Universidad de Oviedo

Intelligent Scheduling and Optimization

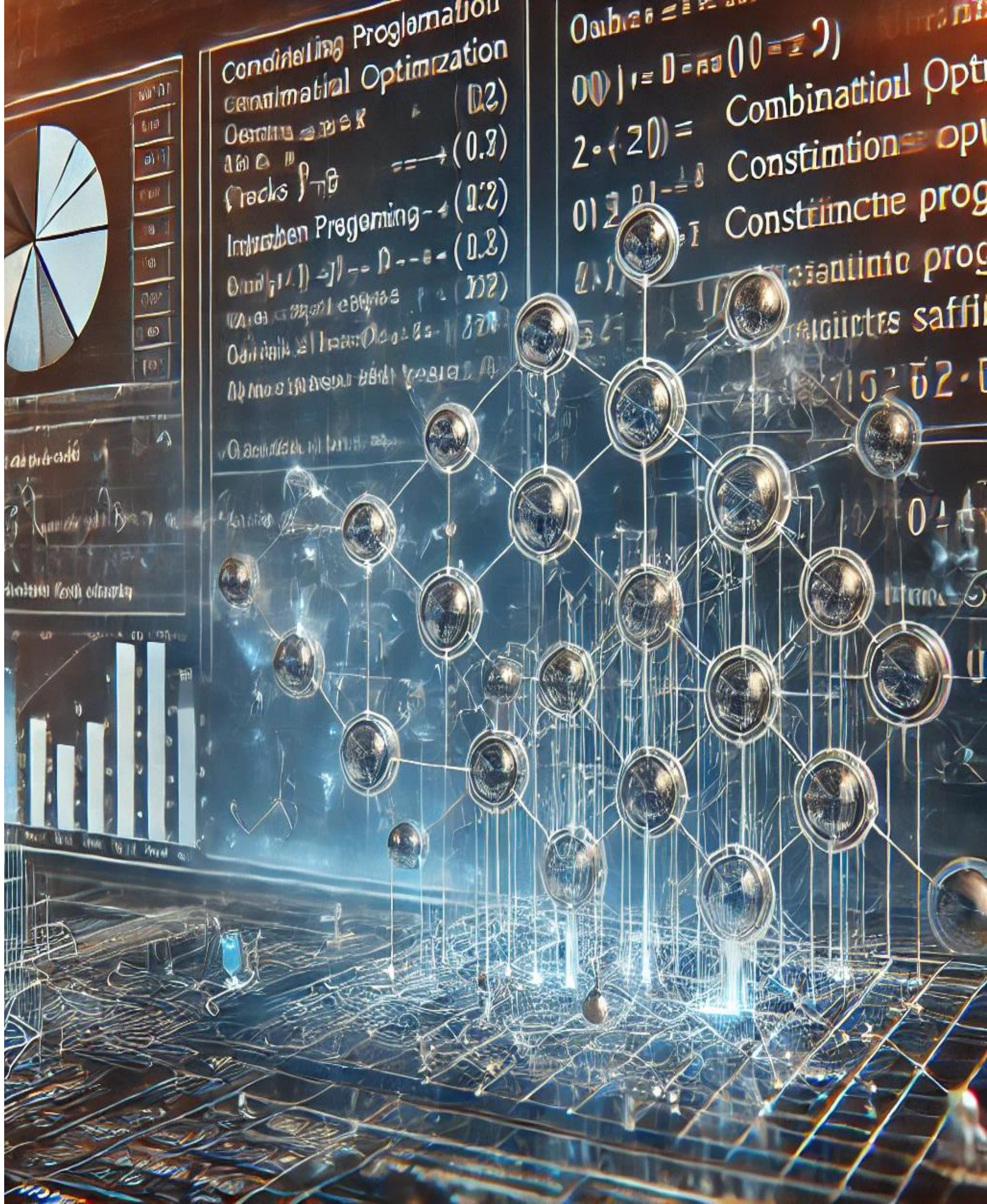
Técnicas de Inteligencia Artificial para la Optimización y Programación de Recursos

Tema 1: Introducción general

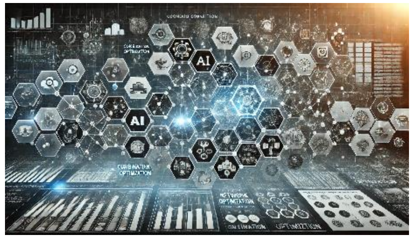
Ramiro Varela Arias
ramiro@uniovi.es

Ciencia de la Computación e Inteligencia Artificial
Departamento de Informática

ceisIA Centro de Estudios sobre el Impacto Social de la Inteligencia Artificial | Universidad de Oviedo

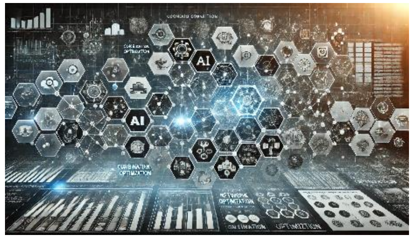


Contenidos del curso



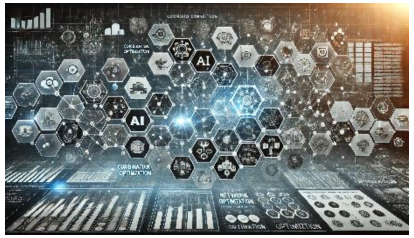
- Introducción a los problemas y técnicas de optimización propias de la IA
- Problemas de scheduling. El problema Job Shop Scheduling
- Algoritmos de Búsqueda Heurística
- Algoritmos Evolutivos
- Programación con Restricciones
- Problemas de scheduling reales (gestión de hidrógeno, control de drones, corte de piezas, …)

