Optimización aplicada a problemas de negocio

Presentación del curso

Objetivos

Explorar los aspectos de la optimización matemática orientada a la resolución de problemas de negocio.

- Introducir la terminología básica de la optimización matemática
- Exponer los elementos básicos de los modelos de optimización: datos, variables de decisión, funciones objetivo y restricciones
- Introducir los tipos de problemas de optimización y su solución
- Explicar las diferentes técnicas de programación matemática: programación lineal, programación entera, programación cuadrática…
- Explicar metaheurísticas para resolución de problemas de optimización complejos, centrándonos en algoritmos genéticos
- Resolver problemas de negocio con programación matemática y algoritmos genéticos

Presentación del curso

Calendario

	Día 1	Día 2	Día 3	Día 4
09:00	▪ Introducción al curso	▪ Repaso y dudas	▪ Repaso y dudas	▪ Repaso y dudas
10:00	▪ Introducción a la optimización matemática ▪ Ejercicios prácticos	▪ Ejercicios con Pyomo	▪ Detalle de las fases de los algoritmos genéticos e implementación	▪ Proyecto final con programación matemática
	▪ Tipos de optimización matemática		▪ Ejercicio práctico de algoritmos genéticos	▪ Proyecto final con algoritmos genéticos
11:00	DESCANSO (15')			
12:00	▪ Introducción a soluciones con programación matemática ▪ Sintaxis Pyomo	▪ Ejercicios con Pyomo	▪ Ejercicios de algoritmos genéticos con Python	▪ Proyecto final con algoritmos genéticos
13:00	▪ Ejercicios con Pyomo	▪ Introducción a los algoritmos genéticos		▪ Detalle de casos reales con algoritmos genéticos

Índice

1. Fundamentos de la optimización matemática

1. Fundamentos de la optimización matemática

¿Qué es optimización?

Tiende a malinterpretarse, ya que **cada persona parece tener su propia** interpretación de lo que significa:

- “Prueba y error"
- Enumerar posibilidades de diseño y elegir la mejor
- Hacer sugerencias cualitativas que conducen a un mejor diseño de producto.
- …

Los métodos anteriores pueden ser considerados de optimización en un sentido amplio. Pero definiremos la **optimización** como una metodología cuantitativa y sistemática para **buscar el mejor diseño entre numerosas posibilidades al tiempo que se satisfacen las limitaciones dadas**.

1. Fundamentos de la optimización matemática

Introducción

La optimización es una **técnica matemática** para encontrar un valor **máximo o mínimo de una función** de varias variables independientes sujetas a un conjunto de restricciones.

Variables de decisión

- Variables **independientes**: sus valores se pueden cambiar para modificar el comportamiento del sistema
- Variables **dependientes**: su valor está determinado por las variables independientes
- **Tipos** de variables: discretas (enteras, booleanas), continuas (reales)
- **Dominio** (conjunto de valores que pueden tomar las variables)

Función objetivo

- En negocio se puede interpretar como el objetivo comercial
- Se calcula a partir de las variables de decisión

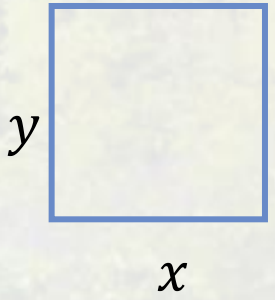
Restricciones

- En negocio, reglas comerciales
- Restricciones e interacciones que **limitan los valores de las variables de decisión**. En ocasiones, el dominio vendrá definido a partir de restricciones.
- Si existen restricciones se considera un problema de "*Constrained optimization"* y, en caso de no existir, se considera de *"Unconstrained Optimization"*

1. Fundamentos de la optimización matemática

Ejercicios de formulación

- Un hombre tiene 300 metros de valla, un gran patio y un perro. Quiere crear un recinto rectangular para su perro con la valla que proporcione el área máxima. ¿Qué dimensiones proporcionan el área máxima?



  - Variables de decisión
    - Variables independientes

x, y

  - Función objetivo
    - $\max area(x, y)$

donde $area(x, y) \coloneqq xy$

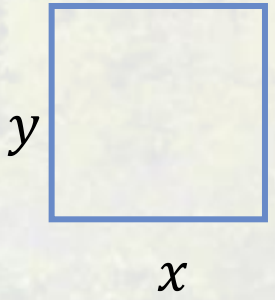
  - Restricciones
    - $2x + 2y \leq 300$

  - Dominio de las variables
    - $x, y \geq 0$

1. Fundamentos de la optimización matemática

Ejercicios de formulación

- Un hombre tiene 300 metros de valla, un patio de 60x100 y un perro. Quiere crear un recinto rectangular para su perro con la valla que proporcione el área máxima. ¿Qué dimensiones proporcionan el área máxima?



  - Variables de decisión
    - Variables independientes

x, y

  - Función objetivo
    - $\max area(x, y)$

      donde $area(x, y) := xy$

  - Restricciones
    - $2x + 2y \leq 300$
    - $x \leq 60$
    - $y \leq 100$

  - Dominio de las variables
    - $x, y \geq 0$

Equivalentemente:

- Restricciones
  - $2x + 2y \leq 300$ ➡ $x + y \leq 150$

- Dominio de las variables
  - $0 \leq x \leq 60$
  - $0 \leq y \leq 100$

1. Fundamentos de la optimización matemática

Ejercicios de formulación

Los propietarios de una empresa de alquiler de automóviles han determinado que si cobran a los clientes p euros por día por alquilar un automóvil, la demanda de automóviles que alquilan por día se modela por $demanda(p) := 1000 - 5p$. ¿Cuánto deberían cobrar para maximizar sus ingresos?

- Variables de decisión
  - Variable independiente

  p
  - Variable dependiente

  $demanda(p)$

- Función objetivo
  - $\max\ p \cdot demanda(p)$

- Restricciones/dominio
  - $0 \leq p \leq 200$

1. Fundamentos de la optimización matemática

Ejercicios de formulación

Una compañía petrolera produce dos mezclas de combustible mezclando tres aceites y desea maximizar las ganancias de la producción y la venta. Cada litro de combustible 1 se puede vender a 1.10 € y cada litro de combustible 2 se puede vender a 1.20 €. Se requiere producir al menos 10000 L de cada combustible.

Combustible	Requerimientos de calidad
Fuel1	Al menos 30% de A Como mucho 50% de B Al menos 30% de C
Fuel2	Como mucho 40% de A Al menos 35% de B Como mucho 40% de C

Aceite	Coste (€/L)	Cantidad disponible (L)
A	0.28	6000
B	0.46	10000
C	0.41	12000

Variables de decisión

Variables independientes: $A_1, B_1, C_1, A_2, B_2, C_2$

Variables dependientes: $F_1 = A_1 + B_1 + C_1$

$$F_2 = A_2 + B_2 + C_2$$

Función objetivo

Maximizar $1.1F_1 + 1.2F_2 - [0.28(A_1 + A_2) + 0.46(B_1 + B_2) + 0.41(C_1 + C_2)]$

- Restricciones

Cantidades

$$F_1 \geq 10000$$
$$F_2 \geq 10000$$
$$A_1 + A_2 \leq 6000$$
$$B_1 + B_2 \leq 10000$$
$$C_1 + C_2 \leq 12000$$

Calidad combustible

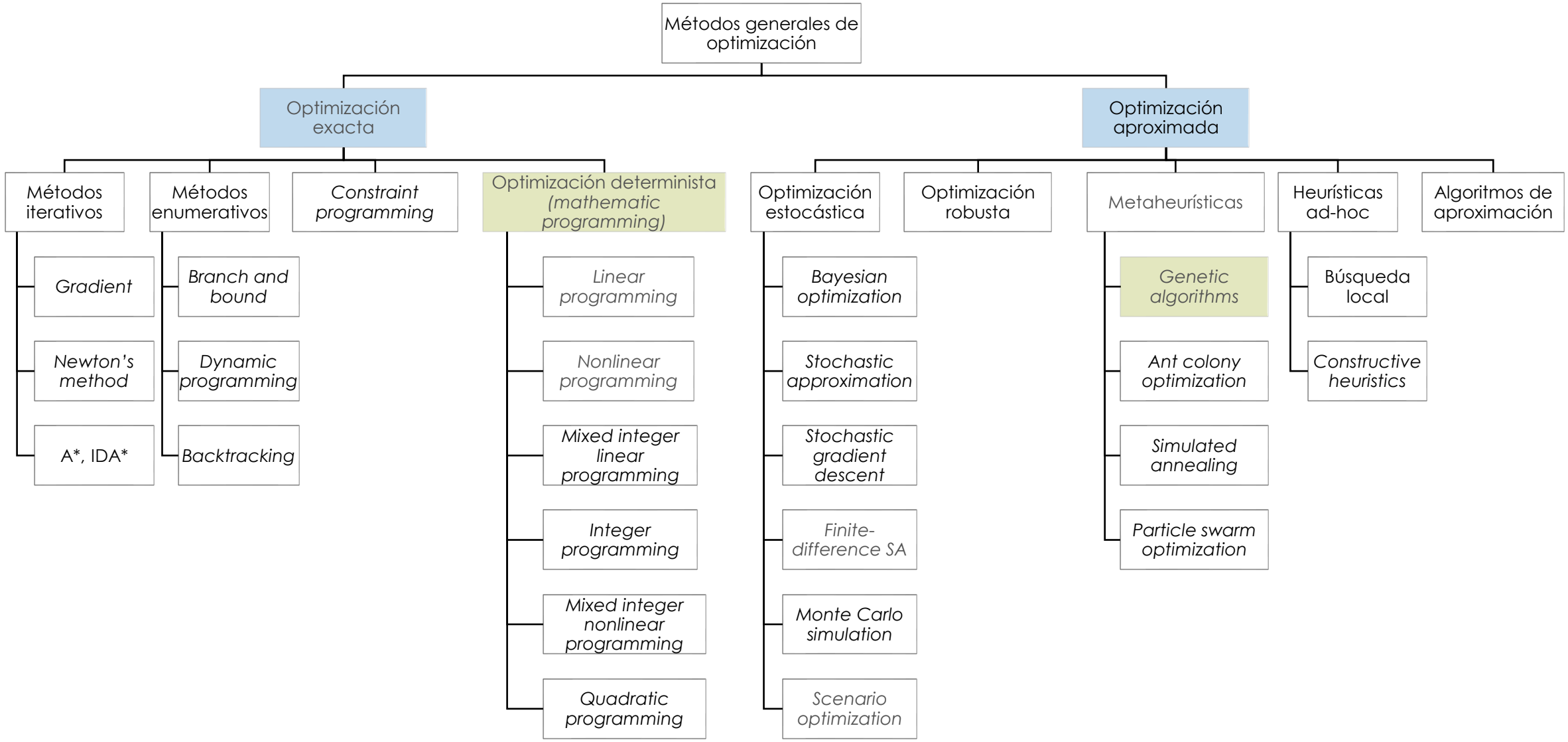
$$A_1 \geq 0.3F_1$$
$$B_1 \leq 0.5F_1$$
$$C_1 \geq 0.3F_1$$
$$A_2 \leq 0.4F_2$$
$$B_2 \geq 0.35F_2$$
$$C_2 \leq 0.4F_2$$

No negativos

$$A_1, A_2, B_1, B_2, C_1, C_2 \geq 0$$

1. Fundamentos de la optimización matemática

Taxonomía de los métodos de optimización



1. Fundamentos de la optimización matemática

Óptimo local y global

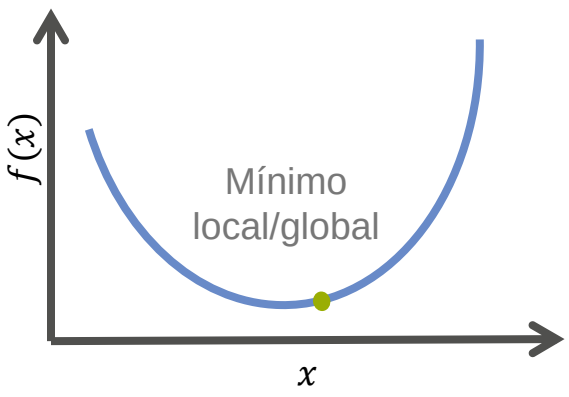
Óptimo local

- Un mínimo (resp. máximo) local de una función es un punto donde el valor de la función es menor (resp. mayor) que en su entorno, pero no necesariamente el valor más pequeño de todo el dominio de las variables.
    - *Greedy*
    - *Hill-climbing*
    - *Heuristics*
    - …

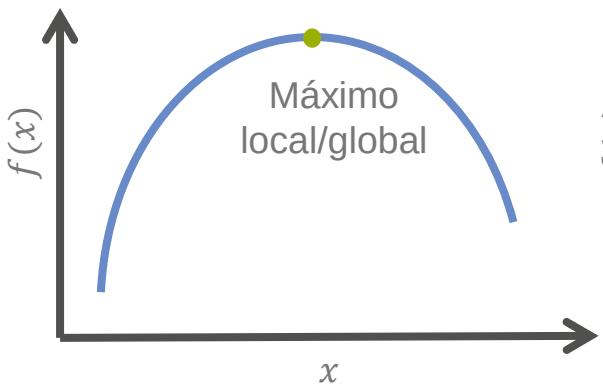
Óptimo global

- Un mínimo (resp. máximo) global es un punto donde el valor de la función es menor (resp. mayor) que en todos los demás puntos factibles.
    - Linear/non-linear programming
    - Mixed integer programming
    - Bayesian optimization
    - Genetic algorithms
    - Ant colony optimization

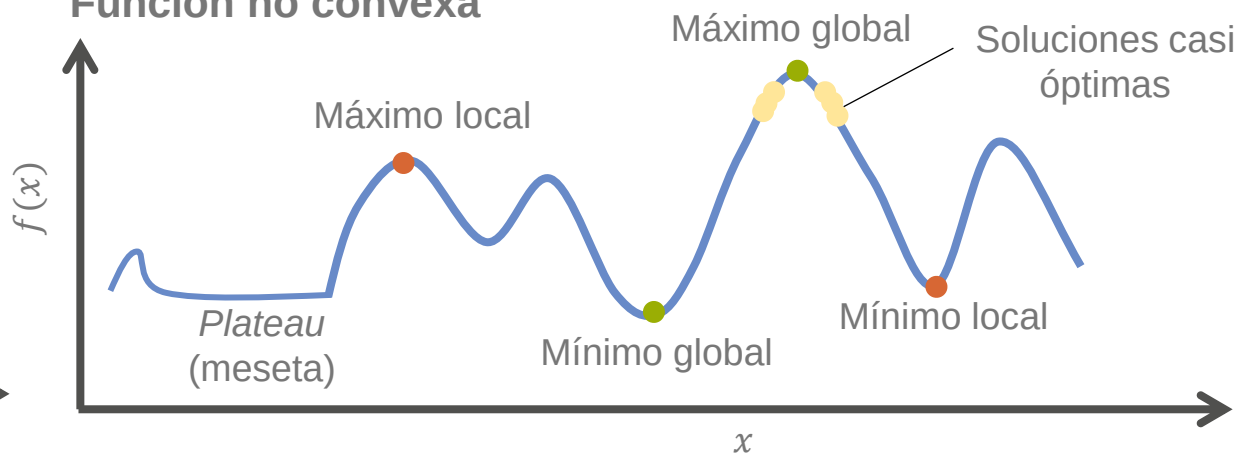
Función convexa



Función cóncava



Función no convexa



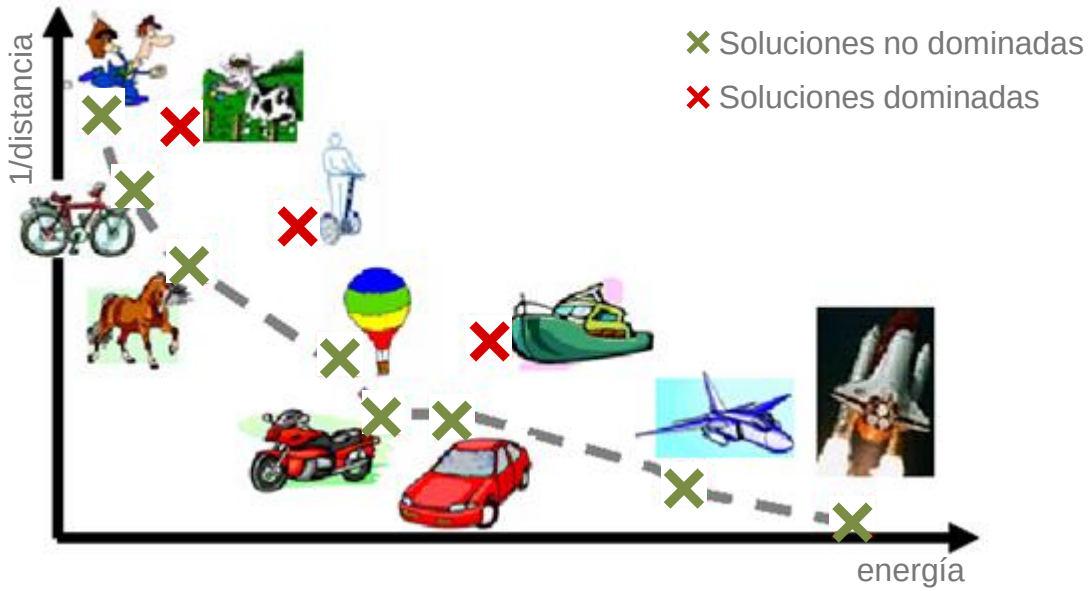
1. Fundamentos de la optimización matemática

Funciones objetivo

La mayoría de los problemas de optimización tienen una única función objetivo, sin embargo, hay casos en los que los problemas de optimización no tienen una función objetivo o tienen múltiples funciones objetivo.

