UNIVERSIDAD TECNOLÓGICA METROPOLITANA
FACULTAD DE INGENIERÍA
ESCUELA DE INFORMÁTICA

HEURÍSTICAS Y COMPLEJIDAD

TRABAJO DE TITULACIÓN PARA OPTAR POR EL TÍTULO DE
INGENIERO EN INFORMÁTICA

AUTOR:
FICA PARDO, NICOLÁS FRANCISCO

PROFESOR GUÍA:
PINCHEIRA CONEJEROS, HECTOR MANUEL

SANTIAGO - CHILE
2021

AUTORIZACIÓN PARA LA REPRODUCCIÓN

Nombre del(os) alumno(s) .....
Rut .....
Dirección .....
E-mail .....
Teléfono .....
Titulo del trabajo .....
Escuela .....
Carrera o programa .....
Titulo al que opta .....

a) Este trabajo de titulación no puede reproducirse o transmitirse bajo ninguna forma o por ningún medio o procedimiento, sin permiso escrito del(os) autor(es), exceptuando la cita bibliográfica, resumen y metadatos que acreditan al trabajo y a su(s) autor(es). Es decir, que la Universidad no puede publicar y/o reproducir el trabajo de titulación o parte de su contenido, pero sí dar a conocer su existencia.

Fecha : _____ Firma : _____

b) Se autoriza la reproducción total o parcial de este trabajo de titulación, con fines académicos, por cualquier medio o procedimiento, incluyendo la cita bibliográfica que acredita al trabajo y a su autor. Es decir, que la Universidad está autorizada para dar a conocer, reproducir y/o publicar el trabajo de titulación sin perjuicio del derecho de autor.

En consideración con lo anterior, marque con una X:

	Inmediata
	A partir de la siguiente fecha: _____ (Mes/Año)

Fecha : _____ Firma : _____

Esta autorización se otorga bajo el marco de la ley N°17.336 sobre Propiedad Intelectual.

NOTA OBTENIDA: _____

_____

Firma y timbre autoridad responsable.

Índice general

Índice de tablas

Índice de figuras

RESUMEN

Palabras clave

Complejidad Computacional (CC), Complejidad algorítmica (CA), Heurística (H) , Algoritmos Heurísticos (AH), Desarrollo de software (DS), Recocido simulado, Algoritmo de colonias de Hormigas.

ABSTRACT

Keywords

Computational Complexity (CC), algorithmic Complexity (AC), Heuristics (H), Heuristic Algorithms (AH), Software Development (DS), Simulated Annealing, Ant Colony Algorithm.

1. PRESENTACIÓN

1.1 Descripción del proyecto

El proyecto consiste en una investigación aplicada, que en su primera parte investigativa se centrará en el estudio de la heurística y la teoría de la complejidad, desde los fundamentos necesarios para la resolución de problemas de optimización en ingeniería, hasta un estudio de los tipos de problemas y su complejidad. Esto, para dejar en evidencia la relevancia, causa y uso de los modelos heurísticos para la resolución en optimización, pasando por un estudio de algunos algoritmos heurísticos utilizados y sus tipos, desde sus orígenes, fundamentos matemáticos/Computacionales.

En la segunda parte y de aplicación del proyecto, se aplicarán los conceptos estudiados para el desarrollo de una aplicación, que permitirá hacer uso de los algoritmos estudiados y sus tipos, sobre un esquema definido y limitado a cierto contexto dado por el tipo, problemas y variaciones estudiadas de los algoritmos, para la resolución de problemas y la comparación de resultados en este tipo de modelos. La aplicación tiene un fin didáctico, en el sentido de que permitirá estudiar los algoritmos heurísticos, siendo una aplicación interactiva.

1.2 Objetivos

1.2.1 Objetivo general

Desarrollar una aplicación, con fines didácticos, que mediante un conjunto de implementaciones simples, muestre la caracterización de los modelos heurísticos, los fundamentos de la teoría de la complejidad y la relación entre ellos.

1.2.2 Objetivos específicos

- Investigar sobre la teoría de la complejidad, los tipos de complejidad y su relación con las problemáticas.
- Investigar sobre los métodos Heurísticos en la resolución de problemas.
- Estudiar sobre los principales algoritmos heurísticos y sus tipos.
- Determinar objetivos específicos de la aplicación a desarrollar.
- Determinar requerimientos de la aplicación.
- Investigar sobre herramientas auxiliares que permitan el desarrollo de la aplicación.
- Estudiar los algoritmos heurísticos en un lenguaje que permita el desarrollo.

1.3 Alcances y limitaciones

1.3.1 Alcances

- Conocer sobre heurísticas, su relevancia y aplicación en informática y en general en la resolución de problemas.
- Conocer los principales algoritmos heurísticos, problemáticas que estos resuelven y su diferencia con los no heurísticos, en cuanto a ventajas, desventajas y limitaciones.
- Desarrollar una aplicación interactiva, didáctica e intuitiva, que permita hacer estudio y uso de los algoritmos heurísticos, en la resolución de problemáticas.

1.3.2 Limitaciones

- El estudio algorítmico es sobre algoritmos heurísticos existentes.
- El análisis de algoritmos no profundizará en el estudio de costes espaciales.
- La aplicación es web y utilizara implementaciones simples.

1.4 Metodología

La metodología a implementar en el desarrollo del proyecto, seguirá el siguiente esquema de 3 etapas (I Estudio, II Desarrollo, III Síntesis) y sus respectivas fases descritas a continuación.

Fase	Nombre	Descripción
1	Introducción	Se genera un esquema básico del tema, a partir de lectura en diversas fuentes, imagen, vídeo y documentos de texto.
2	Recopilación de información y reconocimiento de requisitos	Se recopila información considerando el esquema básico detectado, reconociendo requisitos técnicos, teóricos y prácticos que abordar.
3	Análisis de información	Se estudia el tema, se realiza la redacción de la investigación y se definen los algoritmos a profundizar.
4	Estudio de aplicación	Se estudian los algoritmos seleccionados.
5	Construcción de tabla	En una tabla se recopilan nombre, tipo, descripción, costo, ventajas y desventajas de los algoritmos estudiados (En relación a la redacción previa de conceptos del tema)

Tab. 1.1: Fases de estudio (I)

Fase	Nombre	Descripción
1	Definición objetivos generales y específicos	Se define que hará la aplicación en especifico.
2	Definición esquema	Se definen los procesos necesarios para el desarrollo, como el lenguaje al cual llevar los algoritmos estudiados, el entorno de trabajo para el propio desarrollo, y la interfaz que se presentara en vista de la fase anterior.
3	Desarrollo aplicación	Ejecución de procesos y codificación para el desarrollo de la aplicación.

Tab. 1.2: Fases de desarrollo (II)

Fase	Nombre	Descripción
1	Preparación de presentación	Se extrae lo fundamental del proyecto, teniendo en cuenta sus fines, y se desarrolla una presentación.
2	Revisión	Se realiza una revisión general del desarrollo.

Tab. 1.3: Fases de síntesis III

2. ESTUDIO BIBLIOMÉTRICO

2.1 Bibliografía base seleccionada

Los documentos a continuación corresponden a los utilizados para la construcción de la propuesta y son parte del proceso de desarrollo del informe (Marco teórico), en adición a lo que consideraremos en el presente capítulo, basándonos en el estudio de bases de datos pertinentes.

N	Autores	Título	Institución
[1]	Aitana Vidal Esmorís	Algoritmos Heurísticos en optimización	Universidad de Santiago Compostela, España
[2]	Mario Morán Cañón	Introducción a la teoría de la complejidad	Universidad de Valladolid, España
[3]	Pablo M. Rabanal Basalo	Algoritmos heurísticos y aplicación a métodos formales	Universidad complutense de Madrid, España
[4]	Seymour Lipchutz, Ph.D	Teoría y problemas de topología general	Universidad de Temple.
[5]	Hamdy A. Taha(2012)	Investigación de operaciones	University of Arkansas, Fayetteville.
[6]	Carlos Ivorra Castillo (2011)	Lógica y teoría de conjuntos	
[7]	Atocha Aliseda Llera (2000)	Heurística, hipótesis y demostración en matemáticas	Universidad Nacional Autónoma de México.

Tab. 2.1: Bibliografía básica

En [7] se habla de la heurística en matemáticas, de un punto de vista mas teórico y formal y como naturalmente se extiende en otros contextos como la computación, en [1] se introduce la heurística de una manera práctica, se da una

clasificación de tipos de algoritmos heurísticos, y en particular, se habla sobre el algoritmo genético, recocido simulado, colonias de hormigas, búsqueda local y búsqueda tabú, incluyendo la resolución a partir del algoritmo de colonias de hormigas del TSP o problema del viajante. En adición, en [1] se incluye el código en MATLAB de diversos algoritmos, en específico, algoritmo voraz, búsqueda local, búsqueda Tabú, algoritmo genético, sistema de hormigas y recocido simulado.

En [2] se tiene una introducción a la complejidad en computación, se habla desde las máquinas de Turing y sus tipos, para introducir formalmente a los algoritmos como un programa en instancias de la máquina de Turing, permitiendo definir una manera de medir a los algoritmos de las máquinas que resuelven un problema, como también, la complejidad de un problema del punto de vista de los algoritmos que se tienen disponibles para solución. En específico, acá se formaliza la definición de complejidad de un algoritmo, en función de un tiempo representativo de costo, mediante notación asintótica, y las clases de complejidad de los problemas P, NP, NTIME, EXP, NP-duro, NP-completo, entre otras. Como auxiliar y complemento a [2] se tiene [6], donde se cuenta con información acerca de las máquinas de Turing, de manera complementaria, la definición de funciones para las cuales existe un algoritmo, y algunos otros conceptos de lógica que son de utilidad.

En [3] se estudian los algoritmos basados en nubes de partículas, basados en la formación de ríos y de los algoritmos mencionados en [1], añadiendo el concepto de mejora básica en estos algoritmos. El objetivo de este documento es la aplicación de algoritmos heurísticos en problemas NP-duros, presentando una lista de resúmenes de otros trabajos citados, y una breve descripción de diversos algoritmos relacionados al tema, como ciertos campos y contextos en los cuales se aplican. En este trabajo, se estudia el concepto de Método evolutivo y Método formal.

En [4] se tienen conceptos básicos de topología, que permiten tener una definición formal de conceptos como espacio, cercanía, distancia, longitud, conjunto abierto, cerrado, entre otros. Esto sera relevante a la hora de generalizar

los problemas, del punto de vista, de las soluciones que se tienen, condiciones de existencia o suficiencia de valores de interés, definición de procedimientos, o definición de conceptos mas complejos que requieren de este contenido básico.

En [5] se tiene una descripción de lo que es la investigación de operaciones, su fundamentos principales, los modelos utilizados, y su relación con la optimización.

De estos documentos, se han seleccionado las siguientes palabras clave:

> Algoritmo Heurístico, Metodo Heurístico, Modelo Heurístico, Recocido Simulado, Algoritmo Genético, Optimización Heurística, Complejidad Computacional, Eficiencia Computacional, Clase NP, Clase P, Búsqueda tabú, Búsqueda local.

2.2 Bases de datos

Para el estudio bibliométrico se ha seleccionado la base de datos **Scopus** (ScP). Sin embargo, y como se explicará posteriormente, también se recurrirá a **Web Of Science** (WoS) y **Arvix** (Arv).

2.3 Indicadores bibliométricos

Mediante las palabras clave antes mencionadas, se encuentran 103 documentos, considerando un filtro de búsqueda en las secciones de title, abstract y keywords, que considera documentos en español, y publicaciones en scopus desde el año 1996 hasta la actualidad.

2.3.1 Sobre la actividad

Basándonos en la herramienta de análisis con la que cuenta Scopus, vemos a continuación las siguientes tablas, que respectivamente describen la actividad del tema "Heurísticas y complejidad"(Considerando las palabras clave y filtros), en cuanto a los tipos de documentos presentes, las publicaciones anuales desde 1996 hasta la fecha, y la presencia de esta actividad en países de habla española.

Tipo	Frec.	%
Article	71	68.9
Conf. Paper	29	28.2
Note,Review	3	2.9
Total	103	100

Tab. 2.2: Actividad por documento

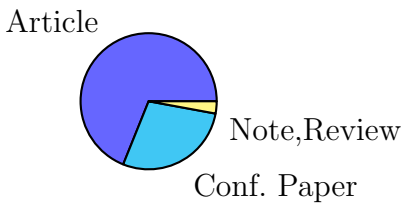


Fig. 2.1: Gráfico circular de documentos

De la población de documentos, contamos con una presencia mayoritaria de artículos, en específico con un 68.9 % del total con 71 documentos. Le siguen los conference Paper con 29 y un 28.2 %, siendo relevante las cantidades presentes y su representación en la población generada, puesto que, serán los tipos de documentos de mayor interés, ya que, representan documentación de investigación o de comunicación científica, lo que responde al enfoque del proyecto.

Cabe destacar, que del total de la población analizada, existen documentos que no cuentan con acceso abierto desde ScP, sin embargo, se ha conseguido el acceso en caso de representar interés, por medio de solicitudes de acceso en otras plataformas como ResearchGate.

N	Año	Frec.	%
1	1996,1997	1	0.97
2	1998,1999	0	0
3	2000,2001	1	0.97
4	2002,2003	4	3.89
5	2004,2005	3	2.91
6	2006,2007	6	5.83
7	2008,2009	12	11.65
8	2010,2011	11	10.68
9	2012,2013	10	9.70
10	2014,2015	19	18.44
11	2016,2017	11	10.7
12	2018,2019	19	18.44
13	2020,2021	6	5.82
	Total	103	100

Tab. 2.3: Actividad por año

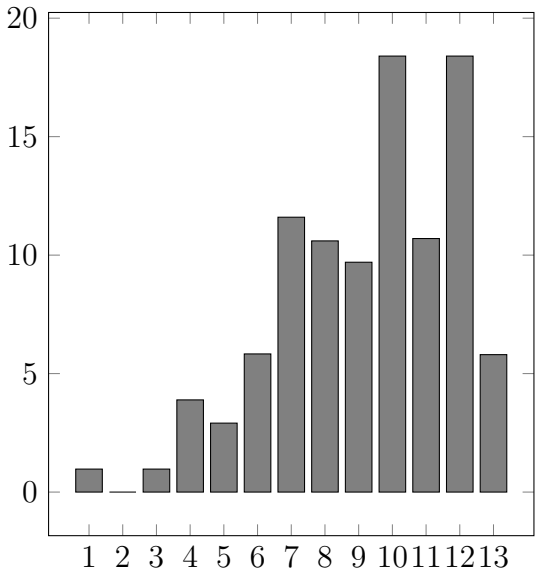


Fig. 2.2: Gráfico circular por año

Con la idea de obtener una documentación que fuera desde los conceptos mas básicos y sus análisis, hasta documentación mas específica sobre una aplicación o situación de aplicación de los algoritmos, se específico un rango grande de épocas de interés. Se observa una mayor actividad del tema en los últimos 11 años, presentando un comportamiento relativamente equilibrando entre el 11.65 % y el 18.44 % anual, representando un 85.4 % aproximadamente del total de documentos obtenidos.

País	Frec.	%
Chile	12	10.71
Colombia	32	28.60
México	22	19.64
España	16	14.28
Cuba	7	6.25
Peru	7	6.25
Brasil	5	4.46
Argentina	4	3.57
Venezuela	4	3.57
Ecuador	3	2.67
Total	112	100

Tab. 2.4: Actividad por país

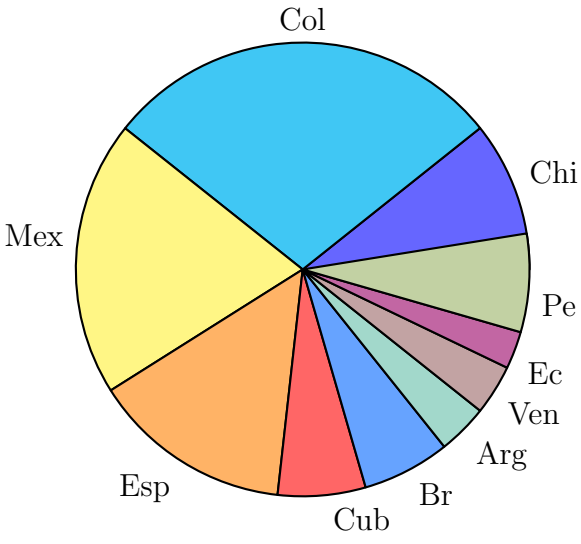


Fig. 2.3: Gráfico circular por país

Finalmente, la actividad del tema considerando los filtros establecidos, se encuentra distribuido en 10 países de Latinoamérica y Europa, presentándose de manera muy amplia en Colombia con un 28.60 % representado a Latinoamérica, y un 14.28 % de España para el caso de Europa.

Resulta interesante observar que existe una relación entre las publicaciones de los países, en el sentido de que, las citas y referencias sobre los documentos en ScP, suelen presentar citas cruzadas, o tener autores de diversos países trabajando sobre un mismo documento.

2.3.2 Selección de fuentes

La documentación [1]-[7] hacen referencia a lo explicado previamente, mientras que los documentos [8]-[12], son necesarios para la compresión de los algoritmos de un punto de vista práctico y mas particular, teniendo en consideración la variación que los algoritmos pueden tener en favor del problema que se esta abordando, y en algunos casos, como ciertos métodos permiten la mejoría, además de representar un complemento de la bibligrafía base.

N	Autores	Título	Peso	Ac
[1]	Aitana Vidal Esmorís	Algoritmos Heurísticos en optimización	16 %	16 %
[2]	Mario Morán Cañón	Introducción a la teoría de la complejidad	14 %	30 %
[3]	Pablo M. Rabanal Basalo	Algoritmos heurísticos y aplicación a métodos formales	5 %	35 %
[4]	Seymour Lipchutz, Ph.D	Teoría y problemas de topología general	5 %	40 %
[5]	Hamdy A. Taha(2012)	Investigación de operaciones	5 %	45 %
[6]	Carlos Ivorra Castillo (2011)	Lógica y teoría de conjuntos	5 %	50 %
[7]	Atocha Aliseda Llera (2000)	Heurística, hipótesis y demostración en matemáticas	10 %	60 %
[8]	Rodrigo Linfati, John Willmer Escobar,Bernardo Cuevas	Un algoritmo basado en búsqueda tabú granular para el problema de balanceo de bicicletas públicas usando múltiples vehículos	5 %	65 %
[9]	Miguel Jiménez-Carrión	Algoritmo genético simple para resolver el problema de programación de la tienda de trabajo (job shop scheduling)	5 %	70 %
[10]	S. Martínez, J. París, I. Colominas, F. Navarrina, M. Casteleiro	Optimización mixta de estructuras de transporte de energía: aplicación del algoritmo de recocido simulado	5 %	75 %
[11]	Johanna Rodríguez León ,Jabid Eduardo Quiroga Méndez, Nestor Raul Ortíz Pimiento	Performance comparison between a classic particle swarm optimization and a genetic algorithm in manufacturing cell design	5 %	80 %
[12]	Marcos Gestal	Introducción a los algoritmos genéticos	10 %	90 %

Tab. 2.5: Bibliografía del estudio bibliométrico

La columna "Peso" hace referencia al porcentaje de relevancia que se le ha asignado en vista, para el desarrollo del presente documento, lo que se puede ver más claramente en las siguientes gráficas:

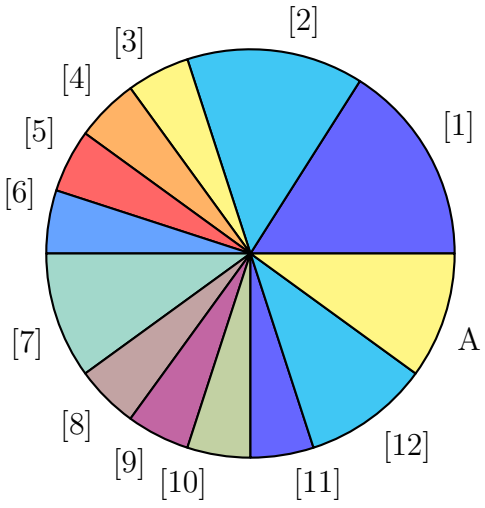


Fig. 2.4: Gráfico circular de fuentes

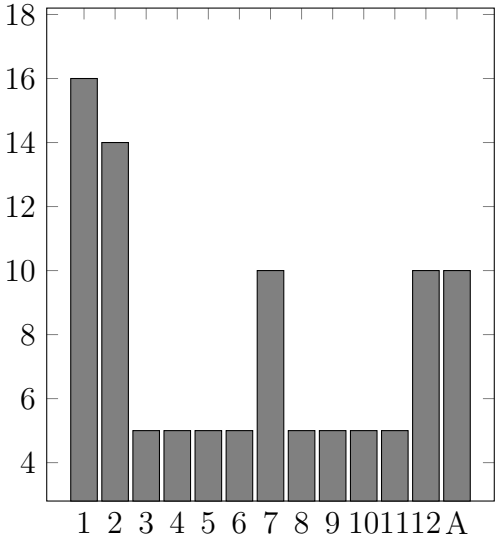


Fig. 2.5: Gráfico de barra de fuentes

Donde A corresponde con auxiliar, lo que hace referencia a otros documentos alternativos, como documentación de cursos impartidos en instituciones, vídeos explicativos de instituciones referentes a los temas de estudio, y otras fuentes que se relacionan no con el contenido propio del informe, si no más bien con su correcto desarrollo.

Toda esta documentación sera indicada en la sección de bibliografía en auxiliares, con su correspondiente explicación de uso.

3. MARCO TEÓRICO

A continuación se hablara de la investigación de operaciones para introducir a la optimización, como una actividad a desarrollar sobre un elemento general, que llamamos sistema.

Se tocaran los elementos esenciales que permiten comprender la optimización y como ésta deriva en procesos que se pueden automatizar mediante la computación, ademas, de una clasificación en vista de los resultados que se entregan o que son capaces de obtener, es decir, si entregan una solución con exactitud, se asemejan a la solución, etc.

Se estudiara la complejidad computacional, para cuantificar el costo de utilizar los algoritmos en la solución de problemas de optimización, para finalmente, introducir a la heurística como alternativa a problemas de costo no razonable.

3.1 Optimización

3.1.1 Investigación de operaciones

La investigación de operaciones (IDO) es una disciplina científica, que aplica la teoría de sistemas, el método científico y variadas disciplinas, para establecer una gestión y control sobre un sistema organizado, sea artificial o natural, mediante la toma de decisiones.

Decimos que aplica la teoría de sistemas, puesto que de un sistema real o también llamado mundo real, se busca una abstracción de éste, que trata de capturar el comportamiento del sistema, en un conjunto de partes y relaciones, que permitan decidir sobre un aspecto que depende de este conjunto de partes.

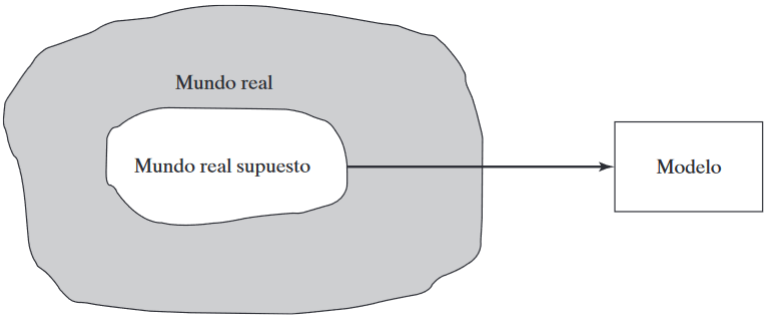


Fig. 3.1: Abstracción de un sistema
Fuente : Investigación de operaciones, HAMDY A. TAHA

Luego, aplica el método científico y la teoría de sistemas, puesto que en general las etapas que se definen para aplicar la investigación de operaciones son:

1. Definición del problema : Donde se identifica un problema, y se analiza el sistema real en busca de factores incidentes en el problema e interrelaciones.
2. Construcción del modelo : Se construye un modelo de un sistema simplificado acorde a la abstracción de factores e interrelaciones. Cabe destacar que algunos factores o interrelaciones pueden o no ser representados de manera directa, y habrá que identificar factores y/o interrelaciones representantes.
3. Solución del modelo : Se debe establece una manera de abordar una búsqueda de la o las soluciones al problema en el modelo construido.
4. Validación del modelo : Se hace una comparativa de los resultados del modelo, con los resultados históricos, factibles, razonables en el mundo real. En caso de no obtener validez, sera necesario volver a etapas anteriores.
5. Implementación : Se aplica la gestión y control sobre el sistema real.

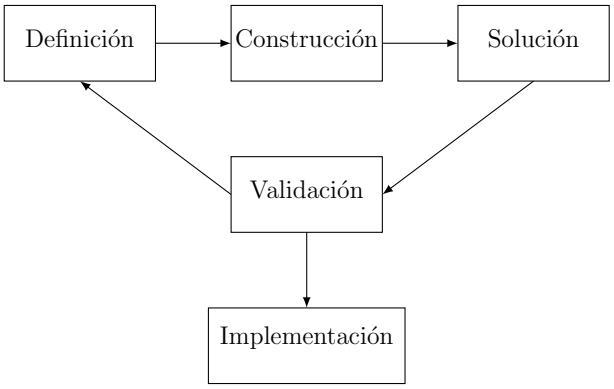


Fig. 3.2: Etapas de investigación de operaciones
Fuente: Elaboración propia

Finalmente, en IDO es importante el uso de la creatividad y experiencia de variadas disciplinas que pueden ir desde la matemática hasta la psicología, con el objetivo de la extracción de fundamentos de sistema supuesto del real, para la preparación de un modelo.

Dicho todo esto, la investigación de operaciones, se resume en la construcción de una sucesión finita de modelos, hasta tener un modelo que tenga cierta "validez" respaldada por datos sobre el sistema real, y nos permita tomar las "mejores" decisiones. En especifico, la etapa de construcción de un modelo, puede ser muy diversa y dependerá de las limitaciones, conveniencia, comodidad y capacidad del o los autores de la aplicación de IDO.

En particular, es de interés el siguiente tipo de modelo en IDO:

3.1.2 Modelo de optimización

Esta formado por $f : E \to S$, una $r-$tupla $X = (X_1, ..., X_n, Y_1, ..., Y_k) \in E$ de variables, con $r = n+k$, y una función vectorial $g : E^m \to S^m$ tal que $g(X) \leq b$ donde $b \in S^m$ y $\leq \subseteq S^m \times S^m$.

- f es la función objetivo, y su expresión $f(X) = U$ cuantifica en un valor lo que nos interesa del sistema en función de factores considerados.
- $X_1, ..., X_n, Y_1, ..., Y_k$ son las variables de decisión, las que representan factores simplificados o factores extraídos del sistema real. En general para la definición de un modelo, no se distingue la existencia de variables controlables X_i y no controlables Y_j, de modo que se suele decir que hay $X_1, ..., X_r$ variables de decisión para $r = n + k$ y $X_{n+j} = Y_j$.
- $g(X) \leq b$ definirá una colección $g_i(X) \leq b_i, i = 1, ..., m$ mediante la desigualdad por coordenadas, siendo las restricciones del modelo y representantes de las interrelaciones existentes entre las variables de decisión.
- Sobre la función objetivo, se establece un operador OPT que proviene de la optimización y hace referencia que deseamos un valor "optimo" que nos permita obtener un beneficio en el sistema real.

La manera en la que se presenta es la siguiente:

$$\begin{cases} \text{OPT}(f(X)) \\ \text{Subject to :} \\ \quad g(X) \leq b \end{cases}$$

o simplemente OPT$\{f(X) \mid g(X) \leq b\}$

Para comprender de mejor manera la representación general que hemos descrito, veremos el siguiente problema de aplicación:

Agroavícola Chile posee un criadero de gallinas ponedoras y compra dos tipos de alimentos para las aves (Campihuevo y Dumor), cuyos costos respectivos son $5000[\frac{\$}{kg}]$ y $2500[\frac{\$}{kg}]$ respectivamente. Al estudiar la dieta requerida por las aves se de terminó que como mínimo debía suministrarles $8[kg]$ de avena, $15[kg]$ de afrecho y $7[kg]$ de maíz por día. El contenido de estos elementos en los alimentos es el siguiente:

Elementos	Campihuevo	Dumor
Avena$[\frac{g}{kg}]$	0,1	0,3
Afrecho$[\frac{g}{kg}]$	0,3	0,4
Maíz$[\frac{g}{kg}]$	0,3	0,1

Tab. 3.1: Composición de alimentos, problema aplicación IDO

¿Que dieta es la óptima satisfaciendo los requerimientos al mínimo costo ?

En este caso en particular, se solicita optimizar el rendimiento de una función de utilidad asociada al costo.

Si la avena, el afrecho y el maíz no deben superar los $8[kg], 15[kg]$ y $7[kg]$ por día, y conocemos que cada una de estas sustancias se pueden obtener en funcíon del Campihuevo y Dumor, pues según la tabla cada una de estas esta presente en $[\frac{g}{kg}]$ de Campihuevo y Dumor según corresponda, si consideramos que $X := [\frac{kg}{dia}]$ de Campihuevo y $Y := [\frac{kg}{dia}]$ de Dumor, se tienen las siguientes expresiones para la avena, el afrecho y el maíz respectivamente:

$$(0{,}1X + 0{,}3Y) \geq 8000 \left[\frac{g}{dia}\right]$$
$$(0{,}3X + 0{,}4Y) \geq 15000 \left[\frac{g}{dia}\right]$$
$$(0{,}3X + 0{,}1Y) \geq 7000 \left[\frac{g}{dia}\right]$$

De este modo, tenemos expresiones que restringen al sistema, o mejor dicho, interrelaciones que permiten establecer reglas sobre el sistema que deseamos simular, por tanto, estas 3 ecuaciones representan las ecuaciones a las que se sujeta nuestro sistema.

Luego, el costo por día considerando las definiciones de X e Y es:

$$Z\left[\frac{\$}{dia}\right] = (5000X + 2500Y)\left[\frac{\$}{kg}\right]\left[\frac{kg}{dia}\right]$$

Finalmente, si consideramos que en el contexto en el que nos encontramos, no nos interesa que $X, Y < 0$ por razones obvias, podemos agregar una cuarta expresión a la que nuestra función de utilidad Z se encuentra sujeta, obteniendo el siguiente modelo para el sistema de interés:

$$\begin{cases} \text{MIN}(Z) = 5000X + 2500Y \\ \text{Subject to :} \\ 0{,}1X + 0{,}3Y \geq 8000 \\ 0{,}3X + 0{,}4Y \geq 15000 \\ 0{,}3X + 0{,}1Y \geq 7000 \\ X, Y \geq 0 \end{cases}$$

con $g(X,Y) = -(0{,}1X + 0{,}3Y\ ,\ 0{,}3X + 0{,}4Y\ ,\ 0{,}3X + 0{,}1Y\ ,\ X\ ,\ Y)$ y $b = -(8000, 15000, 7000, 0, 0)$ para seguir la forma general $g(X,Y) \leq b$.