Introducción

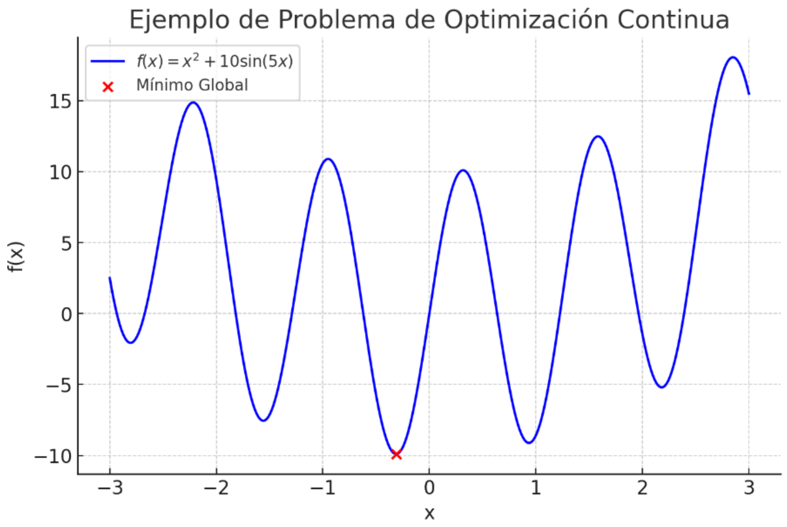


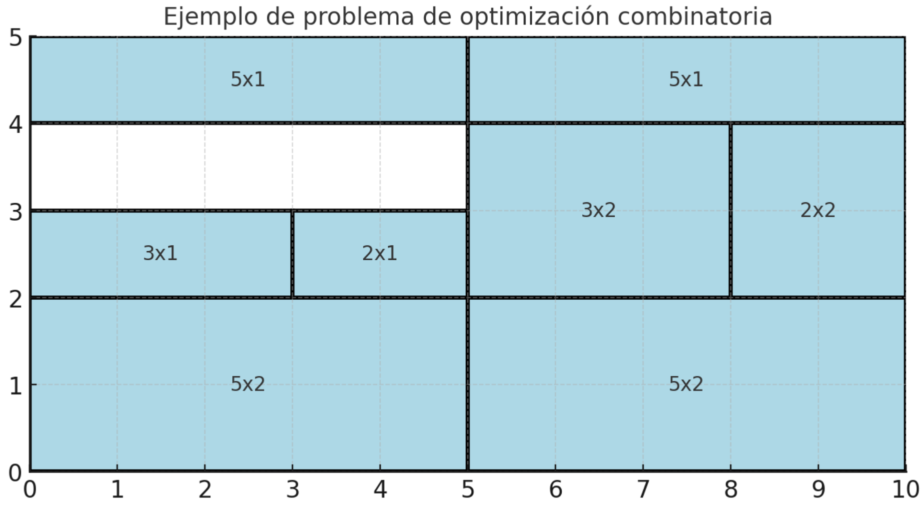
- Algunos problemas de optimización en ciencias e ingeniería
  - Optimización de funciones numéricas
  - Optimización combinatoria
- Formulación de problemas de optimización y satisfacción de restricciones
- Algunas técnicas de resolución propias de la IA
  - Métodos exactos
  - Métodos aproximados
  - Herramientas computacionales

1.- Problemas de Optimización

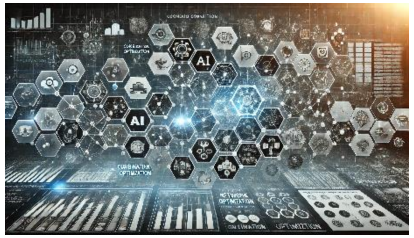


- Problemas en los que se trata de encontrar la mejor solución posible entre un conjunto de soluciones



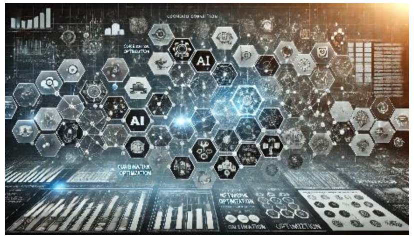


Problemas de Optimización
Ejemplos de aplicaciones reales



- Control de sistemas de distribución de energía eléctrica
- Planificación de cortes de bobinas de film de plástico
- Gestión de contenedores en un gran puerto
- Organización de turnos de trabajo
- Asignación de conductores y autobuses a líneas de transporte
- Planificación de recarga de vehículos eléctricos
- Problemas en biología: protein folding y sequence alignment

Control de sistemas de distribución eléctrica



- Ejemplo: OPF (Optimal Power Flow)
  - Se trata de ajustar parámetros en los generadores y condensadores, para obtener unas condiciones de potencia y voltaje en las cargas, minimizando el consumo de energía

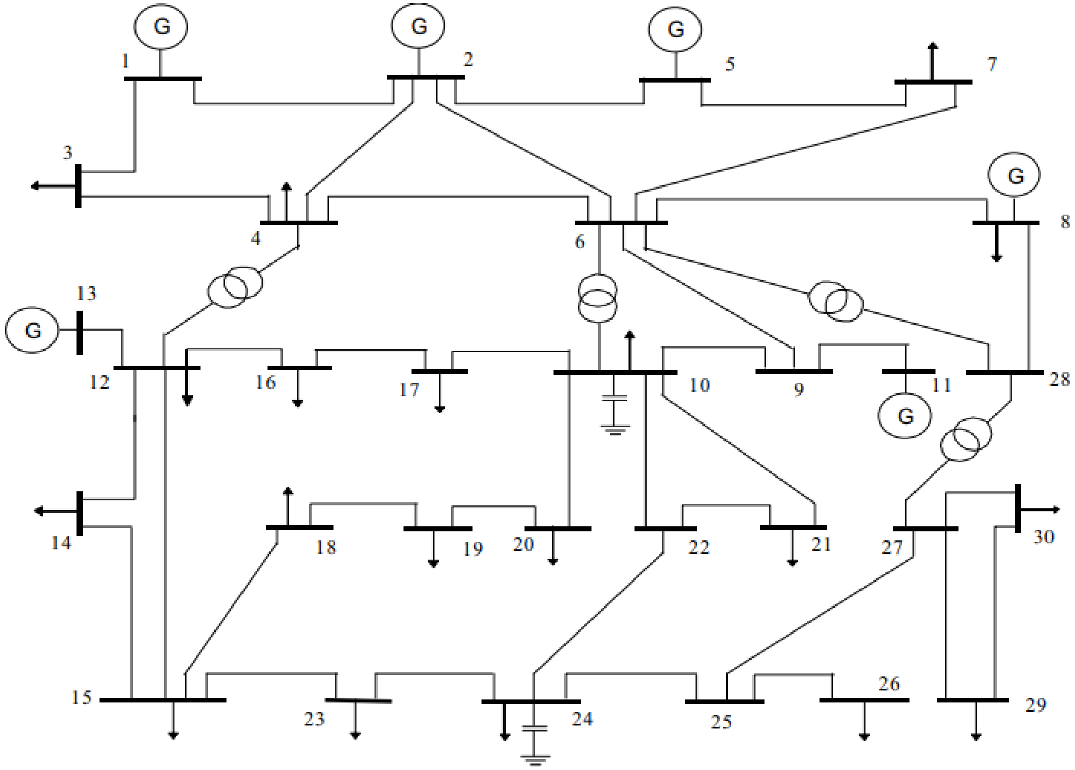
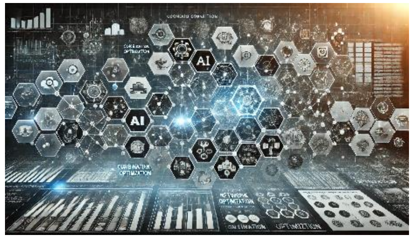
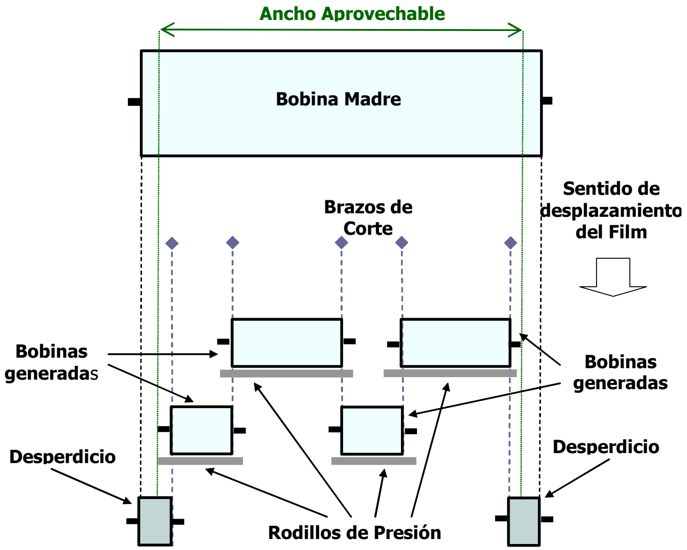