- Problemas de **viabilidad:** son problemas en los que el objetivo es encontrar valores para las variables que satisfagan las restricciones de un modelo sin un objetivo particular a optimizar.

- Problemas de **optimización multiobjetivo:** presentes en campos como la ingeniería, la economía y la logística, cuando es necesario tomar decisiones óptimas en presencia de compensaciones entre dos o más objetivos en conflicto (objetivos no disjuntos).
  - Encontrar **soluciones no dominadas** (no existe una solución mejor que el resto en todos los objetivos)
  - **Reformular** como problemas de un solo objetivo (formando una combinación ponderada de los diferentes objetivos o reemplazando algunos de los objetivos por restricciones)



P. Ngatchou, Anahita Zarei, M. El-Sharkawi (2005), Pareto Multi Objective Optimization, in *Proceedings of the 13th International Conference on, Intelligent Systems Application to Power Systems.*

1. Fundamentos de la optimización matemática

Métodos de solución

Métodos exactos

- Resuelven un problema de optimización **siempre** de **manera óptima** (si tiene las condiciones para ser resuelto por el algoritmo).
- Para un problema de optimización NP-*hard* no pueden ejecutarse en tiempo polinomial
    - P.e., problemas que necesitan una cantidad exponencial de tiempo para ser resueltos (n^m).

Métodos de aproximación

- Encuentran soluciones aproximadas a problemas de optimización (en particular, problemas NP-hard) con garantías demostrables sobre la distancia de la solución encontrada respecto a la óptima.

(Meta)heurísticas

- Encuentran soluciones razonablemente buenas (aproximadas) para diferentes situaciones (datos) de un problema, pero no proporcionan una indicación clara de cuándo pueden tener éxito o fallar.
- La diferencia entre algoritmos aproximados (*approximate*) y de aproximación (*approximation*) es que los primeros tratan de obtener una buena suposición de la solución de un problema, pero que realmente no se sabe qué tan buena será, sin embargo, los segundos garantizan la obtención de una solución con un límite de error demostrable en el peor de los casos.

1. Fundamentos de la optimización matemática

Optimización determinista y optimización estocástica

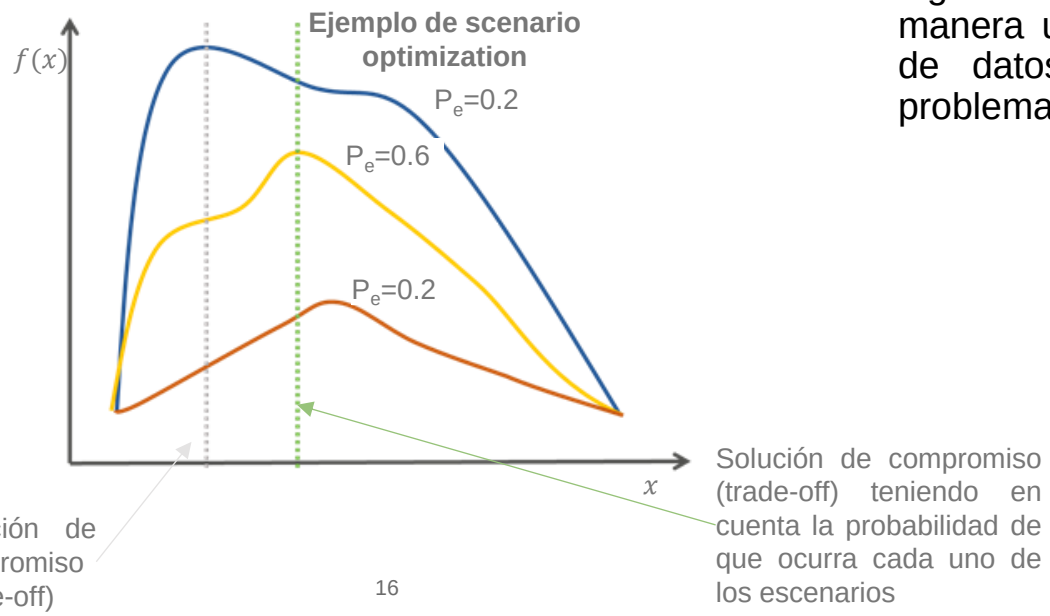
Optimización determinista

Los datos para el problema se conocen con precisión.

Sin embargo, en muchos problemas reales, los datos no se pueden conocer con precisión (e.g., errores de medición, datos que representan situaciones futuras y que no se pueden conocer con certeza).

Optimización estocástica

Si las distribuciones de probabilidad de los datos son conocidas o pueden estimarse, el objetivo es encontrar una solución que sea factible para todas (o casi todas) las posibles instancias de datos y optimice el rendimiento esperado del modelo



Optimización bajo incertidumbre

(Randomized search methods)

Para acelerar el progreso y acercarse a un óptimo global tratando de evitar un óptimo local hay métodos que introducen aleatoriedad en el proceso de búsqueda .

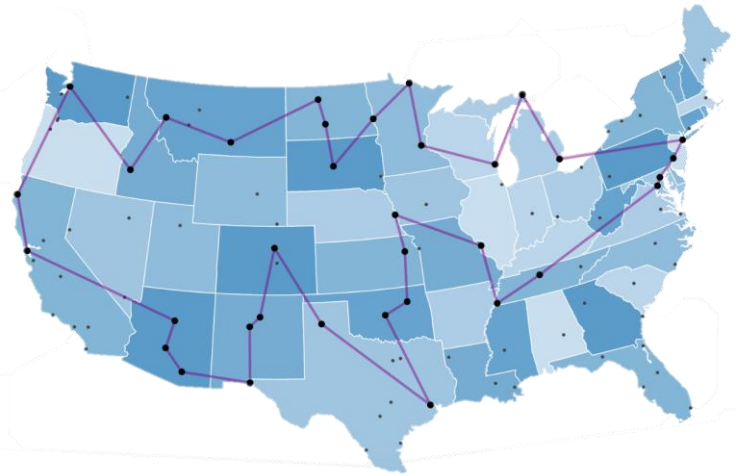
Este principio de aleatorización es una forma simple y eficaz de obtener algoritmos con un buen rendimiento de manera uniforme en muchos conjuntos de datos y para muchos tipos de problemas.

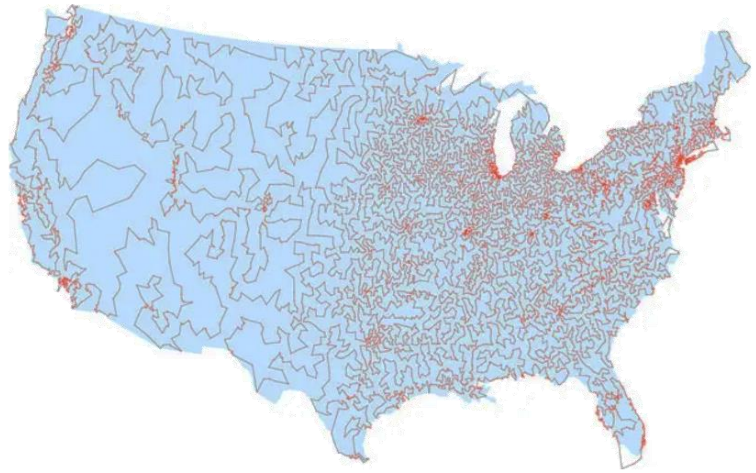
1. Fundamentos de la optimización matemática

Limitaciones de los métodos exactos

- Problema del viajante de comercio (*Travelling Salesman Problem*, **TSP**)
- Dada una lista de ciudades y las distancias entre cada par de ciudades, se debe encontrar la **ruta más corta** posible que visite **cada ciudad únicamente una vez** y regrese a la ciudad de origen.
- Tiempo de ejecución
  - Método de fuerza bruta: $O(n!)$
  - Dynamic programming: $O(n^2 \cdot 2^n)$
    - ✓ 10 ciudades: 0.2 segundos usando intel core i7
    - ✓ 15 ciudades : 13 segundos usando intel core i7
    - ✗ 100 ciudades : $2.94 \cdot 10^{139}$ años

    (edad de la Tierra es $4.54 \cdot 10^9$ años)
  - Es necesario aplicar métodos aproximados (*near-optimal solutions*)
    - Lectura recomendada para algoritmos de aproximación: https://www.wired.com/2013/01/traveling-salesman-problem/
    - Nos centramos en los métodos metaheurísticos (*Metaheuristic Optimization*)





1. Fundamentos de la optimización matemática

Métodos metaheurísticos

- Métodos eficientes para problemas difíciles(1) de optimización; i.e., problemas que no se pueden resolver de manera óptima utilizando un enfoque determinista dentro de un límite de tiempo razonable
- Sus propósitos principales son:
  - Resolución rápida de problemas.
    - Son simples de diseñar, flexibles y fáciles de implementar.
    - No imponen exigencias a la formulación del problema.
  - Resolver problemas con una gran combinatoria en un tiempo razonable
    - Mejor equilibrio entre la calidad de la solución y el tiempo de computación
  - Obtención de algoritmos más robustos y adaptables
    - Al definirse en términos generales, que pueden adaptarse a las necesidades cambiantes de la mayoría de los problemas de optimización de la vida real.
- Algunos algoritmos:
  - *Simulated annealing*
  - *Particle swarm optimization*
  - *Ant colony optimization*
  - *Tabu search*
  - ***Evolutionary computation (genetic algorithms)***
  - …

(1) Más info sobre complejidad computacional en el anexo.

2. Programación matemática

2. Programación matemática

Introducción

- La programación matemática se refiere a **modelos matemáticos utilizados para resolver problemas** como problemas de decisión, en contraste con otras técnicas (e.g., heurísticas) que resuelven los problemas mediante la implementación de algoritmos diseñados específicamente para un problema determinado.

- Por programación matemática, consideramos **enfoques declarativos**. Esto significa que se considera una separación entre la representación del problema mediante un modelo matemático y su resolución. La idea es que la resolución se puede realizar a través de métodos generales, como los métodos de ramificación (*branching*), utilizando el modelo matemático diseñado para describir el problema.

- Muchos problemas del mundo real en áreas tan diferentes como la producción industrial, el transporte, las telecomunicaciones, las finanzas o la planificación del personal pueden presentarse en la forma de un problema de Programación Matemática: un conjunto de variables de decisión, restricciones sobre estas variables y una función objetivo que se debe maximizar o minimizar.

2. Programación matemática

Tipos de problemas

Tipo de problema	Variables	Restricciones	Función objetivo
Linear programming (LP)	Reales	Lineales	Lineal
Integer linear programming (ILP)	Enteras	Lineales	Lineal
Mixed integer progamming (MIP)	Enteras y reales	Lineales	Lineal
Quadratic progamming (QP)	Reales	Lineales	Cuadrática
Nonlinear programming (NLP)	Reales	Cualquiera	Cualquiera
Constraint programming	Enteras	Cualquiera	-

2. Programación matemática

Datos, modelado y solvers

Formato de datos

Definición de los valores de datos que definen una instancia particular del problema.

Ejemplos: MPS, nl, sol.

Herramienta de modelado

Librerías y softwares que permiten desarrollar y mantener los modelos de optimización: definición de las variables, restricciones, función objetivo…

Ejemplos: AMPL, Mathematica, PuLP, Pyomo.

***Solver* (motor de optimización)**

Los *solvers* son aplicaciones diseñadas para resolver problemas de optimización genéricos mediante algoritmos conocidos adecuados al tipo de problema (LP, NLP, QP…).

Ejemplos: CPLEX, Gurobi, SciPy, GLPK, IPOPT…

2. Programación matemática

Herramientas de modelado en Python

Módulo	Objetos propios	Solver externo	Procesado en paralelo	Modelado de optimización lineal	Modelado de optimización multiobjetivo	Modelado de optimización no lineal
Scipy				X		
Pulp	X	X		X		
Pyomo	X	X	X	X	X	X
jMetalPy	X	X	X	X	X	

Pyomo (Python Optimization Modeling Objects):

- Estilo orientado a objetos para formular los modelos
- Los modelos de optimización se pueden inicializar con datos de Python y las fuentes de datos externas se pueden definir mediante hojas de cálculo, bases de datos y varios formatos de archivos de texto.
- Admite tanto modelos abstractos, que se definen sin datos, como modelos concretos, que se definen con datos.
- Soporta diferentes solvers.
- Open-source.

2. Programación matemática

Solvers

Solver	El uso académico / no comercial es gratuito	Puede usarse en aplicaciones propietarias	Licencia	Descripción
CPLEX	Sí	Sí	Commercial, academic, trial	IBM CPLEX Optimization Studio es un conjunto de motores de optimización (CPLEX para programación matemática y CP Optimizer para programación de restricciones), un lenguaje de modelado (OPL) y un entorno de desarrollo integrado.
Gurobi	Sí	Sí	Commercial, academic, trial	Maneja problemas lineales de enteros mixtos, restricciones cuadráticas convexas, y optimización de un objetivo y multiobjetivo. Fiable frente a solvers open-source que no consiguen obtener una solución correcta en tiempo razonable (+ 10,000% más rápido que otros solvers de código abierto).
IPOPT	Sí	Sí	Eclipse Public License	IPOPT es un paquete de software de código abierto para optimización no lineal (NLP) a gran escala.
GLPK	Sí	No	GPL	Biblioteca gratuita para programación lineal (LP) y programación de enteros mixta (MIP).
Couenne	Sí	No	GPL	Couenne es una biblioteca de optimización no convexa de código abierto desarrollada para encontrar óptimos globales de problemas de programación de enteros mixta no lineal (MINLP).
SciPy	Sí	Sí	BSD	Biblioteca de computación científica y numérica de propósito general para Python.

https://en.wikipedia.org/wiki/Comparison_of_optimization_software

2. Programación matemática

Tipos de modelos en Pyomo

Modelo Concrete

- Se construye en un paso (datos y modelo a la vez).
- Pyomo construirá cada componente en orden en el momento en que se declare.
- Todos los datos deben estar presentes antes de que Python comience a procesar el modelo.
- Proceso lógico y sencillo; fácil de programar.

Ejemplos simples de modelo Concrete y Abstract:
Simple Models — Pyomo 6.4.0 documentation

Modelo Abstract

- Se construye en dos pasos (primero el modelo y luego los datos).
- Pyomo almacena las declaraciones del modelo básico, pero no construye los objetos reales.
- En el "momento de la creación", los datos se aplican a la declaración abstracta para crear una instancia concreta (los componentes también se construyen en el orden de la declaración).
- Requiere un grado de abstracción y complejidad superior.
- **Fomenta el modelado genérico** y la reutilización de modelos, por ejemplo, el modelo se puede utilizar para distintos datos y, en particular, datos de diferente tamaño.
- Similar a la manera de modelar en AMPL.