3.1.3 Optimización

La optimización, busca encontrar valores específicos que permitan maximizar el beneficio o minimizar la perdida. De este modo, bajo el contexto introducido, se trata de la búsqueda de valores en las variables de decisión controlables, sujetas a un numero finito de restricciones, que permiten maximizar o minimizar la función objetivo (dependiendo si representa beneficio o perdida respectivamente).

Sea $\text{OPT}\{f(X) \mid g(X) \leq b\}$ el modelo de optimización, se define:

- **Conjunto factible**: Los puntos que satisfacen las restricciones y son parte del dominio de la función objetivo, son llamados **Puntos factibles** del modelo, luego el conjunto de ellos es $\Omega = \{X \mid g(X) \leq b, X \in E\}$ y se llama conjunto factible. En general se suele omitir la pertenencia en E.
- **Problema de optimización**: Encontrar un X_0 factible de manera que la función f sea lo mayor posible o menor posible, dependiendo del caso. Ahora bien, en la búsqueda de X_0 se pueden dar los siguientes casos:

  Problema no factible: $\Omega = \oslash$.

  Problema no acotado: Para cualquier $Y \in \Omega$ existe un $X \in \Omega$, de modo que $f(X) < f(Y)$ (Minimización) o $f(Y) < f(X)$ (maximización), esto es, sin importar el valor que se tome, siempre existe otro valor que mejora la función objetivo.

  Problema con óptimo: Existe un $X_0 \in \Omega$ para el que para todo $X \in \Omega$ se cumple que $f(X_0) \leq f(X)$ o en otro caso $f(X_0) \geq f(X)$ según corresponda, al que llamamos **óptimo global**. Por otro lado, si

se encuentra un X_0 para el que existe un $\varepsilon \in \mathbb{R}^+$ de modo que para todo $X \in \Omega$ tal que $\|X - X_0\| < \varepsilon$ se tiene que $f(X_0) \leq f(X)$ o $f(X) \leq f(X_0)$, se habla de un **óptimo local** para el problema, siendo $\|\cdot\|$ una norma.

Para la fase de solución de un problema de optimización, se tienen los siguientes conceptos relevantes a la hora de caracterizar un problema y el modelo proporcionado:

Existencia y unicidad: Se habla de existencia de óptimo, cuando se tiene certeza de que existe al menos uno, y se habla de unicidad para asegurar que existe solo uno. Esto es relevante para un problema de optimización, pues esto nos permite saber que la solución encontrada en un conjunto (Ω u otro mas pequeño) es la única que puede haber o no lo es, lo que es útil en diversas situaciones.

Condiciones de optimalidad: Son proposiciones (generalmente implicaciones), que por medio del cumplimiento de una premisa o proposición en función del modelo definido, podrán dar información acerca de la unicidad y/o existencia de óptimos en el modelo. Por ejemplo, si M es un modelo de optimización y φ una premisa, se tiene:

$$\varphi(M) \rightarrow (M \text{ Tiene única solución})$$

Luego se vera, que las condiciones de optimalidad son útiles pero no en todos los modelos, pues al cumplirse dan certeza, pero al no cumplirse no suelen dar información adicional, o en otros casos, al cumplirse no dicen nada sobre el modelo y al no cumplirse dan certezas (En el ejemplo, es necesario que se cumpla la premisa para tener información)

Estabilidad : En los modelos de optimización, se tienen ciertos valores que acompañan a las variables de decisión y suelen estar fijos (Dependiendo el tipo de optimización como se vera mas adelante), estos son influyentes en los óptimos a encontrar. El estudio respecto la variación de estos parámetros, manteniendo soluciones optimas, es la estabilidad del modelo.

3.1.4 Condiciones necesarias y suficientes de óptimos

En la práctica, para corroborar la optimalidad, se recurre a condiciones necesarias y suficientes, las que pueden utilizarse de manera general, es decir, tomar un conjunto de estudio, o sobre puntos específicos en los cuales se tienen conjeturas , o se sospecha mas bien que pueda haber un óptimo.

Una condición necesaria para la optimalidad, es aquella que debe cumplirse para que exista la posibilidad de que exista un óptimo, así mismo, se cumple para puntos no óptimos y para óptimos. Caso contrario, una condición suficiente para la optimalidad, es aquella que al cumplirse, asegura la optimalidad, así mismo, solo se cumple en puntos óptimos.

En la práctica, las condiciones necesarias permitirán descartar optimalidad, pues, estas condiciones serán de la forma, (si hay óptimos o X es un optimo) entonces P, de modo que, si P es falso no habrá óptimos o X no lo sera según corresponda. Por el contrario las condiciones suficientes, responden a la implicación P entonces hay óptimos o X es un optimo, lo que es una implicación directa.

3.1.5 Algunos problemas de ejemplo en optimización

El problema presentado como ejemplo de modelo de optimización, es un tipo de problema particular donde la función objetivo $Z : \mathbb{R}^k \to \mathbb{R}$ y las restricciones $g_j : \mathbb{R}^k \to \mathbb{R}$ con $j = 1, ..., m$, son de la forma $Z(X) = \sum_{i\in[k]} B_i X_i$ y $g_j(X) = \sum_{i\in[m]} B_{ij} X_i$ para $j = 1, ..., m$ y se llaman lineales. De esta manera, lo que define una instancia de este tipo de problemas, es el valor que toman las variables B_i, B_{ij} para $i = 1, ..., k, j = 1, ..., m$, las cuales se consideraran fijas para el siguiente estudio:

Si $Z(X) - Z(X_0) = \nabla Z(X_0)(X - X_0)$ entonces:

$$\lim_{X\to X_0} \frac{\|Z(X) - Z(X_0) - \nabla Z(X_0)(X - X_0)\|}{\|X - X_0\|} = 0$$

Esto es, la función objetivo es diferenciable en cualquier $X_0 \in \mathbb{R}^k$, lo que fácilmente se extiende también a cada restricción.

Usando la condición necesaria de optimalidad para una función objetivo $f : A \subseteq \mathbb{R}^n \to \mathbb{R}$ diferenciable, definida como, si X_0 es un optimo entonces $\nabla f(X_0) = 0$, se tiene que :

$$\nabla Z = (B_1, ...B_k) = 0 \Leftrightarrow B_i = 0, i = 1, ..., k$$

Con ello no existe optimo en la función sin restricciones, así mismo, no habrá en Int(Ω). Esto implica, que de haber óptimos para el problema, estos están en $\partial\Omega$.

La función objetivo $Z = \sum_{i\in[k]} B_i X_i$ define un hiperplano $k-$dimensional, y las restricciones de la forma $g_i(X) = \sum_{i\in[k]} B_{ij} X_i \leq 0$ cuando $g_i(X) = 0$ definen un hiperplano $k-1$ dimensional. Dicho esto, al tomar un par de puntos $w, t \in \Omega \subseteq \mathbb{R}^{k-1}$ con $g_j(w) = g_j(t) = 0$, se tiene un segmento $\alpha w + (1-\alpha)t, \alpha \in [0,1] \subset \mathbb{R}$ sobre la restricción g_j que es parte de $\partial\Omega$, y se observa lo siguiente:

$$\alpha w + (1-\alpha)t = (\alpha w_i + t_i - \alpha t_i), i = 1, ..., k$$
$$\rightarrow Z = \sum_{i\in[k]} B_i(\alpha w_i + t_i - \alpha t_i) \leftrightarrow Z(\alpha) = \sum_{i\in[k]} w_i B_i \alpha + B_i t_i - \alpha B_i t_i$$

De esta manera, la función a pasado a ser de una variable, que dará un valor para la función Z dependiente de α, es decir, los óptimos de Z sobre el segmento de recta formado, son los óptimos de $Z(\alpha)$.

$$\frac{dZ}{d\alpha} = 0 \leftrightarrow \sum_{i\in[k]} B_i w_i = \sum_{i\in[k]} B_i t_i$$
$$\rightarrow Z = \sum_{i\in[k]} B_i w_i = Z(\alpha = 1) \vee Z = \sum_{i\in[k]} B_i t_i = Z(\alpha = 0)$$

Esto quiere decir, que los candidatos a óptimos de la función Z limitada al segmento, están en los extremos del segmento de recta.

Lo analizado anteriormente, será de utilidad para la resolución de problemas de optimización con función objetivo y restricciones lineales, donde se tienen 2 variables de decisión, así mismo, $\Omega \subseteq \mathbb{R}^2$. A continuación, se resuelve el problema de ejemplo para la definición de modelo de optimización, siendo el procedimiento extensible a otros problemas lineales con 2 variables de decisión, siendo conocido como método gráfico.

Para el modelo :

$$\begin{cases} \text{MIN}(Z) = 5000X + 2500Y \\ \text{Subject to :} \\ 8000 - 0{,}1X - 0{,}3Y \leq 0 \\ 15000 - 0{,}3X - 0{,}4Y \leq 0 \\ 70000 - 0{,}3X - 0{,}1Y \leq 0 \\ -X, -Y \leq 0 \end{cases}$$

Lo primero es conocer al conjunto Ω, para ello, se grafican las restricciones del problema sobre el plano XY:

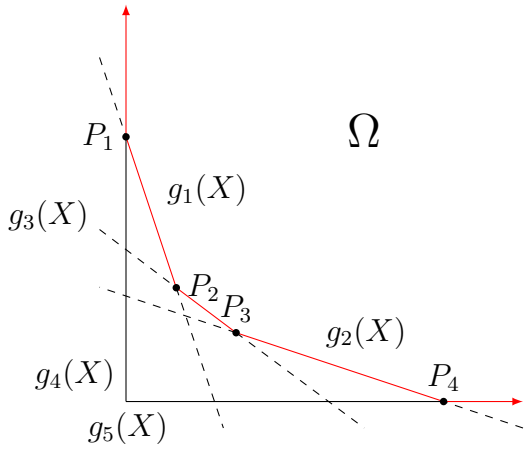


Fig. 3.3: Conjunto factible y puntos de interés

Según lo visto previamente, cualquier segmento $P_i\alpha + (1-\alpha)P_{i+1}$ para $i = 1, 2, 3$ tendrá candidatos a óptimos en P_i, P_{i+1}, lo que permite descartar cualquier punto intermedio en el segmento como candidato de la función Z, pues, esos puntos serían óptimos para el segmento que se ha fijado y no en totalidad para la recta o segmento que se tiene entre P_i, P_{i+1}. Luego, se puede tomar cualquier punto $(0, y)$ y definir $(0, y)\alpha + (1-\alpha)P_1$, obteniendo como candidatos a $(0, y), P_1$, ahora, se podría hacer $(0, y+1)\alpha + (1-\alpha)P_1$ lo que descartaría al punto $(0, y)$ como óptimo, lo que se puede repetir infinitamente, por ende, se descarta otro punto sobre la recta $x = 0$ (distinto de P_1) en la frontera de Ω, lo que es análogo para el caso $(x, 0)\alpha + (1-\alpha)P_4$, descartando puntos candidatos (distintos de P_4) sobre la recta $y = 0$ sobre Ω.

Los puntos después de un procedimiento básico son los siguientes:

$$P_1 = (0, 70000) \ ; \ P_2 = \frac{1}{9}(130000, 240000)$$

$$P_3 = \frac{1}{5}(130000, 90000) \ ; \ P_4 = (80000, 0)$$

Luego, para saber cual es el optimo de estos candidatos, se evalúan en la función objetivo y se escoge el que de un menor valor para Z.

$$Z(P_1) = 175 \cdot 10^6 \ ; \ Z(P_2) = \frac{1250 \cdot 10^6}{9} \approx 138 \cdot 10^6$$

$$Z(P_3) = 175 \cdot 10^6 \ ; \ Z(P_4) = 400 \cdot 10^6$$

$$\rightarrow \text{ El optimo es } P_2$$

Respondiendo al problema de optimización, con $\frac{1}{9}130000$ kilogramos de Campihuevo y $\frac{1}{9}240000$ kilogramos de Dumor, se cumplen los estándares de alimentación de las gallinas, superando los 8 kilogramos de avena, 15 kilogramos de afrecho y 7 kilogramos de maíz, a un costo de aproximadamente 138 millones de pesos.

En este caso en particular, la solución corresponde a un punto donde X, Y son distintos de cero, sin embargo, puede haberse dado matemáticamente, debido a que $X, Y \geq 0$ como restricciones, una combinación $(X, 0)$ o $(0, Y)$ como solución, lo que claramente no tiene sentido en la realidad, pues, la avena, el afrecho y el maíz necesitan de ambos componentes, de este modo, es importante expresar en las restricciones y en la función objetivo todas las condiciones que sean factibles del punto de vista real, lo que en este caso implica que $X, Y > 0$, sin embargo no altera el resultado.

El problema a solucionar a continuación, es un problema bastante común en la teoría de grafos y en su aplicación en otras disciplinas, este tipo de problemas obedece a una lógica diferente al caso anterior, de como abordar el problema de optimizar, puesto que el conjunto factible es discreto, y con ello no se tiene el concepto de derivada y sus derivados, al menos hablando de la derivada del punto de vista clásico.

Sea G un grafo, $f : E(G) \to \mathbb{R}^+$ una función de peso que asigna a cada arista un valor positivo real y E_i la etapa $i-$esima, se define lo siguiente:

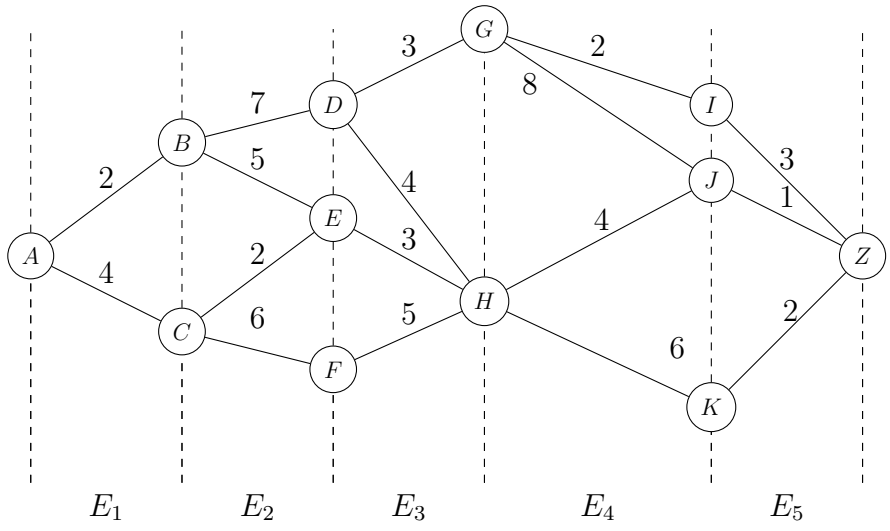


Fig. 3.4: Grafo pesado para el camino menos costoso

G modela el paso de un estado A a un estado Z, en el que existen diferentes caminos, o mejor dicho, diferentes secuencias de estados a diferente costo.

Básicamente, se debe pasar por diferentes etapas, pero dependiendo de cual sea el estado inicial y final de una etapa, se tendrá un costo que se acumulara para todo el proceso. Por ejemplo, para la primera etapa se tienen dos opciones, AB o AC, con ello, existen dos costos para E_1 dados por f_{AB} y f_{AC}, luego, se deberá escoger entre los adyacentes a B para E_2 o los adyacentes a C para E_2 y así sucesivamente.

Es necesario aclarar, que no necesariamente se debe tomar el mínimo costo para cada etapa, por ejemplo, es claro que el costo de AB es menor que el de AC, con lo que se podría pensar en tomar AB y buscar un adyacente, de donde se tienen los caminos ABE y ABD, que son mas costosos que el camino ACE, de este modo, a pesar de que AC cueste mas que AB para las primeras dos etapas conviene más ACE.

El modelo de optimización del problema planteado para minimizar, es el siguiente:

$$\begin{cases} Z = y_1(f_{AB} + y_2(f_{BE} + f_{EH} + y_3(f_{HK} + f_{KZ}) + \overline{y_3}(f_{HJ} + f_{JZ})) + \\ \quad + \overline{y_2}(f_{BD} + f_{DG} + y_4(f_{GI} + f_{IZ}) + \overline{y_4}(f_{GJ} + f_{JZ}))) + \\ \overline{y_1}(f_{AC} + y_5(f_{CE} + f_{EH} + y_6(f_{HJ} + f_{JZ}) + \overline{y_6}(f_{HK} + f_{KZ})) + \\ \quad + \overline{y_5}(f_{CF} + f_{FH} + y_7(f_{HJ} + f_{JZ}) + \overline{y_7}(f_{HK} + f_{KZ}))) \\ y_i \in \{0, 1\} \\ \begin{cases} \overline{y_i} = 0 \text{ si } y_i = 1 \\ \overline{y_i} = 1 \text{ si } y_i = 0 \end{cases} \end{cases}$$

Donde f_{XY} es el peso o costo del par adyacente $(X, Y) \in E(G)$ y las variables y_i sirven para separar las posibilidades de elección al pasar de un estado a otro, por ejemplo, al estar en A se puede elegir B o C, de este modo, si $y_1 = 1$ entonces se forma el camino AB y se descarta AC y cualquier otro que tenga por primeros vértices a AC, luego, en AB se puede tomar D o E, de esta manera, si $y_2 = 1$ entonces se toma ABD y se descarta cualquier camino que empiece por ABE, y así sucesivamente, de modo que vayan quedando en la función sin hacerse nulo, los costos asociados al camino que únicamente se define mediante los valores de $y_i, i = 1, ..., 7$. De esta manera, la solución al problema de optimización consiste en hallar la combinación $X_0 \in \{0, 1\}$ que hace que la función $Z : \{0, 1\}^7 \to \mathbb{R}$ sea mínima.

Reemplazando los costos asociados a esta instancia, haciendo uso de la función particular f definida para este G, y procedimientos básicos de álgebra se tiene:

$$\begin{cases} Z = y_1(2 + y_2(8 + 8y_3 + 5\overline{y_3}) + \overline{y_2}(10 + 5y_4 + 9\overline{y_4})) \\ +\overline{y_1}(4 + y_5(5 + 5y_6 + 8\overline{y_6}) + \overline{y_5}(11 + 5y_7 + 8\overline{y_7})) \\ y_i \in \{0, 1\} \\ \begin{cases} \overline{y_i} = 0 \text{ si } y_i = 1 \\ \overline{y_i} = 1 \text{ si } y_i = 0 \end{cases} \end{cases}$$

Ahora, fácilmente se puede ver que existen únicamente 8 caminos o combinaciones, dependiendo de como se aborda el problema, si desde la gráfica o desde la función, teniendo los siguientes costos:

Punto	Camino	Costo
$(1, 1, 1, y_4, y_5, y_6, y_7)$	$ABEHKZ$	$Z = 18$
$(1, 1, 0, y_4, y_5, y_6, y_7)$	$ABEHJZ$	$Z = 15$
$(1, 0, y_3, 1, y_5, y_6, y_7)$	$ABDGIZ$	$Z = 17$
$(1, 0, y_3, 0, y_5, y_6, y_7)$	$ABDGJZ$	$Z = 21$
$(0, y_2, y_3, y_4, 0, y_6, 1)$	$ACFHJZ$	$Z = 22$
$(0, y_2, y_3, y_4, 0, y_6, 0)$	$ACFHKZ$	$Z = 17$
$(0, y_2, y_3, y_4, 1, 1, y_7)$	$ACEHJZ$	$Z = 14$
$(0, y_2, y_3, y_4, 1, 0, y_7)$	$ACEHKZ$	$Z = 17$

Tab. 3.2: Puntos factibles del problema de estados

De acá es claro, que el camino de estados que generan el menor costo acumulado de etapas es $ACEHJZ$, lo que se puede corroborar de manera exhaustiva sobre el grafo directamente o mediante la función objetivo:

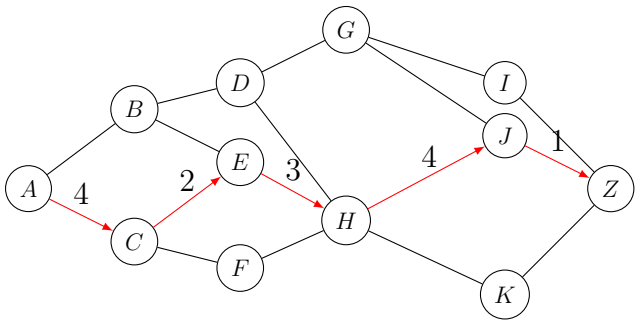


Fig. 3.5: Camino mínimo en rojo

$Z(0, y_2, y_3, y_4, 1, 1, y_7) = 0 \cdot (2 + y_2(8 + 8y_3 + 5\overline{y_3}) + \overline{y_2}(10 + 5y_4 + 9\overline{y_4})) + 1 \cdot (4 + 1 \cdot (5 + 5 \cdot 1 + 8 \cdot 0) + 0 \cdot (11 + 5y_7 + 8\overline{y_7})) = (4 + 1 \cdot (5 + 5)) = 14$

Si se observan los puntos solución en la tabla, en todos existen cuatro variables que no toman un valor, de esta manera, si se toma la primera variable que no tiene valor de izquierda a derecha, se obtienen 2 puntos con 3 variables sin valor, donde la variable tiene valor 0 o 1, si se repite con estos 2 puntos con 3 variables sin valor, se tendrán 4 puntos donde hay 2 variables sin valor, y así sucesivamente. Esto en definitiva, nos dice que existen 2^4 puntos por cada punto en la tabla, luego en la tabla hay 8 y se tienen $8 \cdot 2^4 = 128$ puntos factibles, siendo el total de todos los puntos que se consideran candidatos en Ω, al menos bajo el modelo planteado.

Esto es relevante, pues, si se tuviera que verificar en la función objetivo para cada una de estas 128 combinaciones de manera mecánica, es decir, sin pensar en el contexto, ni en la semántica del problema, y únicamente remitirse a reemplazar y guardar el numero resultante, nos encontraríamos con 16 soluciones, teniendo en cuenta que $(0, y_2, y_3, y_4, 1, 1, y_7)$ representa 16 puntos, y se tendrían 112 puntos no favorables o que no son óptimos, lo que esta muy lejano de las 8 verificaciones echas a mano para un solo óptimo.

3.1.6 Programación matemática

La optimización hasta el momento, ha hecho referencia a la utilización de la matemática para el estudio de un problema (mediante un modelo) y posterior búsqueda de valores para conseguir un beneficio (mediante la solución al modelo). Si bien, se han hecho ciertos guiños hacia la computación, no se ha hablado de manera formal respecto el uso y la inclusión de los métodos computacionales en optimización, entendiendo a un método computacional, como la implementación de un paso a paso que puede ser tan abstracto como se quiera, en un máquina, traduciendo la esencia del paso a paso a las limitaciones de la máquina.

Ahora bien, la optimización siendo rigurosos con la definición, tiene que ver con la búsqueda los óptimos, es decir, se trata mas de la acción que del como, mientras que el estudio de la teoría, de aplicaciones y métodos para la resolución son asociados al concepto de programación matemática, he ahí por que son tratados como sinónimos optimización y programación matemática generalmente en la literatura, y para hablar de la optimización de ciertos tipos, se habla de programación de ese tipo, es decir, se hablaría de programación lineal, mas que de optimización lineal, para citar al campo que se encarga de estudiar la

optimización de problemas con funciones lineales.

La programación matemática, ademas, representa un marco mas amplio que la optimización, puesto que se habla de manera explicita de incluir a la computación, es decir, introducir métodos computacionales que buscan hallar, verificar y facilitar el estudio, solución y validación de los modelos de optimización, con la principal característica de ser automáticos.

Esto implica la necesidad de satisfacer nuevas condiciones adicionales, que son parte de la resolución de un problema pero no son incluidas en el modelo de optimización, pues, tienen que ver con el tiempo y el espacio que son necesarios para dar solución al problema de manera automática en una maquina.

La inclusión de la computación no solo permite automatizar procesos para solucionar problemas, también, permite dar una caracterización de la dificultad de los problemas, teniendo en cuenta, que los problemas que se han presentado como ejemplo, son bastante simples en relación con la realidad, es decir, la computación mas que un apoyo, es fundamental para la resolución de problemas con grandes cantidades de variables y restricciones, ademas de ser capaz de ofrecer soluciones "alternativas".

El paso a paso abstracto del que se hablo, hace referencia al algoritmo. El algoritmo es una secuencia mecánica de pasos sin ambigüedad (aunque como se vera existen variaciones), siendo la implementación o instancia definida en una maquina, el método o procedimiento asociado.

Para ejemplificar la utilización de la implementación de un algoritmo o un método computacional en una computadora moderna, se ve a continuación un método para determinar el camino mínimo de un vértice a cualquier otro dentro de un grafo con ciertas condiciones.

> Sea G un grafo pesado con n nodos no aislados y enumerados, i es el indice del vértice inicial, p una matriz donde p_{kj} es el peso de ir de k a j, a una matriz donde $a_{kj} = 1$ si k es adyacente con j y en caso contrario 0, se tiene lo siguiente:
>
> 1: $i \leftarrow$ Indice del nodo de salida.
> 2: $D \leftarrow (D[0], ..., D[n-1])$
> 3: $V \leftarrow (V[0], ..., V[n-1])$

```
4:  Para j ← 0 hasta n − 1 con paso 1 hacer:
5:      Si (i == j) entonces:
6:          D[j] ← 0
7:          V[j] ← 1
8:      Caso contrario:
9:          V[j] ← 0
10:     Fin si
11:     Si a_ij == 1 entonces:
12:         D[j] ← p_ij
13:     En otro caso
14:         D[j] ← +∞
15:     Fin si
16: Fin Para
17: t ← 0
18: Mientras t == 0 hacer:
19:     s ← 0
20:     M ← +∞
21:     Para j ← 0 hasta n − 1 con paso 1 hacer:
22:         Si (V[j] == 0 ∧ M > V[j]) entonces:
23:             s ← j
24:             M ← V[j]
25:         Fin si
26:     Fin Para
27:     V[s] ← 1
28:     Si (M == +∞) entonces:
29:         t ← 1
30:     Caso contrario:
31:         Para j ← 0 hasta n − 1 con paso 1 hacer:
32:             Si (a_sj == 1) entonces:
33:                 Si (D[s] > D[j] + p_sj) entonces:
34:                     D[s] ← D[j] + p_sj
35:                 Fin si
36:             Fin si
37:         Fin Para
38:     Fin si
39: Fin Mientras
40: Retornar D
```

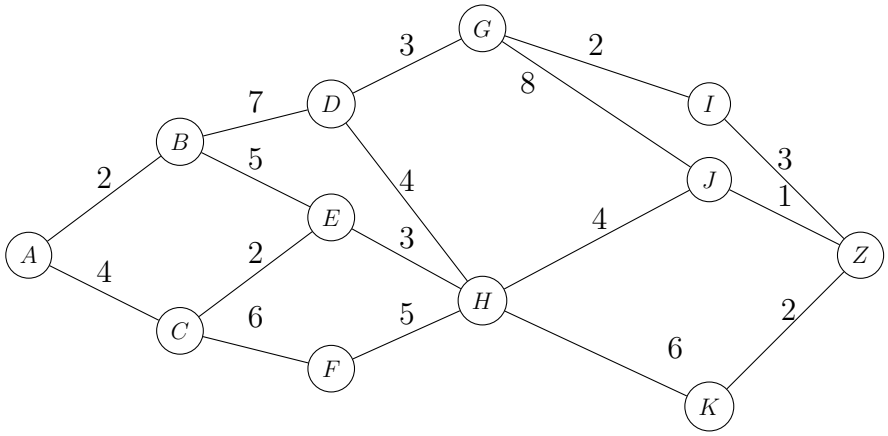
De esta manera, se obtiene un D donde en D_j se tiene el peso mínimo de ir de i hasta j (Si existe), utilizando un V con $V_j = 1$ si ya se tiene calculado un mínimo en D_j o ya fue visitado j y en caso contrario 0. Hay que aclarar que $+\infty$ cumple con un propósito operacional y únicamente debe considerarse como un numero muy alto o estratégico que permita llevar adelante el algoritmo.

Este algoritmo cuenta con una particularidad, y es que al momento de

seleccionar el nuevo vértice a visitar, si existe mas de uno que satisface la condición de valor mínimo en D, se escoge el primero de izquierda a derecha, lo que no necesariamente tiene que ser así, se podría escoger el ultimo o alguno de manera aleatoria.

El núcleo del método se encuentra en las lineas 18-40, es claro las lineas 1-17 sirven para inicializar las estructuras que se han utilizado, esto es, establecer las condiciones iniciales del algoritmo para ajustarlo a las limitaciones. Esta manera de representar un algoritmo, es bastante menos abstracta y precisa que un paso a paso simple definido por frases o indicaciones, esta manera de representar es la mas cercana a su implementación en un lenguaje de programación, donde justamente existe una abstracción del lenguaje a utilizar, pero, teniendo en cuenta que las estructuras tienen definiciones comunes en la mayoría de ellos.

La aplicación del método sobre el grafo a continuación entrega los siguientes resultados:



Con $1 \rightarrow A, 2 \rightarrow B, ..., 12 \rightarrow Z$, se define un esquema para S, D y V, mediante la siguiente tabla:

S	D	V
A	$(0, 2, 4, \infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty)$	$(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$
B	$(0, 2, 4, 9, 7, \infty, \infty, \infty, \infty, \infty, \infty, \infty)$	$(1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$
C	$(0, 2, 4, 9, 6, 10, \infty, \infty, \infty, \infty, \infty, \infty)$	$(1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0)$
E	$(0, 2, 4, 9, 6, 10, \infty, 9, \infty, \infty, \infty, \infty)$	$(1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0)$
D	$(0, 2, 4, 9, 6, 10, 12, 9, \infty, \infty, \infty, \infty)$	$(1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0)$
H	$(0, 2, 4, 9, 6, 10, 12, 9, \infty, 13, 15, \infty)$	$(1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0)$
F	$(0, 2, 4, 9, 6, 10, 12, 9, \infty, 13, 15, \infty)$	$(1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0)$
G	$(0, 2, 4, 9, 6, 10, 12, 9, 14, 13, 15, \infty)$	$(1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0)$
J	$(0, 2, 4, 9, 6, 10, 12, 9, 14, 13, 15, 14)$	$(1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0)$
I	$(0, 2, 4, 9, 6, 10, 12, 9, 14, 13, 15, 14)$	$(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0)$
Z	$(0, 2, 4, 9, 6, 10, 12, 9, 14, 13, 15, 14)$	$(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1)$
K	$(0, 2, 4, 9, 6, 10, 12, 9, 14, 13, 15, 14)$	$(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$

Tab. 3.3: Aplicación de Dijkstra

Obteniendo como respuesta, que el costo mínimo de ir de A hasta Z es 14, lo que obedece a lo conocido anteriormente, sin embargo, el algoritmo no entrega en una estructura o "guarda" el camino llevado.