Figure 3. Single-line diagram of standard IEEE-30 bus test network.

Corte de piezas



- Ejemplo: Corte de bobinas de film de plástico
  - Se trata de planificar el corte de bobinas a partir de una bobina madre, con el objetivo de minimizar el desperdicio y los cambios de configuración de la máquina



Datos

```
;; Hoja 14 de Marzo
11
965     3
700     18
530     4
720     3
670     3
850     7
970     9
1000    7
1110    5
1150    5
1300    33
```

Solución

CORTES	7	3	3	2	1	1
BRAZOS						
1	700	1000	1150	1150	1110	1000
2	700	700	1300	530	1000	700
3	1300	720	1300	1300	1300	1110
4	850	1300	1300	970	1000	1300
5	1300	1110	670	1300	1300	1000
6	970	965	-	530	-	-
7	-	-	-	-	-	-
8	-	-	-	-	-	-
9	-	-	-	-	-	-
10	-	-	-	-	-	-
APROVECHADO						
	5820	5795	5720	5780	5710	5110

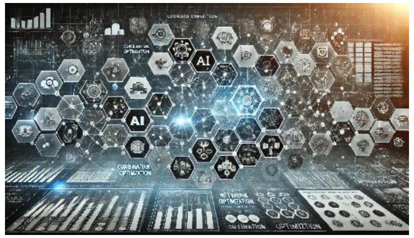
Organización de contenedores

Problema de planificación de actuaciones

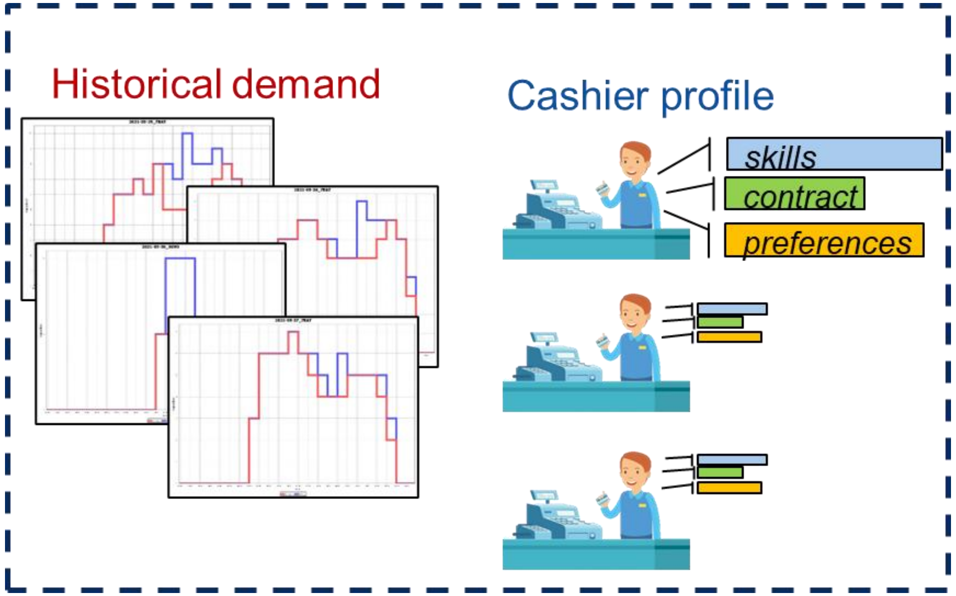
- Se trata de organizar la ubicación de los contenedores de forma que se minimice el número de movimientos y los tiempos de carga/descarga de los buques

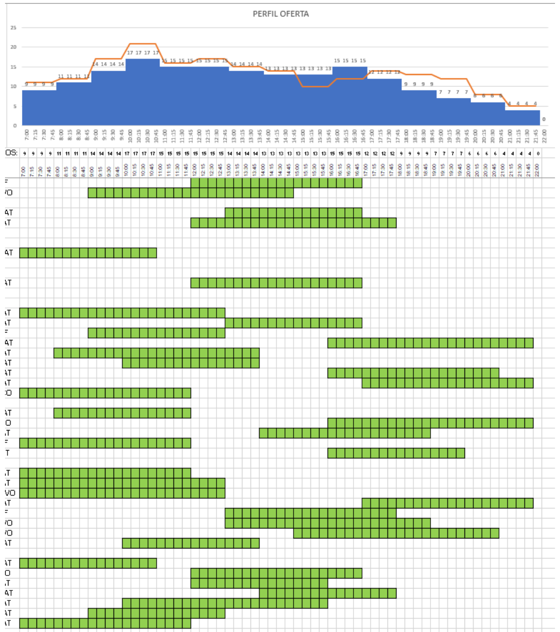
Organización de turnos de trabajo

Rostering

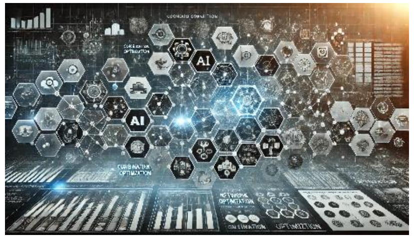


- Ejemplo: Personal de caja en grandes almacenes
  - Se trata de planificar los turnos de trabajo del personal de caja teniendo en cuenta la demanda esperada a distintas horas, minimizando el número de trabajadores y cumpliendo con las condiciones laborales

