	674,76
10	82	80	0,025	681,95
Promedio			**0,042694**	**694,354**

Tabla 13: Resultados para 10 Instancias con N = 60

Adicionalmente se ejecutaron pruebas para un número mayor de instancias con 30 tareas en el proyecto calibrando una población inicial de 50 habitantes y el criterio de parada con 50 generaciones. Los resultados obtenidos se muestran en la tabla 14.

Instancia	Sol AG	Opt	R-Gap	Time (s)
1	46	43	0,069767	130,18
2	51	47	0,085106	113,78
3	47	47	0	88,32
4	62	62	0	148,18
5	39	39	0	124,52
6	48	48	0	123,8
7	60	60	0	128,54
8	54	53	0,018868	105,03
9	49	49	0	129,98
10	45	45	0	109,27
11	38	38	0	105,64
12	51	51	0	146,56
13	43	43	0	92,18
14	43	43	0	124,83
15	51	51	0	121,15
16	47	47	0	116,34
17	48	48	0	116,69
18	54	54	0	117,27
19	65	65	0	123,11
20	59	59	0	137,85
21	49	49	0	106,49
22	60	60	0	119,25
23	47	47	0	119,51
24	57	57	0	119,55
25	59	59	0	121,39
26	45	45	0	115,86
27	56	56	0	132,14
28	55	55	0	121,23
29	38	38	0	111,28
30	48	48	0	113,98
Promedio			**0,005791**	**119,464**

Tabla 14: Resultados para 30 Instancias con N = 30

Los resultados obtenidos nos permiten sacar a la luz que al algoritmo funciona eficazmente con instancias del problema con un número pequeño de tareas en la composición del proyecto alcanzado las soluciones óptimas en la mayoría de los casos y en un tiempo de cómputo relativamente corto.

Con las instancias de 60 tareas podemos notar que el algoritmo ya no alcanza mayoritariamente el óptimo y que el tiempo de cómputo crece rápidamente conforme a la dificultad del problema.

6. Conclusiones y Recomendaciones

En este trabajo se desarrolló un algoritmo genético para resolver el problema de secuenciación de proyectos con recursos limitados y con relaciones de precedencia del tipo fin – inicio. A través de los experimentos se comprobó que nuestro algoritmo resultó ser una herramienta adaptativa muy eficaz para resolver este tipo de problemas hallando las soluciones óptimas para problemas de tamaño relativamente pequeños en un intervalo corto de tiempo.

Para instancias de mayor tamaño el algoritmo encuentra buenas soluciones con un gap no mayor al 20% con un tiempo de ejecución aceptable en la práctica, esto es, no mayor a dos horas.

Por medio de los experimentos se constató que la convergencia al óptimo de las instancias depende en cierta medida de qué tan buenos o malos eran los fitness de los individuos que conformaban la población inicial, esto es, si el mejor individuo en la población inicial ya era la solución óptima del problema, entonces el algoritmo con toda certeza llegaría al óptimo debido a que se decidió guardar el mejor individuo que naciera en cada generación reemplazando al anterior si éste no era mejor que la nueva secuencia generada.

En el caso que no estuviera la secuencia óptima en la población inicial, el algoritmo llegaría a generarla dependiendo de la cantidad de generaciones que estén programadas así mismo como de los fitness de los individuos de la población inicial y el número de tareas del proyecto.

La adecuada programación de las tareas de un proyecto es de vital importancia en las áreas operativas de una empresa debido a que una mala planificación puede acarrear grandes pérdidas económicas para la compañía. Gracias a la tecnología existente se pueden evitar estos problemas porque la planificación de los proyectos ya no se realizarán usando el sentido común del planificador que hasta ahora lo hacía usando papel y lápiz, o en su defecto un software no optimizador como Microsoft Project, con el cual usando una instancia de 30 tareas nos demoramos

media hora en sólo ingresar las precedencias de las tareas y no se consideran las limitaciones impuestas por la escasez de los recursos.

Por lo mencionado anteriormente, salta a la luz la necesidad de desarrollar aplicaciones eficientes y fáciles de usar de manera que la planificación elaborada por el software sea sofisticada y optimice los recursos de la empresa. Esto es minimice el gasto que se incurre en la utilización de los recursos y el tiempo que se necesitaría para terminar el proyecto, lo cual mejora el nivel de servicio ofrecido por la empresa a sus clientes.

Finalmente, nuestro algoritmo genético puede ser utilizado como motor para la creación de este tipo de soluciones empresariales.

Para futuros trabajos de investigación dentro de este ámbito damos las siguientes recomendaciones:

- Cambiar el operador de cruce de uno a más puntos para mejorar las soluciones.
- Sugerimos usar dos puntos fijos para instancias de 50 a 70 tareas y, 3 puntos fijos o más para instancias mayores.
- Recordar que el operador de cruce que se diseñe debe mantener las relaciones de precedencia de las tareas.
- Modificar el operador de mutación incrementando la probabilidad de que un individuo sea sujeto de una alteración en su secuencia genética.

7. Referencias Bibliográficas

[1] Valdés, R. y Tamarit, J., *Algoritmos Heurísticos Deterministas y Aleatorios en Secuenciación de Proyectos con Recursos Limitados*, Universidad de Valencia, 1989.

[2] Artigues, C. et al., "Resource Constrained Project Scheduling", Iste and Wiley, Primera Edición, 2008, pp. 23 – 24.

[3] Londoño, N., *Algoritmos Genéticos*, Universidad Nacional de Colombia, 2006.

[4] Reeves, C. y Rowe, J., "Genetic Algorithms Principles and Perspectives", Kluwer Academics Publishers, Primera Edición, 2003.

[5] PSP Lib., http://129.187.106.231/psplib/.

[6] Wolfram Research Inc., www.wolfram.com.

[7] Vega, J. "Claves para Administrar Proyectos", Microsoft Corporation, www.microsoft.com.

[8] Microsoft Corporation, "Planificación y Gestión de Proyectos con Microsoft Project", pp. 3 – 5.

[9] CEPAL, Programación de Proyectos, pp. 57.

[10] Glover, F. et al., "Handbook of Metaheuristics", Kluwer Academic Publishers, Primera Edición, 2003, pp. XI.

[11]. Wikipedia, "Teoría de la Computación", www.wikipedia.org.