Es evidente que los algoritmos de interés, son los que tienen una definición de su paso a paso más cercana al método, pues, es en este caso, que tiene más sentido y facilidad asociar un tiempo y espacio.

Tipos de algoritmos

Los algoritmos tienen dos formas de dividirse que son de interés, la primera de ellas tiene relación con la naturaleza que implica la obtención de resultados y se puede observar en (Rafael Martí Cunquero , "Algoritmos Heurísticos en Optimización Combinatoria") y a continuación :

Algoritmos deterministas: Se establece una correspondencia entre entrada y salida, que se mantiene estática para toda ejecución o uso del algoritmo.

Por ejemplo, en el Dijkstra mostrado, se seleccionaba la primera distancia de izquierda a derecha en D, si al momento de escoger el vértice a visitar,

se tenían varias distancias con mismo valor. Si pensamos en seleccionar de manera aleatoria estas distancias, esto afectará la construcción de los vectores D y V y la asignación de S, sin embargo, no hay que confundirlo con el resultado final entregado por el algoritmo, que seguirá siendo el mismo, lo que lo define como determinista.

Algoritmos no deterministas: Se establece una correspondencia entre entrada y salida, que no se mantiene estática, esto es, existe un conjunto de posibles salidas para una entrada. La salida tendrá carácter aleatorio, de modo que el conjunto de salida se puede conocer, pero no se sabe con exactitud cuales son las correspondencias.

Siguiendo el ejemplo Dijkstra, si los pesos de las aristas en el grafo G, se definieran de manera aleatoria entre un conjunto posible o intervalo, claramente el resultado final no sería el mismo, así mismo D no sería el mismo en cada ejecución al tomar X_i como partida, lo que lo convierte en uno no determinista.

Y la segunda, tiene que ver con como se caracteriza la solución a un problema de optimización, dando una etiqueta o nombramiento al algoritmo que permitió hallar esa solución:

Algoritmos exactos: Se devuelve la solución óptima.

Algoritmos aproximados: Se obtiene una solución que es un porcentaje de la solución óptima. Esto se suele formalizar como:

$$\begin{cases} X^* \leq X' \leq \varepsilon X^* \text{ con } \varepsilon > 1 \text{ (Minimizar)} \\ \varepsilon X^* \leq X' \leq X^* \text{ con } \varepsilon < 1 \text{ (Maximizar)} \end{cases}$$

Siendo X^* la solución optima, X' la aproximada y ε un real. En general, a un algoritmo que aproxima mediante una constante ε siguiendo el esquema, se le llama $\varepsilon-$aproximado.

Algoritmos Heurísticos: Ofrecen salidas sin garantía de que la solución obtenida sea la esperada y haciendo mas particular el algoritmo al problema que se quiere abordar, en favor de un mejor tiempo de respuesta. Esto se logra como se vera posteriormente, y en sencillas palabras, construyendo un razonamiento sobre la solución en el que no se intenta abordar de manera completa al total de consideraciones sobre el problema, se ajusta el procedimiento a lo que se sabe del problema en especifico, perdiendo credibilidad y generalidad Al mismo tiempo, se habla de algoritmos metaheurísticos, a los que obedecen la misma lógica, pero intentando abordando a una colección de problemas de esta manera.

Por ende, si se tiene una entrada X para un algoritmo, se tendrá una sola salida o solución (determinista) que puede ser exacta, aproximada o con menos garantía de su optimalidad(heurística), o por otro lado, se puede tener una solución de un conjunto posible(no determinista), que puede ser exacta, aproximada o con menos garantía de optimalidad, lo que se ve en el siguiente esquema:

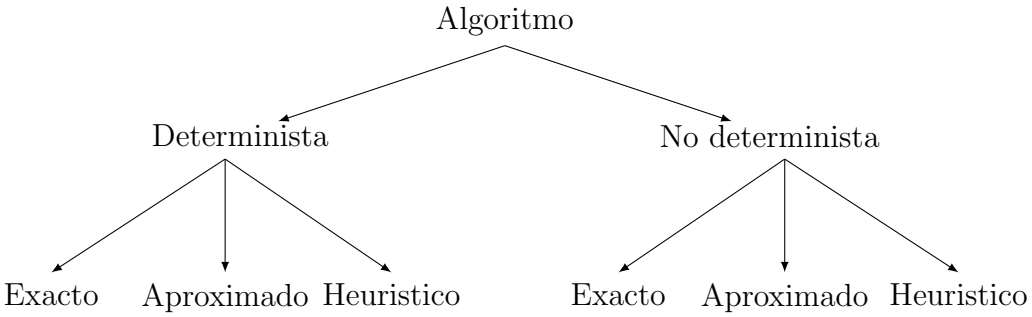


Fig. 3.6: Esquema de tipos de algoritmos

El añadido de un costo y tiempo a la ejecución de un algoritmo, relacionara la dificultad de los problemas con el costo de tiempo y espacio necesario de los algoritmos que existen para ese problema. Ahora bien, resulta que dependiendo del tipo de algoritmo utilizado, se tendrá un menor tiempo o costo, en una relación inversa con la cantidad posible, garantía y exactitud de las soluciones que entrega, para ser más preciso, existe una disminución de tiempo en favor del "empeoramiento" de la solución que se obtiene, que ocurrirá desde izquierda a derecha en el esquema. Es por ello, que es intuitivo pensar, en la utilización de un algoritmo de manera estratégica en vista de cuales son los costos o equivalentemente, en vista de la dificultad del problema.

Calidad de un algoritmo

Anteriormente, se hablaba de una cierta calidad de las soluciones, y como esta se ve afectada por los tipos de soluciones que entrega un algoritmo. Dicho esto, para medir la calidad de un algoritmo, se tiene una relación intima entre algoritmo y tipo de solución, con las siguientes maneras de calificar un algoritmo:

Algoritmo eficiente: El costo computacional del algoritmo es aceptable, razonable o realista. Un algoritmo tendrá asociado un costo abstracto a la

maquina que se utiliza, sin embargo, dependerá de lo que se ha definido como paso a paso y cual es el costo de este paso a paso, como se vera mas adelante. Lo relevante es que, existe cierta vara de medida que incluye aspectos no puramente matemáticos, pues, en algunos algoritmos se tendrán tiempos finitos para entregar una solución, pero no factible en el mundo real, o se podrá tener asegurada una solución exacta y precisa, pero el costo de tiempo parece inaceptable.

Algoritmo bueno: La solución en promedio debe estar cerca del optimo. De acá es evidente que todo algoritmo exacto es bueno, no así un algoritmo aproximado que su definición de bueno dependerá de un cierto factor ε que se utiliza como indicador de que tan cercana está la solución hallada del óptimo.

Algoritmo robusto: La probabilidad de que se obtenga una solución lejana al optimo es baja. Esto es transversal al igual que los casos anteriores, es claro que un algoritmo exacto sera robusto pues la probabilidad de que se tenga una solución lejana al optimo es 0, sin embargo, para los casos donde se recurre a conjunto de soluciones, a la aleatoriedad del no determinismo, esto sera esencial para medir que tan bueno es un algoritmo en función de la probabilidad mencionada.

En particular, para calificar a un algoritmo heurístico, se tienen maneras adicionales de medir su calidad, basadas en comparaciones con otro tipo de algoritmos o ciertas situaciones puntuales, como las que vemos a continuación:

- **Comparación con un algoritmo exacto truncado**: Si no se utiliza un algoritmo exacto para hallar una solución, es por dos razones, o no se conoce (existe) tal algoritmo exacto, o no cumple con un costo computacional razonable. Ahora, mediante una restricción del algoritmo exacto se puede perder esta condición de ineficiente, truncando su ejecución, esto es, acortando la ejecución a un cierto tiempo límite, acortando la ejecución mediante la utilización de un parámetro de control, entre otros. Esto permite, cuando el algoritmo exacto entrega resultados a lo largo de su ejecución y no únicamente al terminar, comparar los resultados de un algoritmo heurístico, con los resultados de uno exacto hasta cierto punto.
- **Comparación con la solución óptima** : Un algoritmo exacto podrá ser ineficiente, sin embargo, la medida que se utiliza para asociar o no la eficiencia, como se vera mas adelante, no implica que en todos los casos del algoritmo, se tenga un tiempo no razonable, puede ocurrir que se puedan conseguir algunos resultados en un tiempo aceptable. De esta manera, la comparación de los resultados de un algoritmo heurístico, se harían con los resultados de estos casos o ejemplos del algoritmo exacto.

- Comparación con una cota: Se comparan los resultados de un heurístico, con un valor de referencia que se sabe es bueno o cercano a la solución.
- Comparación con otros algoritmos heurísticos.

3.2 Complejidad

La resolución de un problema, se puede abordar, como la construcción de un modelo que permite definir un procedimiento para resolverlo. Lo interesante es que exista un modelo para el cual se puedan definir procedimientos que resuelven variados problemas y de diversas formas, pues de esta manera, se podrían introducir formas de medir la dificultad de estos problemas resueltos en este modelo.

De esta manera si M es una máquina (modelo) "configurable", en el sentido de que se cambian ciertos factores α en ella pero sigue siendo la misma máquina, la cual puede resolver cualquier problema de una colección $\mathcal{P}$ ajustando estos factores, ofreciendo para cada posible solución X para un problema $P \in \mathcal{P}$ una respuesta 0 o 1 (No es solución o si lo es).

Si consideramos que M entrega 0 o 1 en una cantidad de "pasos" u operaciones internas, podríamos decir que la dificultad de resolver un problema P es la cantidad de pasos que se tarda, o la cantidad de operaciones internas que necesita M para entregar 0 o 1. Ahora bien, uno puede pensar que para P hay muchos posibles candidatos X que pueden o no ser solución, por esta razón, se define que la dificultad del problema P es la cantidad de pasos que demora M en entregar 0 o 1 para la posible solución que más pasos cuesta.

Dicho esto, es intuitivo pensar que si M ajusta sus factores de dos formas α y β para resolver un problema P, tomemos el caso de M que menos pasos cuesta para la posible solución mas costosa. Ademas, es intuitivo pensar, en clasificar a los problemas en función de la cantidad de pasos, es decir, si $P_1, ..., P_k$ son resueltos en una misma cantidad de pasos T, entonces $P_1, ..., P_k$ están en la clase de problemas con dificultad T.

Ahora bien, la máquina M representa un caso particular, pues M verifica que cierto conjunto de posibles soluciones X es solución o no lo es, por tanto, mas

que resolver un problema P en tiempo T, se diría que verifica P en un tiempo T. Sin embargo, podríamos considerar perfectamente una máquina M' que no entrega como salida un 0 o un 1 para no es solución o si lo es, si no mas bien, que para un problema P, cierto ajuste de factores α' y entrada Y, entrega una salida X que es la solución al problema. Con esto ultimo, diríamos que M' resuelve el problema P en un tiempo T'.

Si se piensa que estas máquinas son automáticas, es decir, que por si solas después de una configuración previa entregan una respuesta, utilizando un cierto tiempo que se desprende del costo de tiempo del paso a paso, y que utiliza cierta "memoria" para realizar y guardar cálculos, se tiene lo siguiente:

- **Algoritmo**: Secuencia mecánica de pasos finitos y ordenados a seguir.

  El algoritmo es el paso a paso, mientras que el algoritmo implementado en la máquina, es el método de la máquina, aunque se suele hablar sin distinguir estos aspectos.

- **Complejidad algorítmica**: Tiempo y/o espacio necesarios para un algoritmo.

  Esto es, el tiempo de respuesta y la cantidad de memoria requerida, sin aun tener definido a que se refiere la memoria, o la capacidad de "guardar" datos.

- **Complejidad de problema**: Será una clase de problemas que son resueltos por un método o algoritmo, utilizando un tiempo y/o espacio necesarios. Esto es, se puede tener una clase C de problemas que son resueltos en un tiempo T, si existe un algoritmo o método para un problema P que lo resuelve en tiempo T entonces $P \in C$.

  De acá es intuitivo pensar en formalizar las clases de complejidad, formalizar un problema de modo que pueda pertenecer a una clase, y el tiempo y/o espacio asociado a la utilización de un método.

Ahora, lo interesante, es tener un modelo $\mathcal{M}$ que responde positivamente a las siguientes interrogantes:

- ¿$\mathcal{M}$ permite representar problemas que otros modelos representan? . Esto con la intención de que se generalice la representación de los problemas, es decir, conocer si no tiene limites de representación el modelo, teniendo en cuenta, que se podría pensar en un finito numero de problemas que representa un modelo, lo que en definitiva define una limitación.

- ¿$\mathcal{M}$ permite solucionar problemas que otros modelos solucionan? . Es interesante que el modelo, no solo represente otros modelos, si no que resuelva lo que esos modelos resuelven.
- ¿La dificultad de un problema P según el modelo $\mathcal{M}$, es similar a la dificultad de P en otros modelos?. Esto es, que la "simulación" de un modelo y un problema resuelto en el, no modifique (o si lo hace saber en cuanto) sustancialmente la medición sobre los algoritmos, de modo que la dificultad sea al menos "similar", al llevarla a $\mathcal{M}$

La pregunta es ¿ Existe tal modelo ?, la respuesta es que si, y en nuestro caso es llamado **Máquina de Turing** (MT).

3.3 Máquinas de Turing

Una cinta será una cantidad infinita de posiciones o espacios que permiten escribir un único símbolo de un conjunto de símbolos o alfabeto Σ, o alguno de los símbolos notables $\triangleright$ y $\square$, que permiten representar el inicio de la cinta y un espacio vació donde puede ir un símbolo.

Además, una cinta tendrá un cabezal $\triangledown$ que es capaz de leer un símbolo de una posición, escribir un símbolo en una posición y moverse en algún sentido de la cinta después de leer y/o escribir, mediante el paso de un espacio o posición a otra (un movimiento), o manteniéndose en la posición actual.

La interpretación gráfica de una cinta es la siguiente:

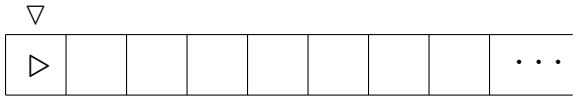


Fig. 3.7: Cinta inicializada
Fuente: Elaboración propia

Esta cinta está en un estado inicializado, es decir, tiene escrito el cursor, tiene el cabezal en el cursor, y ademas, toda posición posterior o a la derecha del cursor tiene escrito el símbolo $\square$, ahora, cuando se hable de cinta, se habla de una cinta inicializada, salvo que se diga otra cosa.

Siguiendo el ejemplo de la imagen, se puede dar el siguiente ejemplo:

Si $\triangledown$ lee el símbolo $\triangleright$, escribe el mismo símbolo $\triangleright$ y se mueve a la derecha, posterior a eso, cada vez que $\triangledown$ lea el símbolo $\square$, escribe el símbolo 1 (que debería estar en Σ), quedando la siguiente cinta:

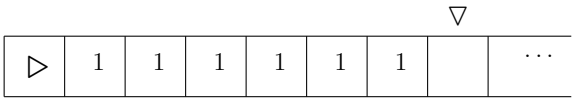


Fig. 3.8: Cinta con 111111
Fuente: Elaboración propia

Este seria el resultado, luego de 6 pasos definidos por leer $\square$, escribir 1 y moverse a la derecha.

Dicho todo esto, una máquina de Turing (MT) es una colección de $k+1$ cintas, donde se tiene una cinta CE de entrada en la que el cabezal únicamente puede leer y moverse y $k \geq 0$ cintas de trabajo CT_i con un cabezal normal. Formalmente, una MT es una $5-$tupla $M := (\Sigma, Q, q_0, F, \delta)$, donde:

- Σ: Es el alfabeto de símbolos, siendo $\triangledown$ y $\square$ auxiliares de Σ. (En los auxiliares, únicamente su semántica es esencial, que es lo se interpreta de ellos, mientras que los símbolos de Σ serán el contenido propio de usar una MT)
- Q : Es un conjunto de etiquetas finito $q_0, q_1, ..., q_{|Q|}$ que tienen relación, con cual es el símbolo que debe leer $\triangledown$ en un cierto instante. Por ejemplo, q_0 considerando que las $k+1$ cintas están inicializadas, lee $\triangleright$ en cada cinta, ese sería un estado, el primero. Básicamente un estado, es el contenido que se tiene previo a realizar un paso en la MT, teniendo en cuenta, que un paso en una MT es igual a un paso en cada cinta.
- q_0 : Es el estado inicial.
- F : Es el conjunto de estados, que finalizan la máquina, en otras palabras, que símbolos deben leerse en las cintas para que se termine el proceso de la máquina.
- δ : Es una función $\delta : (Q - F) \times \Sigma^{k+1} \to Q \times \Sigma^k \times \{-1, 0, 1\}^{k+1}$ que toma una tupla $(q, X_1, ..., X_k, X_{k+1})$ donde q es un estado no final y cada X_i es un símbolo, y retorna una tupla $(q', Y_1, ..., Y_k, Z_1, ..., Z_{k+1})$, que en q' tiene el estado actual después de haber leido los símbolos $X_1, ..., X_{k+1}$ en las $k+1$ cintas, $Y_1, ..., Y_k$ es lo que escribe en las cintas $CT_1, ..., CT_k$ respectivamente y $Z_1, ..., Z_{k+1}$ es el movimiento que realiza el cabezal luego de la escritura para cada cinta, pudiendo ser -1 izquierda, 0 centro o 1 derecha. Intuitivamente, la función de transición es la que gobierna el comportamiento de los cabezales.

Se suele definir como elemento adicional a la $5-$tupla, al conjunto C_M de configuraciones de la máquina:

- C_M : Es un conjunto de $2k+3-$tuplas de la forma $(q, X, Y_1, ..., Y_k, n_0, ..., n_k)$ con $q \in Q$,$X, Y_1, ..., Y_k \in \Sigma^*$ y $n_0, ..., n_k \in \mathbb{N}$, donde q es un estado de la MT, X es la cadena de símbolos que se tiene en la cinta CE para ese estado, cada Y_i es la cadena de símbolos que se tiene en la cinta CT_i para ese estado, y cada n_j es la posición que tiene el cabezal en cada cinta (n_0 para CE y n_j para CT_j con $j > 0$), siendo la posición del cursor 0 a su derecha la posición 1 y así sucesivamente.

Y ademas, se suele tener las siguientes consideraciones:

- Si la función de transición δ en alguna cinta lee el símbolo $\triangleright$, necesariamente escribe el mismo símbolo $\triangleright$ y se mueve a la derecha. Esto es para evitar que se elimine el inicio y se salga de la cinta.
- Si la función de transición δ en alguna cinta lee un símbolo $\neq \triangleright$, el símbolo a escribir debe ser $\neq \triangleright$. Esto para evitar que se ocupe el símbolo en otro espacio.

En la práctica, la cinta CE tendrá escrita una cadena de símbolos previamente, estando únicamente las cintas de trabajo en el estado inicial:

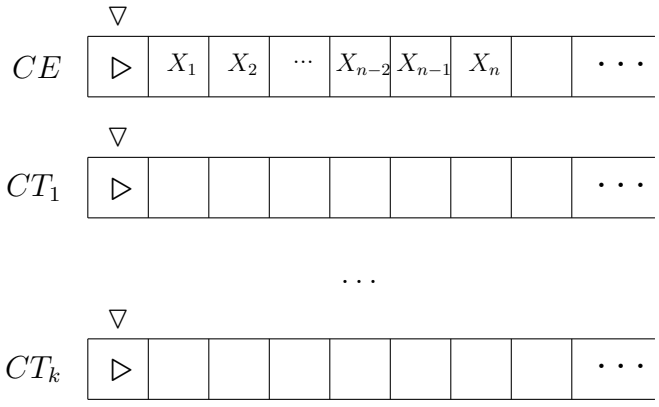


Fig. 3.9: Máquina de Turing con $k+1$ cintas con entrada $X = X_1...X_n$
Fuente: Elaboración propia

Teniendo definida la máquina de Turing, sera de interés de esta conocer inicialmente los siguientes elementos:

- **Configuración inicial**: La configuración $(q_0, \triangleright\sigma, \triangleright, ..., \triangleright, 0, ..., 0)$ se llama configuración inicial y se denota por $I(\sigma)$. Es importante aclarar que no se

escribe nada después de σ o después de $\triangleright$, pero hay que entender que de ahí vienen infinitos símbolos $\square$, simplemente no se escriben por simpleza, y se asume que se sobreentiende que están.

Ahora, dependiendo de cual es la configuración inicial $I(\sigma)$ y como ha sido definida la función δ, se puede obtener o no una salida o configuración final que tiene en la cinta CT_k una cadena Y.

- **Función de respuesta** : Se define una función $\text{Res}_M : L \subseteq \Sigma^* \to \Sigma^*$ para una máquina M, como la función que retorna a cada cadena X la salida Y si existe, siendo L lo que se llama lenguaje de la máquina.

  Cuando se habla de que puede o no haber una salida, se refiere a que existe la posibilidad de que la máquina se detenga después de una cantidad finita de pasos o no se detenga, así mismo, hay una cantidad infinita o una cantidad demasiado alta de pasos, que dan la intuición de que no se detiene, por la espera que implica.

  Acá se observa, que intuitivamente se alinea la ejecución de una MT con el concepto de algoritmo previamente definido, es decir, existe una secuencia de pasos mecánicos, que consisten en leer, escribir y moverse, lo que esta explicitamente indicado por la función de transición que gobierna.

- **Algoritmo**: Se trata de la secuencia finita de pasos que indican el procedimiento de cálculo de la imagen de la función de respuesta para una cadena de entrada.

- **Conjunto de parada**: El paso de una configuración inicial $I(X)$ para una cadena X, a una configuración final C mediante la aplicación del algoritmo de una MT M, se denota $I(X) \vdash C$. Luego, si por facilidad se considera que $\text{Final}(C) \equiv (C \in C_M$ tiene estado $q \in F)$, el conjunto de parada de M es $P(M) = \{X \in \Sigma^* \mid I(X) \vdash C, \text{Final}(C)\}$.

3.3.1 Complejidad algorítmica

La complejidad de un algoritmo será una clase de funciones, que permite caracterizar y acotar el crecimiento de la función de tiempo de las MT donde se implementa el algoritmo. Es a partir de la complejidad del mejor algoritmo disponible para un problema, que se asigna una clase de complejidad para el problema.

Ahora bien, para formalizar todo ello, siendo M una MT, primero es necesario definir lo siguiente:

- **Tiempo de computación de una cadena**: Para la entrada $X \in P(M)$, es la cantidad de veces que se ejecuta la función de transición, así mismo,

la cantidad de veces que se cambia de configuración al ir desde $I(X)$ hasta la configuración final (Considerando una posible repetición). Este se define mediante una función $t_M : P(M) \to \mathbb{N}$, en la que se incluye una dependencia sobre la cadena, debido a que la computación de X tendrá un tiempo determinado por ciertos factores relevantes de X, como el tamaño, pues al menos tiene que terminar de leerse una vez X, y el orden y símbolos presentes en la cadena.

Esta definición, asocia para una cadena X de cierto tamaño, la cantidad de pasos necesarios para que un algoritmo $\mathcal{A}$ dé una respuesta para X. Ahora, un calculo particular no es un buen representante, de los tiempos que es capaz de ofrecer M utilizando $\mathcal{A}$, pues, si se tiene un intervalo $[t_{min}, t_{max}]$, desde el tiempo mínimo al máximo obtenido para una cadena X cualquiera al usar $\mathcal{A}$ en M, se sabe que es lo que ocurre para el peor caso posible y el mejor caso posible de tiempos o pasos, e incluso, tener un tiempo promedio. Estos me permiten responder a las preguntas ¿ Me conviene utilizar M para resolver, si puede que mi entrada represente el peor caso? o incluso, ¿Me conviene usar M viendo el tiempo del caso promedio?, entre otras.

- **Tiempo de computación notables**: Sea $n \in \mathbb{N}$ la longitud de un problema de referencia , así mismo, la longitud de una cadena que se computa en una MT mediante un algoritmo y $T : \mathbb{N} \to \mathbb{N}$, se define lo siguiente:
  - T_M **peor caso**: $T_M(n) = \text{Max}\{t_M(X) : X \in P(M), |X| \leq n\}$, para cualquier n natural.
  - T_m **mejor caso**: $T_m(n) = \text{Min}\{t_M(X) : X \in P(M), |X| \leq n\}$, para cualquier n natural.
  - T_p **caso promedio** $T_p(n) = \frac{T_M(n)+T_m(n)}{2}$ o $T_p(n) = \frac{1}{|S|}\sum_{X \in S} t_M(X)$ si $S = \{X \mid |X| \leq n, \varphi(X)\} \subseteq P(X)$ es finito para alguna condición φ, o cualquier otra variación de promedio, para todo n natural.

Básicamente, la función T ofrecerá para cada n una cantidad de pasos notables (maximos, minimos o medios), lo que definirá una sucesión que describe la tendencia de aumento de la cantidad de pasos a medida que aumenta el tamaño de la cadena, o el tamaño del problema.

Por ejemplo, se puede tomar a T para el peor caso de las MTs M_1, M_2, M_3, teniendo la siguiente situación:

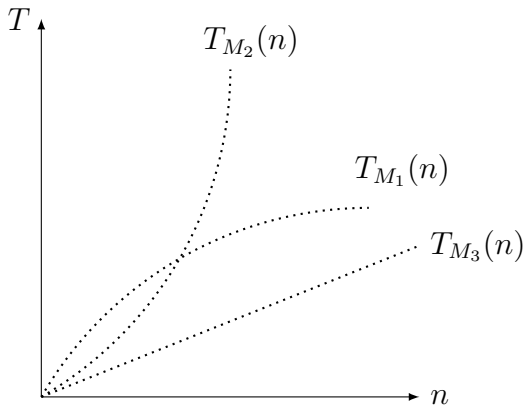


Fig. 3.10: $T(n)$ para los algoritmos de las máquinas M_1, M_2 y M_3
Fuente: Elaboración propia

El algoritmo que necesita menor cantidad de pasos es M_3, ahora, esto es para los primeros valores de n, pues como se aprecia en las gráficas, si se sigue la tendencia a medida que $n \to \infty$ es mejor el algoritmo de M_1. A pesar de que en general es mejor M_1, no impide que se utilice M_3 para los primeros valores en los cuales es mas rápido, es decir, se puede definir una sucesión a trozos, hallando el $m \in \mathbb{N}$ para el cual $T_{M_3}(n) = T_{M_1}(n)$ de modo que:

$$T(n) = \begin{cases} T_{M_3}(n) \text{ si } n \leq m \\ T_{M_1}(n) \text{ si } n > m \end{cases}$$

Define la función de tiempo, que resulta de utilizar para tamaños del problema $n \leq m$ la máquina M_3 y luego para problemas con $n > m$ la máquina M_1, así mismo, utilizar el algoritmo de M_3 y luego el algoritmo de M_1. Ahora bien, viéndolo del punto de vista de las MTs, se tendría que tener una manera de comprobar cuando el problema es de un tamaño de modo que convenga usar una maquina u otra, lo mismo para un algoritmo en computadora moderna.

Notación asintótica

La notación asintótica, va a permitir caracterizar el comportamiento de las funciones de tiempo en MTs, permitiendo asociar a cada MT y su algoritmo, un crecimiento representativo cuando $n \to \infty$. Dicho esto, sea $T : \mathbb{N} \to \mathbb{R}_0^+$ una función creciente, se define lo siguiente:

- **Cota superior asintótica**: g es la cota superior asintótica de T si y sólo si $\exists(n_0, c) \in \mathbb{N} \times \mathbb{R}^+$ tal que $\forall n \geq n_0$ se tiene que $cT(n) \leq g(n)$. Básica-

mente, una cota superior asintótica, es una función que sirve como vara de medida, respecto al crecimiento que tiene una función, en el sentido de que, se asegura que la función (T en este caso), no superara los valores que tiene la cota g cuando $n \to \infty$ principalmente.

- **Cota inferior asintótica**: g es la cota inferior asintótica de T si y sólo si $\exists(n_0, c) \in \mathbb{N} \times \mathbb{R}^+$ tal que $\forall n \geq n_0$ se tiene que $cg(n) \leq T(n)$. Este es el caso opuesto a la cota anterior, es decir, en este caso, se asegura que la función (T en este caso) no estara por debajo de una función o cota g para valores $n \to \infty$.
- **Cota ajustada asintótica**: g es cota ajustada asintótica de T si y sólo si g es cota superior e inferior asintótica de T.

Dicho lo anterior, se podría pensar en una clase de funciones que satisfacen la condición de cota asintótica para una función f, lo que es evidente si consideramos que una función es claramente una cota asintótica de si misma, y existe una especie de orden, pues, si una función tiene por cota superior asintótica a g, toda función que está sobre o por encima de g sera cota superior asintótica de f, del mismo modo si g es cota inferior asintótica, toda función que está sobre o por debajo de g sera cota inferior asintótica. Dicho esto, se define:

- $O(g(n)) = \{T(n) \mid \exists(n_0, c) \in \mathbb{N} \times \mathbb{R}^+, \forall n \in \mathbb{N}(n \geq n_0 \to cT(n) \leq g(n))\}$.
- $\Omega(g(n)) = \{T(n) \mid \exists(n_0, c) \in \mathbb{N} \times \mathbb{R}^+, \forall n \in \mathbb{N}(n \geq n_0 \to cg(n) \leq T(n))\}$.
- $\Theta(g(n)) = \{T(n) \mid T(n) \in O(g(n)), T(n) \in \Omega(g(n))\}$.

Estas clases se llaman respectivamente **Big**$-O$, **Big-**Ω y **Big-**Θ en la literatura, y no son las unicas clases que existen, también se suelen definir **little-**o para la cota superior asintótica estrictamente mayor, **little-**ω para la cota inferior asintótica estrictamente menor, y de manera análoga para **little-**θ, simplemente, reemplazando el orden usual $\leq$ por el orden estricto $<$ en las definiciones.