2. Programación matemática

Sintaxis en Pyomo

Modelo Concrete

- import pyomo.environ as pyo
- from pyomo.opt import SolverFactory
- Modelo: model = pyo.**ConcreteModel**()
- Variables: model.nombrevariable = pyo.**Var**()
  - Argumentos: (index/components), bounds, domain/within, initialize
  - model.x = pyo.Var([1, 2, 3, 4, 5], domain=pyo.Binary)
- Función objetivo: model.fobj = pyo.**Objective**()
  - Argumentos: expr/rule, sense (pyo.minimize by default)
  - model.OBJ = pyo.Objective(expr = 2*model.x[1] – 3*model.x[2], sense=pyo.maximize)
- Restricciones: model.constraint_name = pyo.**Constraint**()
  - Argumentos: expr/rule
  - model.c1 = pyo.Constraint(expr = model.x[2] + model.x[3] == 1)
- Solvers: opt = **Solverfactory**()
  - opt = Solverfactory(‘glpk’)
  - opt.solve(model)

Modelo Abstract

- Imports, variables, función objetivo, restricciones y solver como en modelo concrete.
- Modelo: model = pyo.AbstractModel()
- Conjuntos: model.S = pyo.Set() (pyo.RangeSet())
  - Argumentos: dimen, initialize, within, …
  - model.S1 = pyo.Set(initialize=[‘e1', ‘e2', ‘e3'])
  - model.S2 = pyo.RangeSet(1, 10, 2)
- Parámetros/constantes/coeficientes: model.p = pyo.Param()
  - Argumentos: (index/components), initialize, default
  - model.p1 = pyo.Param([1, 2, 3], default=0)

2. Linear and nonlinear programming

Información adicional

- Data Portals class: Posibilita la importación de datos desde otros archivos distintos a .dat.
- Asignación de un nombre al modelo:
  - model.name = “Nombre del modelo”
- Conexión de google Colab con drive:
  - from google.colab import drive
  - drive.mount('/content/drive')
- Acceso a la carpeta deseada desde colab:
  - !pwd
  - %ldir
  - %cd /content/drive/MyDrive/optimizacion/Alumno/

2. Linear and nonlinear programming

Ejercicios prácticos

Pyomo_Valla_Alumno.ipynb

Pyomo_AlquilerCoche_Alumno.ipynb

Pyomo_Combustible_Alumno.ipynb

Pyomo_MochilaConcrete_Alumno.ipynb

Pyomo_MochilaAbstract_Alumno.ipynb + mochila.dat

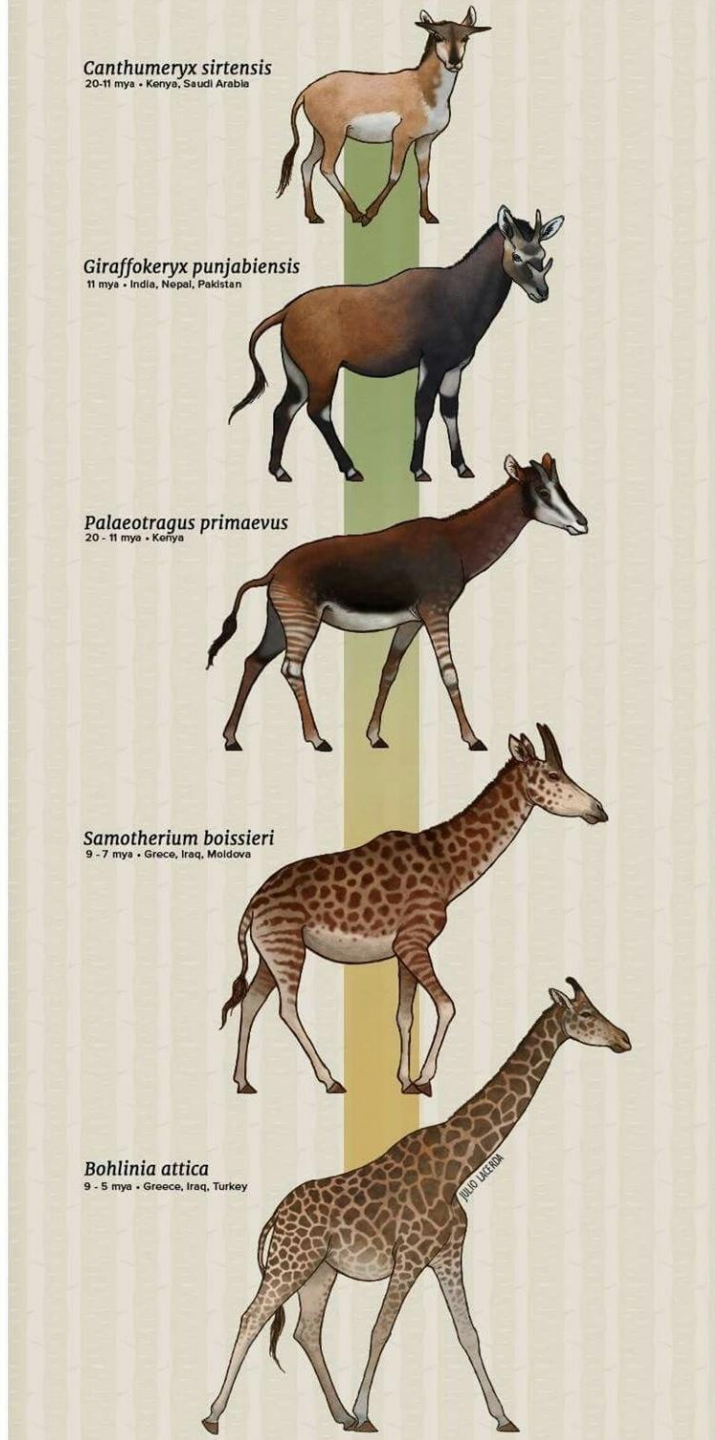
Pyomo_DietaAbstract_Alumno.ipynb + diet.dat

3. Algoritmos genéticos

3. Algoritmos genéticos

Computación evolutiva

- La computación evolutiva es una familia de métodos de búsqueda y optimización
- Optimización global (soluciones casi óptimas)
  - Optimización con restricciones
  - Optimización no lineal
- Optimización metaheurística basada en población
  - Robusta
  - Eficiente y eficaz para explorar espacios de búsqueda grandes, complejos y desconocidos
- Replica el mismo proceso que realiza la naturaleza
  - Teoría de la evolución por selección natural
  - Herencia
- Conceptos genéricos
  - Una población de soluciones candidatas que compiten entre ellas
  - Combinación aleatoria y alteración de estructuras potencialmente útiles para generar nuevas soluciones
  - Un mecanismo de selección para aumentar la proporción de mejores soluciones
- Ejemplos:
  - **Algoritmos genéticos** (genetic algorithms)
  - Estrategias evolutivas (evolution strategies)
  - Programación evolutiva (evolutionary programming)
  - Programación genética (genetic programming)



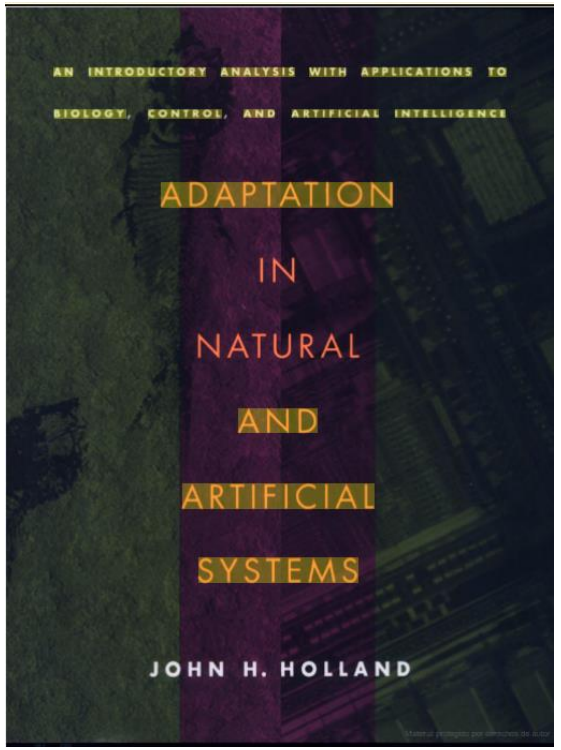
3. Algoritmos genéticos

Historia

- Fueron propuestos por primera vez por Holland en la década de 1960.

- Holland, J. H. (1975). Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. MIT Press, Cambridge, MA, USA.

- Goldberg, D. E. (1989). Genetic Algorithm in Search, Optimization, and Machine Learning. Addison-Wesley, Inc., Boston, MA, USA.



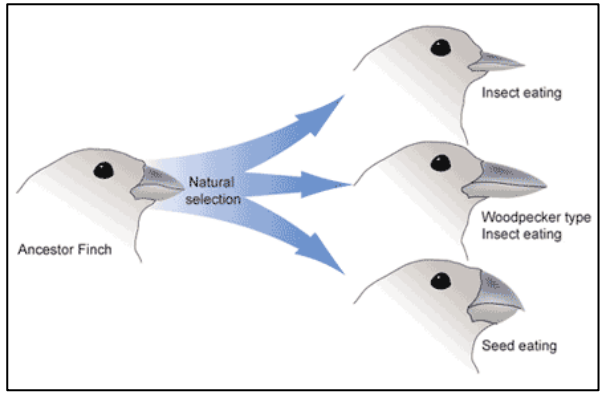
- Probado teórica y empíricamente para proporcionar una búsqueda sólida en espacios complejos
- Son un enfoque válido a los problemas que requieren búsquedas eficientes y efectivas.
- No garantizan encontrar la solución global óptima a un problema, pero en general son buenos para encontrar soluciones aceptablemente buenas (casi óptimas).

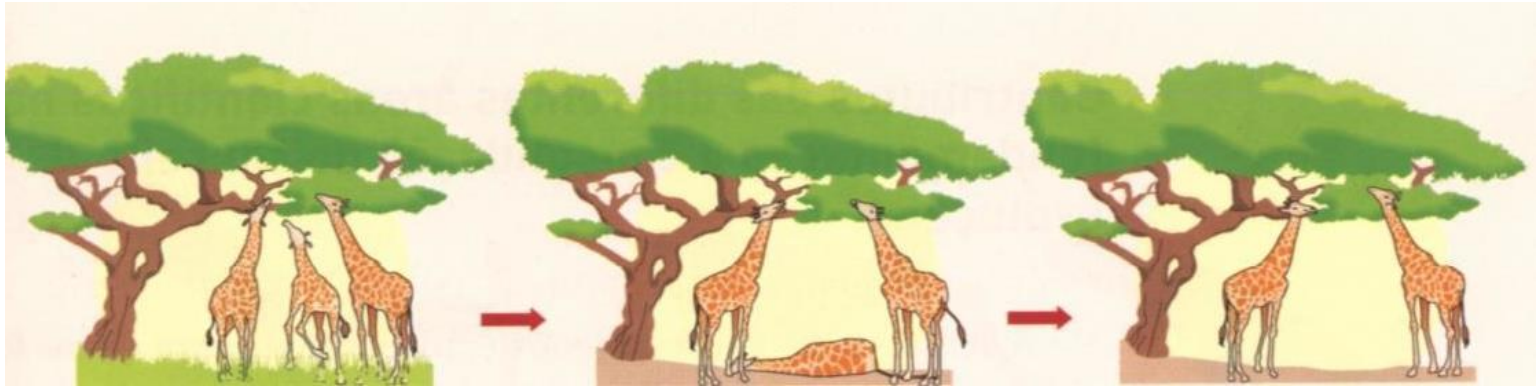
3. Algoritmos genéticos

Teoría de la evolución por selección natural

- Introducido por Charles Darwin en 1859
  - *On the Origin of Species*

- Explica cómo los seres vivos evolucionan a partir de pequeños cambios y de la selección de los individuos más aptos.
  - *Survival of the fittest*
    - Herbert Spencer utilizó por primera vez esta frase en sus *Principles of Biology* (1864).

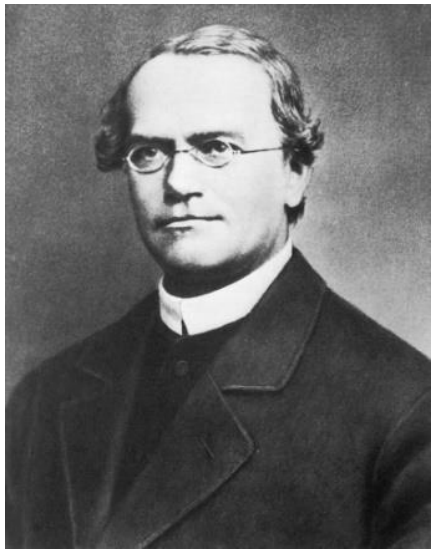


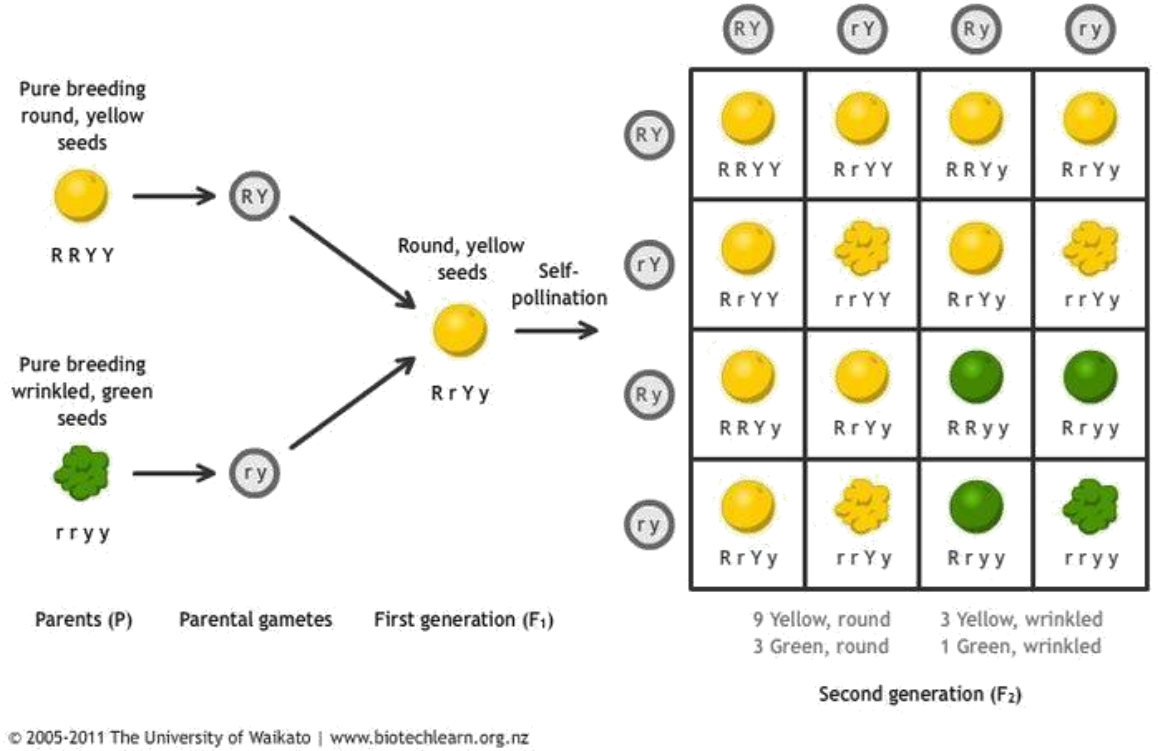


3. Algoritmos genéticos

Herencia

- Propuesto por Gregor Mendel en 1865
- Principios básicos de la transferencia de factores hereditarios de padres a hijos
- Leyes de Mendel
  - En 1900, su trabajo fue "redescubierto" por Hugo de Vries, Carl Correns y Erich von Tschermak.
- Es considerado el padre de la genética moderna.