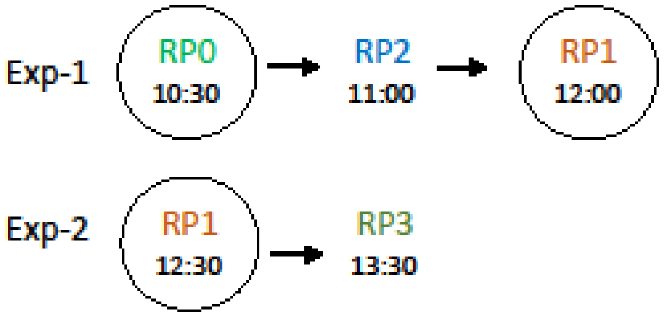


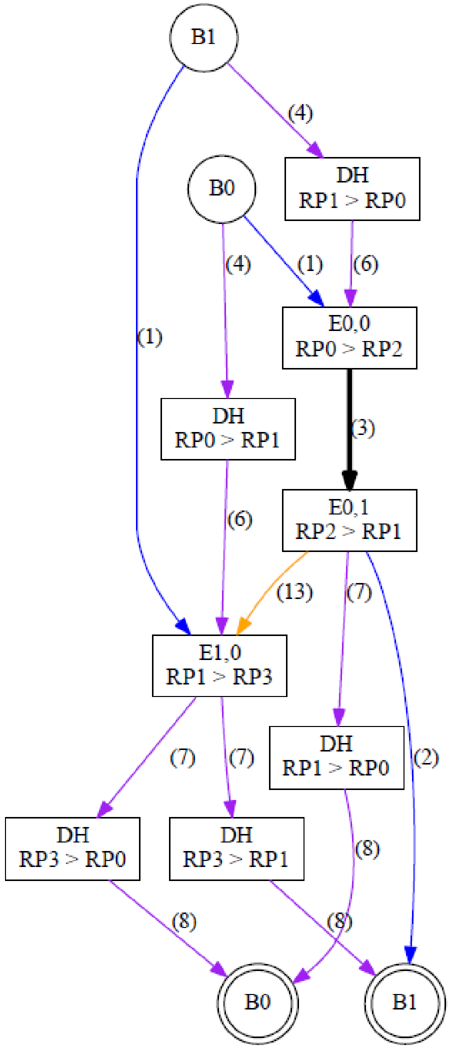


Asignación de vehículos y conductores a rutas

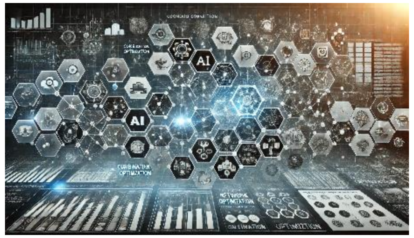


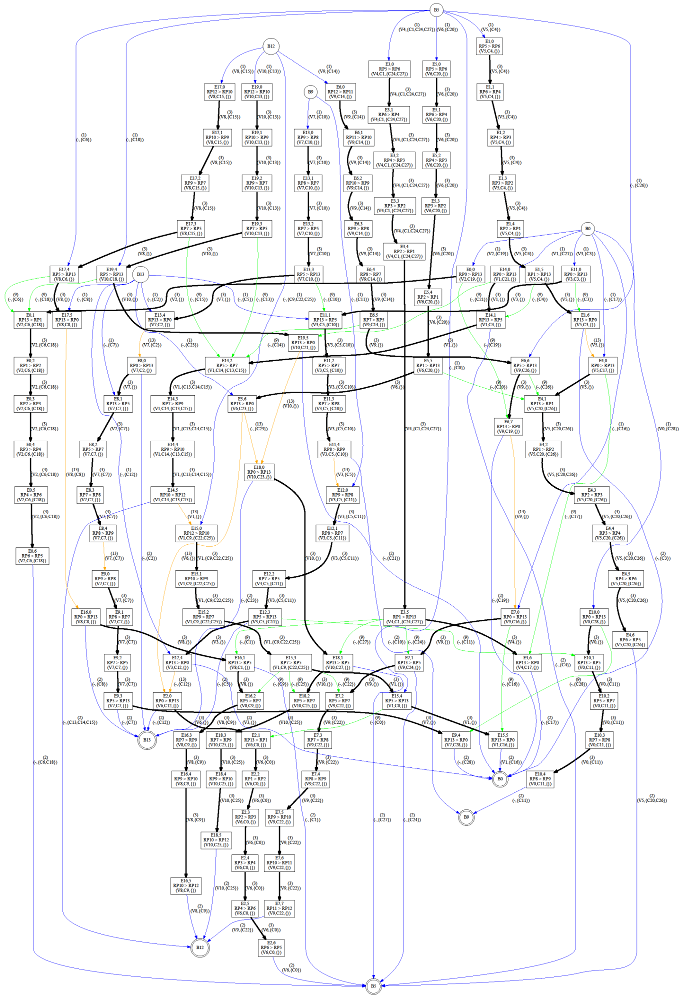
- Dadas unas líneas de transporte predefinidas, se trata de asignar conductores y autobuses a las líneas, minimizando el número de conductores y vehículos, y cumpliendo la normativa laboral.



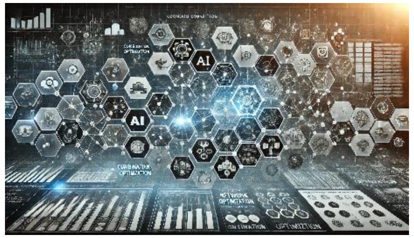


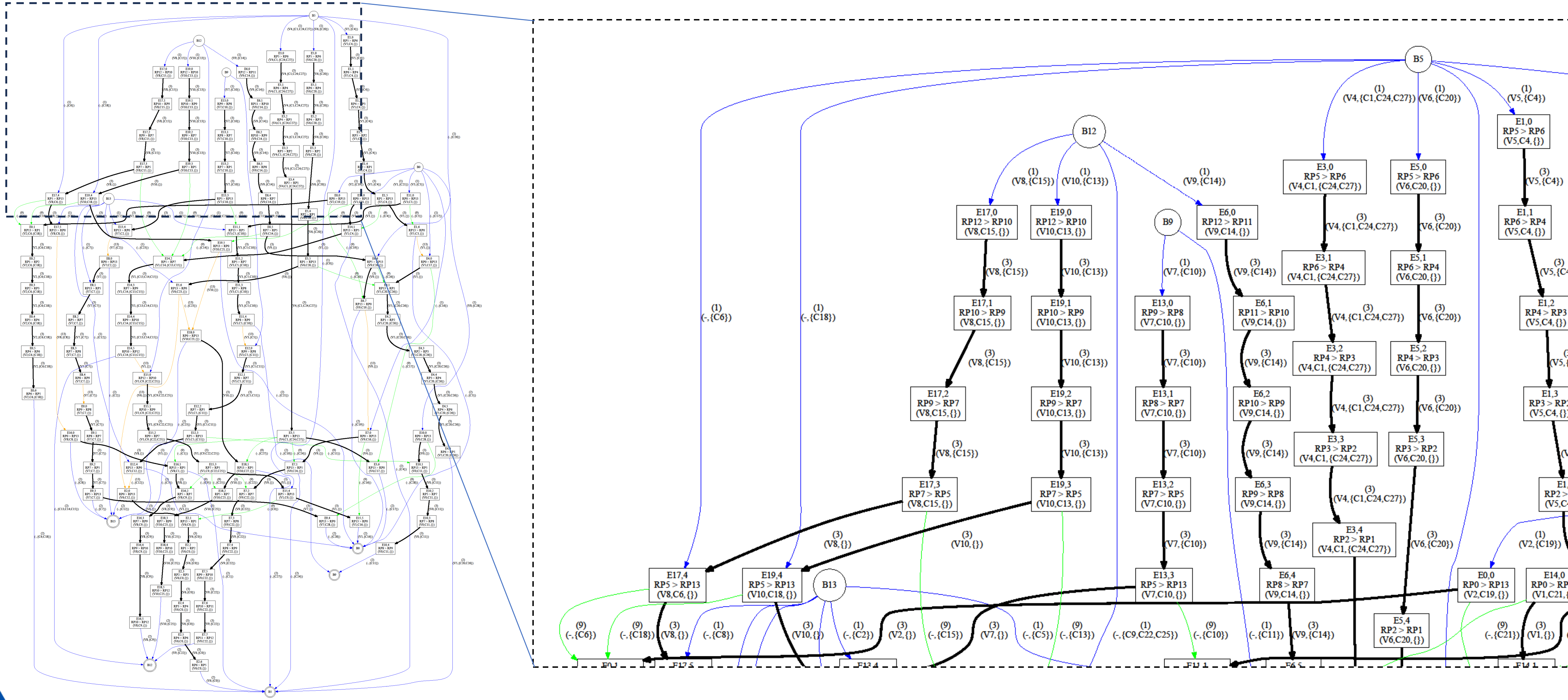
Asignación de vehículos y conductores a rutas