Algunas propiedades de las notaciones asintóticas que serán de utilidad, con $\varphi(f(n)) \equiv \varphi(f)$ siendo $\varphi \in \{O, \Omega, \Theta\}$ para facilitar la notación, son las siguientes :

Para cualquier función f entonces $f \in \varphi(f)$.

$f \in O(g) \to O(f) \subseteq O(g)$.

$f \in \Omega(g) \rightarrow \Omega(f) \subseteq \Omega(g)$.

$f \in \Theta(g) \rightarrow \Theta(f) = \Theta(g)$.

$\varphi(f) = \varphi(g) \leftrightarrow f \in \varphi(g), g \in \varphi(f)$.

$f \in \varphi(g), g \in \varphi(h) \rightarrow f \in \varphi(h)$.

$f \in O(g), O(h) \rightarrow f \in O(\text{Min}(g, h))$.

$f \in \Omega(g), \Omega(h) \rightarrow f \in \Omega(\text{Max}(g, h))$.

$f_1 \in O(g), f_2 \in O(h) \rightarrow f_1 + f_2 \in O(\text{Max}(g, h))$.

$f_1 \in \Omega(g), f_2 \in \Omega(h) \rightarrow f_1 + f_2 \in \Omega(g + h)$.

$f_1 \in \Theta(g), f_2 \in \Theta(h) \rightarrow f_1 + f_2 \in \Theta(\text{Max}(g, h))$.

$f_1 \in O(g), f_2 \in O(h) \rightarrow f_1 f_2 \in O(gh)$.

Sea $\lim_{n\to\infty} \frac{f(n)}{g(n)} = L$, se tiene:

$L \neq 0, L < \infty$ entonces $\varphi(f) = \varphi(g)$

$L = 0$ entonces $O(f) \subset O(g)$ $(g \notin O(f))$, $\Omega(g) \subset \Omega(f)$ $(f \notin O(g))$ y los ordenes exactos son distintos.

Ahora, para poder asociar una complejidad a los algoritmos, se seleccionara una lista de funciones notables, y con ello una lista de clases con notación O, teniendo en cuenta los siguientes puntos:

- Existe una relación de equivalencia $f \sim_O g \leftrightarrow O(f) = O(g)$, con ello se definen clases de equivalencia, pudiendo escoger una función f tal que $O(f)$ es representante de todas aquellas para que contienen o son cotas superiores asintóticas de las mismas funciones que f.
- Para las constantes $\alpha, \alpha_i \in \mathbb{R}$ y las funciones f_i con $i = 1, ..., k$ se cumple:

  $O(\alpha) = O(1)$
  $O(\sum_{i\in[k]} \alpha_i f_i) = O(\text{Max}(f_i)_{i\in[k]})$

Esto se traduce, en que O funciona como un operador sobre las funciones en su 'interior', seleccionando la función de mayor crecimiento cuando $n \rightarrow \infty$ y no considerando las constantes. De esta manera, si $f_i = n_i$ con $i = 0, 1..., k$ entonces $O(\alpha_0 + \alpha_1 n + ... + \alpha_k n^k) = O(n^k)$, probando no solo la igualdad, si no también,

que el representante mas simple cuando $n \to \infty$ para funciones polinomicas, es n^k, lo que se puede extender fácilmente a otras funciones.

Dicho lo anterior, la complejidad de un algoritmo de función de tiempo $T(n)$, es la primera clase que tiene por función notable una cota superior asintótica de T, de la siguiente lista.

Clase	Orden
$O(1)$	Constante
$O(\log(n))$	Logarítmico
$O(n\log(n))$	Quasilineal
$O(n^k)$	$(k \geq 1)$ Polinomial
$O(a^n)$	Exponencial $(a \geq 2)$
$O(n!)$	Factorial
$O(a^{n^k})$	Expo-polinomial $(a \geq 2, k \geq 2)$
$O(\mathcal{E}(a^k, r))$	$r-$exponencial $(a \geq 2, k \geq 1)$

Tab. 3.4: Ordenes de complejidad algorítmica

Siendo $\mathcal{E}(X^Y, n) = X^{\mathcal{E}(X^Y, n-1)}$ con $\mathcal{E}(X^Y, 0) = Y$. Estas clases son algunas de las existentes, sin embargo, queda claro que cualquier clase adicional satisface la condición de notable, al ser un representante de un clase de funciones que son cotas de una misma clase de funciones, y además, obedece al representante mas simple cuando $n \to \infty$.

3.3.2 Máquina de Turing no determinista

Una MT no determinista con $k+1$ cintas es una $5-$tupla $(\Sigma, Q, q_0, F, \delta)$, donde Σ, Q, q_0, F obedecen a la definición dada para máquina determinista, y δ generaliza su definición de función perdiendo la unicidad de imagen, de manera que, para al menos algún estado $q_i \notin F$ y una sucesión de símbolos $x_0, ..., x_{k+1}$ donde x_i se lee en la cinta $i-$sima, existe un conjunto no unitario de salidas para $\delta(q_i, x_0, ..., x_{k+1})$. Existe un proceso de selección o adivinación de la salida que le corresponde a este punto, es decir, no existe una lectura y escritura para cada

posible salida en simultaneo o secuencial, si no que, existe una salida que define un cambio a una nueva configuración que excluye a las otras que se pudieron tomar, durante la computación o ejecución en curso.

Si se define un digrafo G que tiene por vértices a las configuraciones por las que pasa una MT determinista M al tener de entrada a X desde la inicial $I(X)$, siendo el paso por una arista, el cambio hacia una configuración sucesora, mediante la utilización de una transición $\cdot \to \delta(\cdot)$, se tiene el siguiente árbol:

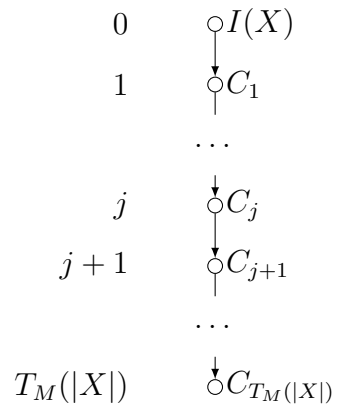


Fig. 3.11: Camino de computo de una MT determinista

Se observa un árbol $1-$ario en el que las alturas a la izquierda, concuerdan con la cantidad de pasos del algoritmo para encontrar la salida de X (si existe). Si se replica esta idea, pero siendo el estado de la configuración C_j, un estado con t posibles salidas mediante la transición, se tiene el siguiente resultado:

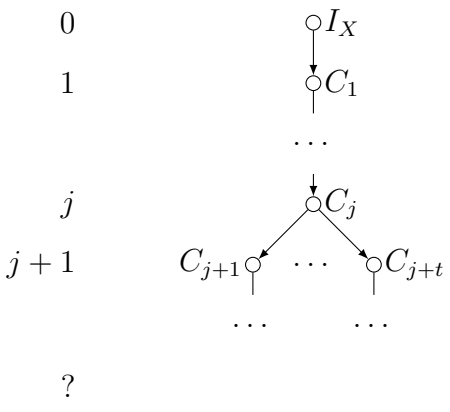


Fig. 3.12: Árbol de computo de una MT no determinista

Lo que se observa, es que al llegar a la configuración C_j, se adivina o selecciona cual es la siguiente configuración de entre $C_{j+1}, ..., C_{j+t}$ de acuerdo a la función de transición de una MT no determinista, definiéndose t caminos posibles. Lo que define a su vez, un árbol de una altura no fija a diferencia del caso anterior, pues, dependerá de cual es el recorrido que se toma por el digrafo.

Ahora, es necesario notar, que la situación planteada solo nos dice que existen al menos t caminos, puede ocurrir que en algún camino se vuelva a repetir esta situación para algún otra configuración. Lo interesante es que cada camino posible queda definido como:

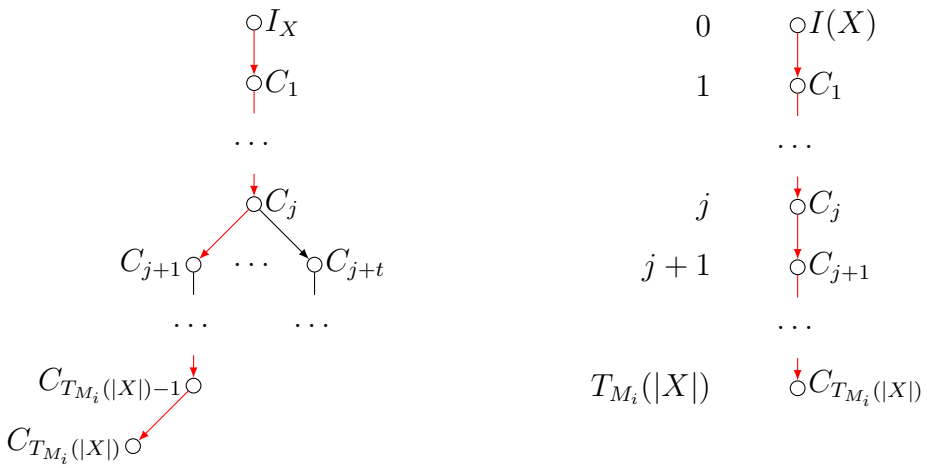


Fig. 3.13: Camino particular de una MT no determinista

Es decir, si para las configuraciones que tienen mas de una salida se fija una única salida mediante la transición asociada por δ, lo que se define es el computo de una MT determinista, con ello, las MTs deterministas, pueden ser vistas como una caso particular de una MT no determinista.

Cuando se habla de proceso de selección o adivinación, existen variadas maneras de formalizar esta situación, desde establecer una probabilidad que se reparte entre los candidatos a configuración sucesora o salida de una transición, lo que pasaría a llamar la MT como probabilística, hasta definir una MT que tiene una parte no determinista y otra determinista, simplemente, tomando nota de la ejecución de la MT no determinista, para en definitiva determinar el camino

que se tendría en esa ejecución, y luego repetirlo directamente mediante pasos deterministas, lo que llamaríamos MT adivinatoria.

3.3.3 Complejidad algorítmica no determinista

Según la definición de algoritmo, el procedimiento de una MT no determinista no puede llamarse algoritmo, puesto que, no existe una secuencia de pasos mecánica en la que no existe ambigüedad. Es por ello que se introducen los algoritmos no deterministas, que directamente se definen a partir de una MT no determinista, como el paso a paso de una MT que permite determinar un valor de una colección de posibles, mediante un camino de los posibles definidos por el árbol de computo de la MT.

Dicho esto, para una entrada X en una MT no determinista M, la complejidad del algoritmo no determinista inherente, es $O(T_M(|X|))$ si a lo sumo la profundidad o altura del arból de computo de M es $T_M(|X|)$.

3.3.4 Problema de decisión

Una MT determinista con función δ_D, se suele describir a partir de una tabla $Q \times \Sigma^{k+1}$ con $|\Sigma^{k+1}| = r$ columnas y $|Q| = s+1$ reglones. Si los estados son enumerados por $q_0, ..., q_s$ con q_i^* si q_i es final, y las cadenas por $X_1, ..., X_r$ entonces en la posición ij se tiene $\delta(q_i, X_{1j}, ..., X_{k+1j})$ siempre que exista. Para el caso de una MT no determinista, la distinción esta en una función de transición δ_{ND} de modo que, en la posición ij se tendrán todas las posibles salidas. Por ejemplo:

> Sea $\{a, b, 0, 1\}$ el alfabeto con $\square$ un símbolo auxiliar, de 2 cintas de lectura. Se tiene la siguiente tabla para una M determinista:

$q_i \backslash X_j$	a	b
q_0	$(q_1, 1, 1, 0)$	$(q_3, 0, 1, 0)$
q_1	$(q_2, 1, 1, 0)$	$(q_0, 1, 1, 0)$
q_2	$(q_2, 1, 1, 0)$	$(q_3, 0, 1, 0)$
q_3*	$(q_3, 0, 1, 0)$	$(q_3, 0, 1, 0)$

Tab. 3.5: Ejemplo de tabla de estados

Esta tabla corresponde a una MT determinista que para cadenas X con $|X| > 0$ formadas por $\Sigma = \{0, 1, a, b\}$ entrega como respuesta en la cinta de trabajo, 1 si la cadena es de la forma $(ab)^n a^m$ para algún $n \geq 0, m \geq 1$ y 0 si la cadena en caso contrario. En otras palabras, verifica que la palabra o cadena de ingreso este en $L = \{(ab)^n a^m \mid n \geq 0, m \geq 1\}$.

De esta manera, si se ingresa la palabra *ababaaa* se obtiene como respuesta 1 lo que es evidente.

De la tabla de estados de un MT, se puede construir un digrafo G que llamamos diagrama de estados de la MT, donde el conjunto de vértices $V(G)$ es el conjunto de estados de la máquina, con un símbolo $\triangleright$ para estado inicial junto al vértice y un circulo adicional al vértice de mismo centro para indicar que es final, cada arista tiene una (o mas si MT no es determinista) etiqueta $X \mid Y \mid m_1, ..., m_{k+1}$ donde X es la cadena que se lee con X_i en la cinta $i-$esima, Y es la cadena que se escribe con Y_i el símbolo que se escribe en la cinta $i-$sima y cada $m_j \in \{-1, 0, 1\}$ es una etiqueta que hace alusión a que hace el cabezal $j-$esimo.

Para la tabla antes mencionada, se tiene el siguiente sistema de transición:

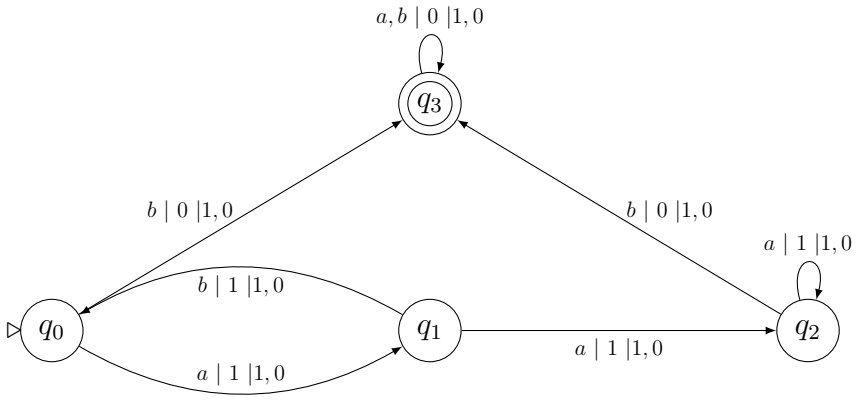


Fig. 3.14: Diagrama de estados de ejemplo

Dicho esto, es de interés representar un problema como un lenguaje L, en el que se tienen posibles soluciones formadas por símbolos de un alfabeto Σ, con la intención, de que se pueda verificar mediante una MT si una palabra en particular se encuentra en el lenguaje L o no, mediante la entrega de un símbolo 0 o 1 respectivamente para rechazar y aceptar. El lenguaje L, pasa a llamarse

problema, y la pregunta de si existe o no una MT que permita esta verificación, es lo que se conoce como problema de decisión.

Sea M una MT que como salida entrega un 1 o 0, si para toda palabra $X \in L \subseteq \Sigma^*$ cumple que $X \in P(M)$, se define:

- Lenguaje aceptado: Es el conjunto $L(M) = \{X \in \Sigma^* \mid \text{Res}_M(X) = 1\}$.
- Codificación: La codificación de un conjunto A mediante un alfabeto Σ, es una función inyectiva $f : A \to \Sigma^*$. Esta función sera relevante, puesto que permitirá representar los elementos de un conjunto cualquiera mediante una cadena de símbolos, por ejemplo, A podría tratarse del conjunto factible Ω de un problema de optimización, y su representación en cadenas de símbolos podría permitir verificar las soluciones. Aun más, la función de codificación podria ir a un modelo de computación que no corresponda con una MT, como en el caso de una computadora moderna, por ello, es que también se define $f : A \to \mathcal{B}$ tal que $f(a) = \lfloor a \rfloor$ siendo $\lfloor a \rfloor$ la representación de a en la computadora moderna, por ejemplo, seria $f : \mathbb{R}^n \to$ Arrays tal que $f(X_1, ..., X_n) = [x_1, ..., x_n]$, siendo $[x_1, ..., x_n]$ la representación de un vector en algunos lenguajes de programación.
- Problema recursivamente numerable: L es recursivamente enumerable si y sólo si existe una MT M de modo que $L = L(M)$. Intuitivamente, una MT podra verificar un conjunto si este es enumerable, pues debe poderse asignar un natural a cada miembro del conjunto, para que se pueda ir verificando uno a uno cada elemento. Recordemos que un conjunto que no es enumerable, no tiene algo como un sucesor definido, pues entre cualquier Y que se considere mayor que X, por muy cercano que esté, siempre habrá otro mas cercano. De este modo, esta definicion de enumerable es algo mas fuerte, que la definición usual de conjunto enumerable.
- Problema decidible : L es decidible o recursivo si y sólo si L y L^c son recursivamente enumerables (Para un universo Σ^*). Al ser enumerables ambos lenguajes y verificables, se puede tener certeza mediante un algoritmo, de cuáles son soluciones y cuáles no para un problema.

3.3.5 Complejidad de problemas

Para facilitar la notación de las clases de complejidad, se considera $M_\varphi\Sigma$ donde $\varphi \in \{D, ND\}$ siendo D determinista, ND no determinista y Σ el alfabeto. Dicho esto, se tiene la siguiente lista notable de clases de complejidad de problemas:

- Tiempo determinista (**DTIME**) : Es la clase de los lenguajes (problemas de decisión) para los cuales existe una MT determinista que verifica cuando

una entrada esta en L (cuando una entrada es aceptada en el problema) o no en un tiempo de orden $f(n)$ con f monótona creciente de $\mathbb{N}$ a reales positivos.

$$\mathbf{DTIME}(f(n)) := \{L \subseteq \{0,1\}^* \mid \exists M_D\{0,1\}, \mathrm{Res}_M = \mathcal{X}_L, T_M(n) \in O(f(n))\}$$

La expresión $\mathcal{X}_L$ es la función característica, tal que $\mathcal{X}_L(X) = 1 \leftrightarrow X \in L$ y 0 en otro caso. Esta clase se interpreta como la clase de problemas que se verifican en tiempo determinista $f(n)$.

Definido el tiempo determinista, se definen las siguientes clases en función de una cota superior de tiempo notable:

- (Polinomial)$\mathbf{P} := \bigcup_{k\in\mathbb{N}} \mathbf{DTIME}(n^k)$
- (Exponencial)$\mathbf{E} := \mathbf{DTIME}(2^{O(n)})$
- (Exponencial polinomial)$\mathrm{EXP} := \bigcup_{k\in\mathbb{N}} \mathbf{DTIME}(2^{n^k})$
- ($r > 1-$Expo polinomial) $r - \mathbf{EXP} := \bigcup_{k\in\mathbb{N}} \mathbf{DTIME}(\mathcal{E}(2^{n^k}, r))$

  Con $\mathcal{E}(X^Y, m) = X^{\mathcal{E}(X^Y, m-1)} = ... = X^{X^{X^{\cdots X^Y}}}$.

- Tiempo no determinista (**NTIME**) : Es la clase de problemas que se verifican en tiempo no determinista $f(n)$ con f monótona creciente de $\mathbb{N}$ a reales positivos, por una MT en binario.

$$\mathbf{NTIME}(f(n)) := \{L \subseteq \{0,1\} \mid \exists M_N D\{0,1\}, \mathrm{Res}_M = \mathcal{X}_L, T_M(n) \in O(f(n))\}$$

Definido el tiempo no determinista, se definen las siguientes clases en función de una cota superior de tiempo notable:

- (Polinomial) $\mathbf{NP} := \bigcup_{k\in\mathbb{N}} \mathbf{NTIME}(n^k)$
- (Exponencial) $\mathbf{NE} := \mathbf{NTIME}(2^{O(n)})$
- (Exponencial polinomial) $\mathbf{NEXP} := \bigcup_{k\in\mathbb{N}} \mathbf{NTIME}(2^{n^k})$
- ($r > 1-$Expo polinomial) $r - \mathbf{EXP} := \bigcup_{k\in\mathbb{N}} \mathbf{NTIME}(\mathcal{E}(2^{n^k}, r))$

- (Complementario-**C**) $\mathbf{Co} - \mathbf{C} := \{L \mid L^c \in \mathrm{C}\}$

La clase **P** y **NP** son fundamentales para la teoría de la complejidad, lo primero es comprender que la pertenencia a la clase **P** se puede dar de dos maneras, obviamente que $L \in \mathbf{P}$, y la otra es reducir un problema L a otro L' que es polinomial.

Reducción: Una función $f : A \to B$ con $A, B \subseteq \Sigma^*$ computable, es una reducción de A a B, si para todo $x \in \Sigma^*$ se tiene que $x \in A \leftrightarrow f(x) \in B$. Se denota la reducción como $A \leq B$

Dicho esto, es claro que L debe ser reducido por una función f de tiempo de computo polinomial a L' (lo que se denota como $\leq_p$), de esta manera, es que se probaría que L esta realmente en **P**, pues el tiempo de transformación es polinomial y el tiempo de resolución de L' es polinomial. Luego, se utiliza a la clase **P** para definir lo siguiente:

Problemas tratables: Un problema L es **tratable** si es posible resolverlo en tiempo polinomial determinista y es intratable en caso contrario.

La tratabilidad de un problema es un característica intrínseca. Hablamos de problemas tratables cuando pueden ser resueltos por un algoritmo para un tamaño de problema o n grande, e incluso mejorando hardware o el propio algoritmo que se utiliza para resolverlos, permite aumentar el rango o el tamaño n para los cuales se resuelven en tiempo razonable. Por el contrario, los problemas intratables son aquellos para los cuales solo (si existe) hay solución para n pequeños, y cambios o mejoras en los algoritmos conocidos y en el harware, no son sustanciales a la hora de resolverlos.

Respecto de la clase **NP**, este tipo de problemas tiene la principal característica de que si bien no pueden ser resueltos en un tiempo polinomial determinista, si se puede verificar una solución en tiempo polinomial determinista.

3.3.6 Problemas NP-Completos y NP-Duros

Para comprender la clasificación de tiempo polinomial no determinista completa y dura, se deben definir los siguientes conceptos:

Clase cerrada: $\mathcal{C}$ es cerrada si para todo $L \in \mathcal{C}$ vale que $L' \leq L \to L' \in \mathcal{C}$.

Lenguaje completo: $L \in \mathcal{C}$ es completo si para todo $L' \in \mathcal{C}$ se tiene que $L' \leq L$. Se dice que L es $\mathcal{C}$-Completo.

Lenguaje separador: El lenguaje $L \in \mathcal{C}_2$ es separador de las clases $\mathcal{C}_1, \mathcal{C}_2$ si $L \in \mathcal{C}_2, L \notin \mathcal{C}_1$.

Dicho esto, los problemas **NP**-completos serán problemas de decisión a los cuales se pueden reducir todos los problemas de **NP**. Luego, los problemas

NP-hard serán los problemas de optimización que tienen problema de decisión **NP**-completo.

De lo anteriormente dicho, y de las definiciones dadas, se pueden extraer varias cosas interesantes. La clase **P** es cerrada, pues todo problema reducible a **P** en tiempo polinomico sera un problema en **P**. Luego, si pensamos en el no determinismo generalizando un algoritmo determinista es claro que $\mathbf{P} \subseteq \mathbf{NP}$, no siendo claro por el contrario, cual de estas relaciones es la cierta, es decir $\mathbf{P} = \mathbf{NP}$ o $\mathbf{P} \subset \mathbf{NP}$.

Una manera de resolver esta incógnita proviene de considerar que los problemas **NP**-completos son candidatos a lenguajes separadores entre **P** y **NP**, lo que implica que si un problema **NP**-completo tiene solución polinomial entonces todo problema en **NP** tiene solución polinomial y es trivial que $\mathbf{P} = \mathbf{NP}$. Esto deja en evidencia la relevancia que tienen los problemas **NP**-completos, pues el estatus que se asigne a uno de ellos respecto la existencia o la certeza de la no existencia de algoritmos en tiempo polinomico, permitirá mostrar cual de las proposiciones antes mencionadas es cierta.

3.4 Problemas de optimización notables

3.4.1 Travelling Salesman Problem

Dado un conjunto de ciudades finito y costos de viaje entre pares de ciudades, encontrar la forma mas barata de visitar todas las ciudades sola una vez y volver al punto de partida.

Formalizando esta situación, se tiene un grafo G donde $V(G)$ es un conjunto de etiquetas de ciudad de cardinal n, $E(G)$ es un conjunto de pares de ciudades no ordenado y $f : E(G) \to \mathbb{R}$ es una función de asignación de de costo por viaje entre ciudades, se busca hallar una permutación α de $V(G)$ tal que $\sum_{i=1}^{n} f(V_{\alpha_i}, V_{\alpha_{i+1}}) + f(V_{\alpha_n}, V_{\alpha_1})$ sea mínima.

Si utilizamos la fuerza bruta, esto es, revisar toda combinación posible para el grafo G con un tamaño de problema n se tiene lo siguiente:

Si fijamos como vértice inicial $V_j := V_0$ para algún j al tomar $n = 1, n =$

$2, n = 3$ se hace intuitivo suponer que $n \to (n-1)!$, ahora bien, para probarlo se puede utilizar inducción, si suponemos que $n \to (n-1)!$, al agregar un vértice adicional o ciudad adicional en G, para cada camino ya existente existen n nuevas combinaciones que se derivan de introducir el vértice nuevo en alguna posición de la cadena de vértices que define el camino, con ello es claro que si $n \to (n-1)!$ entonces $n+1 \to n!$, luego como caso base se toma cualquier caso pequeño por ejemplo $n = 3$ y se termina la demostración.

Probado que la cantidad de caminos para n ciudades es $(n-1)!$, podemos pensar en un algoritmo de revisión de cada una de estas posibilidades, es decir, si cada permutación tiene n vértices y se tienen que revisar $(n-1)!$ caminos, el costo computacional en el peor de los casos, considerando como paso a paso en una computadora moderna un costo constante 1 y que tenemos que revisar todas las posibilidades para conocer el camino de costo mínimo, es de $O(n!)$.

Dicho esto, si se asume como ejercicio que el tiempo necesario para ejecutar la operación simple revisión de una solución es 0,01$[s]$, se tiene lo siguiente:

- $n = 10 \approx 10^{3,5}[s]$
- $n = 50 \approx 10^{62}[s]$
- $n = 100 \approx 10^{157}[s]$

El tiempo que lleva el universo existiendo es de aproximadamente $13 \cdot 10^9$ años lo que implica un tiempo de $4 \cdot 10^{17}[s]$. De esta manera, el tiempo necesario para resolver el TSP por fuerza bruta para $n \geq 50$ es superior al tiempo de existencia del universo. Es muy claro desde que se define el orden de complejidad que el problema es intratable, sin embargo, lo interesante es que al tener soluciones permutacionales pensándolo del punto de vista mas práctico (como se utilizara en el desarrollo), la comprobación o verificación de una solución es lineal y esto a su vez nos dice que el TSP es **NP**.

Mas aun, este problema es un problema al que se pueden reducir el resto de problemas, es decir, se trata de un problema **NP**-duro, pensando en su naturaleza, la necesidad de optimizar el recorrido al mínimo.

3.5 Heuristica

La heurística es el arte o la ciencia del descubrimiento, la cual en su aplicación en la resolución de problemas, basándose en el caso de las matemáticas pero extensible a las demás ciencias, consiste en la utilización de conjeturas en busca de una solución, sobre las cuales se busca aumentar o disminuir la credibilidad, por verificación de consecuencias y patrones de manera estratégica, o por observación y analogía al compararlas entre si u otras, con el objetivo, de que a través de estos razonamiento, se concluya con una solución.

Por ejemplo:

> Se puede pensar en el proceso de demostración de una formula f, que es la construcción de una sucesión $f_1, ..., f_n$ de formulas, donde cada f_i es obtenido de ciertas reglas, ahora bien, existen dos procesos asociados. El primero, es un proceso basado en conjeturas, que permita identificar cuáles y en que orden deben ir las formulas o expresiones que construyen la sucesión que permiten la demostración, y el segundo proceso, es la entrega formal del resultado, que es lo que técnicamente llamaríamos demostración.

Ahora, en la definición se habla de credibilidad, verificación, observación y analogía, estos son 4 pilares fundamentales de la aplicación de la heurística y del razonamiento plausible, siendo los siguientes algunos patrones definidos en [7](“Heurística, Hipótesis y demostración en Matemáticas”) que se relacionan con los 4 pilares mencionados:

- **Patrones inductivos**: La verificación de la consecuencia, hace que la conjetura sea mas creíble. Si se tiene una expresión $A \rightarrow B$, siendo A una conjetura, es decir, cuenta con una probabilidad o credibilidad asociada pero no es una certeza su valor de verdad, si se da la implicación mencionada, cuando B se verifica, así mismo es cierta, la credibilidad de A sube. Un ejemplo, podría ser considerar que un conjunto de números solo tiene números primos (conjetura), una consecuencia seria que al tomar un subconjunto de este conjunto todo numero es primo. Entonces, si se toma un subconjunto y concuerda con la consecuencia, esto es, todos los números son primos, no permite confirmar la conjetura, pero aumenta la credibilidad.
- **Verificación sucesiva**: La verificación de una nueva consecuencia cuenta más o menos si la nueva consecuencia difiere mas o menos de anteriores. Es decir, citando el caso anterior de números primos, si ahora el subconjunto

no es elegido por alguien, si no de manera aleatoria (subconjunto estricto) y también resulta que los números son primos, esto reafirma aun más la credibilidad de la conjetura, pues no esta sujeta a la arbitrariedad de alguien.

- **Verificación de consecuencias improbables**: La verificación de una consecuencia cuenta más o menos dependiendo de si es más o menos probable. Si se toma un conjunto unitario y este tiene un primo, aumenta menos la credibilidad si el conjunto que se toma es de 2, o 3, pues se acerca a que la conjetura sea cierta, todo número del conjunto es primo.
- **Inferencia por analogía**: Una conjetura se hace mas creíble si una conjetura análoga es verdadera.