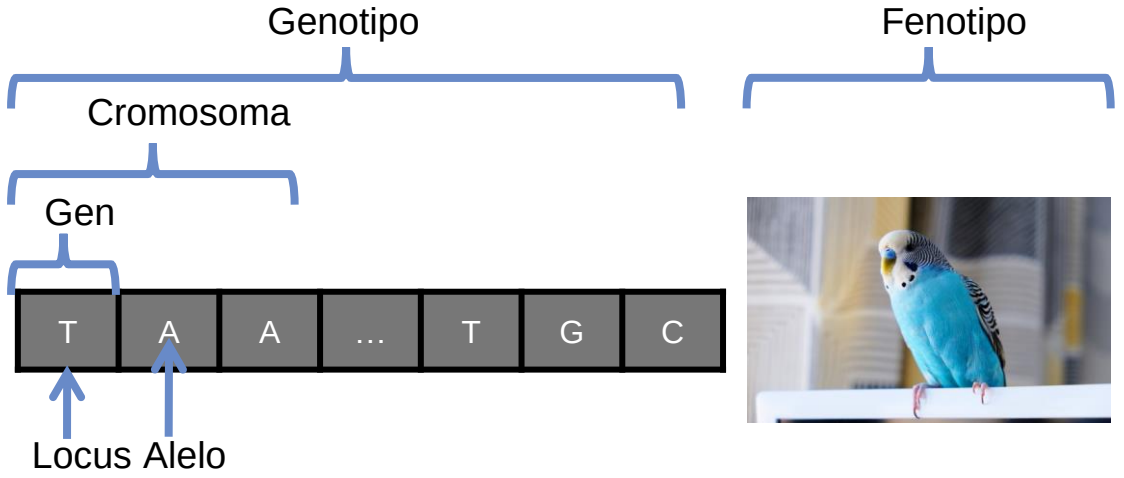


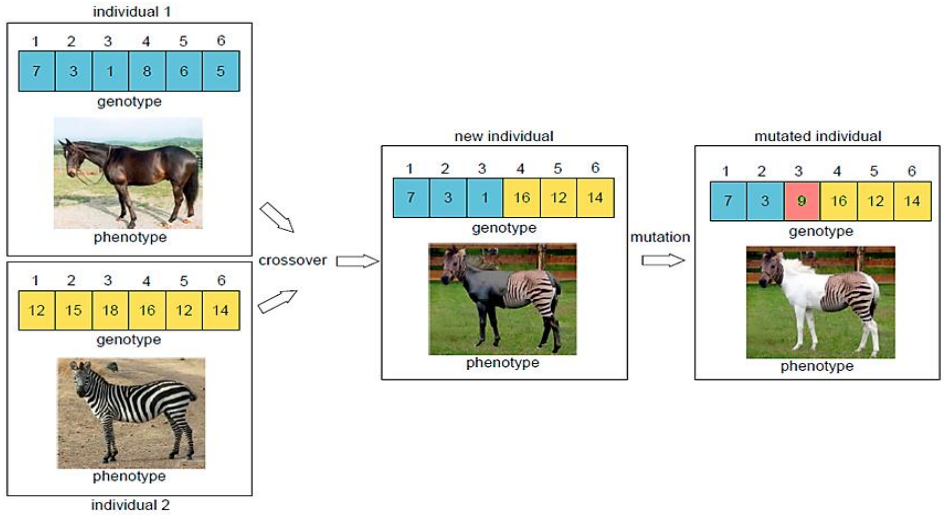


3. Algoritmos genéticos

Aplicación de los términos biológicos

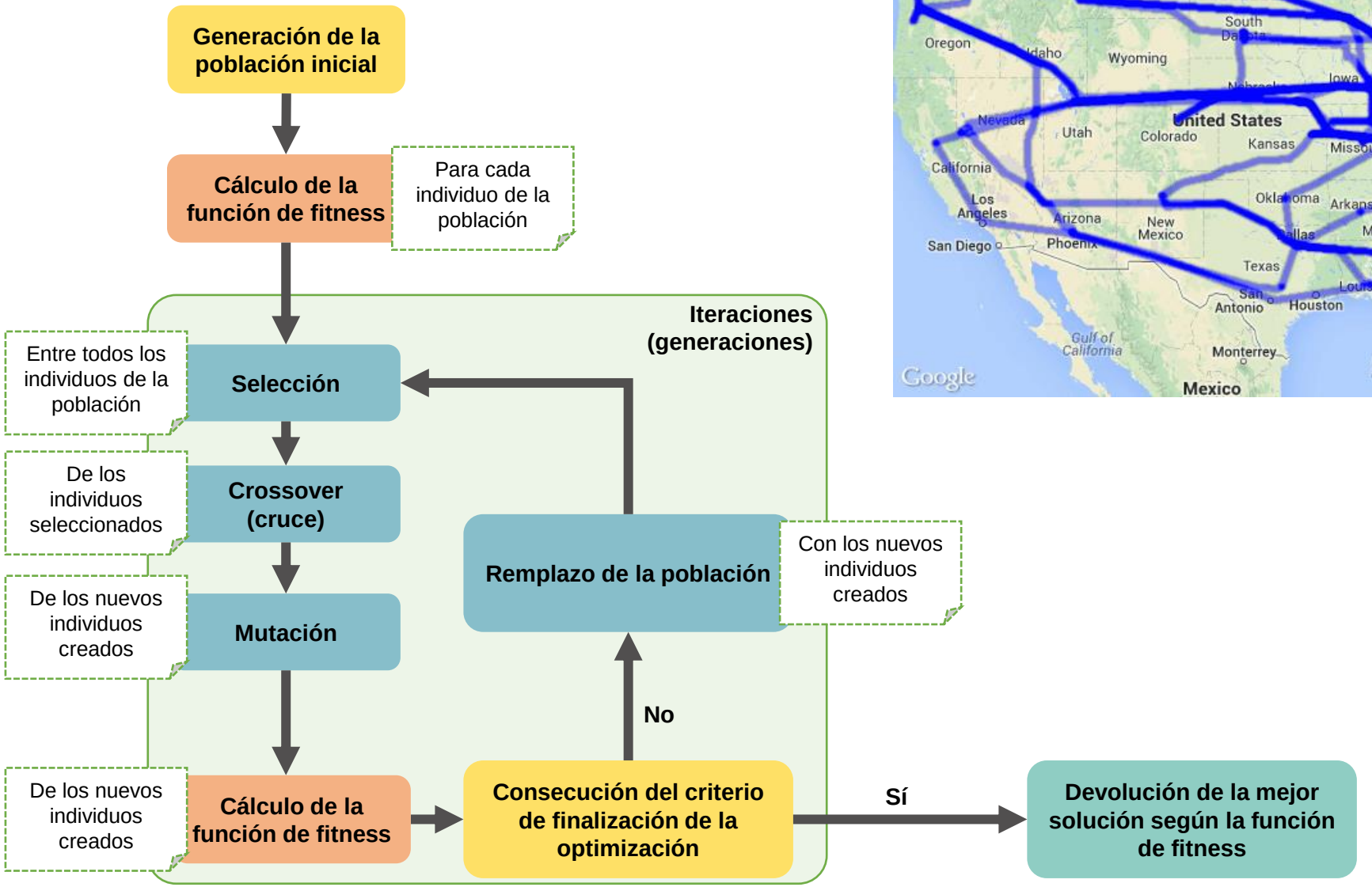
- Cromosoma / genotipo
  - Conjunto de parámetros (variables independientes) que codifican una solución candidata al problema de optimización.
  - Por ejemplo: cadena binaria
- Fenotipo
  - Aquello representado por el genotipo
  - Por ejemplo: color del pájaro, altura, tamaño del pico, …
- Gen
  - Parámetro específico de la solución
  - Por ejemplo: en una cadena binaria, un simple bit
- Alelo
  - El valor del gen
  - Por ejemplo: en una cadena binaria, 0 ó 1
- Locus
  - La posición del gen
  - Por ejemplo: posición en la cadena





3. Algoritmos genéticos

Algoritmo



3. Algoritmos genéticos

Aplicación de los términos biológicos

Evolución y herencia

- Recombinación de dos cromosomas diferentes
- Mutación de genes para generar nuevos rasgos
- Selección de individuos mejor adaptados

Operadores genéticos

Supervivencia del más apto

- El término fitness (aptitud) describe la calidad de un individuo

Un algoritmo genético procesa una población de cromosomas (individuos) de una generación a la siguiente

- El fitness de un individuo se describe mediante una función objetivo escalar
- Los individuos con mejor fitness tienen más probabilidades de ser seleccionados para reproducir descendencia en la próxima generación
- El crossover (cruce) y la mutación generan nuevos individuos
    - Durante el ciclo evolutivo, surgen cada vez más soluciones adecuadas.

3. Algoritmos genéticos

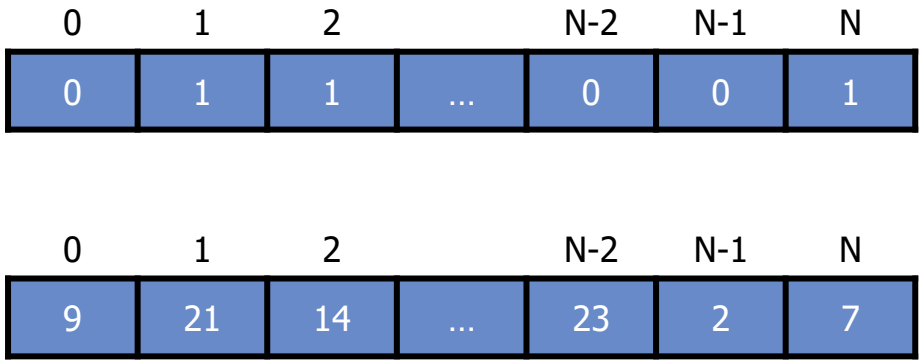
Consideraciones sobre el diseño

- Representación de la solución candidata
- Generación de la población inicial de soluciones
- Diseño de la función fitness que evalúa a cada individuo
- Diseño de operadores genéticos que generan nuevos individuos
- Configuración de los parámetros del algoritmo genético
    - Tamaño de la población
    - Número de generaciones
    - Probabilidad de aplicar operadores genéticos

3. Algoritmos genéticos

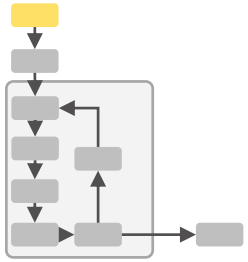
Codificación

- Representación de la solución candidata
- Es una elección muy importante que depende del problema
  - Una representación inapropiada afectará a la calidad y/o rendimiento de la optimización
- Los individuos se representan como cadenas
  - Binarias (0/1)
  - Números enteros
  - Números reales
- Es recomendable aplanar las matrices (flatten)

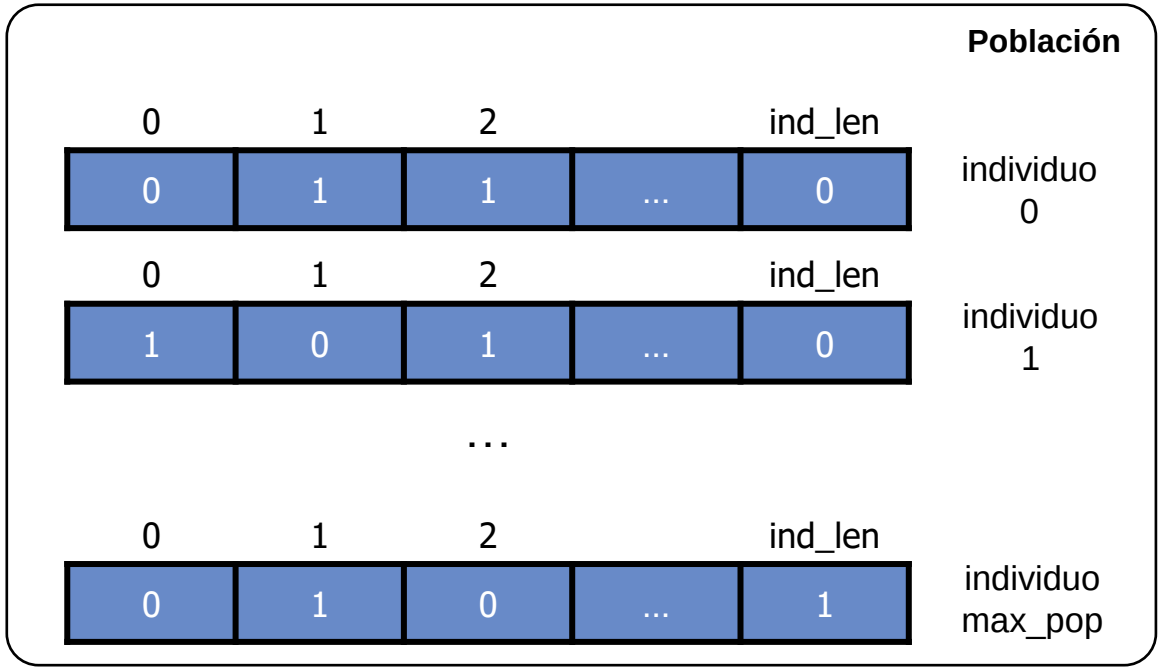


3. Algoritmos genéticos

Generación de la población inicial



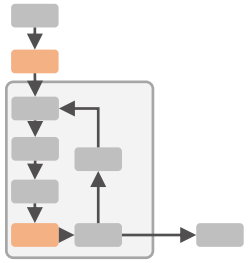
- Los individuos utilizados en la primera generación del algoritmo genético se crean de acuerdo con la representación del individuo elegida.
- Cada individuo es una solución candidata al problema a resolver.



jupyter Algoritmo Genético.ipynb

3. Algoritmos genéticos

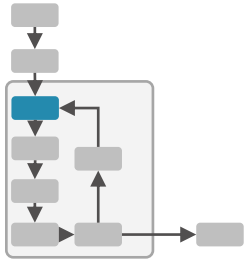
Cálculo de la función de fitness



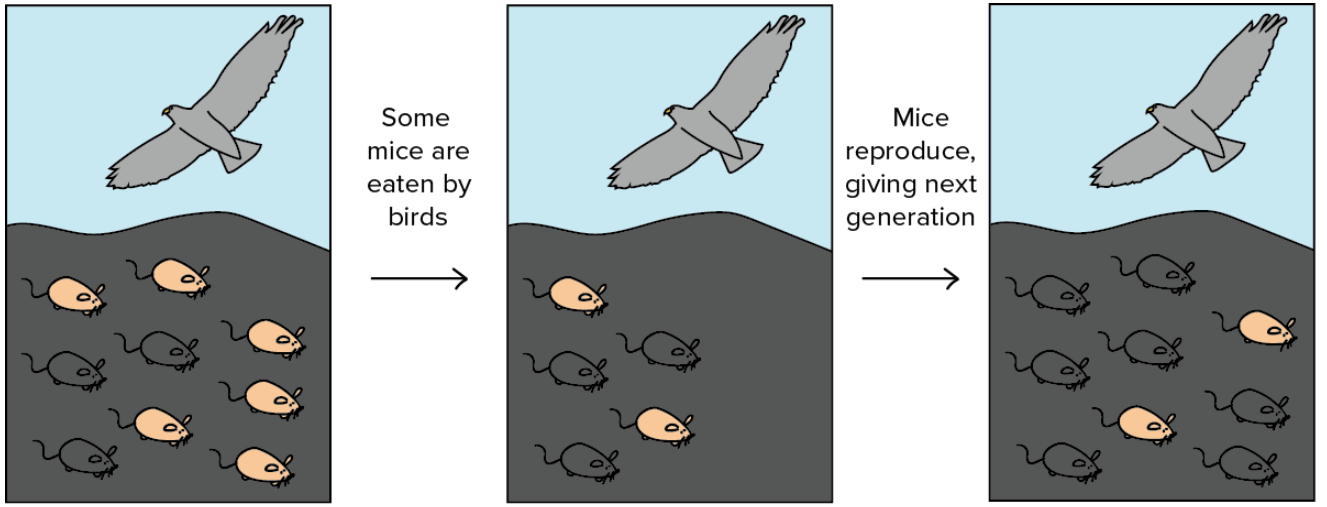
- Evalúa qué tan cerca está una solución dada de la solución óptima
- Requisitos
  - Debe estar claramente definido
  - Debe implementarse de manera eficiente
    - Si es computacionalmente costoso, la eficiencia general del algoritmo genético se reducirá
  - Debe medir cuantitativamente qué tan adecuada es una solución para resolver el problema
  - Debería generar resultados intuitivos
    - Los mejores / peores candidatos deben tener los mejores / peores valores de puntuación

3. Algoritmos genéticos

Selección



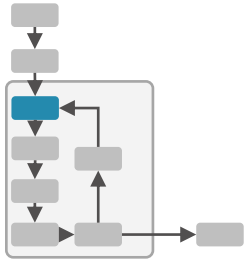
- Tiene como objetivo explotar las mejores características de las buenas soluciones candidatas para mejorar estas soluciones a lo largo de generaciones.
  - Varios métodos de selección:
    - Roulette wheel selection
    - Rank selection
    - Tournament selection
    - ...