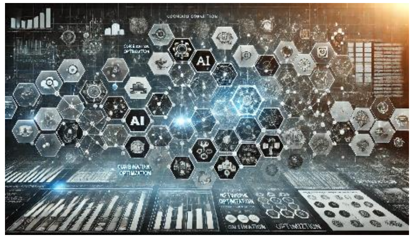
Asignación de vehículos y conductores a rutas



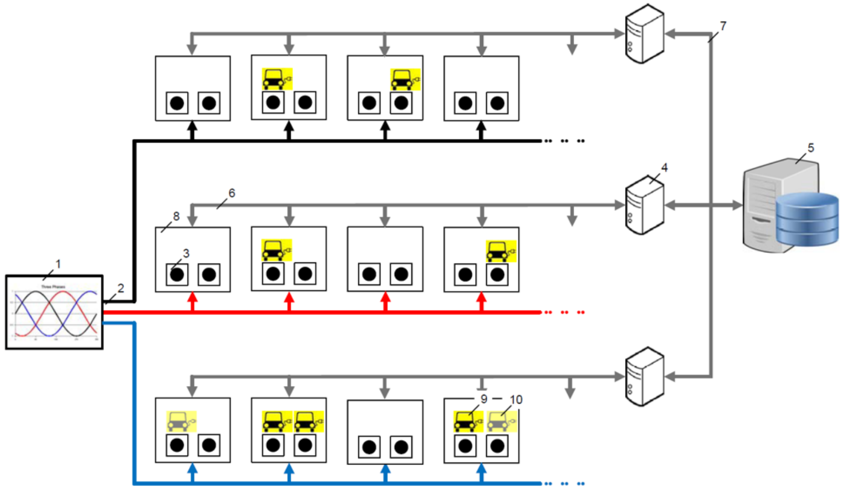


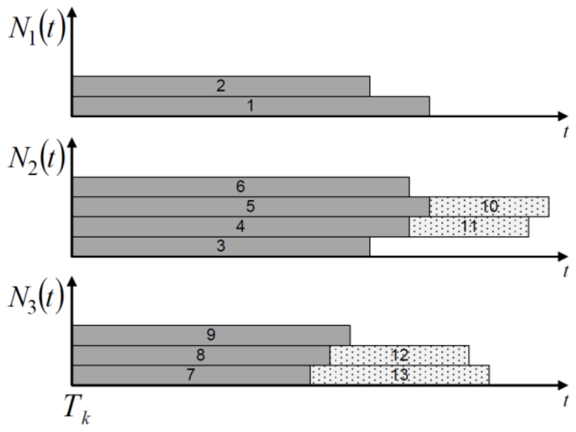
Recarga de Vehículos Eléctricos

Problema dinámico

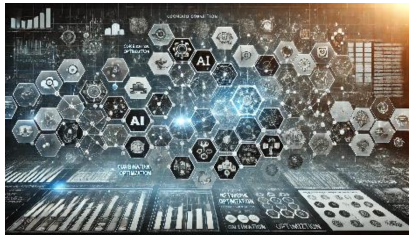


- Ejemplo: Sistema de recarga con alimentación trifásica para comunidades de vecinos
  - Dada la demanda de los usuarios (propietarios) de las plazas, se trata de atender a todos de la mejor forma posible satisfaciendo las restricciones de carga y balance en las tres líneas

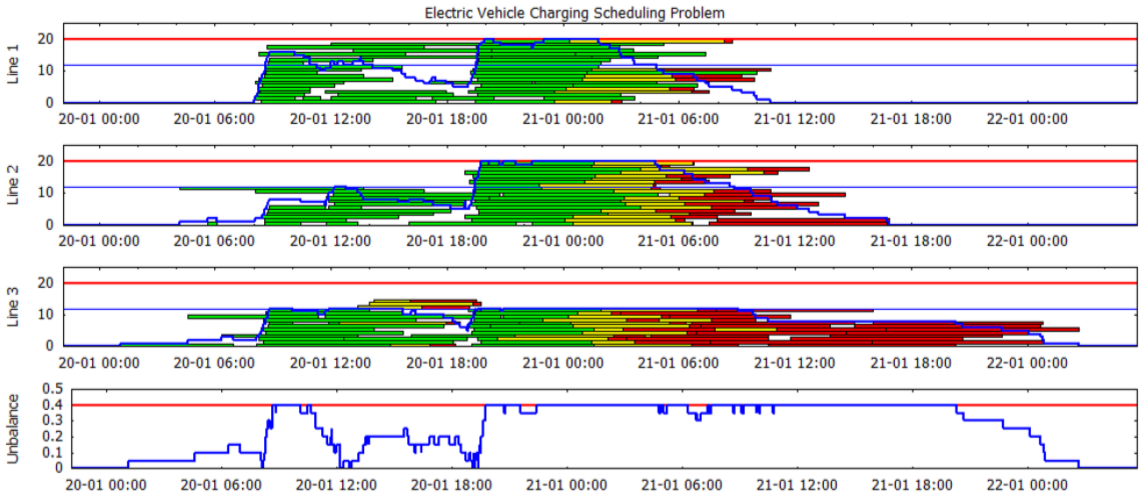




Recarga de Vehículos Eléctricos
Problema dinámico

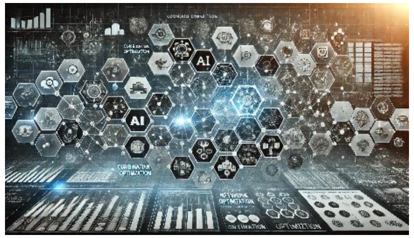


- Ejemplo: Sistema de recarga con alimentación trifásica para comunidades de vecinos
  - Dada la demanda de los usuarios (propietarios) de las plazas, se trata de atendera todos de la mejor forma posible satisfaciendo las restricciones de carga y balance en las tres líneas