La distinción que tiene la heurística, es que a pesar de que existen métodos, formas y procesos, se encuentra sujeta o tiene naturaleza no determinista, pues, las conjeturas dependerán de que o quien las crea, es decir, de la creatividad, experiencia y conocimiento. Para hacer esa distinción de manera mas concreta, se pueden establecer ciertas condiciones, por ejemplo, que el problema sea explicar un concepto, utilizando hojas para cuantificar un espacio utilizado y midiendo el tiempo para cuantificar la demora. Lo interesante, es que las respuestas que se dan, obedecen a una lógica, es decir, podrán haber respuestas aleatorias de quienes no tienen nada, otras respuestas podrán estar basadas en casos análogos o particulares antes ya definidos (Explicar complejidad usando la complejidad computacional por ejemplo), y en otros casos, se tendrá una definición más concreta.

Por ejemplo:

Puede que alguno tenga buenas conjeturas que permitan formalizar el concepto, pero el espacio y el tiempo no son factibles o adecuados, o puede que alguno ocupe muy poco espacio y tiempo pero no parece una respuesta plausible. Es de este punto de vista, que resulta de interés la heurística, es decir, pensar en ella como una forma de construcción de una solución, para determinar una respuesta conveniente en ciertas condiciones establecidas, por ejemplo, para el caso de tiempo y espacio como varas de medida, ¿ Será mejor utilizar menos espacio y tiempo en contraposición a menos credibilidad o garantía, o será mejor utilizar el máximo espacio y tiempo posibles para una credibilidad máxima? (Quedando claro que respuestas sin tiempo y espacio no son razonables del punto de vista práctico)

Otro aspecto relevante e importante de la heurística, es que genera una reducción de la complejidad de los problemas de manera estratégica, es decir, el problema se reduce a una colección de conjeturas sobre las cuales se intenta inferir una manera de resolver un problema, siendo las conjeturas atajos sobre la complejidad, que permiten responder en base a las limitaciones actuales, o en otros casos, en base a las consideraciones que por estrategia se disponen.

Dicho esto, una respuesta heurística a un problema se podría modelar, mediante el siguiente gráfico:

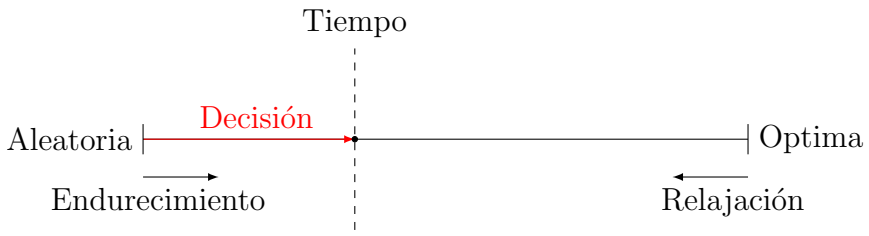


Fig. 3.15: Intervalo [Aleatoria , Óptima]
Fuente: Elaboración propia

Acá se observa un intervalo que tiene por extremos una repuesta aleatoria y una respuesta óptima, entendida como la mejor respuesta que se puede tener. El endurecimiento es el mecanismo de aumento de credibilidad (aumento de garantías y tiempo), y la relajación, el mecanismo de reducción de la credibilidad (disminución de garantías y tiempo). En este caso en particular, se muestra una decisión en rojo, se puede pensar en esa flecha en rojo, como una acumulación de consideraciones para tomar la decisión en el tiempo indicado del punto de vista humano, al tratar de conjeturar sobre una repuesta, siendo claramente una decisión heurística, según lo que se ha planteado, donde existe un segmento de recta de consideraciones no realizadas, que sería lo que se ha reducido la complejidad del problema para decidir. Por el contrario, tomar una recta desde a respuesta óptima en sentido hacia la aleatoriedad, representa una intención de estrategicamente no considerar ciertos elementos, descartar ciertos elementos de un gran sistema que permite obtener esa respuesta óptima, para reducir el tiempo de respuesta.

3.6 Metodos heurísticos

Si pensamos en la heurística de una manera más especifica, de tal manera que la recta que se ha utilizado para describir las respuestas heurísticas o intermedias entre la óptima y la aleatoria, se utiliza para asociar una solución a un problema de optimización, de modo que, la solución aleatoria no tiene ninguna garantía de que se optimice el problema, la solución óptima es la que tiene total garantía de que optimiza, y toda respuesta intermedia o heurística, es una solución que puede o no optimizar el problema, con cierto grado de credibilidad en función del proceso de obtención de esa respuesta.

El proceso de obtención de las soluciones, se hará mediante un algoritmo, que a medida que trata de capturar los aspectos relevantes de la solución a un problema, añade complejidad y endurece la respuesta que se obtiene. A continuación, se da una clasificación no excluyente para asignar un tipo a los algoritmos heurísticos a partir de [1] ("Algoritmos Heurísticos en Optimización"), teniendo en cuenta que generalmente éstos se utilizan sobre problemas en específico y responden a la lógica interna del problema:

Métodos constructivos: La solución se va construyendo a lo largo del procedimiento, dependiendo fuertemente de la estrategia seguida.

- Estrategia voraz: Se va construyendo la solución estratégicamente paso por paso, añadiendo el constituyente que representa la mayor mejora para la solución parcial actual.
- Estrategia de descomposición: Se utiliza la estrategia divide y vencerás, para reducir sistemáticamente un problema de optimización generalmente de manera recursiva en subproblemas triviales.
- Métodos de reducción: Se extrapolan características de buenas soluciones conocidas, para asumir que la solución optima y en general las buenas soluciones tienen esas características.
- Manipulación del modelo: Se extrapolan los resultados obtenidos en una simplificación del original, ya sea por simplificación de la función objetivo, por el añadido de restricciones que simplifican el problema, u otros casos.

Métodos de búsqueda: Desde una solución inicial, se busca mejorarla, teniéndose las siguientes:

- Búsqueda local: Se selecciona, o bien la primera o bien la mejor solución presente en la vecindad de la solución actual, ya sea la inicial o alguna posterior ya seleccionada.
- Búsqueda aleatorizada: Se selecciona de manera aleatoria, ya sea solo aleatoria o combinada con alguna estrategia, en la vecindad de la solución actual una nueva solución actual.

Cuando se establece un marco general en el que se hace dependiente a las heurísticas de ciertos parámetros, procedimientos o más generalmente, de valores o definiciones particulares propias de los problemas, para las cuales sin embargo, existe un lineamiento, una descripción clara de a que se refieren, de tal modo, que bajo ese marco de naturaleza heurística, se permite abordar una colección de problemas, se habla de una metaheurística.

Las metaheurísticas ademas, cuentan con una clara distinción de las heurísticas del punto de vista de los resultados obtenidos para un problema, pues, son utilizadas principalmente como procedimientos o métodos que permiten escapar o evitar soluciones optimas locales. Si bien no se tiene consenso acorde a una clasificación de metaheurísticas, se suele dar las siguiente clasificación no excluyente:

Inspiración

- Con inspiración: Se utilizan metáforas de fenómenos de la naturaleza para definir al método.
- Sin inspiración: Se utilizan propiedades matemáticas.

Número de soluciones

- Poblacional: Se utiliza un conjunto de soluciones en la búsqueda de óptimos.
- Trayectorial: Se utiliza una sola solución que se mejora en la búsqueda del optimo.

Función objetivo

- Estática: Se mantiene la misma función objetivo.
- Dinámica: Se modifica o cambia la función objetivo.

Vecindades

- Única: Se utiliza una vecindad para definir en ciertos puntos en el proceso, para definir conjuntos de puntos cercanos a estos puntos en especifico.

- Mas de una: Se utiliza mas de una vecindad para definir al conjunto de puntos cercanos de ciertos puntos a lo largo del proceso.

Memoria

- No utiliza memoria: Solo se utilizan los valores anteriores.
- Utiliza memoria: Es necesario utilizar estructuras que almacenan y permiten recurrir a sus valores durante el procedimiento.

3.6.1 Búsqueda local

Es una heurística determinista, que de manera iterativa mejora una solución inicial hasta encontrar una solución optima (local o global). Básicamente, el algoritmo toma una solución actual X_i para una iteración i, busca de manera aleatoria en una vecindad de X_i denotada por N_i, soluciones candidatas a nueva solución actual, decidiendo mediante un mecanismo de selección, cual es la nueva solución X_{i+1}, con la que se repite el proceso.

La construcción de un algoritmo de búsqueda local, puede seguir el siguiente esquema:

- **Definición de vecindades**: Para una solución X_i, se distribuye uniformemente la probabilidad de seleccionar una solución X_{i+1} en N_i, por ello, es fundamental la definición de la vecindad, ya que esta determinara como es que se lleva a cabo tal distribución uniforme, sumándole la propia definición de los valores que toman las soluciones como condicionante (No es lo mismo distribuir números discretos que continuos).

  Para el algoritmo, se debe definir una sucesión $N_0, ..., N_M$ de vecindades respectivas de las soluciones $X_0, ..., X_M$ seleccionadas, suponiendo M iteraciones, teniendo en cuenta dos enfoques:

  - Constante: La sucesión de vecindades, se obtiene de una única vecindad, que va definiendo instancias sobre los puntos $X_0, ..., X_M$. Esto es, sea $N(x)$ una vecindad para el punto x variable, la definición de la sucesión, no sera mas que la evaluación de cada X_i esto es $N_i = N(X_i)$.
  - Variable: Se definen $N_1(x), ..., N_M(x)$ vecindades para la variable x, la sucesión se puede definir como $N_i = N_j(X_i)$.

  Dicho esto, considerando que $X_j^{(i)}$ representa la $i-$esima coordenada de la solución $j-$esima, se tienen las siguientes maneras de definir una vecindad acorde a los diferentes tipos de valores de las soluciones:

- Vecindad discreta: Si los valores que pueden tomar las variables de decisión son discretos, la vecindad N_j de la solución $j-$esima es finita para cualquier j, con ello se define que $X_{j+1} \sim U(S \in N_j)$ y la probabilidad para cada punto en N_j es $\frac{1}{|N_j|}$. Dicho esto, se tienen las siguientes vecindades notables:
  - Vecindad inmediata: Se define un $r \in \mathbb{N}^+$ y se hace $X_{j+1} \sim U(X_j^{(i)} - r, X_j^{(i)} + r)$ para definir $N_j = [X_j^{(i)} - r, X_j^{(i)} + r]_{i\in[n]}$, siendo $X_{j+1} = (X_j^{(i)} + \lceil(2r+1)\lambda - r - 1\rceil)$ con $\lambda \in (0,1)$ aleatorio. Se podría hacer $\lambda = \lambda_i$ para definir uno aleatorio por coordenada o en efecto $r = r_i$.
  - Vecindad sin centro: $N_j = \Omega - \{(X_1, ..., X_n)\}$, siendo Ω el conjunto factible. Se construyen $(\frac{i}{|N_j|-1}, \frac{i+1}{|N_j|-1}]$ con $i = 0, ..., |N_j| - 2$, de modo que, se selecciona X_{j+1} si $\lambda \in (\frac{j}{|N_j|-1}, \frac{j+1}{|N_j|-1}]$ para $\lambda \in (0,1)$ aleatorio.
- Vecindad continua: La lógica es igual que en el caso anterior, pero utilizando una distribución uniform continua. En este caso, se necesita al menos un intervalo por coordenada, para distribuir de manera continua en ese intervalo.
  - Vecindad inmediata: Si $X^{(i)} \in [a_i, b_i]$ se hace $X^{(i)} \sim U(a_i, b_i)$ y se tiene $X_{j+1} = ((b_i - a_i)\lambda + a_i)$ para que $X_{j+1}^{(i)} \in [a_i, b_i]$, o siguiendo el esquema inmediato en el caso discreto $X_{j+1} = X_j + (2\varepsilon\lambda - \varepsilon)$ para un $\varepsilon \in \mathbb{R}^+$, de modo que $X_{j+1}^{(i)} \in [X_j^{(i)} - \varepsilon, X_j^{(i)} + \varepsilon]$.
  - Vecindad mediante distribución normal: Siguiendo la idea de la vecindad inmediata se define $X_{j+1} = X_j + (\frac{b_i - a_i}{6} N(0,1))$ siendo $N(0,1)$ un valor en el eje X para una probabilidad aleatoria en $(0,1)$, según la distribución normal.

En general, para seleccionar una solución de la vecindad de N_j de X_j, se genera un numero aleatorio $\lambda \in (0,1)$ y se sigue alguno de los siguientes casos:

- Si $F(S)$ es la función de distribución (acumulada) de los $S \in N_n$ entonces se hace $F(S) = \lambda$ y se obtiene $F^{-1}(\lambda) = S$ como elemento seleccionado.
- Si $F(X_n, \lambda) = X_{n+1}$ es la función que permite definir al sucesor en función del actual, simplemente se utiliza λ.
- Si $N_j = \{S_1, ..., S_k\}$, se toma la probabilidad $\frac{1}{k}$ propia de la distribución uniforme, y sin importar el orden se construyen $(\frac{i}{k}, \frac{i+1}{k}]$ con $i = 0, ..., k-1$, de modo que, se selecciona S_{j+1} si $\lambda \in (\frac{j}{k}, \frac{j+1}{k}]$.

- **Mecanismo de selección**: Este determinara cual solución de la vecindad por medio de la búsqueda uniformemente distribuida, es la que se tomará como solución sucesora X_{j+1} de la actual X_j.

  - Primera selección: X_{i+1} sera la primera solución que mejora el resultado de X_i en $N(X_i)$.
  - Mejor selección: X_{i+1} es la mejor solución en $N(X_i)$. Claramente, esta solución deja de tener sentido si $|N(X_i)| \to \infty$, pues el costo computacional seria muy elevado, es por ello, que esta mejor selección se suele implementar de dos maneras, o bien para una vecindad finita con $|N(X_i)|$ un numero razonable, o bien seleccionar la mejor en un $\eta \subset N(X_i)$ finito, de manera estratégica, es decir, la mejor selección para una cantidad $|\eta| = k$ de candidatas evaluadas.

- **Definir la manera de determinar una mejora**: Dependiendo del problema de optimización, se deberá definir una manera de verificar y comparar soluciones. Esto se debe, a que en algunos casos el problema de optimización esta definido de tal manera, que el costo computacional no es aceptable para la cantidad de evaluaciones que se deben realizar a lo largo del algoritmo, o también, la evaluación tiene una dificultad, pues tiene una cantidad de restricciones o la forma de la función objetivo y las restricciones no se puede abordar de manera directa, mas concretamente de manera aritmética, necesitando recurrir a un método externo.

- **Criterio de parada**: Si el conjunto de soluciones posibles es un numero finito y razonable, se suele abordar el problema de manera exhaustiva, esto es, iterar desde la solución inicial hasta un optimo local o global, sin embargo, cuando la cantidad de soluciones de las vecindades no es finita o razonable, sera necesario adoptar criterios para detener el algoritmo. Esto se hace evidente, si consideramos que de llegar al óptimo, se requerirá revisar cada uno de los elementos de la vecindad.

  - Limitar el número de iteraciones: Se define un número I máximo de soluciones actuales, así mismo, existe una sucesión $(X_0, ..., X_I)$ donde X_I es la mejor.
  - Número de soluciones candidatas sin mejora: Se define un numero J máximo de candidatas que no mejoran la solución actual.
  - Porcentaje de la vecindad sin mejora: Se define un $\alpha \in (0, 1)$, de modo que, si de la vecindad actual existe un $\lfloor \alpha N(S) \rfloor$ de candidatas que no mejoran la solución, entonces se detiene el algoritmo.

Pseudocódigos

- **Calculo de una solución vecina**: Sea M el numero de coordenadas de las soluciones, R un numero entero positivo para definir un intervalo, ε un número real positivo para definir un intervalo, S el punto que se envía para el calculo de un vecino y $N(S)$ su vecindad, se tiene:

1: $r \leftarrow R$

2: $X \leftarrow (X[0], ..., X[M-1])$
3: $\lambda \leftarrow$ Numero aleatorio en $(0, 1)$.
4: Para $i \leftarrow 0$ hasta $M-1$ con paso 1 hacer:
5: $\quad X[i] \leftarrow S[i] + \lceil (2r+1)\lambda - r - 1 \rceil$
6: Fin Para
7: Retornar X

Se considera una vecindad inmediata que para cada $X^{(i)}$ define un intervalo $[X^{(i)} - R, X^{(i)} + R]$ e internamente uno $[X[i] - r, X[i] + r]$. Generalizando esta idea, pero para M valores enteros positivos, se tiene:

1: $R \leftarrow (R[0], ..., R[M-1])$
2: $X \leftarrow (X[0], ..., X[M-1])$
3: Para $i \leftarrow 0$ hasta $M-1$ con paso 1 hacer:
4: $\quad \lambda \leftarrow$ Número aleatorio en $(0, 1)$
5: $\quad X[i] \leftarrow S[i] + \lceil (2R[i]+1)\lambda - R[i] - 1 \rceil$
6: Fin Para
7: Retornar X

Es un procedimiento análogo al caso anterior, pero definiendo intervalos $[X[i] - R[i], X[i] + R[i]]$ de manera interna. Si se recibe la vecindad $N(S)$, siendo ademas finita, se tiene:

1: $q \leftarrow 0$
2: $N \leftarrow$ Tamaño de $N(S)$
3: $\lambda \leftarrow$ Numero aleatorio en $(0, 1)$
4: $i \leftarrow 0$
5: Mientras $(i < N)$ hacer:
6: $\quad q \leftarrow q + \frac{1}{N}$
7: $\quad$ Si $(\lambda \leq q)$
8: $\quad\quad S \leftarrow N(S)[i]$
9: $\quad\quad i \leftarrow N - 1$
10: $\quad$ Fin si
11: $\quad i \leftarrow i + 1$
12: Fin Mientras
13: Retornar S

El número q se define $\frac{i+1}{N}$ para la iteración i, de este modo cuando $\lambda \in (\frac{i}{N}, \frac{i+1}{N}]$, se obtiene el $i-$esimo elemento de $N(S)$. Para el caso de los reales, se tienen lo siguientes códigos:

1: $\varepsilon \leftarrow R$
2: $X \leftarrow (X[0], ..., X[M-1])$
3: $\lambda \leftarrow$ Numero aleatorio en $(0, 1)$
4: Para $i \leftarrow 0$ hasta $M-1$ con paso 1 hacer:
5: $\quad X[i] \leftarrow S[i] + 2\varepsilon\lambda - \varepsilon$

6: Fin Para
7: Retornar X

Se utilizan los intervalos $(X^{(i)}-\varepsilon, X^{(i)}+\varepsilon)$ distribuidos uniformemente para cada i. Si se construyen los intervalos $[a_i, b_i]$ para $i = 1, ..., M$, se tiene:

1: $X \leftarrow (X[0], ..., X[M-1])$
2: $\lambda \leftarrow$ Numero aleatorio en $(0, 1)$
3: Para $i \leftarrow 0$ hasta $M-1$ con paso 1 hacer:
4: $\quad X[i] \leftarrow (B[i] - A[i])\lambda + A[i]$
5: Fin Para
6: Retornar X

Donde $A = (A[0], ..., A[M-1])$ y $B = (B[0], ..., B[M-1])$ con M coordenadas, representan respectivamente mediante los pares $[A[i], B[i]]$ los pares $[a_i, b_i]$.

1: $X \leftarrow (X[0], ..., X[M-1])$
2: Para $i \leftarrow 0$ hasta $M-1$ con paso 1 hacer:
3: $\quad \lambda \leftarrow$ Numero aleatorio inverso en $N(0, 1)$
4: $\quad X[i] \leftarrow S[i] + \frac{B[i]-A[i]}{6}\lambda$
5: Fin Para
6: Retornar X

Finalmente, acá se tiene la selección de un candidato mediante un desplazamiento definido por la distribución normal $N(0, 1)$.

- **Algoritmo de búsqueda**: A continuación se especifican algunas definiciones generales de ciertos tipos del algoritmo de búsqueda local que se pueden dar en función de los criterios de parada, vecindades, y otros elementos. Considere que $\sim$ debe ajustarse al caso particular de interés, minimizar o maximizar, siendo "Calculo(S)" para algún S el valor que tiene al evaluarse en lo que se quiera optimizar, pudiendo en base a lo que se ha planteado, ser un procedimiento, una operación aritmética, entre otros.

1: $S \leftarrow$ Solución inicial
2: $N \leftarrow$ Tamaño de la vecindad
3: $i \leftarrow 1$
4: Mientras $(i < N)$ hacer:
5: $\quad$ Para cada $X \in N(S)$ Hacer:
6: $\quad\quad$ Si (Calculo(S) $\sim$ Calculo(X)) entonces:
7: $\quad\quad\quad S \leftarrow X$
8: $\quad\quad\quad i \leftarrow 1$
9: $\quad\quad\quad N \leftarrow$ Tamaño nueva vecindad
10: $\quad\quad$ Caso contrario:
11: $\quad\quad\quad i \leftarrow i + 1$

12: Fin si
13: Para cada
14: Fin Mientras
15: Retornar S

Este es el caso en el que las vecindades son finitas y se selecciona como solución candidata la que primero mejora el resultado actual. El iterador i se utiliza para determinar cuando ya no se puede mejorar S, así mismo, las N soluciones de la vecindad para la solución actual S no mejoran la solución y se termina el algoritmo.

1: $S \leftarrow$ Solución inicial
2: $N \leftarrow$ Tamaño de la vecindad
3: $i \leftarrow 1$
4: $Y \leftarrow \square$
5: Mientras $(i < N)$ hacer:
6: $Y \leftarrow S$
7: Para cada $X \in N(Y)$ Hacer:
8: Si (Calculo$(S) \sim$ Calculo(X)) entonces:
9: $S \leftarrow X$
10: Fin si
11: $i \leftarrow i + 1$
12: Si $(i == N \wedge Y! = S)$ entonces:
13: $i \leftarrow 1$
14: $N \leftarrow$ Tamaño nueva vecindad
15: Fin si
16: Para cada
17: Fin Mientras
18: Retornar S

En este caso, se selecciona el mejor de una vecindad finita, utilizando una solución auxiliar Y que permite comparar la solución actual en S con el anterior S en Y después de revisar toda la vecindad, si hubo cambio, ahora S es que la vez anterior y se reinician i y el tamaño de la vecindad (y la vecindad claro esta), en otro caso, no se reinicia i quedando con valor $|N(Y)| = N$ y se detiene el algoritmo.

1: $S \leftarrow$ Solución inicial
2: $N \leftarrow$ Tamaño de la vecindad
3: $i \leftarrow 1$
4: $j \leftarrow 1$
5: $J \leftarrow$Numero máximo de iteraciones
6: $Y \leftarrow \square$
7: Mientras $(i < N \wedge j < J)$ hacer:
8: $Y \leftarrow S$
9: Para cada $X \in N(Y)$ Hacer:

10: Si (Calculo(S) $\sim$ Calculo(X)) entonces:
11: $S \leftarrow X$
12: Fin si
13: $i \leftarrow i + 1$
14: Si ($i == N \wedge Y! = S$) entonces:
15: $i \leftarrow 1$
16: $N \leftarrow$ Tamaño nueva vecindad
17: Fin si
18: Para cada
19: $j \leftarrow j + 1$
20: Fin Mientras
21: Retornar S

Se define una combinación, donde se termina el algoritmo si se han revisado los elementos de la vecindad sin mejora, o en efecto, se ha podido mejorar la solución a lo largo de J vecindades, lo que se puede realizar análogamente con el caso de primera selección.

1: $S \leftarrow$ Solución inicial
2: $j \leftarrow 1$
3: $J \leftarrow$ Numero de candidatas sin mejora
4: Mientras ($j < J$) hacer:
5: $X \leftarrow$ SolucionVecindad(S)
6: $j \leftarrow j + 1$
7: Si (Calculo(S) $\sim$ Calculo(X)) entonces:
8: $j \leftarrow 1$
9: $S \leftarrow X$
10: Fin Mientras
11: Fin Mientras
12: Retornar S

El número de candidatas sin mejora, se puede definir directamente, o en otro caso, como se ha mencionado en los criterios de parada, se puede definir $J \leftarrow \lfloor \alpha N(S) \rfloor$. En este caso, se adopta un procedimiento de primera selección, donde se obtienen soluciones candidatas de una vecindad para S, siendo comúnmente este procedimiento, de una manera aleatoria, en la que se puesto como cota superior, un máximo de J soluciones "extraídas" o elegidas en $N(S)$ sin que se pueda reasignar un valor a S que mejore los resultados. Para el caso de mejor selección, se tendría que considerar un ciclo acotado por un numero que puede ser justamente J, teniendo en cuenta que en este caso mas general, $N(S)$ puede no ser finito.

1: $S \leftarrow$ Solución inicial
2: $j \leftarrow 1$
3: $I \leftarrow$ Numero de iteraciones
4: $J \leftarrow$ Numero de candidatas sin mejora

```
5:  Mientras (i < I) hacer:
6:      Para j ← 0 hasta J − 1 con paso 1 hacer:
7:          X ← SolucionVecindad(S)
8:          Si (Calculo(S) ∼ Calculo(X)) entonces:
9:              S ← X
10:         Fin si
11:     Fin Para
12:     i ← i + 1
13: Fin Mientras
14: Retornar S
```

3.6.2 Algoritmos genéticos (AG)

Un algoritmo genético es una metaheurística, basada en la metáfora de considerar las soluciones o conjunto posible de soluciones de un problema de optimización, como las entidades o individuos de un sistema evolutivo.

Un sistema evolutivo, está conformado por una población o colección de individuos, en la que existe diversidad, selección y descendencia, con el fin de producir una población o generación sucesora, que representa una evolución o mejora respecto la anterior, teniendo una forma de medir la aptitud o posibilidad de sobrevivencia de los individuos, a la evolución o selección natural.

- Diversidad: En la población debe haber la suficiente diversidad, que permita representar al total de individuos y sus diferencias, permitiendo que mediante la interacción se definen diversas diversas maneras de responder ante una vara de medida, aptitud o posibilidad de supervivencia.

- Selección: La selección es un proceso de discriminación, valorando a los mejores respecto un cierto interés. Esto para, buscar candidatos a ser padres o ser mutados, con el fin, de que sus descendientes representen una mejoría respecto sus predecesores. De esta manera, la selección sucesiva preservando una diversidad entre los individuos población a población, permite a lo largo de las poblaciones o generaciones, definir individuos cada vez mejores.

  Por ejemplo, se tiene una población de soluciones aleatoria (diversa y representativa) del conjunto posible de soluciones. Se puede tomar como vara de medida la evaluación de esas soluciones en una función a maximizar, de esta manera, lo que se busca es seleccionar de los mejores individuos o soluciones, para definir descendientes que preserven estas condiciones, con la idea de que con el pasar de generaciones o poblaciones se llegue al óptimo, o a un valor muy cercano a el, debido al traspaso de ciertas condiciones internas de las soluciones a lo largo de la descendencia.

Al mecanismo que permite medir la aptitud de un individuo en una población, se le llama función de fitness o simplemente fitness.

- Descendencia: Los individuos poseen una estructura genética inherente llamada genotipo, y su expresión sujeto al ambiente es llamada fenotipo. Llevándolo al caso humano, el genotipo es un cromosoma que contiene el material genético de un ser humano, y es por este, que se produce una expresión del ser humano en diversas situaciones, áreas, o formas, por interacción con el ambiente.

  La descendencia, o el paso de individuos a una nueva generación o población, se da mediante los siguientes procedimientos:

  - Cruce: Se combinan el material genético de dos individuos, dando una cierta cantidad de descendientes. Acá hay 3 elementos que destacar, dependiendo de que población se habla, sera necesario definir que es el material genético, en este caso comúnmente sera un cromosoma, sera necesario definir como ocurre la combinación, esto es, como se obtiene un cromosoma para un descendiente, y finalmente, definir la cantidad de descendientes en el cruce.
  - Mutación: La mutación, es la modificación del material genético de un individuo, de esta manera, siguiendo la idea del cruce, sera necesario definir a que se refiere mutar, entendiéndolo como una modificación de los cromosomas.
  - Conservación: Es de interés conservar a los individuos más destacados para la construcción de una nueva población.

Para el caso práctico un algoritmo genético sera una instancia del esquema dado por la metaheurística, en un problema de optimización. La construcción de un algoritmo genético para un problema, considerando los aspectos necesarios para definir un sistema evolutivo sobre un conjunto de soluciones, puede seguir el siguiente esquema:

- **Representación**: Para cada solución se deberá definir una representación genética o mas precisamente el genotipo, con el fin de replicar las operaciones que se realizan a este nivel en el proceso evolutivo, como pueden ser el cruce y la mutación. Ademas, se deberá definir una representación del fenotipo, que estará relacionada con el genotipo, comúnmente mediante una correspondencia o función, que permite evaluar la aptitud de un individuo o solución, en relación a que tan buena o útil es de cara a la optimización de un problema, dependiendo de como se define es evaluación de fitness.

  - Genotipo: Se representara el genotipo o genoma de un individuo o solución, mediante $k-$tuplas $(X_1, ..., X_k)$ en un conjunto $S \subseteq \mathbb{R}^k$, que

se llaman cromosomas. Cada posición, así mismo cada $i = 1, ..., k$ se llama gen del cromosoma, y cada valor para un gen i dado por X_i, es lo que se llama alelo.

Por ejemplo, se tiene un cromosoma $W = (W_1, ..., W_n)$ donde cada alelo esta en $\mathbb{R}$ y representa el peso de ir de un vértice a otro en un grafo G, existiendo como semántica sobre W que los primeros k alelos, representar el peso de ir de cualquier vértice al vértice 1 según una enumeración en G, siempre que $N_G(i) = k$. De acá se extrae, que un cromosoma no solo es una secuencia de valores, también se pueden definir ciertos condicionales, que añaden una semántica sobre el cromosoma, que se relaciona directamente con el objetivo de definir al cromosoma, que es realizar operaciones evolutivas.

- Fenotipo: El genotipo puede ser coincidente con el fenotipo para fines prácticos, ahora bien, en general sera una codificación de lo que es el fenotipo, de tal manera, que existe una función tal que $(X_1, ..., X_k) \to \phi(X_1, ..., X_k) = Y$, o existe un algoritmo $\mathcal{A}(X_1, ..., X_n)$ que por medio de los alelos retorna un fenotipo.

  Por ejemplo, se tiene un cromosoma $(X_0, ..., X_k) \in \{0, 1\}^{k+1}$ de modo que $2^0 X_0 + ... + 2^k X_k = X$, siendo $X \in \mathbb{R}$ el fenotipo de un individuo o solución.

- **Inicialización de la población**: Debe ser representativa del conjunto de todas las soluciones o individuos en un problema de optimización, por ende se busca que no exista un proceso de selección sesgado, pues esto generaría una menor diversidad y en definitiva condicionaría la evolución buscada, lo que podría significar en la practica que la población se parece y que no se evalúan lo suficientemente las características genéticas y fenotípicas de una amplia mayoría, lo que podría generar una solución no muy buena.