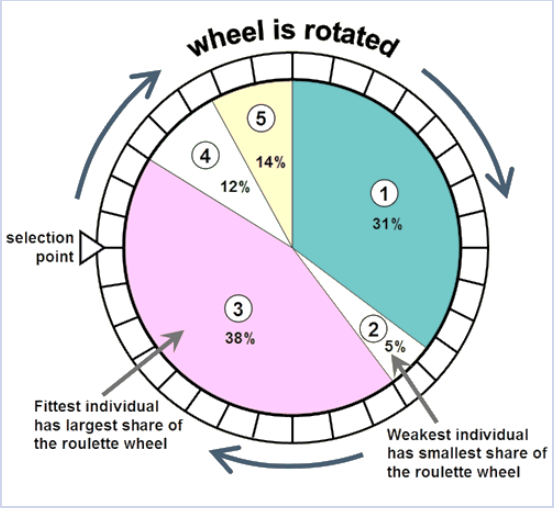
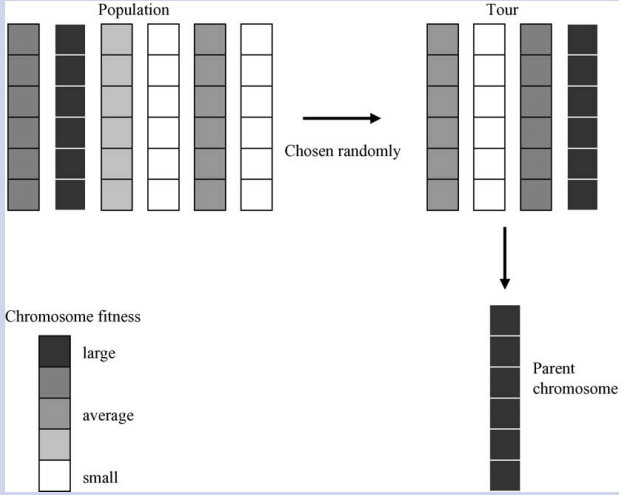


https://www.khanacademy.org/science/ap-biology/natural-selection/natural-selection-ap/a/darwin-evolution-natural-selection

3. Algoritmos genéticos

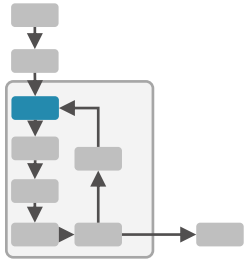
Selección



	Roulette Wheel	Rank	Tournament
Probabilidad selección	Un individuo es seleccionado proporcionalmente a su fitness $p(i) = \frac{f(i)}{\sum_{j=1}^{n} f(j)}$  http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php	Como roulette Wheel pero con fitness relativo en vez de absoluto (utiliza posición en ranking en vez de valor de fitness) $p(i) = \frac{2\,rank(i)}{n \cdot (n+1)}$	1. Selección aleatoria de k individuos 2. De los k individuos, el individuo más apto es seleccionado como padre 3. Todo el proceso se repite n veces para toda la población  Shopova, Elisaveta & Vaklieva-Bancheva, Natasha. (2013). Shopova&Vaklieva BASIC.

3. Algoritmos genéticos

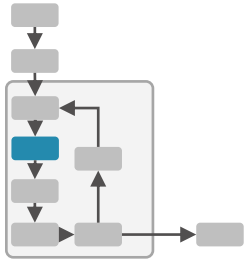
Selección



	Roulette Wheel	Rank	Tournament
Probabilidad selección	Un individuo es seleccionado proporcionalmente a su fitness $$p(i) = \frac{f(i)}{\sum_{j=1}^{n} f(j)}$$	Como roulette Wheel pero con fitness relativo en vez de absoluto (utiliza posición en ranking en vez de valor de fitness) $$p(i) = \frac{2\,rank(i)}{n \cdot (n+1)}$$	1. Selección aleatoria de k individuos 2. De los k individuos, el individuo más apto es seleccionado como padre 3. Todo el proceso se repite n veces para toda la población
Ventajas	Si hay un miembro de la población particularmente apto, esperaríamos que tuviera más éxito en producir descendencia que un rival más débil	Funciona con valores de *fitness* negativos	• Funciona con valores de fitness negativos • Más eficiente de calcular (sin ordenación) • Es fácil de implementar • Funciona en arquitecturas paralelas
Desventajas	• Pueda darse una convergencia prematura en un óptimo local cuando existe un individuo dominante, ya que siempre gana la competición y es seleccionado como padre. • No funciona con valores de fitness negativos	• Costoso de calcular porque hay un proceso de ordenación • Puede conducir a una convergencia más lenta, porque los mejores cromosomas no se diferencian tanto de los demás	La selección de *k* afecta los resultados

3. Algoritmos genéticos

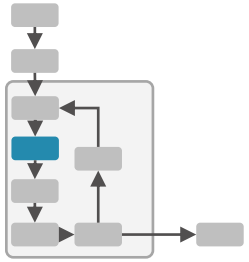
Crossover



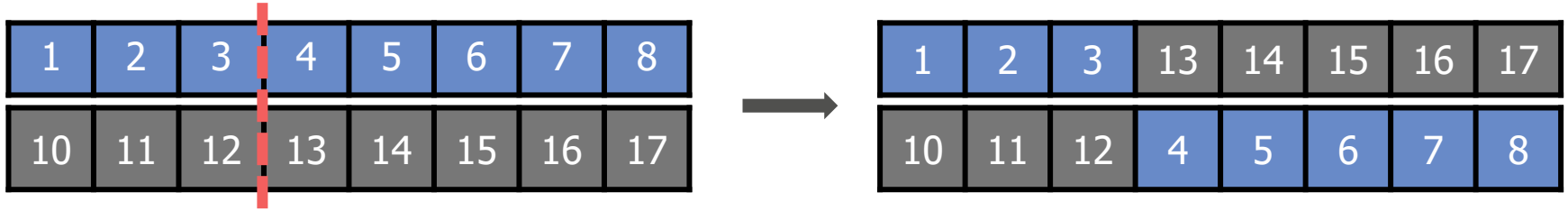
- El crossover (cruce) es un operador genético destinado a generar nuevos individuos mediante la combinación de información genética de individuos anteriores (padres).
- Por lo general, se seleccionan dos individuos como padres para generar dos nuevos individuos (descendencia)
- Se aplica con una probabilidad P_c (e.g., 0.9)
- Algunos padres pueden no cruzarse y pasar íntegramente a la siguiente generación
- Varios métodos de cruce:
    - Crossover de un solo punto
    - Crossover multipunto
    - Cruce uniforme
    - ...

3. Algoritmos genéticos

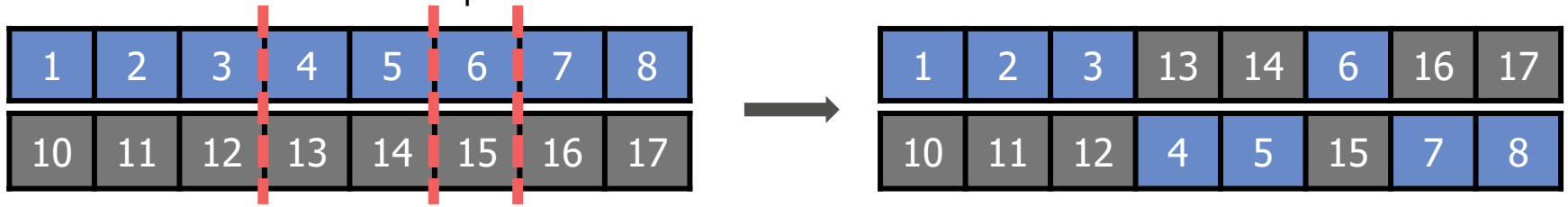
Crossover



- **Single-point crossover**
  - Se selecciona al azar un punto de corte en el cromosoma de los padres
  - Todos los datos más allá de ese punto se intercambian entre los dos padres



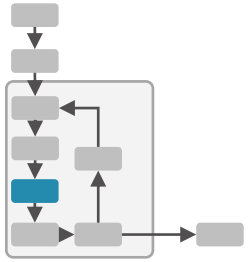
- **Multiple-point crossover** (*k*-point crossover)
  - Se seleccionan al azar *k* puntos de corte en el cromosoma de los padres
  - Los segmentos alternos se intercambian para obtener nuevos individuos
  - (e.g. *k*=3)



- **Uniform crossover**
  - Cada gen se maneja de forma independiente
  - Cada gen del hijo tiene una probabilidad de provenir del primer padre o del segundo padre

3. Algoritmos genéticos

Mutación

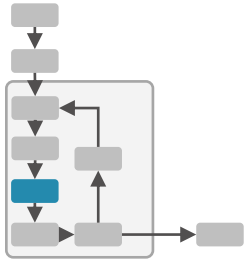


- Es un operador genético destinado a mantener la diversidad genética de una generación a la siguiente.
  - Altera el estado inicial de uno o más genes
  - Se aplica con probabilidad P_μ (e.g., $\frac{1}{\text{tamaño del cromosoma}}$)
  - No es conveniente una P_μ muy grande ya que modificaría mucho el individuo alejándolo de la solución óptima cuando esté cerca

- Varios métodos de mutación:
  - Bit flip mutation
  - Random resetting mutation
  - Swap mutation
  - …

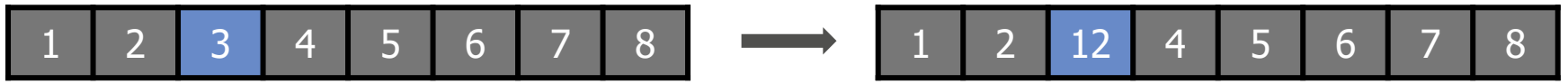
3. Algoritmos genéticos

Mutación



- **Bit flip mutation**
  - Los genes seleccionados se invierten
  - Solo para codificación binaria

- **Random resetting mutation**
  - Se asigna un valor aleatorio del conjunto de valores permitidos a los genes seleccionados
  - Distribución uniforme

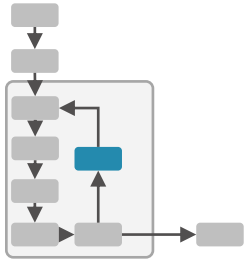


  - Distribución gaussiana
  - Distribución gaussiana centrada en el valor original

- **Swap mutation**
  - Los alelos de dos genes seleccionados se intercambian

3. Algoritmos genéticos

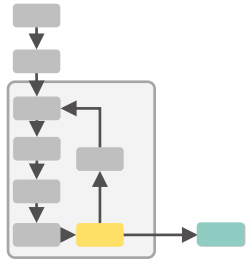
Remplazo de la población



- El reemplazo de la población determina qué individuos pasan a la siguiente generación.
- Los individuos con mejor fitness deben mantenerse en la población
  - También se debe mantener la diversidad
- Elitismo
  - Los buenos candidatos se pueden perder cuando el cruce o la mutación dan como resultado una descendencia con peor fitness que el de los padres
  - Implica copiar a los candidatos con mejor fitness, sin cambios, a la próxima generación.
- Estrategias
  - La descendencia reemplaza a los peores individuos de la generación anterior
  - Se mantiene el mejor individuo de la generación anterior (esto garantiza que al final del proceso se mantiene la mejor solución encontrada teniendo en cuenta todas las generaciones).
  - La descendencia reemplaza a los individuos más antiguos (los que llevan más generaciones)

3. Algoritmos genéticos

Consecución del criterio de finalización

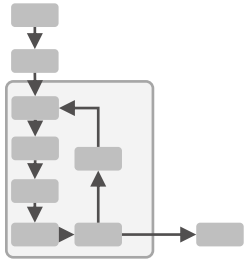


- Los algoritmos genéticos progresan muy rápido, pero tienden a saturarse en las generaciones posteriores, donde las mejoras son muy pequeñas cuando se acercan a la solución óptima
- Estrategias para terminar el ciclo genético:
  - Cuando se alcanza un número de generaciones
  - Cuando no ha habido mejora en la población durante varias generaciones
  - Cuando la función fitness alcanza un cierto valor
  - Después de un tiempo de ejecución específico
  - …

- Cuando se alcanzan las condiciones de finalización, se selecciona como solución final la mejor solución de acuerdo con la función de fitness.

3. Algoritmos genéticos

Parametrización

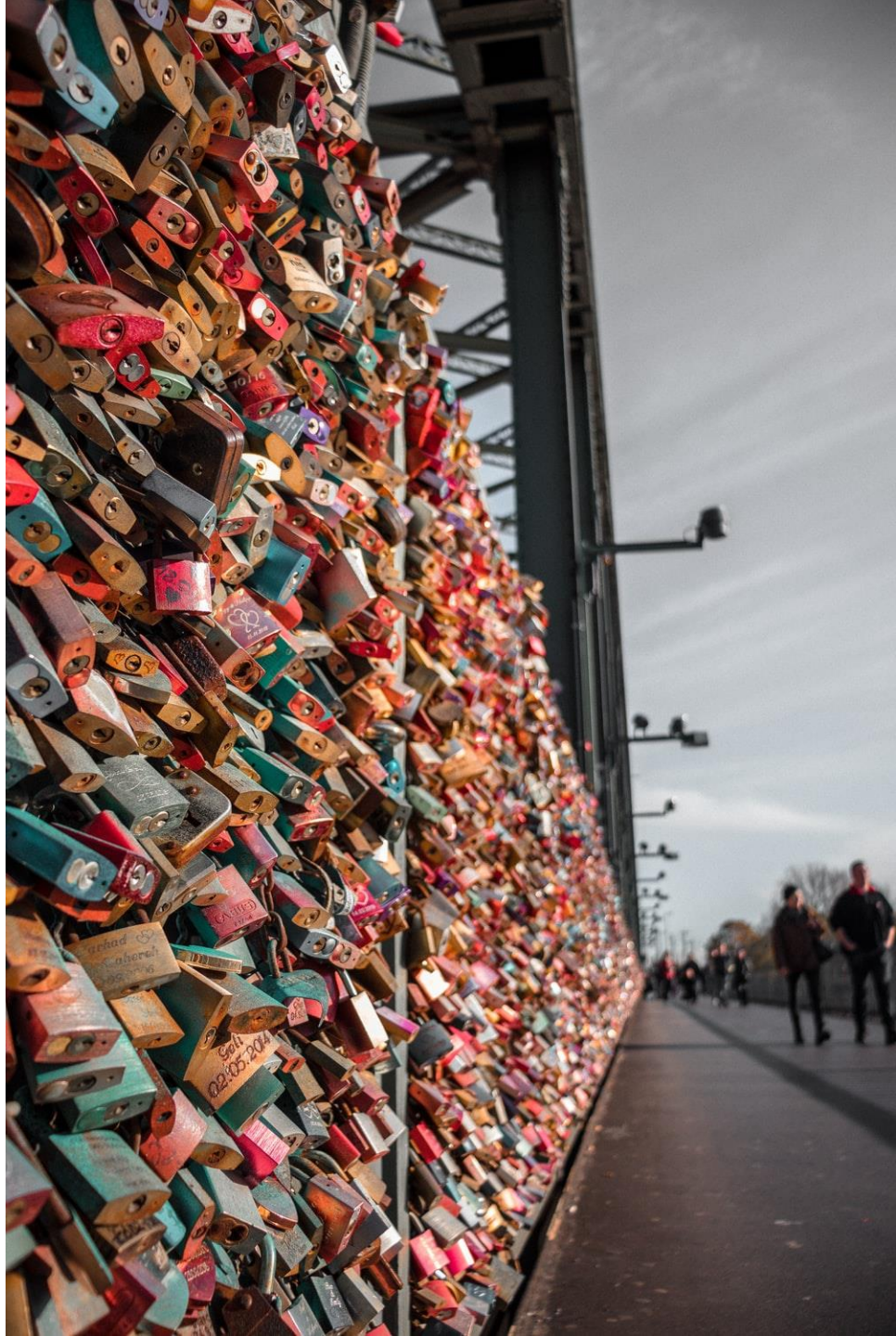


- Tamaño de la población: una población más grande permite una mayor exploración del espacio del problema en cada generación y aumenta la posibilidad de desarrollar una solución
    - Penaliza el tiempo de cómputo

- Número máximo de generaciones: cuanto mayor sea el número máximo de generaciones, mayor será la posibilidad de desarrollar una solución
    - Penaliza el tiempo de cómputo

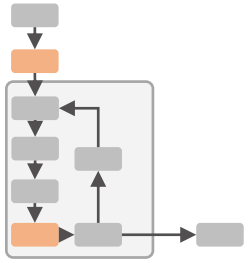
- Probabilidad de cruce: 0.9 (Koza, 1992)
- Probabilidad de mutación: $\frac{1}{\text{tamaño del cromosoma}}$

- Varias investigaciones, por ejemplo:
    - A.E. Eiben, S.K. Smit, Parameter tuning for configuring and analyzing evolutionary algorithms, 2011
    - B. Yuan and M. Gallagher, A hybrid approach to parameter tuning in genetic algorithms, 2005