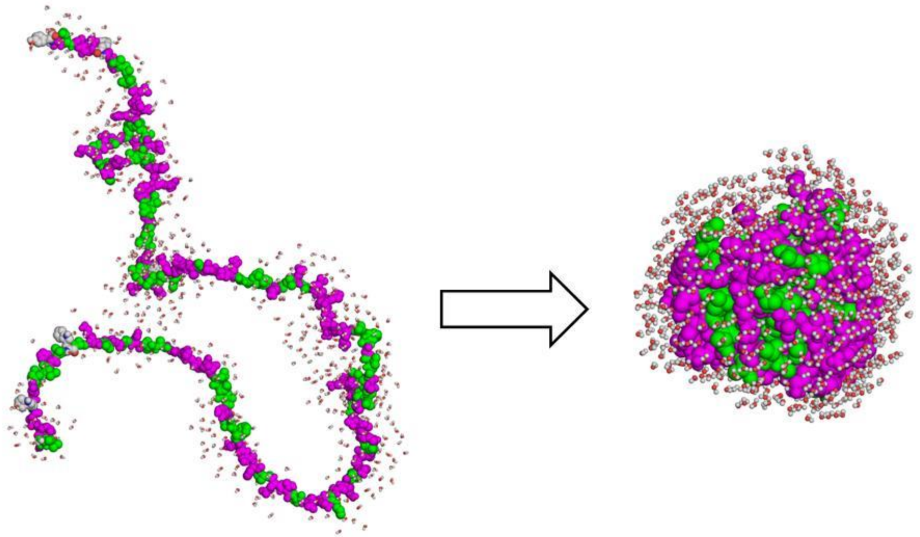


Plegamiento de proteínas

(Protein folding)



- Se trata de predecir la forma final (3D) de una proteína a partir de la secuencia de aminoácidos que la componen



Unfolded **Folded**

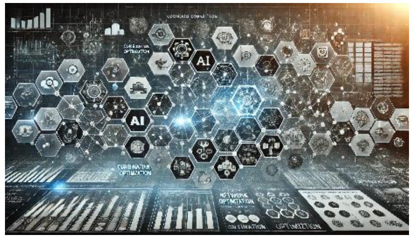
What are the components of the Rosetta Energy Function?

When Rosetta calculates the an energy score for a molecule's conformation, it is actually calculating 12 different kinds of energy terms.

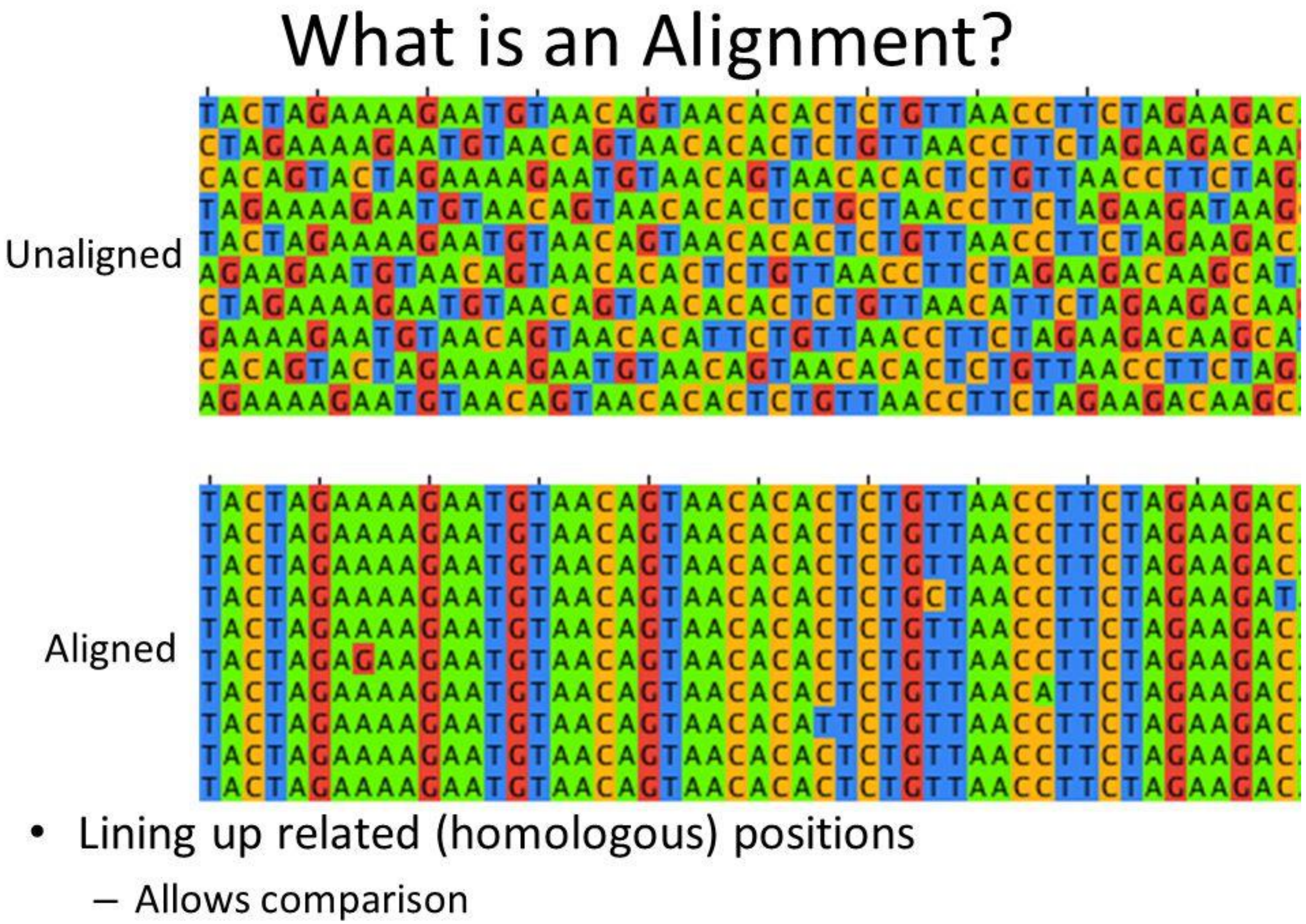
1. Lennard-Jones Attraction and Repulsion Forces between different residues
2. Lennard-Jones Repulsion Force between atoms of the same residue
3. Coulombic Electrostatic Potential
4. Lazaridis-Karplus Solvation Energy
5. Proline Closure Term
6. Hydrogen Bond Energy
7. Disulfide Geometry Potential
8. Ramachandran Preferences
9. Harmonic Constraint for Backbone Omega Dihedral Angles
10. Side Chain Rotamer Term using Dunbrack's 2010 Statistical Library
11. Probability Term of an Amino Acid given a Phi/Psi angle
12. Reference Energy for each Amino Acid – Average energy of each amino acid in the unfolded state