  La selección de la población inicial, se hace considerando que el alelo para cada gen i es un variable aleatoria $X_i \sim \mathcal{D}$ en un cromosoma $(X_1, ..., X_k)$, considerando que los valores posibles del alelo X_i están en un intervalo S_i. Es claro que los valores que puede tomar cada alelo dependerán del problema directamente, y dependiendo de estos conjuntos es que se definirá como se distribuye la probabilidad a lo largo del conjunto. De esta manera, seleccionado un valor para cada alelo a través de las probabilidades que se tienen, se construye cada cromosoma de cada individuo a incluir en la población inicial.

  Comúnmente, se dice que cada alelo se distribuye de manera uniforme, esto es, la probabilidad para cada alelo posible es la misma. Formalmente, si X_i es continua entonces $X_i \sim U(a, b)$ con $S_i = [a, b]$ y la probabilidad de cada valor es $\frac{1}{b-a}$, si X es discreta entonces $X \sim U(x_1, ..., x_r)$ con $S_i = \{x_1, ..., x_r\}$ para $r \geq 1$ y la probabilidad de cada valor es $1/r$.

- **Definición y evaluación de fitness**: La función de fitness permite realizar una discriminación respecto la población, con el fin de tomar a los más aptos

para ser sometidos a los operadores evolutivos. Estos individuos o estas soluciones, serán las soluciones ejemplares a utilizar para la conformación de una nueva población en un estado $t+1$ sucesor al actual t.

La función de fitness puede ser definida de diferentes maneras:

- Expresión analítica: Si lo que se trata de optimizar es una función objetivo, siendo literalmente una función matemática que tiene una cierta expresión y depende de una serie de parámetros y restricciones, se habla de una expresión analítica de la función de fitness.

  En general se suele definir una expresión analítica en función de la función objetivo, o derechamente se utiliza la función objetivo con alguna penalización, teniendo en cuenta, que la evaluación sobre la función objetivo original y con sus restricciones, puede no ser aceptable del punto de vista del costo computacional.
- Rutina externa: Los individuos o soluciones, pueden representar parámetros de una simulación, o de una función objetivo del punto de vista de las computadoras.

Según lo planteado en [12], existen 4 tipos de fitness:

- Fitness Puro: Para una población de $i = 1, ..., N$ individuos para la población $t-$esima, se considera que existen $j = 1, ..., N_c$ casos a considerar para evaluar la calidad de un individuo. De esta manera, para evaluar la calidad de cada individuo i se debe comparar el resultado de este individuo en un caso j, respecto una cierta referencia para ese caso j, definiéndose la calidad por medio de la suma de las diferencias:

  $$r_{it} = \sum_{j \in [N]} |p_{ij} - c_{ij}|$$

  Siendo r_{it} la bondad o calidad del individuo i en la generación t, p_{ij} el resultado del individuo i en el caso j y c_{ij} el caso de referencia o valor esperado para el individuo i durante el caso j. Ahora bien, es claro que si el resultado o la respuesta de un individuo se parece o es igual que el valor esperado o deseado, la diferencia $\to 0$, con ello, se utiliza una estandarización para que la función de fitness, defina un buen individuo o seleccionable en función de que tan alto es su fitness cuando se quiera maximizar, y permita definir a un buen individuo en función de que tan bajo es su fitness cuando se quiera minimizar.
- Fitness Estandarizado: Sea r_{it} el fitness puro de un individuo en t y $r_{\max}$ una cota superior de todo fitness o bondad para la población, se define:

  $$s_{it} = \begin{cases} r_{it} & \text{(Minimizar)} \\ r_{\max} - r_{it} & \text{(Maximizar)} \end{cases}$$

Lo que se llama fitness estandarizado, siendo claro, que la discriminación de dos individuos i, j se hará mediante el orden usual $\leq$ tal que $s_{it} \leq s_{jt}$ implica que i es mejor si se busca minimizar, y $s_{it} \geq s_{jt}$ implica que i es mejor si se busca maximizar.

- Fitness Ajustado: Para ajustar el resultado de la bondad o calidad de los individuos sometidos a cierta evaluación al intervalo $[0, 1]$, se define el fitness ajustado:

$$a_{it} = \frac{1}{1 + s_{it}}$$

Este tipo de fitness y el normalizado son de bastante utilidad, teniendo en cuenta, que se construye una especie de probabilidad en función de la calidad, siendo mas directo en torno a la estrategia de selección, que dependerá de una probabilidad atribuida por el fitness.

- Fitness Normalizado: Para establecer una medida de la calidad de los individuos no unicamente respecto sus resultados si no del resto de la población a evaluar, se utiliza el fitness normalizado:

$$n_{it} = \frac{a_{it}}{\sum_{j \in [N]} a_{jt}}$$

Esto implica que la calidad en torno a un objetivo, se ve distribuida acorde a la calidad particular en torno a valores esperados, es decir, mientras mas alto sea el fitness normalizado, mejor es respecto el resto de individuos, lo que permite hablar de una manera mas concreta respecto la discriminación.

$$\sum_{j \in [N]} n_{jt} = 1$$

- **Estrategia de selección**: La selección de individuos debe realizarse considerando a los mas aptos, lo que se puede diferenciar mediante la función de fitness. La selección tiene como objetivo la construcción de un conjunto Q de individuos o soluciones a cruzar, entendiendo seleccionar como llevar una copia de la población P a Q.

  Ahora bien, sumado a que se debe seleccionar a los mas aptos, también se debe preservar en alguna medida a los individuos no aptos, para preservar la diversidad, siendo una condición necesaria para la evolución, y ademas, puede que un cromosoma que define un fenotipo y una bondad, tenga en general una mala bondad, pero internamente tenga cierta configuración genética que puede ser de utilidad bajo los operadores evolutivos.

  Puede ocurrir que dos soluciones de manera independiente no sean interesantes, sin embargo, un cruce entre ellos defina un nuevo individuo que si lo sea, esto es que tenga buena fitness, de igual manera pueda que una mu-

tación permite modificar la genética de una solución pudiendo definir una solución buena.

También será de utilidad, incluir en una nueva población a individuos que sean destacados del resto, sin alterar el resultado favorable que éstos tienen, mediante un cruce o mutación, que es lo que se verá en el mecanismo de reemplazo, para la determinación de la población nueva.

Se cuenta con una buena cantidad de métodos de selección, sin embargo, los usuales son los siguientes:

- Torneo: Se seleccionan $p \leq N$ individuos de una población de tamaño N, de manera aleatoria en vista de una cierta función de probabilidad asociada al fitness que otorga un valor para cada individuo. Si $\alpha_1, ..., \alpha_p$ son los individuos con fitness $n_{\alpha_1 t}, ..., n_{\alpha_p t}$ seleccionados, para alguna población en t, se toma el α_k tal que $n_{\alpha_k t} = \text{Max}\{n_{\alpha_i t} \mid i \in [p]\}$ o el $n_{\alpha_k t} = \text{Min}\{n_{\alpha_i t} \mid i \in [p]\}$ dependiendo el interés.

  La variación de la cantidad p de individuos a seleccionar es influyente con el conjunto en el que se busca la solución, para ser mas precisos, si $p \to N$ se dice que la selección se hace con una presión alta y elitista, que reduce las posibilidades de los individuos menos aptos, así mismo, define un conjunto de soluciones próximas o cercanas, por el contrario, si $p \to 0$ se dice que la selección se hace con baja presión dando oportunidad a los individuos menos aptos, lo que permite buscar respecto soluciones no necesariamente próximas.

  Si pensamos en puntos cercanos a un monte de una función que se busca optimizar, es claro que los puntos mas aptos deben estar cercanos al monte, lo que implica, que seleccionar puntos con presión alta implica a su vez puntos cercanos al monte o próximos a estos puntos, mientras que una presión baja aumenta la probabilidad de que se tomen puntos que no necesariamente están cerca del monte.

  Ambos extremos no parecen favorables en ciertas situaciones, si la solución optima del monte es una local, no nos interesa solo escoger puntos de ese monte, también una diversidad tal de los puntos, que permitan una búsqueda afuera de los puntos que tienen una imagen en el monte. Luego, tampoco parece favorable seleccionar con muy baja presión, pues, la acotación a un conjunto de mas aptos no parece tan fuerte, o tan concreta.

  A partir del torneo, se pueden generar ciertas variaciones, limitando la cantidad de veces que un individuo puede ser seleccionado por ejemplo.

- Truncamiento: Es un tipo de selección en la que se toma una parte de la población el resto se descarta, no siendo recomendable puesto que se suele disminuir muy rápidamente la diversidad de la población. Se tienen lo siguientes tipos:

  - Selección media: Si $\overline{s_t}$ es la fitness media, entonces todo individuo i tal que $s_{it} \geq \overline{s_t}$ se replica dos veces, en caso contrario, se eliminan

(Para el caso de minimizar es al revés, pues se quiere que el fitness sea bajo). Esto puede producir ciertos problemas técnicos, pues no se asegura que la población siga teniendo N individuos después de la selección.

Selección mediana: Si $s_{i,t}$ para $i = 1, ..., N$ es la fitness de los N individuos de manera ordenada creciente, se toma:

$$\tilde{s_t} = \begin{cases} s_{\frac{N}{2},t} \text{ si } N \text{ es impar} \\ \frac{s_{\frac{N}{2},t} + s_{\frac{N}{2}+1,t}}{2} \text{ si } N \text{ es par} \end{cases}$$

Y se sigue el mismo procedimiento que en selección media, pero con la mediana $\tilde{s_t}$.

- Selección por desviación: Si $\overline{s_t}$ es la media de la fitness, σ es la desviación tipica y s_{it} la fitness de elemento $i-$esimo con $i = 1, ..., N$. Para el caso de maximización se tiene:

$$s_{it} < \overline{s_t} - \sigma \text{ entonces } i \text{ se elimina}$$

$$\overline{s_t} - \sigma \leq s_{it} \leq \overline{s_t} + \sigma \text{ entonces } i \text{ se replica 1 vez}$$

$$s_{it} > \overline{s_t} + \sigma \text{ entonces } i \text{ se replica 2 veces}$$

Para el caso de minimización, se invierten "elimina" y "replica 2 veces".

- **Definición de operadores de cruce**: Se construye un conjunto P_Q intermedio de descendientes por cruce, a través de la aplicación de operadores de cruce. El cruce es un operador aplicado sobre el genotipo de un par de individuos, teniéndose en cuenta lo siguiente:

  - Debe haber herencia de ambos padres, esto es, el genotipo del descendiente debe presentar alelos de ambos.
  - No todos los padres deben ser combinados, puede que la combinación de los genotipos de los padres no de un descendiente apto, destruyendo el proceso de selección previo al que han sido sometidos a lo largo de las poblaciones generadas.
  - La operación cruce debe ser cerrada, esto es, si C es la operación cruce del par i, j y G_P es el conjunto de genotipos de la población P entonces $C(i, j) \in G_P$.

Se define una probabilidad P_c para decidir si cruzar o no cruzar. Se suele tomar $\lfloor \frac{|P|}{2} \rfloor$ como numero de cruces a realizar, considerando que los cruces mas usados suelen generar dos descendientes. Se puede definir un $\alpha \in [0, 1]$ aleatorio, tal que al tener un par de padres seleccionados, si $\alpha < P_c$ no se cruza, y si $\alpha \geq P_c$ se cruza, devolviéndose los individuos obtenidos al cruzar, o en caso contrario, los padres no cruzados.

Evidentemente, en caso de que se utilice un cruce con un solo descendiente o un numero distinto, el numero de selecciones de padres para ver si existe

un cruce debe modificarse, para evitar disminuir o aumentar la población, si se desea.

A continuación, se tiene algunos operadores de cruce usuales:

- Cruce binario: Sean $X = (X_1, ..., X_n)$ e $Y = (Y_1, ..., Y_n)$ genotipos de dos individuos con $X_i, Y_i \in \{0, 1\}$ para $i = 1, ..., n$, se toman k números distintos $n_1, ..., n_k$ de $[n]$ de manera aleatoria. Se obtienen los descendientes o hijos:
  Si k es par:

$$X^* = (X_1, ..., X_{n_1}, Y_{n_1+1}, ...Y_{n_2}, ..., X_{n_k+1}, ..., X_n)$$
$$Y^* = (Y_1, ..., Y_{n_1}, X_{n_1+1}, ...X_{n_2}, ..., Y_{n_k+1}, ..., Y_n)$$

  Si k es impar:

$$X^* = (X_1, ..., X_{n_1}, Y_{n_1+1}, ...Y_{n_2}, ..., Y_{n_k+1}, ..., Y_n)$$
$$Y^* = (Y_1, ..., Y_{n_1}, X_{n_1+1}, ...X_{n_2}, ..., X_{n_k+1}, ..., X_n)$$

  Se suelen utilizar cruces con $k = 1$ y $k = 2$. Para $k \to 0$ se tiende a preservar subconjuntos de alelos secuenciales más grandes en los padres, lo que arrastra de mejor manera las características de los padres al descendiente, por el contrario, cuando $k \to N$ se va perdiendo esta interrelación de alelos secuenciales, sin embargo, el descendiente es un individuo que conlleva alelos de los padres, pero con características más diferenciadas de los padres, lo que implica que se recorre de manera más amplia el conjunto de soluciones.
- Cruce uniforme: Sean $X = (X_1, ..., X_n)$ e $Y = (Y_1, ..., Y_n)$ genotipos de dos individuos, se definen p_{ij} probabilidades con $i \in [n]$ y $j \in [2]$ tal que $p_{i1} + p_{i2} = 1$. De esta manera, se construyen hijos Z por medio de la selección aleatoria de un Z_i para todo $i = 1, ..., n$ considerando que hay una probabilidad p_{i1} de que $Z_i = X_i$ y una probabilidad p_{i2} de que $Z_i = Y_i$.

  Cruce aritmético: Sean $X = (X_1, ..., X_n)$ e $Y = (Y_1, ..., Y_n)$ genotipos de dos individuos, se pueden definir los siguientes hijos:

  Por media:

$$\left(\frac{X_i + Y_i}{2}\right)_{i\in[n]}$$

  Por media geométrica:

$$\left(\sqrt{X_i Y_i}\right)_{i\in[n]}$$

  Por extensión:

$$(\text{Max}(X_i, Y_i) + |X_i - Y_i|)_{i\in[n]} \ , \ (\text{Min}(X_i, Y_i) - |X_i - Y_i|)_{i\in[n]}$$

Cruce BLX$-\alpha$: Se define un $\alpha \in [0,1]$ tal que los descendientes se forman tomando para cada gen un alelo en $[C_{min} - I\alpha, C_{max} + I\alpha]$ siendo $C_{max} = \text{Max}\{X_i, Y_i\}_{i\in[n]}$, $C_{min} = \text{Min}\{X_i, Y_i\}_{i\in[n]}$ con $I = C_{max} - C_{min}$.

Los únicos que permiten definir mas de un descendiente son el cruce por extensión y BLX$-\alpha$.

- Cruce permutacional: Cuando los genotipos son permutaciones, el conjunto de alelos (distintos)de cada genotipo es el mismo, pero, el gen al que son asignados es lo que varía. De esta manera, de esta manera, para cada genotipo existe una biyección α tal que $\alpha(X_1, ..., X_n) = (X_{\alpha_1}, ..., X_{\alpha_n})$ que asigna al alelo X_j que originalmente estaba asignado al gen j, el gen $\alpha(j) = \alpha_j$, considerando que la asignación $X_1, ..., X_n$ que asigna el alelo X_j al gen j es una de referencia para como se ha considerado el conjunto inicialmente y en el orden que se consideraba. Dicho esto, sean α, β permutaciones sobre $X = (X_1, ..., X_n)$, se tienen los siguientes cruces permutacionales:

  - Cruce de orden: Sean α, β permutaciones , se toman dos cruces n_1, n_2 para formar dos descendientes:

$$(Y_{n-n_2+1}, ..., Y_{n-n_2+n_1-1}, X_{\alpha_{n_1}}, ..., X_{\alpha_{n_2}}, Y_1, ..., Y_{n-n_2})$$
$$(Z_{n-n_2+1}, ..., Z_{n-n_2+n_1-1}, X_{\beta_{n_1}}, ..., X_{\beta_{n_2}}, Z_1, ..., Z_{n-n_2})$$

    Con $Y_1, ..., Y_{n-n_2+n_1-1}$ los elementos $X_{\alpha_1}, ..., X_{\alpha_{n_1-1}}, X_{\alpha_{n_2+1}}, ..., X_{\alpha_n}$ según el orden que tienen en X_β y $Z_1, ..., Z_{n-n_2+n_1-1}$ los elementos $X_{\beta_1}, ..., X_{\beta_{n_1-1}}, X_{\beta_{n_2+1}}, ..., X_{\beta_n}$ según el orden en X_α.
    Por ejemplo, si $X_\alpha = (7,3,1,8,2,4,6,5)$ y $X_\beta = (4,3,2,8,6,7,1,5)$ con $n_1 = 2, n_2 = 5$, se toman los elementos $7,3,4,6,5$ de X_α se ordenan como X_β, esto es $4,3,6,7,5$ obteniendo un descendiente $(7,5,1,8,2,4,3,6)$, luego se toman los elementos $4,3,7,1,5$ de X_β se ordenan como X_α esto es $7,3,1,4,5$ obteniendo el descendiente $(4,5,2,8,6,7,3,1)$.

  - Cruce de emparejamiento parcial: Para dos permutaciones X_α y X_β se definen un par n_1, n_2 con $n_1 < n_2$ para hacer un cruce en dos puntos en ambos cromosomas. Se define una función $f : \{X_{\alpha_{n_1}}, ..., X_{\alpha_{n_2}}\} \to \{X_{\beta_{n_1}}, ...X_{\beta_{n_2}}\}$ biyectiva y a partir de ella y su inversa al par de funciones para $j = 1, ..., n_1 - 1, n_2 + 1, .., n$:

$$h_{1j} = \begin{cases} X_{\alpha_j} \text{ si } X_{\alpha_j} \notin \{X_{\beta_{n_1}}, ..., X_{\beta_{n_2}}\} \\ f(X_{\alpha_j}) \text{ si } X_{\alpha_j} \in \{X_{\beta_{n_1}}, ..., X_{\beta_{n_2}}\} \end{cases}$$

$$h_{2j} = \begin{cases} X_{\beta_j} \text{ si } X_{\beta_j} \notin \{X_{\alpha_{n_1}}, ..., X_{\alpha_{n_2}}\} \\ f^{-1}(X_{\beta_j}) \text{ si } X_{\beta_j} \in \{X_{\alpha_{n_1}}, ..., X_{\alpha_{n_2}}\} \end{cases}$$

    Definiendose al siguiente par de descendientes:

$$(h_{11}, ..., h_{1n_1-1}, X_{\beta_{n_1}}, ..., X_{\beta_{n_2}}, h_{1n_2+1}, ..., h_{1n})$$
$$(h_{21}, ..., h_{2n_1-1}, X_{\alpha_{n_1}}, ..., X_{\alpha_{n_2}}, h_{2n_2+1}, ..., h_{2n})$$

- Cruce ciclo: Sean X_α y X_β dos permutaciones, f, g biyecciones tal que $f(X_{\alpha_i}) = X_{\beta_i}$ y $g(X_{\alpha_i}) = X_{\beta_j}$ con $X_{\alpha_i} = X_{\beta_j}$ y h una función tal que $h(X_i) = i$. Se elige un $k \in [n]$ de manera aleatoria con distribución uniforme y se escoge entre X_{α_k} o X_{β_k} con probabilidad también distribuida uniformemente.
  Si se escoge a X_{α_k} se define lo siguiente:

$$X_{\alpha_k} \to f^{-1}g(X_{\alpha_k}) \to (f^{-1}g)^2(X_{\alpha_k}) \to ... \to (f^{-1}g)^r(\alpha_k)$$

  Siendo $(f^{-1}g)^r(\alpha_k) = X_{\alpha_k}$ para algún $r \geq 1$ tal que la secuencia construida es un ciclo. Por el contrario si se escoge X_{β_k} se define:

$$X_{\beta_k} \to fg^{-1}(X_{\beta_k}) \to (fg^{-1})^2(X_{\beta_k}) \to ... \to (fg^{-1})^r(X_{\beta_k})$$

  Siendo $(fg^{-1})^r(X_{\beta_k}) = X_{\beta_k}$ para algún $r \geq 1$, o no se requiere seguir.
  Dicho esto, lo que se hace es construir un descendiente Y inicialmente vació $Y = (-, -, ..., -)$, asignándole un valor a sus coordenadas por medio de los elementos $X_{\alpha_k}, (f^{-1}g)^i(X_{\alpha_k})$ en los índices $k, h((f^{-1}g)^i(X_{\alpha_k}))$ para $i = 1, ..., r-1$, si es que se escogió primero X_{α_k}, en caso contrario análogamente con X_β. Si existen aun índices que no tienen asignado un valor, se repite la generación de un k' aleatorio para los índices donde Y no tiene valor aun, y así sucesivamente hasta que Y este completo.
  Por ejemplo, para $(1, 2, 3, 4, 5, 6, 7, 8)$ y $(2, 4, 6, 8, 7, 5, 3, 1)$ se escoge aleatoriamente la posición $k = 1$ y al valor 1 entonces:

$$1 \to 8 \to 4 \to 2 \to 1$$

  De este modo, $Y = (-, -, ..., -)$ pasa a tener a $1, 8, 4, 2$ en las posiciones $1, 8, 4, 2$ esto es $Y = (1, 2, -, 4, ..., 8)$. Si se repite para $k' \in [8] - \{1, 8, 4, 2\}$, se obtiene $k' = 3$ y se escoge aleatoriamente entre 3 en $(1, 2, 3, 4, 5, 6, 7, 8)$ y 6 en $(2, 4, 6, 8, 7, 5, 3, 1)$, dando 6, con ello:

$$6 \to 5 \to 7 \to 3 \to 6$$

  De este modo, Y pasa a tener $6, 5, 7, 3$ en las posiciones $3, 6, 5, 7$ tal que $Y = (1, 2, 6, 4, 7, 5, 3, 8)$.

- **Definición de un operador de mutación**: Es un mecanismo de inclusión de ruido o diversidad, se construye el conjunto Q' a partir del conjunto P_Q por medio de la mutación aleatoria para un cierto subconjunto de P_Q, lle-

vando los individuos mutados hacia Q' y también aquellos que se decidieron no mutar. La mutación debe satisfacer las siguientes condiciones:

- Que no presente sesgos hacia ciertos individuos en la población, lo que se traduce en que, luego de la mutación de un individuo, éste pueda ser tanto apto como no en alguna medida.
- La mutación debe ser controlada, es decir, debe existir un equilibro, que permita introducir ruido o nuevas características a los individuos de la población que no dependen de lo que se ha generado a lo largo de las poblaciones anteriores, pero que tampoco se vea afectada de tal manera que se pierdan las características arrastradas a lo largo del proceso por un exceso de perturbación de genotipos.

Se define una probabilidad P_m muy baja, generalmente P_m entre el 0,5 % y 2 % de tal manera, que para todo individuo en P_Q obtenido por el cruce o no de padres seleccionados aleatoriamente, se realiza o no una mutación, llevando respectivamente al individuo mutado o no al conjunto de descendientes Q'.

Dicho esto, se tienen a continuación algunos tipos de mutación:

- Mutación binaria: Se define una probabilidad p y se generan k números aleatorios $n_i \in [0,1]$ con $i = 1, ..., k$ para un genotipo $(X_1, ..., X_k)$ con $X_i \in \{0,1\}$. Luego, si se define :

$$\overline{X_i} = \begin{cases} 1 \text{ si } X_i = 0 \\ 0 \text{ si } X_i = 1 \end{cases}$$

$$f(X_i) = \begin{cases} X_i \text{ si } n_i < p \\ \overline{X_i} \text{ si } n_i \geq p \end{cases}$$

  El individuo después de la mutación es $(f(X_1), ..., f(X_n))$.
- Mutación real: Para $(X_1, ..., X_n)$ se define $Y_i = X_i + N(0, \sigma^2)$ con $i = 1, ..., n$, de tal manera, que la mutación genera un individuo $(Y_1, ..., Y_n)$ para alguna desviación.
- Mutación permutacional: Determinar dos números $n_1, n_2 \in [n]$ aleatorios. Para un cromosoma $(X_1, ..., X_n)$ la mutación es intercambiar X_{n_1} por X_{n_2}.

- **Mecanismo de reemplazo**: El reemplazo hace referencia al proceso de construcción de la nueva población, entre las cuales se tienen:
  - Generacional: Si se tiene una población P para un estado t, se construyen respectivamente $Q(t), Q'(t)$. La visión generacional, descarta a la población P_t por $Q'(t)$, de tal manera, que se puede hablar de la

sucesión $P_0, P_1, ...P_t$ como las generaciones del proceso evolutivo, en la que existen ciertas variaciones a la hora de realizar un reemplazo. Por ejemplo, la estrategia generacional elitista, donde se preserva al mejor o los mejores k individuos o soluciones de una generación, para incluirla en la siguiente, copiandolos (siguen pudiendo elegirse como padres o para mutación) directamente en Q' o conjunto de descendientes, implicando que el mecanismo de selección de padres o individuos a mutar en $Q(t)$ sea de $p-1 \leq N$ o $p-k \leq N$ según corresponda.

- Estacionaria: La población P se somete a cambios en vista de los resultados en los descendientes, eliminando aquellos que ya no son de interés e ingresando los individuos o soluciones en P. Se tiene los siguientes tipos:
  - Aleatoria : Se elimina aleatoriamente un individuo en P, y se pasa de Q' a P otro.
  - Reemplazo de padres: Se eliminan de P los padres.
  - Reemplazo de similares: Se selecciona $\alpha_1, ..., \alpha_k$ individuos con fitness similar, y aleatoriamente se eliminan r individuos si necesitamos espacio para r nuevos de Q' (r y k están relacionados).
  - Reemplazo de los peores: Para un $\beta \in [0,1]$ de los peores en P, se eliminan r individuos si necesitamos r espacios (β y r deben estar relacionados).

- **Criterio de parada**: Se tienen a continuación algunas formas de detener el algoritmo genético:
  - Limitar las iteraciones: Definir una cota $T \in \mathbb{N}$ tal que $t \leq T$, así mismo, la cantidad de poblaciones sucesivas o generaciones obtenidas, no supera o es igual a T.
  - Limitar las iteraciones si no existe mejora: Se define un $\varepsilon > 0$ tan pequeño como se desea, un $\triangle_i$ en función de como se ha caracterizado el resultado de un población P, entre la población actual P_i y la anterior P_{i-1} y un $n_0 \in \mathbb{N}$. De esta manera si $\triangle_i < \varepsilon$ durante $j \geq n_0$ se detiene el algoritmo. Cuando se habla de caracterizar el resultado de una población, se habla de como se define el rendimiento de una población, si $s_i t$ es la función de fitness del elemento $i \leq N$ en la población $t-$esima se tiene:
    - Mejor : $\triangle_j = \text{Max}\{s_{ij} : i \in [N_j]\} - \text{Max}\{s_{ij-1} : i \in [N_{j-1}]\}$ (o mínimo dependiendo si se quiere minimizar)
    - Media: $\triangle_j = \overline{s_{ij}} - \overline{s_{ij-1}}$.
    - Media geométrica: $\triangle_j = \sqrt[N_j]{\prod_{i\in[N_j]} s_{ij}} - \sqrt[N_{j-1}]{\prod_{i\in[N_{j-1}]} s_{ij-1}}$.
    - Alguna otra función : La definición de la función para representar el fitness de una población en un t puede ser muy amplia, se podría pensar en el fitness promedio para un porcentaje de la población, la diferencia promedio entre los $k \leq N$ individuos mas aptos, etc.

Se considera en general que existe para cada población $j-$esima una cantidad N_j de individuos, por que existen ciertos casos particulares donde es así, ahora bien, según se recomendó a lo largo del estudio, usualmente se tiene que $N_1 =, ..., N_j = N$.

- Limitar el tiempo de ejecución: Definir una cota $T_M \in \mathbb{N}$ para lo que se haya definido como tiempo T.
- Limitar el tiempo si no existen mejoras: Análogamente al caso de mejores para iteraciones, pero teniendo en cuenta alguna definición de tiempo.

Dicho lo anterior, el esquema general de un algoritmo genético se puede modelar a partir del siguiente grafo:

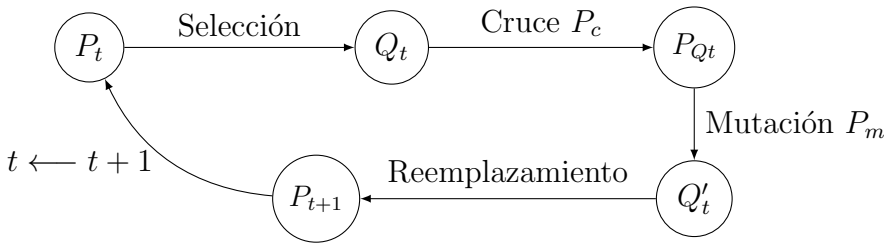


Fig. 3.16: Esquema general de un GA

Pseudocódigos

- **Selección de un individuo aleatorio** : Para seleccionar un individuo en P considerando una distribución uniforme $\frac{1}{N}$, con un tamaño de población $|P| = N$ se tiene:

```
1:  p ← Numero aleatorio en [0, 1]
2:  Y ← □
3:  q ← 0
3:  i ← 0
4:  Mientras (Y == □) hacer:
5:      q ← q + 1/N
6:      Si (p ≤ q) entonces:
7:          Y ← X[i]
8:      Fin si
5:      i ← i + 1
9:  Fin Mientras
10: Retornar Y
```

Básicamente, se asocia para el individuo $i-$esimo en revisarse un intervalo $(\frac{i-1}{N}, \frac{i}{N}]$, de este modo, el individuo seleccionado será el cual tiene a p en su

intervalo, siendo el caso particular cuando $i = 1$ definido el intervalo como $[0, \frac{1}{N}]$.