3. Algoritmos genéticos

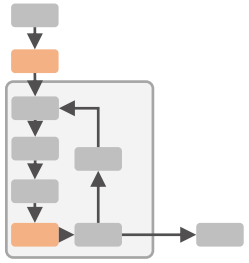
Factibilidad



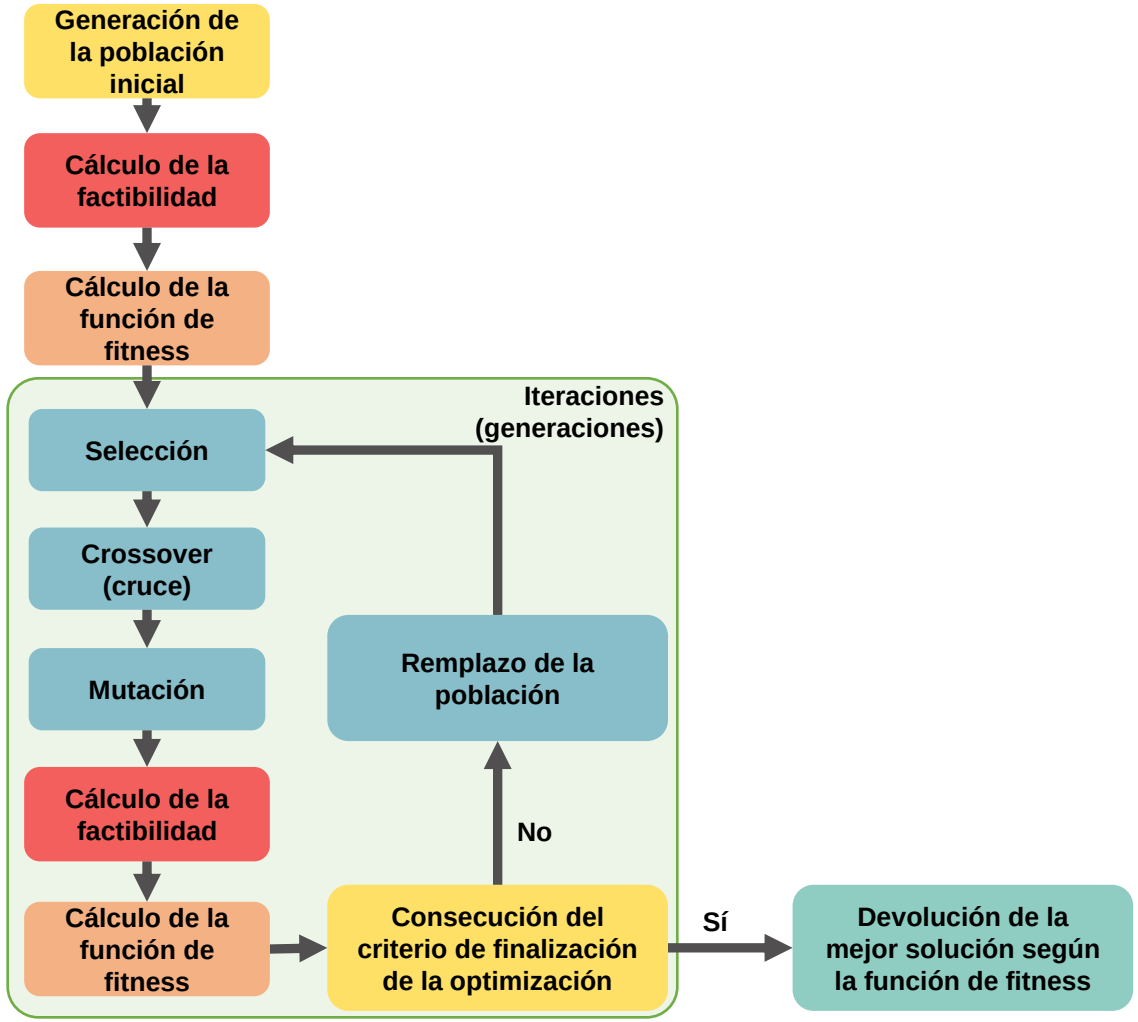
- Durante el proceso genético, se podrían crear individuos inviables
- Población inicial
- Los operadores de cruce y mutación pueden generar individuos inviables a partir de los factibles
- Esto es especialmente relevante en problemas de optimización con restricciones
- Es necesario aplicar algunas medidas especiales
    - Penalizar a individuos inviables (por ejemplo, función de fitness, función de penalización)
    - Reparar los individuos inviables
    - Utilizar operadores genéticos especiales para evitar la creación de individuos inviables
    - Eliminar a los individuos inviables
    - …

3. Algoritmos genéticos

Factibilidad

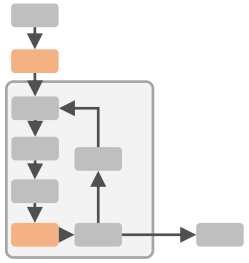


- Penalizar la función de fitness con una función de penalización
  - Para cada individuo
    - Calcular si el individuo es factible
      - Si no es factible, el fitness del individuo es la distancia para ser factible (con penalización)
      - Si es factible, el fitness se calcula en la etapa de cálculo de la función de fitness



3. Algoritmos genéticos

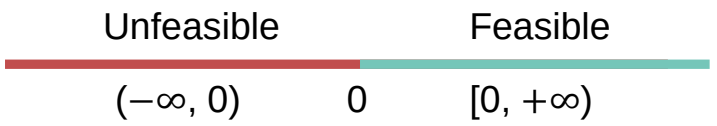
Ejemplo de factibilidad



Encuentra dos números positivos tales que la suma de uno y el cuadrado del otro sea 200 y cuyo producto sea un máximo.

- Variables de decisión
  - Variables independientes
    $x, y \in [0, +\infty)$
- Restricciones
    $x + y^2 = 200$
- Función objetivo
  - Maximizar
    $O(x, y) = x \cdot y$

- Variables de decisión
  - Variables independientes
    $x, y \in [0, +\infty)$
- Restricciones
    $$C(x, y) = \begin{cases} 0, & x + y^2 = 200 \\ |x + y^2 - 200|, & otherwise \end{cases}$$
- Función objetivo
  - Maximizar
    $$O(x, y) = \begin{cases} -C(x, y), & C(x, y) > 0 \\ x \cdot y, & otherwise \end{cases}$$



3. Algoritmos genéticos

Ejercicio práctico básico

Problema de la valla: AlgoritmoGenético_Valla_Alumno.ipynb

3. Algoritmos genéticos

Ejercicio – Planificación de un hospital

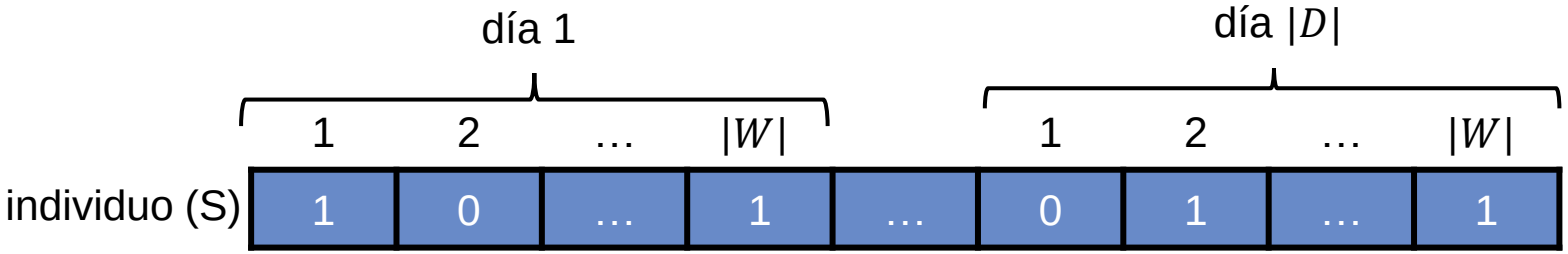
- Planificación de los horarios de un hospital
- Horario diario de los trabajadores (W) de un hospital durante un número específico de días (D)
  - $|W| > |D|$
- Cada trabajador trabaja todo el día
- Optimizar la cobertura en función del forecast (F) de carga de trabajo de cada día (medido en número de trabajadores)
- Debe evitarse la ineficiencia (más trabajadores de los necesarios)
- Los trabajadores no pueden trabajar más de dos días seguidos

3. Algoritmos genéticos

Ejercicio – Planificación de un hospital

- Notación
  - Trabajadores (W), días (D), forecast de trabajadores necesarios por día (F)
- Variables de decisión
  - Variables independientes:
    - Planificación diaria S de tamaño $|W| \times |D|$ con valores binarios (1 trabaja, 0 no trabaja)
- Función objetivo (minimizar)

$$O(S) = \sum_{d \in D} \left(F_d - \sum_{w \in W} S_{w,d} \right)$$



- Restricciones
  - Evitar la ineficiencia

$$\forall d \in D: \; F_d - \sum_{w \in W} S_{w,d} \geq 0$$

No trabajar más de dos días seguidos

$$\forall d \in D: \quad \forall w \in W: \quad S_{w,d} + S_{w,d+1} + S_{w,d+2} \leq 2$$

3. Algoritmos genéticos

Ejercicio – Planificación de un hospital

- Notación
  - Trabajadores (W), días (D), forecast de trabajadores necesarios por día (F)
- Variables de decisión
  - Variables independientes:
    - Planificación diaria S de tamaño $|W| \times |D|$ con valores binarios (1 trabaja, 0 no trabaja)
- Restricciones
  - Evitar la ineficiencia

$$C_1(S) = \begin{cases} 0, & \forall d \in D:\ F_d - \sum_{w \in W} S_{w,d} \geq 0 \\ \sum_{d \in D} \left(\sum_{w \in W} S_{w,d} \right) - F_d\,, & otherwise \end{cases}$$

No trabajar más de dos días seguidos

$$C_2(S) = \begin{cases} 0, & \forall d \in D:\ \ \forall w \in W:\ \ S_{w,d} + S_{w,d+1} + S_{w,d+2} \leq 2 \\ \sum_{d \in D} \left(\sum_{w \in W} S_{w,d} + S_{w,d+1} + S_{w,d+2} \right), & otherwise \end{cases}$$

- Función objetivo (minimizar)

$$O(S) = \begin{cases} (C_1(S) + C_2(S)), & C_1(S) + C_2(S) > 0 \\ \sum_{d \in D} \left(F_d - \sum_{w \in W} S_{w,d} \right), & otherwise \end{cases}$$

→

Función objetivo (minimizar, [0..1] factible, > 1 inviable)

$$O(S) = \begin{cases} 1 + (C_1(S) + C_2(S)), & C_1(S) + C_2(S) > 0 \\ \frac{\sum_{d \in D} \left(F_d - \sum_{w \in W} S_{w,d} \right)}{\sum_{d \in D} F_d}, & otherwise \end{cases}$$

3. Algoritmos genéticos

Ejercicio práctico

Planificación de un hospital: AlgoritmoGenético_SchedulingHospital_Alumno.ipynb

3. Algoritmos genéticos

Librerías

- Python – DEAP:
  - https://deap.readthedocs.io/en/master/
  - https://pypi.org/project/deap/

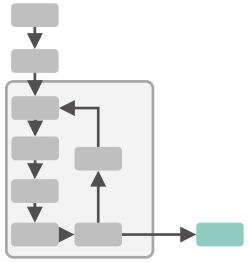
AlgoritmoGenético_DEAP.ipynb

- R – GA:
  - https://cran.r-project.org/web/packages/GA/GA.pdf
  - http://luca-scr.github.io/GA/articles/GA.html

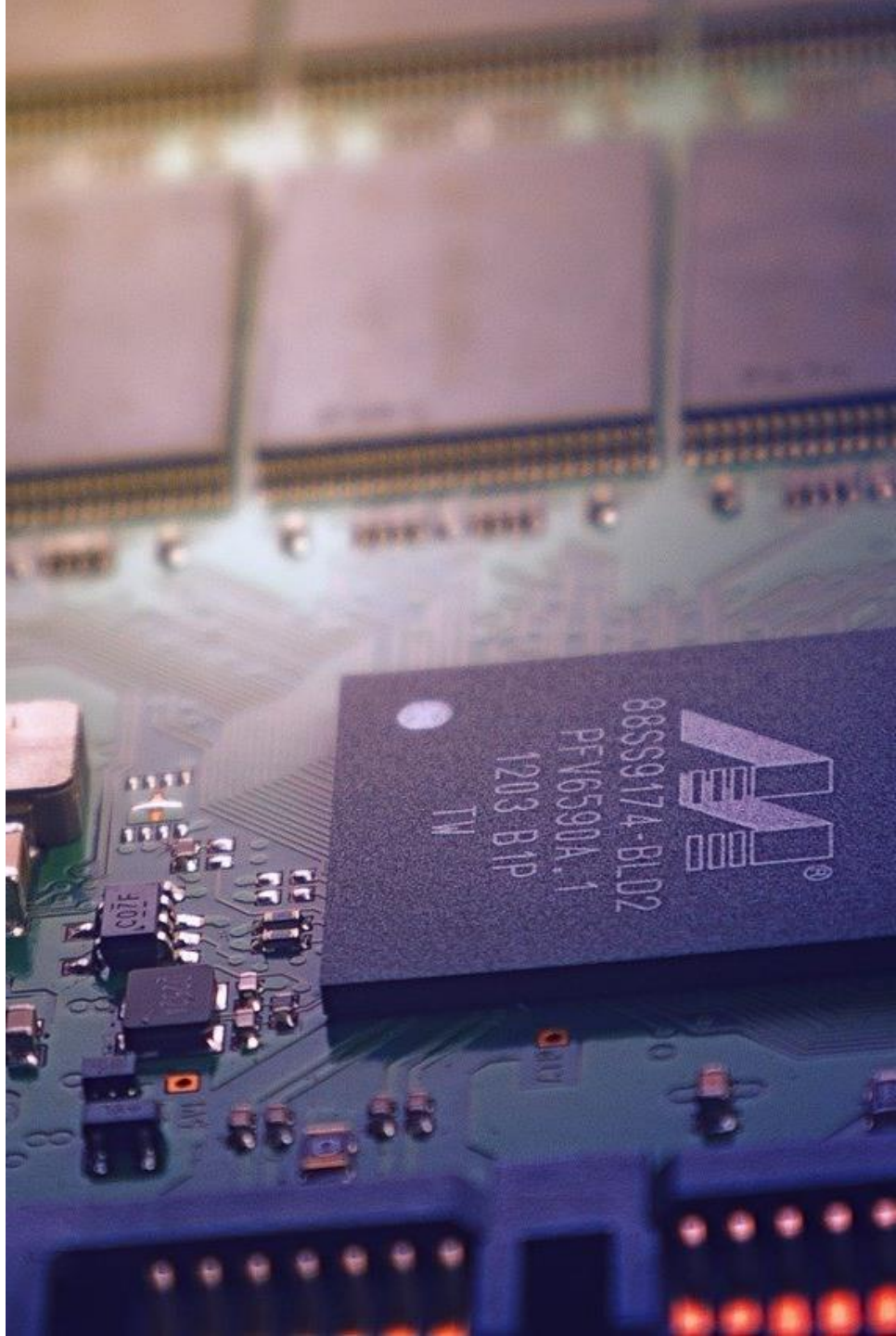
- Python – Interna, basada en DEAP y que puede llamarse desde R:
  - Files · master · ANALYTICS / Assets / Optimization Asset · GitLab (everis.com)
  - Se ha desarrollado y usado en SAS Bidding y en Acciona

3. Algoritmos genéticos

Escalabilidad

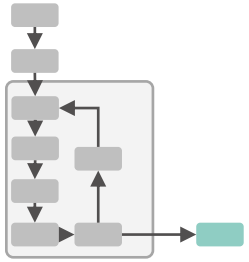


- Cuando los GA están mal diseñados, pueden tener un **alto uso de memoria y tiempo** de cálculo.
- Además, estos efectos también se pueden experimentar en algoritmos genéticos bien diseñados cuando se aplican a grandes conjuntos de datos.
- Algunas **estrategias utilizadas** para superar estas limitaciones:
  - **Simplificar al individuo**. E.g.: Es más eficaz asignar la hora de inicio y la duración a una tarea que asignar una tarea a cada minuto del día.
  - Si es posible, definición (e implementación) de la **función de fitness óptima.**
  - **Fitness surrogate**. Se basa en el uso de una función de fitness más rápida de calcular aunque menos precisa para evaluar a algunos individuos o algunas generaciones.
  - Utilizar un **subconjunto de los datos** disponibles. E.g.: estratificar los datos y cambiar el subconjunto en cada generación.
  - **Paralelizar** algunos pasos
  - ...