Alineamiento de secuencias

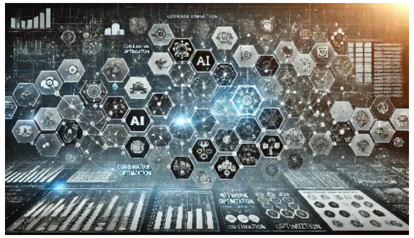
(sequence alignment)



- Se trata de alinear secuencias de símbolos tratando de alinear fragmentos análogos

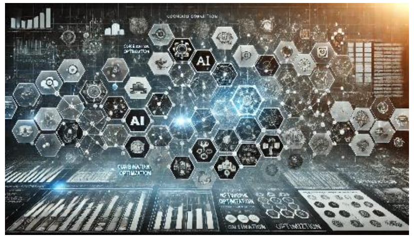


2.- Formulación de problemas de optimización



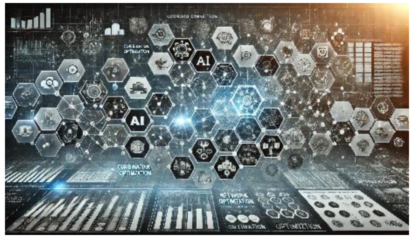
- Todos los problemas anteriores son de la familia de problemas de optimización con satisfacción de restricciones (CSOP, Constraint Satisfaction Optimization Problem)
- La formulación de un CSOP requiere la definición precisa de los siguientes elementos
  - Datos del problema
  - Qué es una solución del problema
  - Restricciones que debe cumplir la solución
  - Función/es objetivo que se deben optimizar

Formulación de problemas de optimización

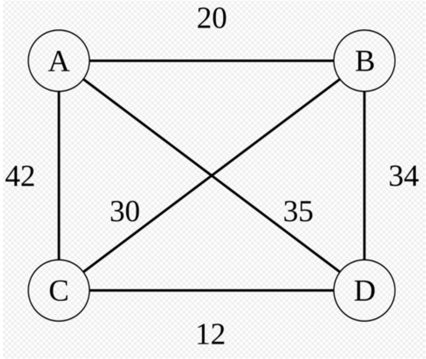


- Ejemplo simple: Planificación de trabajos en distintas máquinas
  - Datos del problema
    - Tenemos un conjunto de *m* máquinas, un conjunto de *n* tareas, y una matriz $n \times m$ con las duraciones de cada tarea en cada máquina. La tarea *i* está disponible en el tiempo r_i
  - Qué es una solución del problema
    - Para cada tarea, la asignación de una máquina y un tiempo de inicio
  - Restricciones que debe cumplir la solución
    - Cada máquina solo puede realizar una tarea a la vez
    - El tiempo de inicio de la tarea *i* debe ser mayor o igual al tiempo de disponibilidad r_i
  - Función objetivo
    - El tiempo de finalización de la última tarea debe ser mínimo

Formulación de problemas de optimización

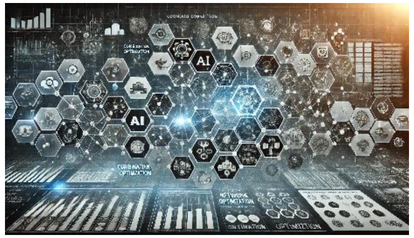


- Ejemplo: El problema del viajante de comercio (TSP, o Travelling Salesman Problem)
  - Datos del problema
    - Un conjunto de *n* ciudades y un grafo con los costes de las conexiones entre cada par de ciudades
  - Solución
    - Una secuencia de *n* arcos del grafo de conexiones
  - Restricciones
    - La secuencia de arcos debe representar un camino que parte de la primera ciudad y regresa a la misma, pasando una y solo una vez por cada una de las *n* ciudades
  - Función objetivo
    - El coste del camino debe ser mínimo



Formulación de problemas de optimización

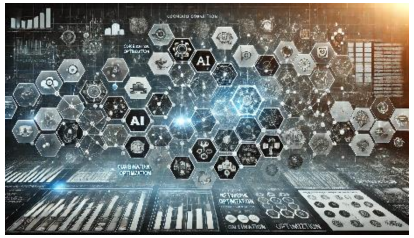
Más formal con notación matemática



- Ejemplo simple: Planificación de trabajos en varias máquinas
  - Datos del problema
    - Un conjunto $\{1,\ldots,m\}$ de máquinas, un conjunto $\{1,\ldots,n\}$ de tareas, disponibles en los instantes $\{r_1,\ldots,r_n\}$
    - Una matriz $Dn\times m$ tal que D_{ij} es la duración de la tarea i en la máquina j
  - Qué es una solución del problema (variables de decisión)
    - $x_{ij} = 1$ si a la tarea i se asigna a la máquina j, 0 en otro caso
    - $t_i =$ tiempo de inicio de la tarea i
  - Restricciones que debe cumplir la solución
    - $\sum_{j=1,\ldots,m} x_{ij} = 1,\ 1\leq i\leq n$
    - $t_i \geq r_i\ 1\leq i\leq n$
    - $t_i + D_{ij} \leq t_k \vee t_k + D_{kj} \leq t_i,\ 1\leq i,k\leq n,\ 1\leq j\leq m,\ x_{ij} = x_{kj} = 1$
  - Función/es objetivo que se deben optimizar
    - $\min\{\max(t_i + D_{ij} * x_{ij}),\ 1\leq i\leq n,\ 1\leq j\leq m\}$

Formulación de problemas de optimización

Más formal con notación matemática



- Ejemplo: El problema del viajante de comercio
  - Formulación en Programación Lineal Entera (ILP)

Dantzig–Fulkerson–Johnson formulation [edit]

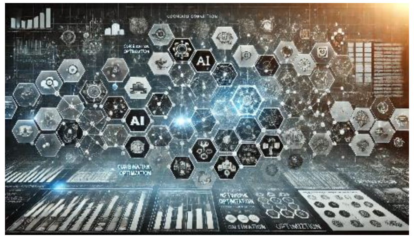
Label the cities with the numbers 1, ..., *n* and define:

$$x_{ij} = \begin{cases} 1 & \text{the path goes from city } i \text{ to city } j \\ 0 & \text{otherwise.} \end{cases}$$