- **Generación de un individuo aleatorio** : Considerando cromosomas $X = (X_1, ..., X_M)$, $\varphi(p, a, b) = \lceil (b - a + 1)p + a - 1 \rceil$ para el caso discreto y $\varphi(p, a, b) = (b - a)p + a$ caso continuo, y $a, b \in \mathbb{R}$ dos valores que sirven como limitadores del espacio de búsqueda o dominio, se tiene:

1: $a \leftarrow$ Limite inferior
2: $b \leftarrow$ Limite superior
3: $n \leftarrow M$
4: $X \leftarrow (X[0], ..., X[n-1])$
5: Para $i \leftarrow 1$ hasta n con paso 1 hacer:
6: $\quad p \leftarrow$ Numero aleatorio en $[0, 1]$
7: $\quad X[i-1] \leftarrow \varphi(p, a, b)$
8: Fin Para
9: Retornar X

Este método, permite construir un cromosoma de números reales y enteros.

1: $n \leftarrow M$
2: $X \leftarrow (X[0], ..., X[n-1])$
3: Para $i \leftarrow 1$ hasta n con paso 1 hacer:
4: $\quad p \leftarrow$ Numero aleatorio en $[0, 1]$
5: $\quad$ Si (0,5 $\geq p$) entonces:
6: $\quad\quad X[i-1] \leftarrow 1$
7: $\quad$ Caso contrario:
8: $\quad\quad X[i-1] \leftarrow 0$
9: $\quad$ Fin si
10: Fin Para
11: Retornar X

Este es para la construcción de un cromosoma aleatorio binario. En ambos casos, se considera el tamaño de los cromosomas, y se va obteniendo un valor aleatorio dentro de un conjunto de posibles, para el caso real y entero $[a, b]$, para el caso binario $\{0, 1\}$, utilizando una probabilidad uniformemente distribuida.

- **Cruce** : Considerando cromosomas $X = (X_1, ..., X_M)$, $f(X_i, Y_i) = \frac{X_i + Y_i}{2}$ o $f(X_i, Y_i) = \sqrt{X_i Y_i}$ para el cruce aritmético, se tiene:

1: $k \leftarrow$ Numero aleatorio en $[M]$
2: Para $i \leftarrow k - 1$ hasta M con paso 1 hacer:
3: $\quad S \leftarrow X[i]$
4: $\quad X[i] \leftarrow Y[i]$
5: $\quad Y[i] \leftarrow S$
6: Fin Para

7: Retornar $\{X, Y\}$

Este corresponde al cruce binario en un solo punto k obtenido de manera aleatoria.

1: $k_1 \leftarrow$ Numero aleatorio en $[M]$
2: $k_2 \leftarrow$ Numero aleatorio en $[M]$
3: Si $(k_1 > k_2)$ entonces:
4: $\quad s \leftarrow k_1$
5: $\quad k_1 \leftarrow k_2$
6: $\quad k_2 \leftarrow s$
7: Fin Si
8: Para $i \leftarrow k_1$ hasta k_2 con paso 1 hacer:
9: $\quad s \leftarrow X[i-1]$
10: $\quad X[i-1] \leftarrow Y[i-1]$
11: $\quad Y[i-1] \leftarrow s$
12: Fin Para
13: Retornar $\{X, Y\}$

SI $k_1 = k_2$ se devuelven los mismos padres con probabilidad $\frac{1}{M}$ de que no se defina un cruce, ahora, si X, Y fueron seleccionados con probabilidad P_c para un cruce, entonces $\frac{P_c}{M}$ es la probabilidad de que no haya cruce.

1: Para $i \leftarrow 1$ hasta M con paso 1 hacer:
2: $\quad p \leftarrow$ Numero aleatorio en $[0, 1]$
3: $\quad$ Si $(p \geq 0{,}5)$ entonces:
4: $\quad\quad s \leftarrow X[i-1]$
5: $\quad\quad X[i-1] \leftarrow Y[i-1]$
6: $\quad\quad Y \leftarrow s$
7: $\quad$ Fin Si
8: Fin Para
9: Retornar $\{X, Y\}$

Este es el cruce uniforme, existiendo una probabilidad p seleccionada aleatoriamente en $[0, 1]$ de que se intercambien los alelos entre los elementos en cruce para algún gen.

1: Para $i \leftarrow 1$ hasta M con paso 1 hacer:
2: $\quad X[i-1] \leftarrow f(X[i], Y[i])$
3: Fin Para
4: Retornar X

Este corresponde al cruce aritmético para la media y media geométrica. Recordemos que estos cruces generan solo un descendiente o hijo.

1: Para $i \leftarrow 1$ hasta M con paso 1 hacer:

2: $m \leftarrow \text{Max}(X[i], Y[i])$
3: $n \leftarrow \text{Min}(X[i], Y[i])$
4: $X[i-1] \leftarrow m + (m - n)$
5: $Y[i-1] \leftarrow n - (m - n)$
6: Fin Para
7: Retornar $\{X, Y\}$

1: $\alpha \leftarrow$ Numero aleatorio en $[0, 1]$
2: $m \leftarrow -\infty$
3: $n \leftarrow +\infty$
4: Para $i \leftarrow 1$ hasta M con paso 1 hacer:
5: $M \leftarrow \text{Max}(X[i], Y[i])$
6: $N \leftarrow \text{Min}(X[i], Y[i])$
7: Si $(m < M)$ entonces:
8: $m \leftarrow M$
9: Fin Si
10: Si $(n > N)$ entonces:
11: $n \leftarrow N$
12: Fin Si
13: Fin Para
13: $I_1 \leftarrow n - (m - n)\alpha$
13: $I_2 \leftarrow m + (m - n)\alpha$
14: Para $i \leftarrow 1$ hasta M con paso 1 hacer:
15: $X[i-1] \leftarrow$ Numero aleatorio en $[I_1, I_2]$
16: $Y[i-1] \leftarrow$ Numero aleatorio en $[I_1, I_2]$
17: Fin Para
18: Retornar $\{X, Y\}$

Estos últimos cruces, son respectivamente el cruce por extensión y BLX$-\alpha$, siendo de tipo aritmético pero representando un mayor interés, considerando que se tienen 2 descendientes o hijos.

- **Mutación** : Considerando cromosomas $X = (X_1, ..., X_M)$, una desviación r y una probabilidad de mutación P_m se tienen:

1: $p \leftarrow P_c$
2: Para $i \leftarrow 1$ hasta M con paso 1 hacer:
3: $q \leftarrow$ Numero aleatorio en $[0, 1]$
4: Si $(p \geq q)$ entonces:
5: Si $(X[i] == 0)$ entonces:
6: $X[i] \leftarrow 1$
7: Caso contrario:
8: $X[i] \leftarrow 0$
9: Fin si
10: Fin si
11: Fin Para

12: Retornar X

Se realiza una mutación, intercambiando el alelo del individuo por el valor contrario, siempre que un número aleatorio para ese gen, se tenga un número aleatorio en $[0, 1]$ inferior a P_m (Lo que es muy poco probable).

1: $p \leftarrow P_c$
2: $\sigma \leftarrow r$
3: Para $i \leftarrow 1$ hasta M con paso 1 hacer:
4: $q \leftarrow$ Numero aleatorio en $[0, 1]$
5: Si $(p \geq q)$ entonces:
6: $X[i] \leftarrow X[i] + N(0, \sigma^2)$
7: Fin si
8: Fin Para
9: Retornar X

Este corresponde al cruce uniforme, siendo la expresión $N(0, \sigma^2)$ el valor en el eje X asociado a un número aleatorio entre $[0, 1]$ en el eje Y.

1: $p \leftarrow P_c$
2: $n_1 \leftarrow$ Numero aleatorio en $[M]$
3: $n_2 \leftarrow$ Numero aleatorio en $[M]$
4: $q \leftarrow$ Numero aleatorio en $[0, 1]$
5: Si $(p \geq q)$ entonces:
6: $s \leftarrow X[n_1 - 1]$
7: $X[n_1 - 1] \leftarrow X[n_2 - 1]$
8: $X[n_2 - 1] \leftarrow s$
9: Fin si
10: Retornar X

Esta ultima mutación, es el caso permutacional. Estas 3 maneras de mutar, siempre se aplicaran sobre los padres o sobre individuos a pasar a una nueva generación o población, es decir, todo individuo se somete al método mutar seleccionado, pero existiendo una probabilidad P_m muy baja de que realmente ocurra.

- **Selección con reemplazo para padres** : Considerando $\sim$ como $<$ si el interes es minimizar y $>$ si se quiere maximizar, se tiene:

1: $p \leftarrow$ Numero de individuos $(1 \leq p \leq N)$
2: $X \leftarrow$ IndividuoAleatorio(P).
3: Para $i \leftarrow 2$ hasta p con paso 1 hacer:
4: $Y \leftarrow$ IndividuoAleatorio(P)
5: Si (Fitness$(Y) \sim$ Fitness(X)) entonces:
6: $X \leftarrow Y$
7: Fin Para
8: Retornar X

Este es el torneo aleatorio, donde se selecciona de p elegidos, el que tiene mejor fitness.

- **Auxiliares**: Sea M el tamaño de los cromosomas, se tiene:

1: $i \leftarrow 0$
2: $s \leftarrow 1$
3: Mientras $(X[i] == Y[i] \wedge i < M - 1)$ entonces:
4: $\quad i \leftarrow i + 1$
5: $\quad$ Si $(X[i]! = Y[i])$ entonces:
6: $\quad\quad s \leftarrow 0$
7: $\quad$ Fin Si
8: Fin Mientras
9: Retornar s

Este método permite obtener un 1 o un 0 respectivamente si hay igualdad o no entre dos cromosomas, este se utiliza bajo el nombre "Igual".

1: $i \leftarrow 0$
3: Para cada $x \in X$ entonces:
4: $\quad i \leftarrow i + 1$
3: Fin Para cada
1: $Y \leftarrow (Y[0], ..., Y[i - 1])$
3: Para $j \leftarrow 0$ hasta $i - 1$ con paso 1 hacer:
4: $\quad Y[j] \leftarrow X[j]$
7: Fin Para
9: Retornar Y

Esta función retorna una copia exacta de un cromosoma de entrada. En algunas situaciones se puede disponer del tamaño del cromosoma y seria innecesario el primer ciclo, sin embargo, agregarlo no afecta el costo computacional cuando $i \to \infty$ (en este caso siendo i el numero interno del método para definir el tamaño de los cromosomas).

A partir de esto es que se definen los siguientes esquemas de algoritmos genéticos, en función de los métodos seleccionados de los anteriores, y de algunas variaciones propias de la implementación:

- **Algoritmo genético generacional sin elitismo**: Sea L el número de individuos para la población, M el tamaño de los cromosomas, $\sim$ es $>$ si se quiere maximizar o $<$ si se quiere minimizar. Se obtiene al mejor individuo por Fitness para un método generacional sin elitismo como sigue:

1: $N \leftarrow L$
2: $P \leftarrow \{\}$
3: Para $i \leftarrow 0$ hasta N con paso 1 hacer:
4: $\quad P \leftarrow P \cup \{\text{NuevoIndividuo}(M)\}$

```
Fin Para
Solucion ← □
Mientras (Parada! = True) hacer:
    Para i ← 0 hasta N con paso 1 hacer:
        CalculoFitness(P[i])
            Si (Solucion == □ ∨ Fitness(Solucion) ∼ Fitness(P[i])) entonces:
                Solucion ← P[i]
        Fin Si
    Fin Para
    Q ← {}
    P_Q ← {}
    Para i ← 0 hasta N/2 con paso 1 hacer:
        P_a ← SeleccionConReemplazo(P)
        P_b ← SeleccionConReemplazo(P)
        P_Q ← Cruce(Copia(P_a), Copia(P_b))
        Para cada C ∈ P_Q hacer:
            Q ← Q ∪ {Mutacion(C)}
        Fin Para cada
        P_Q → {}
    Fin Para
    P ← Q
Fin Mientras
Retornar Solucion
```

- **Algoritmo genético generacional con elitismo**: A la definición anterior, se le añade la condición de definir k individuos con mejor fitness.

```
N ← L
P ← {}
Para i ← 0 hasta N con paso 1 hacer:
    P ← P ∪ {NuevoIndividuo(M)}
Fin Para
Solucion ← □
Mientras (Parada! = True) hacer:
    Para i ← 0 hasta N con paso 1 hacer:
        CalculoFitness(P[i])
        Si (Solucion == □∨Fitness(Solucion) ∼ Fitness(P[i])) entonces:
            Solucion ← P[i]
        Fin Si
    Fin Para
    Q ← MasAptos(k)
    P_Q ← {}
    Para i ← 0 hasta (N−k)/2 con paso 1 hacer:
        P_a ← SeleccionConReemplazo(P)
        P_b ← SeleccionConReemplazo(P)
```

19: $P_Q \leftarrow \text{Cruce}(\text{Copia}(P_a), \text{Copia}(P_b))$
20: Para cada $C \in P_Q$ hacer:
21: $Q \leftarrow Q \cup \{\text{Mutacion}(C)\}$
22: Fin Para cada
23: $P_Q \rightarrow \{\}$
24: Fin Para
25: $P \leftarrow Q$
26: Fin Mientras
27: Retornar Solucion

- **Algoritmo genético estacionario**:

1: $N \leftarrow L$
2: $P \leftarrow \{\}$
3: Para $i \leftarrow 0$ hasta N con paso 1 hacer:
4: $P \leftarrow P \cup \{\text{NuevoIndividuo}(M)\}$
5: Fin Para
6: Solucion $\leftarrow \square$
7: Para $i \leftarrow 0$ hasta N con paso 1 hacer:
8: CalculoFitness($P[i]$)
9: Si (Solucion $== \square \vee$ Fitness(Solucion) $\sim$ Fitness($P[i]$)) entonces:
10: Solucion $\leftarrow P[i]$
11: Fin Si
12: Fin Para
13: Mientras (Parada! = True) hacer:
14: $P_a \leftarrow \text{SeleccionConReemplazo}(P)$
15: $P_b \leftarrow \text{SeleccionConReemplazo}(P)$
16: $P_Q \leftarrow \text{Cruce}(\text{Copia}(P_a), \text{Copia}(P_b))$
17: Para cada $C \in P_Q$ hacer:
18: $C \leftarrow \text{Mutacion}(C)$
19: CalculoFitness(C)
20: Si (Fitness(C) $\sim$ Fitness(Solucion)) entonces:
21: Solucion $\leftarrow C$
22: Fin Si
23: Fin Para cada
24: $P_d \leftarrow \text{SeleccionParaMorir}(P)$
25: $P_e \leftarrow \text{SeleccionParaMorir}(P)$
26: Mientras (Igual(P_e, P_d) $== 1$) entonces:
27: $P_e \leftarrow \text{SeleccionParaMorir}(P)$
28: Fin Mientras
29: $P \leftarrow P - \{P_d, P_e\}$
30: $P \leftarrow P \cup P_Q$
31: $P_Q \rightarrow \{\}$
32: Fin Mientras
33: Retornar Solucion

3.6.3 Recocido simulado (SA)

El recocido simulado es una metaheurística basada en la metáfora, de considerar al conjunto de vecinos de una solución, como posibles posiciones de movimiento de un átomo central o de referencia que coincide con la solución, considerando que la libertad de movimiento, o el rango de movimiento posible al pasar de estar en la solución a una solución sucesora, sera limitado por una metáfora de la temperatura, que es la que regula el cambio de estados de las moléculas y movimientos de los átomos en un sólido, en el procedimiento de recocido, que es el fenómeno en el cual se basa la metaheurística.

El procedimiento de recocido, se basa en elevar la temperatura de un sólido, consiguiendo que los estados moleculares se vean alterados, permitiendo que los átomos se muevan mas libremente de manera aleatoria en la retícula que da forma al sólido, buscando una posición o configuración en el espacio, que permita una energía mínima. La disminución de la temperatura se hace de manera generalmente lenta, permitiendo los ajustes necesarios de los átomos, para que al momento de llegar a la temperatura en la cual los estados moleculares vuelven a su estado fundamental, los átomos se encuentren en posiciones de mínima o cercanas a la mínima energía.

Se establece la siguiente correspondencia de conceptos, entre el fenómeno del recocido y su utilización en la simulación del recocido para solucionar problemas de optimización:

- **Estado del sistema**: Este permite describir cual es la condición actual del solido en función de la temperatura que se tiene, mas precisamente, cual es la energía del sólido para un punto en particular del proceso de recocido. En el caso de la optimización, el estado del sistema sera una solución.
- **Posición molecular**: El estado del sistema, sera claramente determinado por la posición actual de los átomos, es decir, debido a la posición actual, es que se podrá definir un estado del sistema, conociendo cuál es la temperatura. La posición molecular en optimización, será directamente cuáles son los valores de las variables de decisión, que para el estado actual del sistema, o la solución actual, definen tal nivel de energía.
- **Energía** : La energía es claro de que se trata, dependiendo directamente del movimiento conseguido debido a la temperatura. Para el caso de la

optimización, esta será la función objetivo del problema, pues, se tiene la intención de que por medio del movimiento aleatorio con tendencia a la disminución de la energía del recocido, se emplee en la función objetivo a través de una metáfora.

- **Estado fundamental** : El estado de mínima energía para el solido, o estado de estabilidad esperado. Esto corresponde con una solución global para el problema de optimización.
- **Estado metaestable** : Un estado estable condicionado, esto es, se relaciona con un optimo local.
- **Temperatura** : Es un mecanismo de control del movimiento y con ello, de la alteración de los valores de la energía. Este será un mecanismo de control, mas precisamente un parámetro para el algoritmo que simula el recocido, siendo fundamental para la limitación de las soluciones vecinas que se pueden seleccionar.

Un algoritmo que simula el recocido simulado, lo que hace es tomar una solución inicial y una temperatura inicial, y por medio de una búsqueda local, se van tomando soluciones que mejoran la función objetivo, o la empeoran considerando un marco para permitir el empeoramiento, durante un cierto periodo de tiempo, hasta que, se disminuye la temperatura, y se repite el proceso pero con una zona de selección posible cada vez menor, de vecinos para la solución que se tenga seleccionada. El objetivo de esta metaheurística, es permitir la mejoría de la función objetivo, permitiendo que en algunos casos la función objetivo tenga un valor peor, para permitir saltarse los óptimos locales, en los que se cae inevitablemente, si solamente se intenta mejorar la solución.

Dicho esto, un algoritmo de recocido simulado, se definirá en función del problema que se aborda, principalmente asociado a la manera de búsqueda de soluciones vecinas, que se entiende por solución vecina, y algunos aspectos como la manera de disminuir la temperatura, que es la temperatura entre otros. Un marco de elaboración de un algoritmo de recocido simulado, es el siguiente:

- **Definición de la temperatura** : La temperatura denotada por T, tendrá las mismas unidades que tiene la función objetivo del problema, teniendo en cuenta las 3 formas en las cuales se ve la temperatura, la temperatura inicial T_0, la temperatura programada T o la temperatura final T_f.

  La temperatura inicial se puede definir mediante los siguientes métodos:

- Parámetro del algoritmo: T_0 sera una constante, ingresada de manera directa.
- En función de la solución inicial: Si X_0 es la solución inicial para el algoritmo y Estado(X_0) es el valor en la función objetivo, se puede definir $T_0 = k\text{Estado}(X_0)$ donde $k \in (0, 1)$ y es un numero aleatorio.

La temperatura programada, sera una sucesión $(T_0, T_1, T_2, ..., T_N)$ decreciente, esto es $T_{i+1} \leq T_i$ para $i = 0, 1, ..., N$, que es definida mediante la disminución de la temperatura inicial. Existirá una temperatura para cada iteración, entendiendo iteración, como el proceso de salto de una solución a otra dentro de la vecindad definida durante un cierto periodo. A continuación se tienen algunas formas de definir la temperatura programada:

- Directa: Definir explicitamente la sucesión $(T_0, ..., T_N)$.
- Lineal: Se define parámetro $\beta < T_0$ y la siguiente forma cerrada para el tiempo:
$$T_i = T_0 - i\beta$$
Ahora bien, considerando el proceso iterativo que se lleva a cabo, también se podría definir como:
$$T_i = T_{i-1} - \beta$$
β determinara al menos parcialmente la rapidez de convergencia del algoritmo, considerando que existen otros factores (Como la cantidad de aceptaciones de nuevas soluciones para un mismo T_j en la iteración $j-$esima)
- Exponencial: Se define un parámetro $\alpha \in (0, 1)$ para la siguiente sucesión:
$$T_i = \alpha^i T_0 \quad \text{o} \quad T_i = \alpha T_{i-1}$$
- Logarítmica :
$$T_i = \frac{T_0}{\ln(1+i)}$$
- Cauchy: Existen dos versiones de esta sucesión, la original responde a la forma de las anteriores mencionadas, unicamente se enfoca en una forma de decrementar la sucesión de temperatura:
$$T_i = \frac{T_0}{1+i}$$
En cambio, la modificada, incluye una limitación a M iteraciones considerando una temperatura final T_f ya conocida:
$$T_i = \frac{T_{i-1}}{1+\beta T_{i-1}} \quad \beta = \frac{T_0 - T_f}{M T_0 T_f}$$

La temperatura final T_f es clave para que se tenga una convergencia, o exista un tiempo razonable y finito para determinar una buena solución. T_f se puede definir de variadas maneras, generalmente $T_f \to 0$, sin embargo, todas ellas deben tener una condición en común, y es que deben hacer sentido con la sucesión de enfriamiento, o sucesión de temperatura antes mencionadas.

Para los casos donde la temperatura es exponencial, logarítmica y cauchy, el limite resultante cuando $i \to \infty$ es:

$$\lim_{i\to\infty} \alpha^i T_0 = \lim_{i\to\infty} \frac{T_0}{i+1} = \lim_{i\to\infty} \frac{T_0}{\ln(i+1)} = 0$$

Ahora bien, en todos estos casos, se tiende asintoticamente hacia 0, lo que implica que definir una temperatura $T_f \leq 0$ haría que el algoritmo sea divergente, o en otras palabras, nunca se llegue a la temperatura final, con ello, para estos casos es necesario que $T_f > 0$. Para el caso lineal, se tiene el siguiente resultado:

$$\lim_{i\to\infty} T_0 - i\beta = -\infty$$

Esto es un indicador, de que T_f puede ser cualquiera (Con $T_f < T_0$ para que tenga sentido), pues la sucesión puede disminuir perfectamente a cualquier valor. Por último, la sucesión de cauchy modificada, considerando que se tiene una forma recursiva, se tiene lo siguiente:

$$T_i = \frac{T_{i-1}}{1+\beta T_{i-1}} \to T_i = \frac{T_{i-k}}{1+k\beta T_{i-k}} \to T_i = \frac{T_0}{1+i\beta T_0}$$

De donde es claro, que $T_f > 0$ para asegurar convergencia y que tenga sentido.

- **Evaluación del costo**: Si la función objetivo se puede evaluar en un punto, con un coste computacional aceptable, entonces directamente se toma la función objetivo f, para el cálculo de las expresiones que incluyen al costo, como $e^{-\frac{-\delta}{T}}$ o T_0 entre otras. Lo cierto es que, ese cálculo puede tener asociado una expresión aritmética lo que es inmediatamente eficiente, o un método, en el que puede haber penalizaciones u otras modificaciones para simplificar la operación, o el mismo costo computacional.

  Más generalmente, el esquema de evaluación del costo, es similar al caso del fitness en los GA, se recurre a la expresión mas conveniente, sea aritmética, sea un algoritmo eficiente, o en casos especiales, se recurre a la obtención de parámetros para una simulación, o para englobar todo, a procesos externos.

- **Mecanismo de búsqueda** : Para definir a la solución inicial X_0, se tienen los siguientes mecanismos:

- Búsqueda heurística: Utilizar una heurística que resulte eficiente en conjunto con la metaheurística del recocido, permitiendo definir una solución optima como inicial (que al menos sera local), dependiendo del punto inicial aleatorio que existe al definir una vecindad N fija o variable a través de las iteraciones en la heurística.
- Conocimiento experto: Mediante el estudio previo del problema, determinar un punto conveniente.

En general, es necesario que el mecanismo sea eficiente, a partir de esta condición se puede extender la forma cuanto se quiera. La vecindad N o mas generalmente, la sucesión de vecindades $N_0, N_1, ..., N_p$ que se definen en caso de utilizar la búsqueda heurística, sera de vital importancia, puesto que a partir de ella se determina una heurística eficiente en costo temporal o no, acorde a lo que se espera.

Esto es relevante, pues, la búsqueda de soluciones intermedias, mas bien, selección de puntos candidatos a una solución cualquiera X_n, se realiza de manera análoga a como se hace en una búsqueda heurística, con la distinción, de que el mecanismo de selección (que se vera posteriormente) es lo que se ve alterado.

De esta manera, la vecindad juega el mismo papel fundamental en el proceso de enfriamiento, y permite decir, que el mecanismo de búsqueda de candidatos, es una búsqueda heurística.

- **Mecanismo de selección** : Los puntos o soluciones en la vecindad $N(X_n)$ de la solución actual X_n, se pueden clasificar de dos maneras, mejoran/mantienen el costo, o lo empeoran. Cuando se tiene una solución X_{n+1} que mantiene o mejora, esta se acepta inmediatamente, y reemplaza a la solución actual, cuando por el contrario la solución empeora el resultado, ésta se somete a un proceso aleatorio para decidir si reemplazar a la actual o no.

  Dicho esto, se define la probabilidad de que una solución o posición X_{n+1} sea aceptada siempre que $X_{n+1} \in N(X_n)$ (este en el conjunto de vecinos de la actual), como sigue:

$$P(T, \delta) = P(X_{n+1} | X_{n+1} \in N(X_n)) = \begin{cases} 1 \text{ si } & \delta < 0 \\ e^{-\frac{\delta}{T}} & \delta \geq 0 \end{cases}$$

$$\delta = \text{Estado}(X_{n+1}) - \text{Estado}(X_n)$$

  De esta manera, si X_{n+1} mejora a X_n la probabilidad de aceptación es 1 y en caso contrario es $e^{-\frac{\delta}{T}}$. Si se analiza la proporción $-\frac{\delta}{T}$ se observa lo siguiente:

$$\lim_{\delta \to 0} \frac{\delta}{T} = 0 \to e^{-\frac{\delta}{T}} \to 1$$

$$\lim_{\delta \to +\infty} \frac{\delta}{T} = 0 \to e^{-\frac{\delta}{T}} \to 0$$

La probabilidad de aceptar una solución va en decrecimiento a medida que va en aumento la cantidad que se empeora, por el contrario, la probabilidad de aceptar una solución, va en aumento cuando la cantidad que se empeora va en disminución o se habla de pequeñas cantidades cercanas a 0. Ahora bien, en la practica T jugará un rol fundamental, pues no se tomaran casos extremos llevados al límite, y dependiendo del tamaño de T, el que sabemos es inicialmente alto, se tendrá una diferencia en los costes alta o baja, considerando que $-\frac{\delta}{T}$ es la proporción, así mismo, la cantidad de veces que representa $-\delta$ a T.

Dicho lo anterior, si μ es un numero aleatorio en $[0, 1]$, se puede modelar la asignación de un nuevo valor o posición para la solución actual X_n y una candidata X_{n+1} como sigue:

$$X_n = \begin{cases} X_n \text{ si } \delta \geq 0 \\ \begin{cases} X_{n+1} \text{ si } \mu < e^{\frac{-\delta}{T}} \\ X_n \text{ si } \mu \geq e^{\frac{-\delta}{T}} \end{cases} & \delta < 0 \end{cases}$$

Por ultimo, es interesante ver que si N es el conjunto de vecinos de una solución, M es el conjunto de puntos que mejoran o mantienen y E el conjunto de puntos que empeoran la solución, se puede definir una combinación convexa $|E| = \alpha|E| + \beta|M|$ con $\alpha + \beta = 1$ y $\alpha, \beta \in [0, 1]$, que modela el comportamiento de los tipos de soluciones al ir variando la solución que se encuentra en el centro de la vecindad N como actual, desde un máximo hasta un mínimo.

Esto se puede relacionar con la naturaleza heurística, es decir, en el máximo la solución aleatoria siempre se acepta, de algún modo existe un determinismo del resultado de tomar una solución candidata entre acepto o rechazo, coincidiendo con una búsqueda local, por el contrario, en ningún caso hay seguridad de que se acepte una solución o no al estar en un optimo o mínimo, osea hay total indeterminismo respecto al acepto o rechazo.

- **Definición de un límite de movimientos**: Para cada temperatura T_j de la sucesión $(T_0, ..., T_N)$, se tendrá un cierto número de soluciones generadas de manera aleatoria, dada por un valor L que sera una cota limite. L sera un criterio de enfriamiento, o de iteración de la sucesión $(T_0, ..., T_N)$, en el sentido de que cuando la cantidad de soluciones generadas sea L, entonces T_j pasa a ser T_{j+1}, ajustándose las probabilidades de aceptación de soluciones que empeoran a un rango más pequeño, lo que se repetirá sucesivamente, hasta que la temperatura llegue a su último elemento, o se cumpla cierta condición con L, de modo que, L también puede actuar como criterio de parada del algoritmo, no siendo necesario en este caso, definir la temperatura T_f por ejemplo.

Como criterio de enfriamiento, L se puede definir de las siguientes maneras:

- En función del tiempo: Para cada tiempo T_j se define un límite $L(T_j)$.
- Como una constante: L es un valor constante que define cuantas soluciones candidatas se generan.

Esta es una manera directa de utilizar L como un número máximo de soluciones generadas, o como límite para un iterador $i = 1, ..., L$. Ahora bien, se puede añadir una condición de termino o parada adicional a esta lógica iterativa, tomando el numero máximo de soluciones aceptadas, con esto, se puede definir una condición de enfriamiento a partir de una proporción entre soluciones generadas y aceptadas. Para ello, se utiliza un cierto $\lambda \in (0, 1)$, otro iterador j y a L, de modo que si L^* es un número definido bajo la misma lógica que L, pero para el número de soluciones aceptadas para una temperatura, se enfría la temperatura, hasta que j llega a $L^*(T) = \lfloor \lambda L(T) \rfloor$ o en efecto i llega a $L(T)$.

- **Criterio de parada**:
  - Cota de temperatura : Se define un T_f, de modo que mientras que $T \geq T_f$ se continua con el proceso.
  - Limitar el numero de iteraciones: Se define un número M de cambios de tiempo a lo largo en el proceso, esto es, si la sucesión es $(T_0, ..., T_N)$ cuando $T = T_M$ se detiene el proceso. Recordemos que este M también se puede definir inherente a la sucesión de tiempo, como en el caso de la sucesión de Cauchy modificado.
  - Limitar la ejecución por porcentaje de soluciones aceptadas: Se establece un contador k, que se aumenta en 1 cada vez que para una temperatura actual o fijada, se tiene una cantidad de soluciones aceptadas estrictamente menor que un máximo ($< L^*(T)$). De esta manera, si durante una cantidad K fija de enfriamientos, el porcentaje de aceptación esta bajo el considerado, entonces se detiene el algoritmo ($k = 1, ..., K$). En este caso, $L^*(T)$ debe definirse responsablemente, considerando que si es muy alto, el algoritmo puede converger muy rápidamente. Como añadido o de manera independiente a este criterio, también se suele considerar, que si la cantidad de aceptadas es 0, entonces es conveniente terminarlo.