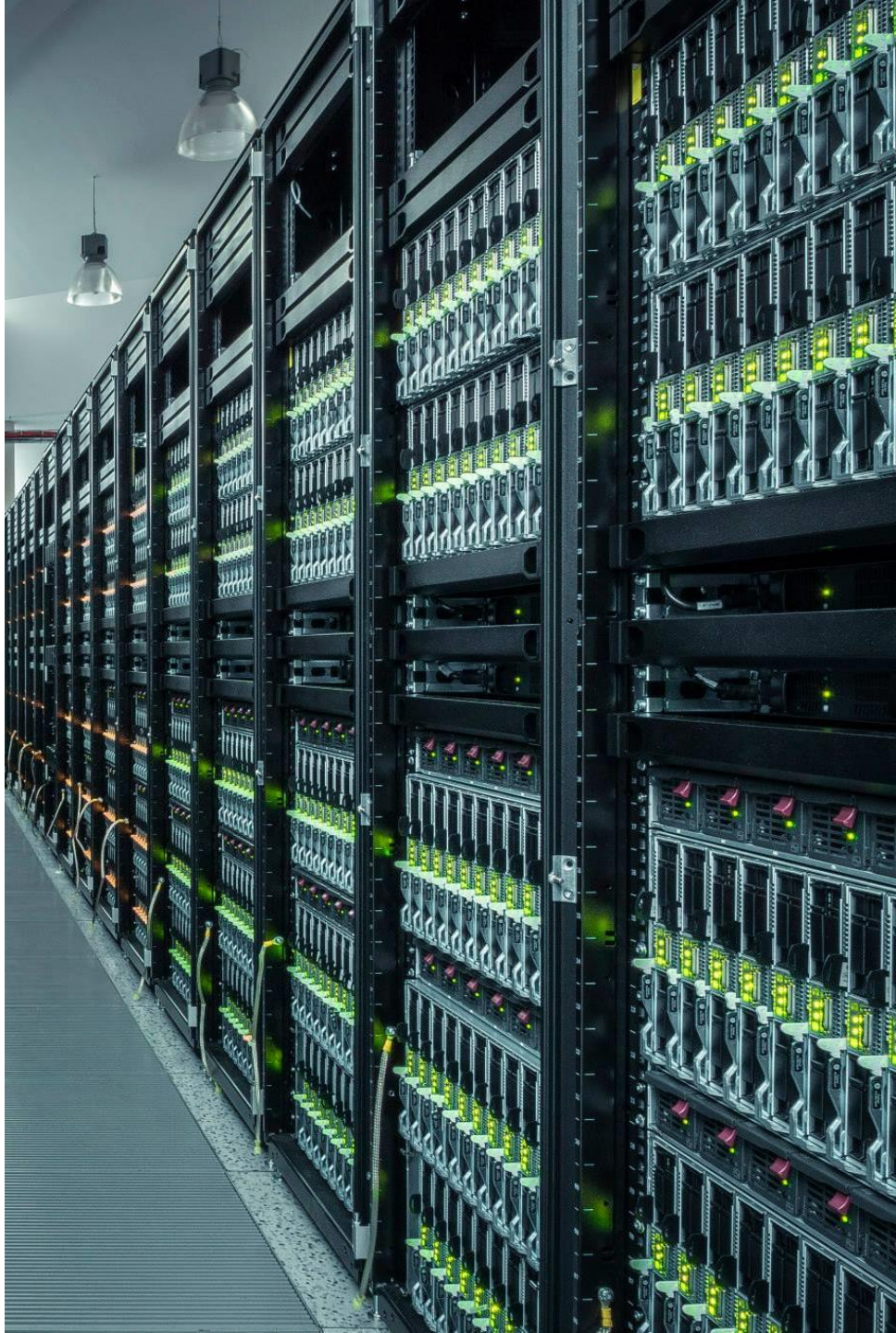


3. Algoritmos genéticos

Paralelismo



- Uno de los principales cuellos de botella en el ciclo de los algoritmos genéticos es la evaluación de los individuos (fitness y restricciones)
- Es posible distribuir la carga computacional entre varios procesadores
  - Cada procesador evalúa un subconjunto diferente de la población
  - Cada procesador evalúa a cada individuo con un subconjunto de los datos
  - ...
- Implica un costo de comunicación adicional que podría disminuir el rendimiento logrado con la distribución computacional
- La implementación puede ser trivial con algunas bibliotecas de paralelización



4. Proyecto final

4. Proyecto final

Pyomo y algoritmos genéticos

- El Hospital del Mar tiene un problema con el laboratorio de análisis de muestras. El laboratorio tiene disponibles **tres máquinas** que pueden analizar diferentes muestras de fluidos. Últimamente la demanda de análisis de sangre se ha incrementado de tal forma que el director del laboratorio tiene problemas para tener los resultados a tiempo y hacer frente al mismo tiempo a las analíticas de los restos de fluidos. El laboratorio trabaja con **5 tipos diferentes de muestras** de fluidos. Cada máquina puede ser usada para analizar cualquier tipo de muestra, **pero el tiempo (min) que tarda cada una depende del tipo de muestra**, según se indica en la siguiente tabla:

Tiempo procesado (min/mL)	Máquina A	Máquina B	Máquina C	Volumen (mL) 1mL=1Muestra
Muestra 1	3	5	2	80
Muestra 2	4	3	5	75
Muestra 3	4	5	3	80
Muestra 4	5	4	3	12
Muestra 5	3	5	4	60

http://www-eio.upc.es/~heredia/

- **Cada máquina se puede usar un máximo de 8h al día**. Las muestras recogidas un día dado llegan al laboratorio y esperan durante la noche a ser procesadas al día siguiente. Al comienzo de cada día, el director del laboratorio debe determinar cómo repartir las muestras entre las diferentes máquinas. La cantidad de muestras a analizar esta mañana se indica en la última columna de la tabla anterior:

1) Formulad el problema que permite encontrar cómo distribuir de forma óptima las muestras entre las máquinas.

2) Implementad la solución con Pyomo.

3) ¿Qué pasaría si el problema actual pasase a minimizar el tiempo del procesado y no la maximización de número de muestras?

4) Considerad que se quiere obtener la distribución óptima de muestras teniendo en cuenta las siguientes limitaciones:

   a) No se quiere que ninguna muestra ocupe más del 50% del tiempo total de funcionamiento de una máquina.

   b) No se quiere que ninguna máquina realice más del 40% de volumen total de las pruebas.

5) El tiempo proporcionado por el fabricante de las máquinas es una aproximación, ya que existen cambios en el rendimiento de las máquinas cuando trabajan varias horas con un mismo tipo de muestra. Para hacer un cálculo más realista, el fabricante nos ha proporcionado un simulador que calcula la duración real de cada análisis, pero desconocemos cómo funciona.

4. Proyecto final

Pyomo y algoritmos genéticos

Pyomo_HospitalDelMar_Alumno.ipynb

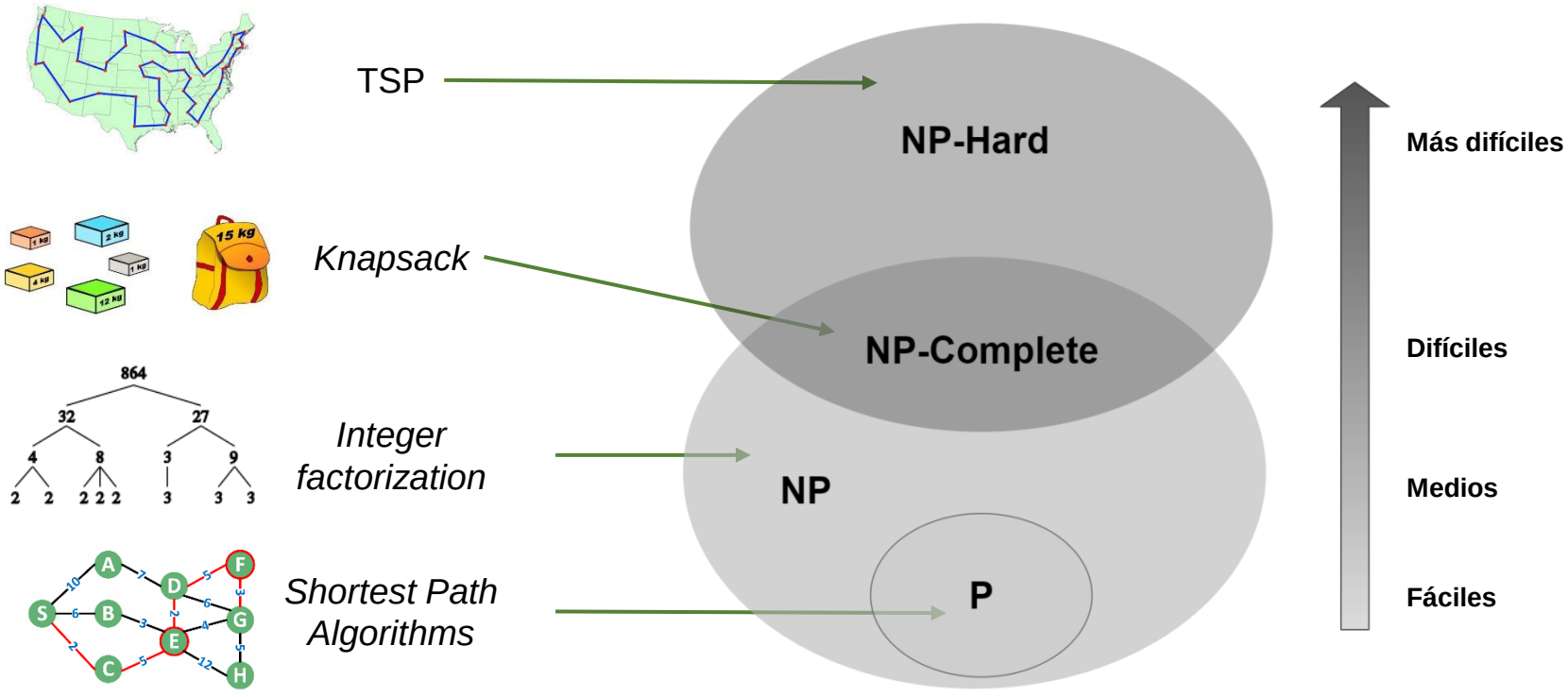
AlgoritmoGenético_HospitalDelMar_Alumno.ipynb

Anexo

1. Fundamentos de la optimización matemática

Clases de complejidad

- En teoría de la complejidad computacional, NP es el acrónimo de *nondeterministic polynomial time* (tiempo polinomial no determinista). Es el conjunto de problemas que pueden ser resueltos en tiempo polinómico por una máquina de Turing no determinista.
- Intuitivamente (e informalmente), un problema está en NP si es fácil verificar sus soluciones.
- Por otro lado, un problema es NP-*hard* si es difícil de resolver o encontrar una solución.
- Ahora, un problema es NP-*complete* si es tanto NP como NP-*hard*. Por lo tanto, tiene dos propiedades intuitivas clave para ser NP-*complete*: fácil de verificar, pero difícil de encontrar soluciones.



https://www.baeldung.com/cs/p-np-np-complete-np-hard

2. Programación matemática

Herramienta de modelado - AMPL

- Antes de que se pueda invocar cualquier rutina de optimización, se debe realizar un esfuerzo considerable para formular el modelo y generar las estructuras de datos requeridas.

- Una herramienta de modelado es AMPL (A Mathematical Programming Language):
- Es un lenguaje diseñado para hacer estos pasos más fáciles y menos propensos a errores
- Se parece mucho a la notación algebraica simbólica
- Utiliza un traductor que toma como entrada un modelo AMPL y sus datos asociados, y produce una salida adecuada para *solvers* de optimización.
- Soporta diferentes solvers.
- Para usarlo desde Python es necesario usar la API de AMPL, que es una interfaz que permite a los desarrolladores acceder a las funciones del intérprete de AMPL.
- Versión comercial y libre (académica)

```
var x >= 0;
var y >= 0;

maximize area: x * y;

subject to perimeter: 2*x + 2*y <= 300;
subject to max_x: x <= 60;
subject to max_y: y <= 100;
```

3. Algoritmos genéticos

Aplicaciones del mundo real

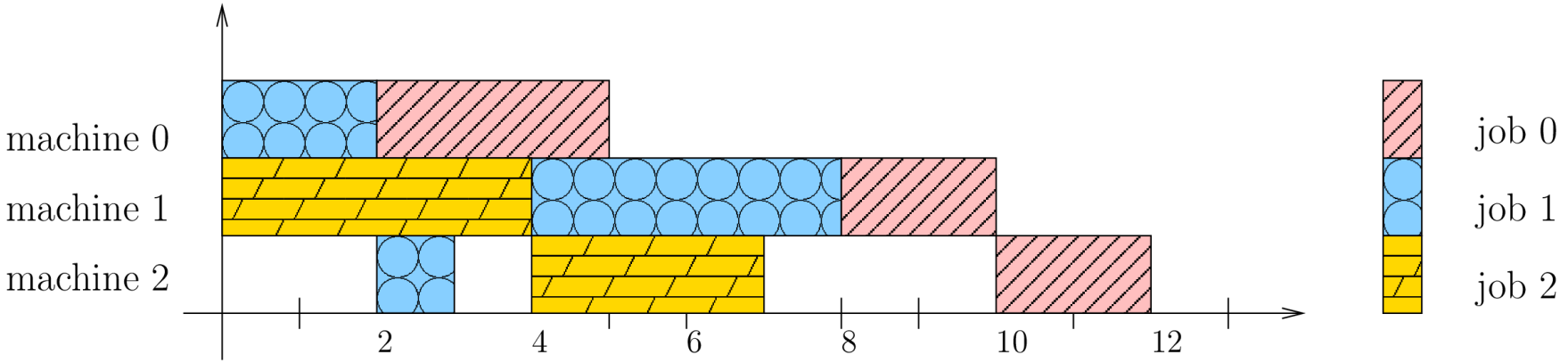
- Referencias:
  - Job shop scheduling
  - Programación de quirófanos
  - Transporte público en Utah
  - Sensores de calidad del agua
  - Préstamos bancarios
  - Jugando a Mario Bros

- List of genetic algorithm applications - Wikipedia

3. Algoritmos genéticos

Aplicaciones en el mundo real

- **Job shop scheduling** basado en optimizar la eficiencia y el consumo energético
- Job shop scheduling es un problema de optimización en el que los trabajos se asignan a los recursos en momentos particulares (los trabajos tienen una duración, dependencias entre ellos, solo se pueden ejecutar en algunos recursos en función de sus características, las máquinas necesitan tiempo de inactividad entre trabajos)
    - Minimizar makespan (tiempo para terminar todos los trabajos)
    - Minimizar el consumo energético (cada máquina y configuración tiene un consumo diferente)



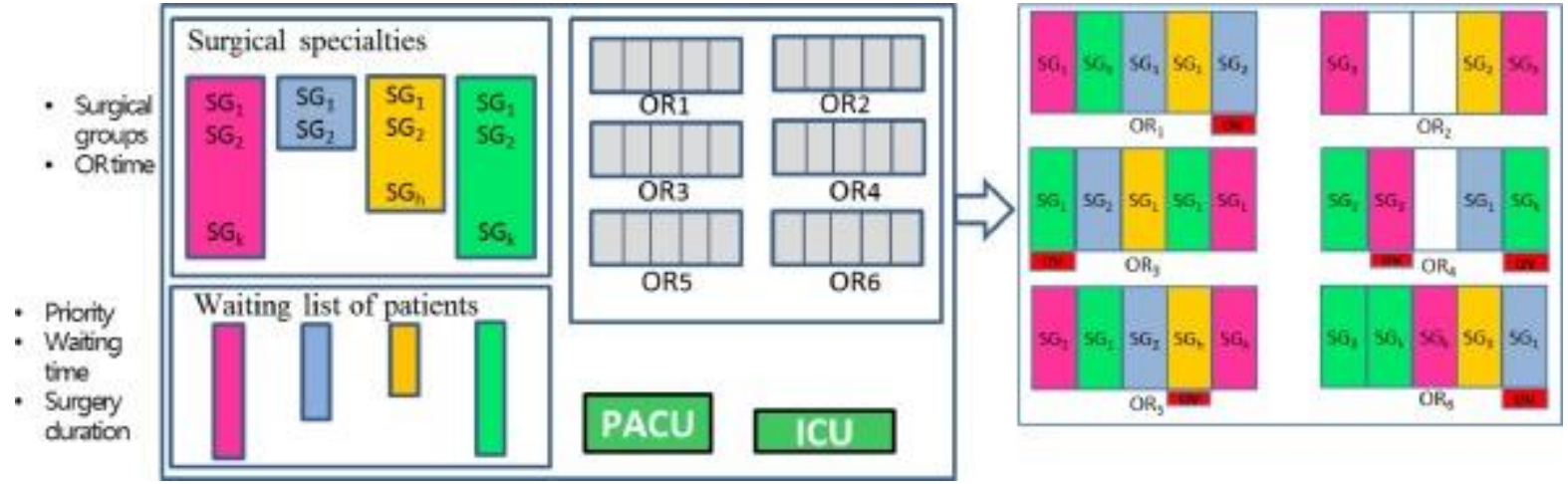
https://developers.google.com/optimization/scheduling/job_shop

Paper G.May, B. Stahl, et al., Multi-objective genetic algorithm for energy-efficient job shop scheduling, 2015

3. Algoritmos genéticos

Aplicaciones en el mundo real

- **Planificación y programación de quirófanos**
  - Algoritmo genético multiobjetivo que determina:
    - El tiempo de quirófano asignado a cada especialidad quirúrgica
    - El tiempo de quirófano asignado a cada equipo quirúrgico
    - La planificación de la admisión a la cirugía (lista de espera)
    - La programación de la cirugía



Paper R. Guido and D. Conforti, A hybrid genetic approach for solving an integrated multi-objective operating room planning and scheduling problem, 2017

3. Algoritmos genéticos

Aplicaciones en el mundo real

- Planificación del **transporte público de Utah**
  - Optimizar tiempos de viaje para acceder a los bienes y servicios más importantes
  - Sujeto a la demanda de viajes, que depende de la ubicación y de la densidad de población de cada zona.

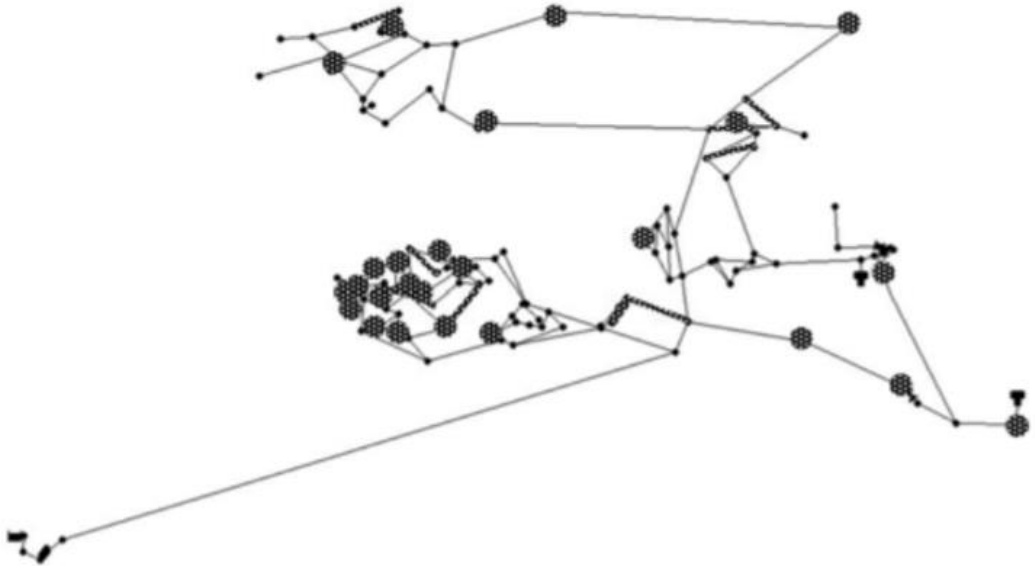
Min Travel Time Plan

Paper R. J. Balling, M. Lowry, M. Saito, Regional Land Use and Transportation Planning with a Genetic Algorithm, 2003

3. Algoritmos genéticos

Aplicaciones en el mundo real

- Ubicación óptima del sensor de calidad del agua para detectar **contaminación accidental de la calidad del agua**
  - Determinación de las ubicaciones óptimas de los sensores en función de la sensibilidad de la dirección del flujo bajo diferentes demandas de agua
    - Minimizar el tiempo de detección de una contaminación
    - Minimizar la cantidad de sensores necesarios



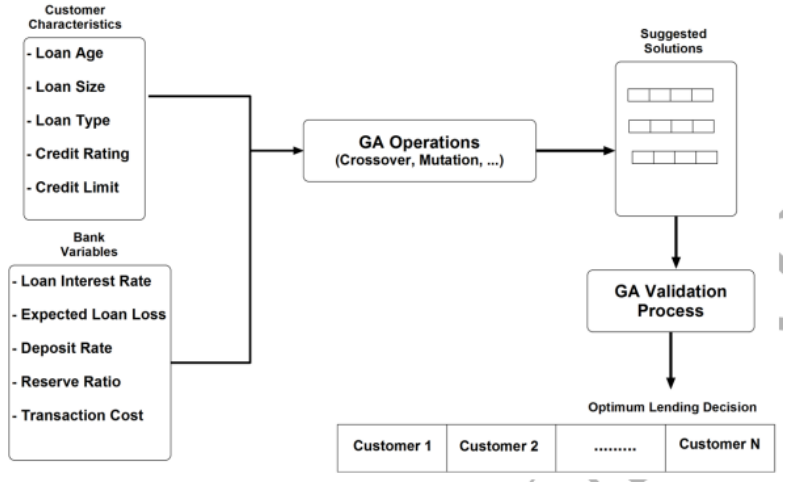
Paper D. Yoo, G. Chung, et al., Applications of Network Analysis and Multi-objective Genetic Algorithm for Selecting Optimal Water Quality Sensor Locations in Water Distribution Networks, 2015

3. Algoritmos genéticos

Aplicaciones en el mundo real

Optimización de **préstamos bancarios**.

- Algoritmo genético que determina si conceder o no un préstamo a cada uno de los clientes en cartera (individuo con codificación binaria).
- Objetivo de maximizar los ingresos del banco sujeto a un límite de dinero prestable.



Paper N. Metawa, M. K. Hassan and M. Elhoseny, Genetic Algorithm Based Model For Optimizing Bank Lending Decisions, 2017

3. Algoritmos genéticos

Aprendiendo niveles de Mario

El individuo representa la secuencia de acciones realizadas por Mario en un nivel específico:

- Para controlar al personaje, se usan 6 controles: un D-Pad con cuatro posiciones (arriba, izquierda, abajo, derecha) botón A (saltar) y botón B (correr y disparar). Con estos controles e introduciendo algunos conocimientos del dominio:
  - a) se pueden pulsar diversos controles a la vez (ej. avanzar y saltar)
  - b) el botón hacia arriba no realiza ninguna acción en el juego
  - c) algunas combinaciones no son factibles, como presionar izquierda y derecha al mismo tiempo

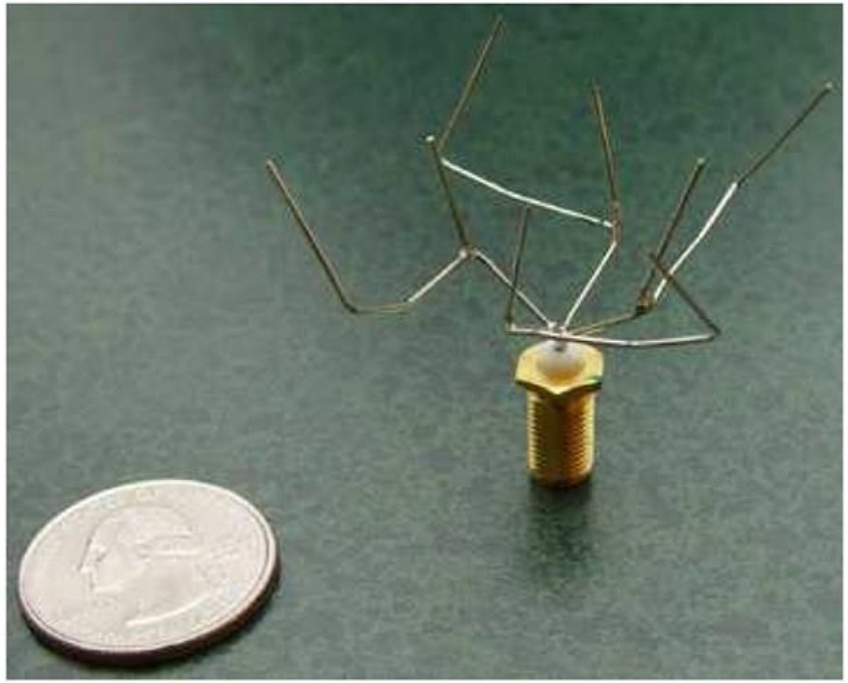
  el número de acciones posibles queda en 22. **Cada gen es un número entero en el rango de 0 a 21**.
- El tiempo máximo para completar un nivel son 200 segundos y que cada segundo se puede discretizar en 15 pulsaciones → **Longitud del individuo=3000**.
- Fitness del individuo
  - Puntuación devuelta por el **simulador Mario AI** tras realizar la secuencia de movimientos del individuo.
- Desventaja:
  - Si los enemigos son aleatorios, el rendimiento se reducirá al ejecutar la mejor solución.

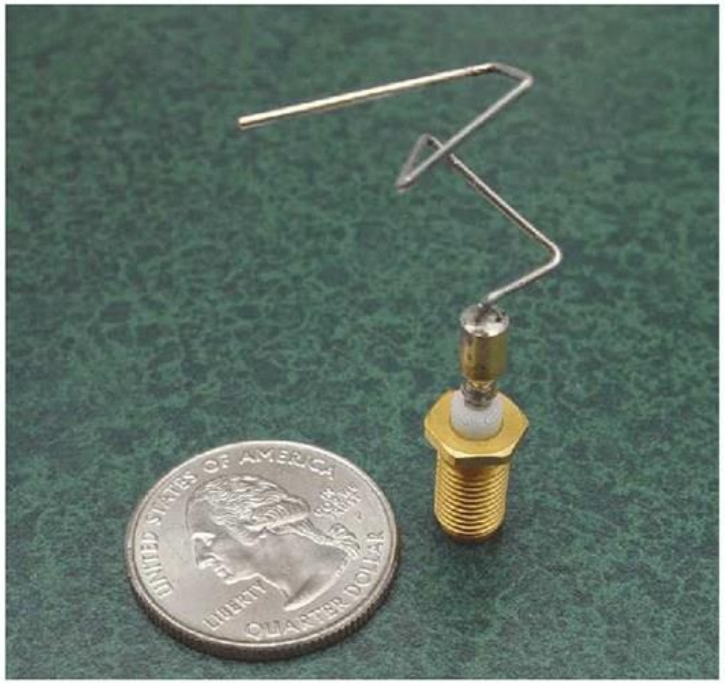
Paper Algoritmo genético para aprender niveles del simulador Mario AI, basado en el juego Super Mario World de Nintendo
Video ilustrativo

3. Algoritmos genéticos

Aplicaciones en el mundo real

- Evolución de la antena de la banda X para la misión ST5 de la NASA
- G. S. Hornby, D. S. Linden and J. D. Lohn, Automated Antenna Design with Evolutionary Algorithms, 2006
    - https://ti.arc.nasa.gov/m/pub-archive/1244h/1244%20%28Hornby%29.pdf
    - Optimizar la combinación de ancho de haz y ancho de banda de impedancia para cubrir las frecuencias de uplink (comunicación Tierra a satélite) y downlink (comunicación satélite a Tierra) en la banda X.

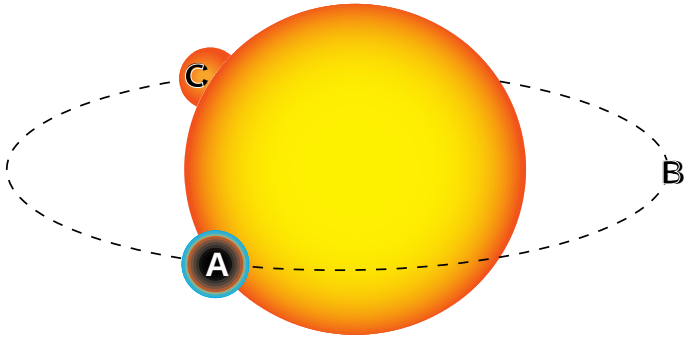


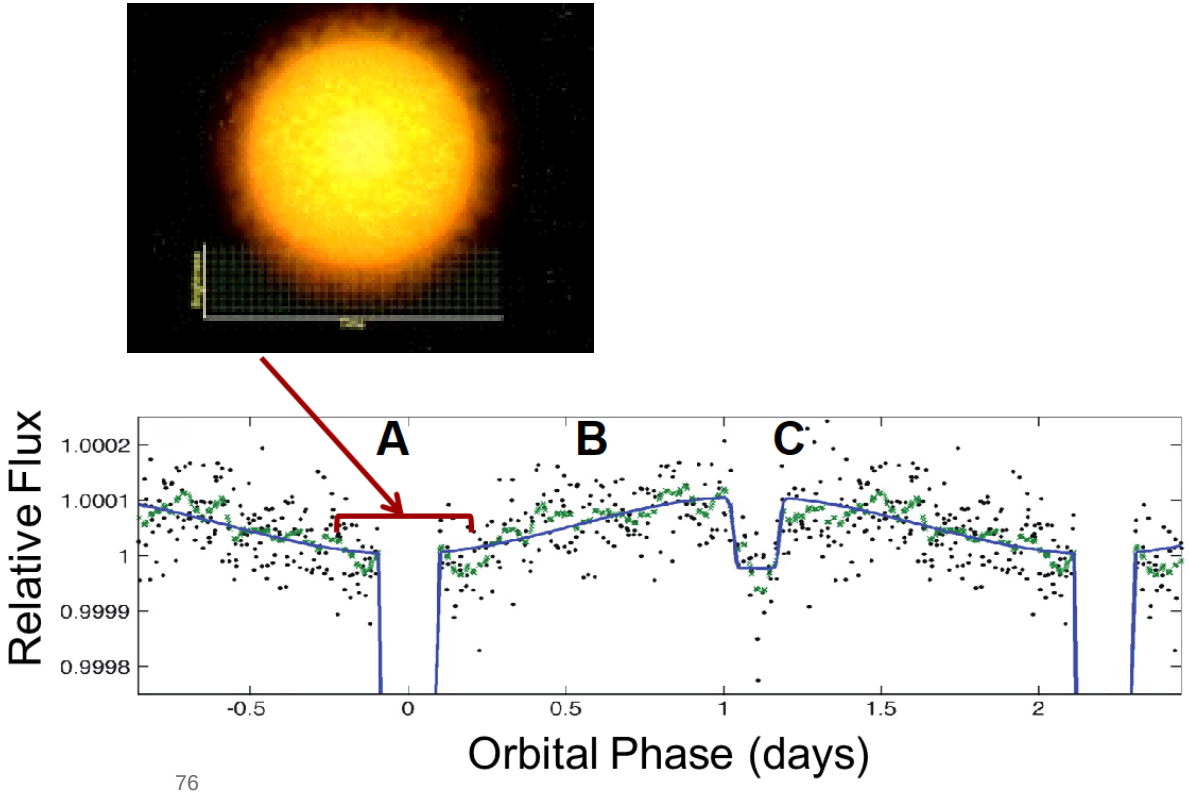


3. Algoritmos genéticos

Aplicaciones en el mundo real

- Planificación y programación de misiones espaciales de observación de exoplanetas
  - A. Garcia-Piquer, J. Colomé, and I. Ribas, Artificial intelligence for the EChO mission planning tool, 2015.
    - https://link.springer.com/article/10.1007%2Fs10686-014-9411-4
  - Maximizar el número de estrellas observadas en situación adecuada y teniendo en cuenta criterios como los objetivos de la misión o la producción científica
    - Las estrellas han de observarse durante los eventos: tránsitos (A) y ocultaciones (C)





3. Algoritmos genéticos

Aplicaciones en el mundo real

- Evolucionar Deep Neural Networks
  - E. David and I. Greental, Genetic Algorithms for Evolving Deep Neural Networks, 2006
    - https://arxiv.org/abs/1711.07655
  - Deep Neural Network
    - Red neuronal artificial con múltiples capas ocultas entre las capas de entrada y salida
  - Propone un método basado en algoritmos genéticos para deep learning que mejora el rendimiento de una deep neural network
    - Cada cromosoma de la población del algoritmo genético es un conjunto de pesos para la red neuronal