Pseudocódigos

Las líneas de los pseudocódigos a continuación son relativas, esto es, no representan el número dentro del algoritmo como tal, entendiendo que estas secciones de código serán para reemplazar en el pseudocodigo del algoritmo que simula el recocido, según sea el interés.

- **Tiempo inicial** : Se tienen las siguientes formas de definir el tiempo inicial:

  1: T_0 (Con T_0 fijo o global)

  Este es el caso, donde se asume una temperatura fija y en general no recomendable, a menos que se tenga un grado de conocimiento sobre el problema.

  1: $k \leftarrow$ Numero aleatorio en $(0, 1)$
  2: $T_0 \leftarrow k\text{Estado}(X_0)$

  Definición aleatoria a partir de un factor k, considerando ademas el estado del sistema, o valor parcial hasta el momento de la función objetivo, o proceso externo que se quiere minimizar.

- **Enfriamiento**: Se tienen respectivamente, las maneras de enfriar, o métodos de enfriamiento, basándose en las sucesiones definidas anteriormente, respectivamente lineal, exponencial, logarítmica, Cauchy y Cauchy modificado.

  1: $\beta \leftarrow$ Parámetro con mismas unidades que la temperatura.
  2: $T \leftarrow T - \beta$

  1: $\alpha \leftarrow$ Parámetro en $(0, 1)$
  2: $T \leftarrow \alpha T$

  1: $T \leftarrow \frac{T_0}{\ln(i+1)}$

  1: $T \leftarrow \frac{T_0}{i+1}$

  1: $M \leftarrow$ Numero de iteraciones
  2: $\beta \leftarrow \frac{(T_0 - T_f)}{M T_0 T_f}$
  3: $T \leftarrow \frac{T}{1+\beta T}$

- **Solución candidata**: Se utiliza un proceso de búsqueda heurística como la búsqueda local, utilizando un vecindad o en efecto un punto con el que se simula la vecindad en un proceso externo.

- **Algoritmo de Recocido simulado**: Entendiéndose $< \cdot >$ como una expresión a reemplazar por una linea o una sección de código según corresponda, teniendo en cuenta las definiciones anteriores, se tiene lo siguiente:

  1: $X_0 \leftarrow$ SolucionInicial()
  2: <Inicialización de T_0 >
  3: $T \leftarrow T_0$
  4: $i \leftarrow 1$
  5: Mientras $(T \geq T_f)$ hacer:

7: Mientras ($i \leq L(T)$) hacer:
8: $S \leftarrow$ SolucionCandidata(X_0)
9: $\delta \leftarrow$ Estado(S) − Estado(X_0)
10: $\mu \leftarrow$ Numero aleatorio en $(0, 1)$
11: Si ($\mu <$ EXP($-\frac{\delta}{T}$) $\vee$ $\delta < 0$) hacer:
12: $X_0 \leftarrow S$
13: Fin Si
13: $i \leftarrow i + 1$
14: Fin Para
15: <Iteración de la sucesión de temperatura T >
16: Fin Mientras
17: Retornar X_0

Este corresponde al algoritmo de recocido, considerando como criterio de parada una temperatura final T_f y una cota $L(T)$ fija o variable en función de T, para limitar la cantidad de soluciones candidatas.

1: $X_0 \leftarrow$ SolucionInicial()
2: <Inicialización de T_0 >
3: $T \leftarrow T_0$
4: $i \leftarrow 1$
6: $j \leftarrow 0$
7: $M \leftarrow 0$ Numero de cambios de temperatura.
8: Mientras ($j < M$) hacer:
9: Mientras ($i \leq L(T)$) hacer:
9: $S \leftarrow$ SolucionCandidata(X_0)
10: $\delta \leftarrow$ Estado(S) − Estado(X_0)
11: $\mu \leftarrow$ Numero aleatorio en $(0, 1)$
12: Si ($\mu <$ EXP($-\frac{\delta}{T}$) $\vee$ $\delta < 0$) hacer:
13: $X_0 \leftarrow S$
14: Fin Si
15: $i \leftarrow i + 1$
16: Fin Para
17: <Iteración de la sucesión de temperatura T >
18: Fin Mientras
19: Retornar X_0

Este mantiene como criterio para cambiar de temperatura un límite de candidatas $L(T)$, sin embargo, el número de cambios de temperatura está fijo para M. Es claro que la iteración de la sucesión de temperatura tiene que ser una distinta a la de Cauchy modificada, o ser la de Cauchy modificada con el mismo límite de iteraciones.

1: $X_0 \leftarrow$ SolucionInicial()
2: <Inicialización de T_0 >
3: $T \leftarrow T_0$

4: $i \leftarrow 1$
5: $k \leftarrow 0$
6: $K \leftarrow$ Cota para k
7: $\lambda \leftarrow$ Numero entre $(0, 1)$
8: $L^*(T) \leftarrow \lambda L(T)$
9: Mientras $(T \geq T_f \wedge k < K)$ hacer:
10: $a \leftarrow 0$
11: Mientras $(i \leq L(T))$ hacer:
12: $S \leftarrow$ SolucionCandidata(X_0)
13: $\delta \leftarrow$ Estado$(S) -$ Estado(X_0)
14: $\mu \leftarrow$ Numero aleatorio en $(0, 1)$
15: Si $(\mu < \text{EXP}(-\frac{\delta}{T}) \vee \delta < 0)$ entonces :
16: $X_0 \leftarrow S$
17: $a \leftarrow a + 1$
18: Fin Si
19: $i \leftarrow i + 1$
20: Fin Mientras
21: Si $(a < L^*(T))$ entonces :
22: $k \leftarrow k + 1$
23: Fin Si
24: Si $(a == 0)$ entonces :
25: $k \leftarrow K$
26: Fin Si
27: <Iteración de la sucesión de temperatura T >
28: Fin Mientras
29: Retornar X_0

Acá se introduce un contador a de candidatas aceptadas, un contador k como contador del numero de veces que $a < \lambda L(T)$ y una cota para definir el tercer criterio de parada, considerando la misma cantidad límite para la generación de candidatas $L(T)$. En este caso, análogamente se puede añadir una cantidad M límite de iteraciones y un iterador j para la condición $j < M$.

1: $X_0 \leftarrow$ SolucionInicial()
2: <Inicialización de T_0 >
3: $T \leftarrow T_0$
4: $i \leftarrow 1$
5: $j \leftarrow 1$
6: $k \leftarrow 0$
7: $K \leftarrow$ Cota para k
8: $\lambda \leftarrow$ Numero entre $(0, 1)$
9: Mientras $(T \geq T_f \wedge k < K)$ hacer:
10: $a \leftarrow 0$
11: Mientras $(i \leq L(T) \wedge j \leq \lambda L(T))$ hacer:
12: $S \leftarrow$ SolucionCandidata(X_0)

13: $\delta \leftarrow$ Estado(S) − Estado(X_0)
14: $\mu \leftarrow$ Numero aleatorio en $(0, 1)$
15: Si $(\mu < \text{EXP}(-\frac{\delta}{T}) \vee \delta < 0)$ entonces :
16: $X_0 \leftarrow S$
17: $a \leftarrow a + 1$
18: Fin Si
19: $i \leftarrow i + 1$
20: Fin Mientras
21: Si $(a < \lambda L(T))$ entonces :
22: $k \leftarrow k + 1$
23: Fin Si
24: Si $(a == 0)$ entonces :
25: $k \leftarrow K$
26: Fin Si
27: <Iteración de la sucesión de temperatura T >
28: Fin Mientras
29: Retornar X_0

Acá se añade una limitación para la generación de candidatas, mediante una cantidad limite de candidatas aceptadas, mediante un iterador j con limite $\lambda L(T)$. La condición $T \geq T_f$ se puede modificar para una cantidad fija M de cambios de temperatura, trivialmente.

Finalmente, para generalizar la definición del algoritmo de recocido, se define como:

1: $X_0 \leftarrow$ SolucionInicial()
2: <Inicialización de T_0 >
3: $T \leftarrow T_0$
4: $i \leftarrow 1$
5: <Inicialización de auxiliares >
6: Mientras (< CriterioParada >) hacer:
7: < Reasignación de contadores >
8: Mientras $(i \leq L(T) \wedge <$ CriterioParadaGeneradas $>)$ hacer:
9: $S \leftarrow$ SolucionCandidata(X_0)
10: $\delta \leftarrow$ Estado(S) − Estado(X_0)
11: $\mu \leftarrow$ Numero aleatorio en $(0, 1)$
12: Si $(\mu < \text{EXP}(-\frac{\delta}{T}) \vee \delta < 0)$ hacer:
13: $X_0 \leftarrow S$
14: Fin Si
15: $i \leftarrow i + 1$
16: Fin Para
17: <Control de aceptación de candidatas y/o terminoT >
18: <Iteración de la sucesión de temperatura T >
19: Fin Mientras
20: Retornar X_0

3.7 Síntesis y resumen

Tarea próxima: Tabla de tipos de optimización, tabla de ordenes de complejidad, tabla de complejidad de problemas, tabla de ordenes para los algoritmos estudiados, tabla para los problemas estudiados con la clase del problema

3.8 Terminología

4. DESARROLLO DE LA APLICACIÓN

Se desarrollara un sitio web que tendrá fines informativos, descriptivos y de aplicación. La especificación de a que se refiere cada apartado mencionado se ve a continuación:

- **Informativo**: Existirá una descripción simple de cada uno de los algoritmos que se han estudiado en el presente informe, en conjunto de imágenes, gifs y/o vídeos que permitan mostrar visualmente de que se trata o que caracteriza a cada uno de ellos. Luego, cada una de estas breves descripciones tendrá una pagina destinada a profundizar el funcionamiento de cada uno de estos algoritmos, desde describir los procesos internos necesarios para el algoritmo, como puede ser la búsqueda de una solución vecina, el cruce entre dos soluciones para obtener un par nuevo, etc, hasta la visualización de sus resultados.

  Ademas, se tendrá una pagina para la descripción del TSP o Travelling Salesman Problem puesto es este el problema que se abordara en general, y que se utilizara para ver y comparar resultados.

- **Descriptivo**: Existirá una descripción de los resultados de los algoritmos para la resolución del TSP o Travelling Salesman Problem, como también, una comparativa de tiempo de resolución para los mejores casos que se encuentren en estos algoritmos, teniendo en cuenta la cantidad de parámetros de los cuales estos dependen. De esta manera, deberá existir una forma de gestionar los resultados que se vayan calculando para los algoritmos, en busca de una especie de optimización y presentación de los mejores resultados, teniendo en cuenta alguna combinación conveniente de parámetros.

  Mas específicamente, para cada algoritmo existirá una tabla de resultados obtenidos (Los 10 mejores hasta ese entonces) y gráficas que muestren y describan los resultados, utilizando imágenes, gifs y vídeos para ello.

- **Aplicativo** : Se podrá utilizar el algoritmo genético para resolver el TSP de manera particular, mas específicamente aplicado sobre el mapa de EEUU para el viaje mas corto entre los diferentes estados. Este realizara un calculo de la solución en tiempo real al darle a ejecutar, obteniéndose un vídeo del proceso de obtención de la mejor solución, permitiendo visualizar el proceso de cambio de la solución y como esta va mejorando desde una aleatoria o una fija inicial. Adicionalmente, se permitirá para cada algoritmo, ajustar el TSP

a parámetros de interés, esto es, la cantidad de nodos, sus posiciones y el costo de ir entre cada par, permitiendo así obtener una solución a cualquier problema, entendiéndose obtener una solución, como la mejor posible en vista de las limitaciones del algoritmo y también, de ciertas limitaciones que se impondrán por la sencillez del alcance de este trabajo.

Para este sencillo desarrollo, se utilizaran los siguientes recursos:

- Hardware: Se utilizara un equipo personal para su desarrollo, descrito por Intel(R) Core(TM) i3-5020U CPU @2.20GHz y 4GB de ram.
- Software: El desarrollo web y de algoritmos se hará por separado, esto es, se desarrollaran los algoritmos necesarios para su presentación en el sitio en Python, posteriormente se hará una integración con el framework web para su presentación.
  - FrameWork: Laravel 8 (Php, Html5 , Css y Js)
  - Editor de código: Visual Studio Code
  - Desarrollo algoritmos: Python + GoogleColab
  - Gestión Db: Microsoft SQL Server

  De manera adicional a esta lista de softwares, se listan a continuación las librerías necesarias en Python, ademas de software adicional para ciertos elementos, como por ejemplo, poder escribir ecuaciones matemáticas al estilo Latex en el sitio.

  - Matplotlib
  - Numpy
  - MathJax

4.1 Metodología de desarrollo

Para el desarrollo de la aplicación, se ha de utilizar la metodología ágil SCRUM, la cual se detalla de manera breve a continuación:

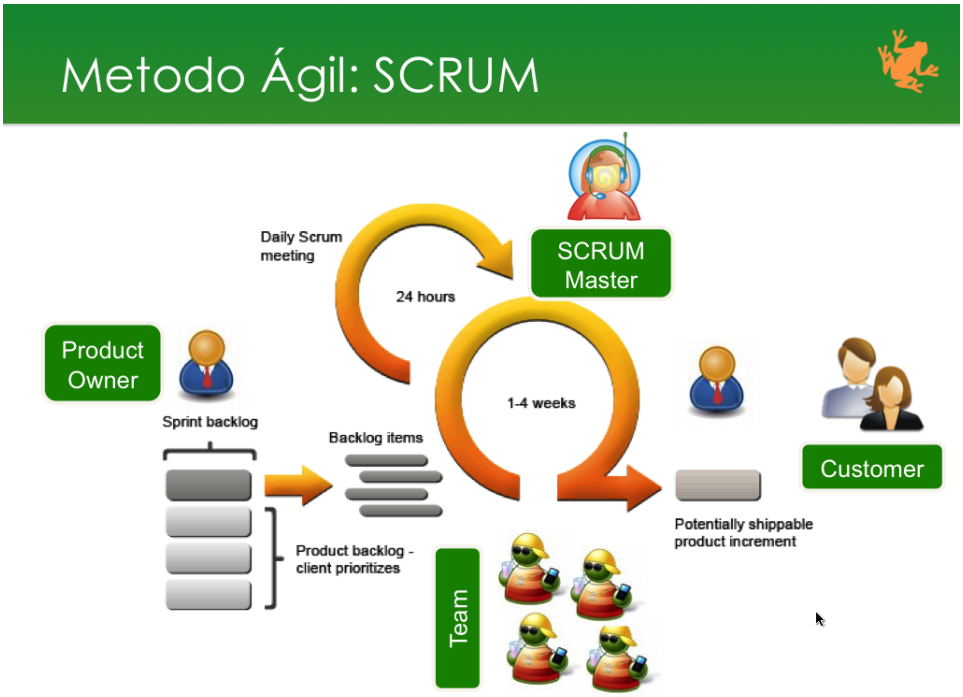


Fig. 4.1: Metodología Scrum
Fuente : Ayselucus.es

De la imagen anterior, se identifican los siguientes agentes o participantes del proceso SCRUM en un equipo de trabajo:

- Product Owner: Es la representación del cliente en el equipo.(Definir las necesidades del cliente en el "Producto Backlog")
- SCRUM Master: El control del proceso de desarrollo, del punto de vista de las practicas de SCRUM.
- Team o Development Team: Encargados de los SPRINT
- Customer: Cliente

A continuación una breve explicación del proceso interno en el que se basa SCRUM para el desarrollo:

- El Product Owner define un documento, o lista de ideas, necesidades y requisitos que satisfacen las solicitudes del cliente, el que se llama Sprint Backlog.
- Sobre este Sprint Backlog se realiza una clasificación por prioridad, extrayendo de el los elementos mas relevantes, o que tienen una prioridad de desarrollo. Este conjunto de elementos se llama Backlog Items.

- El Development Team trabaja durante un periodo definido de semanas (1-4), donde se desarrolla lo especificado en el Backlog Items de manera iterativa, considerando dos mecanismo de control y revisión establecidos, el Daily Scrum y el Sprint Review.
    - El Daily SCRUM es una reunion diaria para realizar seguimiento (Development Team y Scrum master).
    - El sprint REVIEW es la revisión de la entrega o producto incremental. (Scrum Team)
- Por ultimo se realiza una reunión posterior al sprint, para incrementar o iterar el proceso de SPRINT al iniciar nuevamente el ciclo. De esta manera, se incluyen mas elementos o nuevos elementos del Sprint Backlog en el Backlog items a ingresar en el proceso de Sprint.

4.2 Diseño

El sitio web estará constituido por 8 paginas o vistas (considerando el esquema de Laravel), que se detallaran a continuación:

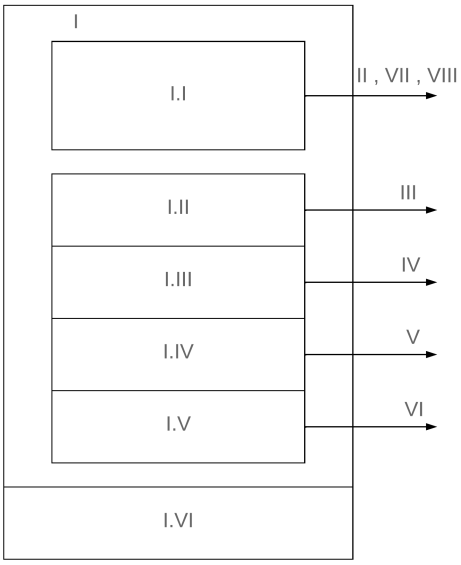


Fig. 4.2: Esquema de vista de inicio
Fuente : Elaboración propia

En la pagina de inicio (I) se observan 6 elementos que se describen a continuación:

I.I Vídeo obtenido de una animación en Python recientemente calculada, que entrega la mejor solución para el TSP fijado para un cierto numero de nodos, costos entre ellos y función de calculo del costo total. Este vídeo se podrá

recalcular cuanto se quiera, pudiendo obtenerse una solución distinta acorde al no determinismo del algoritmo que estará en el Back-end, que para este caso sera el algoritmo genético.

Bajo este vídeo hay 3 botones, el primero de ellos de izquierda a derecha guiara a la pagina web II que permitirá hacer uso de los algoritmos estudiados y programados, acorde a las necesidades del usuario, desde poder seleccionar el numero de nodos, costos entre los nodos e incluso los mecanismos internos de selección de soluciones (cruce, mutación, sus probabilidades, etc).

El segundo botón o botón central es el que permitirá recalcular el vídeo recalculando el algoritmo para el problema fijado. Y finalmente, el ultimo botón permitirá acceder a información sobre el TSP o Travelling Salesman Problem en la pagina web VI.

Sobre este vídeo también existirá un botón, que tendrá información sobre la heurística y la complejidad, a modo de introducción a la lógica de por que aplicar los algoritmos heurísticos y demás elementos en la pagina VII.

I.II Este bloque esta destinado a describir brevemente a la búsqueda local o local search. Se tendrán dos gifs acompañados de un texto descriptivo de ellos. El primer gif mostrara como aparecen soluciones candidatas y elegidas a lo largo de una ejecución de la búsqueda local, y otro gif mostrara como se va disminuyendo el valor a minimizar a medida que sucesivamente se van tomando las soluciones del gif anterior.

En este bloque se incorporara un botón en la esquina inferior derecha para el acceso a la pagina web III que describirá mas en profundidad la búsqueda local, ademas de tener el resumen de resultados para el TSP y un histórico de pruebas realizadas en un problema fijado.

I.III Este bloque describirá al recocido simulado, de manera similar al caso anterior, únicamente diferenciándose en destinar el gif que muestra el comportamiento del valor a minimizar (Para este caso particular) y que este puede ir en aumento o disminución debido al agregar una forma de selección generalizada. De igual manera al caso anterior, abra un botón para acceder una pagina exclusiva referente a este algoritmo. (pagina IV)

I.IV Este bloque contendrá descripción y gifs asociados al algoritmo genético, mostrándose en uno de ellos, como van en evolución los individuos de la población a medida que pasan las generaciones (convergencia), mientras que el otro gif mostrara el valor a minimizar. También se tendrá un botón de acceso a la pagina V donde se profundizara en el algoritmo genético.

I.V Bloque de acceso a la comparativa de resolución del TSP.

I.VI Footer donde se tendrán datos del alumno y la universidad.

A continuación se detalla el esquema que se utiliza para describir los algoritmos, esto es, el esquema de las paginas III, IV y V, que respectivamente reemplazan al símbolo ? en la figura a continuación:

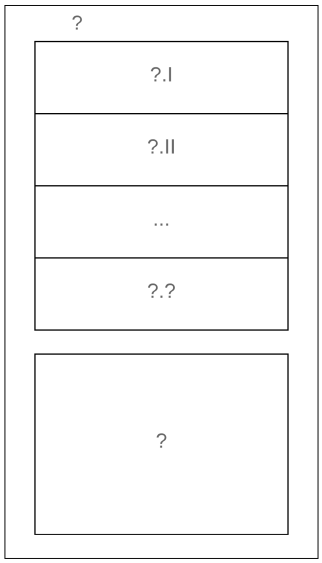


Fig. 4.3: Esquema de vista de descripción
Fuente : Elaboración propia

Los primeros bloques, representaran el mismo esquema de bloque que describe los algoritmos de manera sencilla o breve en el inicio, esto es, usar gifs o imágenes y una descripción, pero en este caso para referirse a los procesos internos del algoritmo, como la forma en la que se escoge una solución vecina por ejemplo, pensando en la búsqueda local o el recocido simulado. De esta manera, se deja variable la cantidad de estos bloques, por que esto dependerá de que tanto se necesite describir al algoritmo, siendo claro que el genético tiene mas procesos internos, tenemos el cruce, la mutación, la selección, el calculo del fitness, etc.

Finalmente, el ultimo bloque indicado con ? mostrara los resultados de aplicarse el algoritmo descrito en la pagina en cuestión, en el TSP, y en efecto lo que se muestra en este bloque dependerá de lo que el algoritmo utiliza para el calculo, parámetros, indicadores, etc.

Las paginas VII y VIII tendrán una estructura similar al caso recién mostrado, exceptuando que no tendrán este ultimo bloque con resultados, por razones

obvias. De esta manera, tendremos dos paginas en los que se describirán al TSP y la heurística y complejidad.

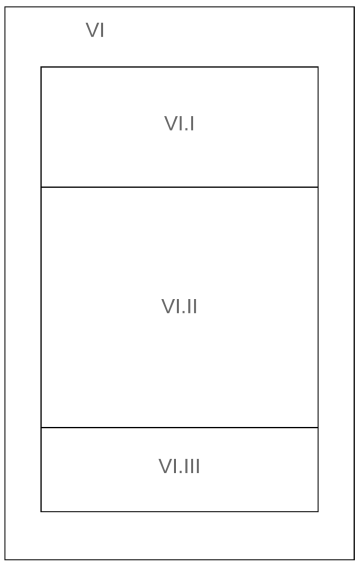


Fig. 4.4: Esquema de vista de comparación
Fuente : Elaboración propia

Acá se observa el esquema de la pagina VI para la comparativa, donde se observan los siguientes elementos:

VI.I Gráficas que muestran las soluciones halladas por cada uno de los algoritmos.

VI.II Tabla comparativa de los algoritmos, considerando sus tiempos, que tan buena es la solución en comparación con la esperada y características propias de los algoritmos.

VI.III Conclusiones.

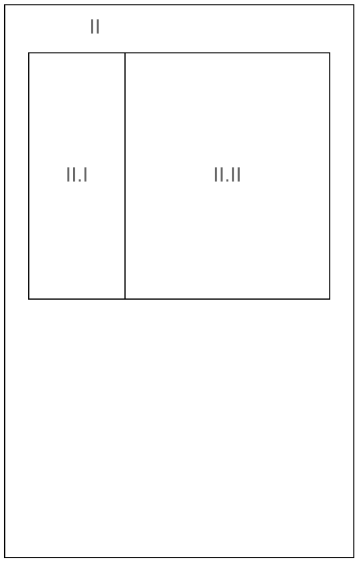


Fig. 4.5: Esquema de vista de aplicación
Fuente : Elaboración propia

Finalmente, se muestra el esquema de la pagina II para el uso de los algoritmos: Donde se tienen los siguientes elementos:

II.I Formulario para la selección del algoritmo, parámetros y métodos.

II.II Resultados obtenidos.

4.2.1 Implementación

EXPLICACIÓN DEL SITIO YA CONSTRUIDO Y MUESTRA DE SUS FUNCIONES Y RESULTADOS.

4.3 Pruebas y resultados

RESULTADOS EN LOS ALGORITMOS, COMPARATIVA DE TIEMPOS, ETC, ETC, ETC.

RESULTADOS DEL SITIO WEB CONSTRUIDO

5. CONCLUSIONES

6. **BIBLIOGRAFÍA**

[1] Aitana Vidal Esmorís (2013) "Algoritmos Heurísticos en Optimización", Universidad de Santiago de Compostela, España.

[2] Mario Morán Cañón "Introducción a la teoría de la complejidad", Universidad de Valladolid, España.

[3] Pablo M. Rabanal Basalo (2010) "Algoritmos heurísticos y aplicación a métodos formales" Universidad complutense de Madrid, España.

[4] Seymour Lipchutz, Ph.D, "Teoria y problemas de topología general", Universidad de Temple.

[5] Hamdy A. Taha(2012), "Investigación de operaciones", University of Arkansas, Fayetteville.

[6] Carlos Ivorra Castillo, "LÓGICA Y TEORÍA DE CONJUNTOS".

[7] Atocha Aliseda Llera (2000). "Heurística, hipótesis y demostración en matemáticas", Universidad Nacional Autónoma de México.

[8] Rodrigo Linfati, John Willmer Escobar,Bernardo Cuevas, "Un algoritmo basado en búsqueda tabú granular para el problema de balanceo de bicicletas públicas usando múltiples vehículos".

[9] Miguel Jiménez-Carrión, "Algoritmo genético simple para resolver el problema de programación de la tienda de trabajo (job shop scheduling)" .

[10] S. Martínez, J. París, I. Colominas, F. Navarrina, M. Casteleiro, "Optimización mixta de estructuras de transporte de energía: aplicación del algoritmo de recocido simulado".

[11] Johanna Rodríguez León ,Jabid Eduardo Quiroga Méndez, Nestor Raul Ortíz Pimiento, "Performance comparison between a classic particle swarm optimization and a genetic algorithm in manufacturing cell design"

[12] Marcos Gestal, "Introducción a los algoritmos genéticos".

[13] Universidad Tecnológica Metropolitana, Sistema de Bibliotecas (2019). Pauta para la presentación de trabajos de titulación. Santiago, Chile. Recuperado de: https://biblioteca.utem.cl/wp-content/uploads/2019/10/Tesis_Normalizacion_04092019.pdf

6.1 Documentación adicional

6.1.1 Recopilación de fuentes alternativas

Para el desarrollo del Marco Teórico:

Para la definición básica de IDO y sus etapas:

https://docplayer.es/29999410-Introduccion-a-la-investigacion-de-operaciones.html

Lectura complementaria para la definición de condiciones necesarias y suficientes: https://es.slideshare.net/josediarte71/investigacin-de-operaciones-31775907

Propiedades y definiciones sobre optimización: http://verso.mat.uam.es/~joser.berrendero/cursos/Matematicas-IO/io-tema4-16.pdf

Continuidad y topología usual de $\mathbb{R}^n$: https://www.dmae.upct.es/~gabi/Civil/ApuntesCivil.pdf

Optimización no lineal y en general, las condiciones necesarias y suficientes para optimalidad en varias variables: https://www.dmae.upct.es/~jose/mastering/optimizacion.pdf

Algoritmos y funciones computables: https://www.cs.us.es/cursos/cc-2018/tema-01.pdf

Sobre optimización dinamica: https://eprints.ucm.es/id/eprint/33921/1/0208.pdf

Matrices semidefinidas: https://www.cimat.mx/~joaquin/mn11/clase07.pdf

Sobre Support vector machine: https://gtas.unican.es/files/docencia/APS/apuntes/07_svm_kernel.pdf

Capitulo 1 sobre propiedades de complejidad algoritmica : http://www.lcc.uma.es/~av/Libro/

Sobre diferenciabilidad en varias variables: https://personal.us.es/pnadal/Informacion/leccion2Diferenciabilidad.pdf

Sobre colonias de hormigas: https://www.youtube.com/watch?v=knYGNpjXqLc

Sobre P vs NP: https://www.youtube.com/watch?v=b7eEFfhkzhM

Listas de reproducción con contenido de ayuda (Algoritmos, heurística,etc): https://www.youtube.com/channel/UCdyoG8-b6G7aLkYb800x6ww

Lista de reproducción sobre problemas NP: https://www.youtube.com/watch?v=ocTZNDPAio0&list=PL6VsnkqbwvkLZ1-DclB_MPzuTm3FMzdkW

Listas de reproducción sobre grafos y algoritmos: https://www.youtube.com/channel/UCOIbYAbmvAN8Sou1UH3FHNw

Lista de reproducción de optimización combinatoria (UPV): https://www.youtube.com/watch?v=wtw_B_3lrjE&list=PL4189642A2A18B4D7&index=4

Sobre cristalización simulada UPV: https://victoryepes.blogs.upv.es/2017/11/07/optimizacion-cristalizacion-simulada/

Para el desarrollo en Latex:

Graficar con tikz: https://www.youtube.com/watch?v=-1048GBUaSY

Sobre tablas: https://manualdelatex.com/tutoriales/tablas http://minisconlatex.blogspot.com/2012/01/mas-sobre-tablas-2.html https://es.overleaf.com/learn/latex/Text_alignment https://es.overleaf.com/learn/latex/Multiple_columns

Notas al pie: https://manualdelatex.com/tutoriales/notas-al-pie

Modificación de fuente: http://elclubdelautodidacta.es/wp/2012/02/latex-capitulo-20-modificando-el-tamano-de-la-fuente/ https://es.overleaf.com/learn/latex/Text_alignment http://minisconlatex.blogspot.com/2012/05/como-cambiar-el-color-de-una-palabra.html

Sobre inserción de imagenes: https://www.overleaf.com/learn/latex/Inserting_Images_es