Take $c_{ij} > 0$ to be the distance from city *i* to city *j*. Then TSP can be written as the following integer linear programming problem:

$$\begin{aligned} \min \sum_{i=1}^{n} \sum_{j\neq i, j=1}^{n} c_{ij} x_{ij} \colon & \\ \sum_{i=1, i\neq j}^{n} x_{ij} = 1 & \qquad j = 1, \ldots, n; \\ \sum_{j=1, j\neq i}^{n} x_{ij} = 1 & \qquad i = 1, \ldots, n; \\ \sum_{i\in Q} \sum_{j\neq i, j\in Q} x_{ij} \leq |Q| - 1 & \qquad \forall Q \subsetneq \{1, \ldots, n\}, |Q| \geq 2. \end{aligned}$$

Problemas multiobjetivo



- Tienen varias funciones que se deben optimizar simultáneamente
  - Ejemplo 1: en el problema anterior, además de minimizar el tiempo de fin de las tareas, podemos querer minimizar la máxima diferencia del número de tareas en diferentes máquinas
  - Ejemplo 2: en el problema de cortes de bobinas, queremos minimizar el desperdicio de plástico, los cambios de configuración de la máquina, el número de palets abiertos, …
- Los objetivos suelen ser contradictorios
  - Se pueden considerar de forma combinada
  - O de forma jerárquica
  - O se puede buscar un conjunto de soluciones “no dominadas”

Problemas multiobjetivo

- Tienen varias funciones que se deben optimizar simultáneamente
- Los objetivos suelen ser contradictorios
  - Conjunto de soluciones "no dominadas": el Frente Pareto

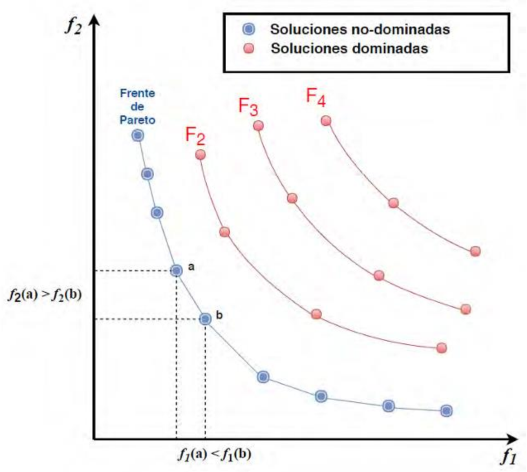
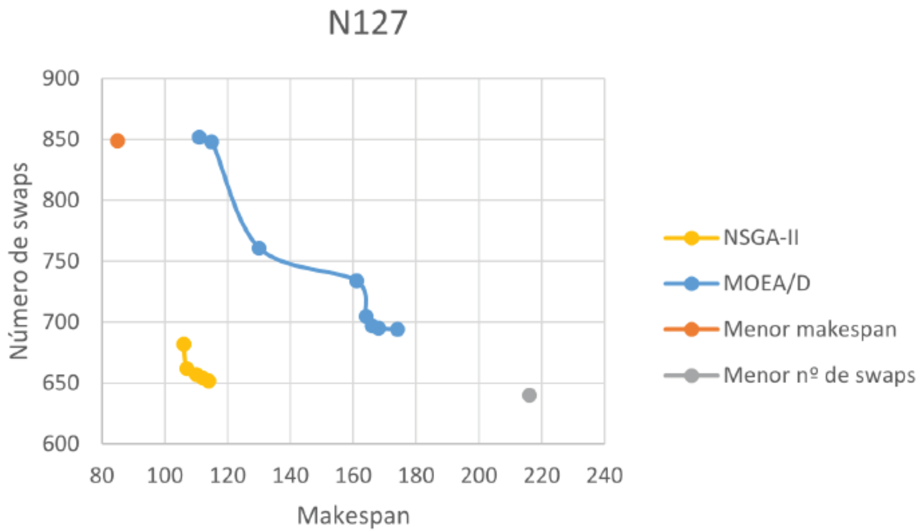
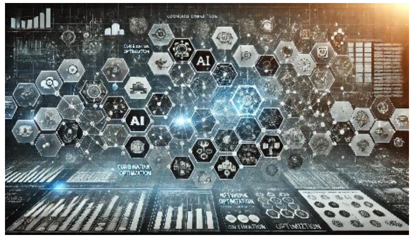


Figura 4. Minimización de f_1 y f_2

Ejemplo: Compilación de circuitos cuánticos



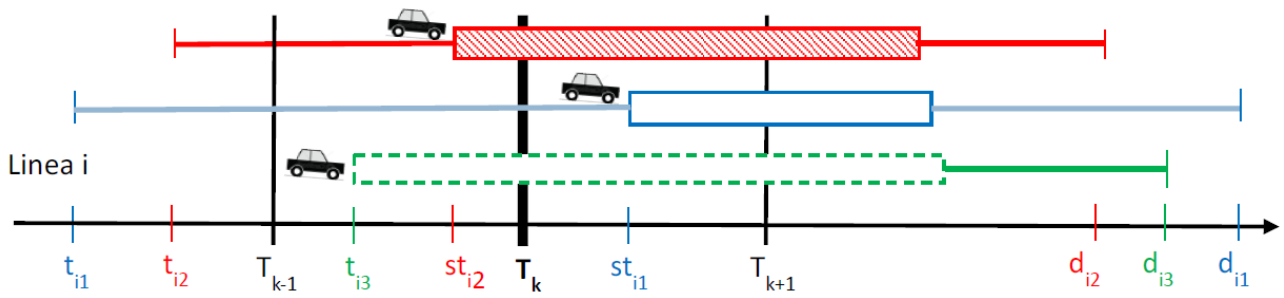
Problemas dinámicos



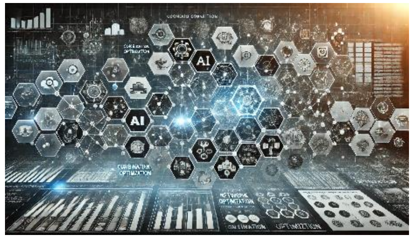
- Los datos, las restricciones o las funciones objetivo pueden cambiar con el tiempo, normalmente de forma gradual
- Se pueden tratar como una secuencia de problemas estáticos
  - La solución del problema sirve hasta que se produzca un cambio
  - Ante un cambio, se calcula una nueva solución (si el cambio es pequeño, quizá baste con modificar la solución anterior)
- Ejemplo: En el problema de recarga de vehículos eléctricos los vehículos llegan de forma asíncrona, o un vehículo puede terminar su período de carga antes de lo previsto, …

Problemas dinámicos

- Ejemplo: En el problema de recarga de vehículos eléctricos los vehículos llegan de forma asíncrona, o un vehículo puede terminar su período de carga antes de lo previsto, …
  - Secuencia de problemas estáticos en instantes T_0, …, T_k, ….
  - En cada instante T_k se planifican los vehículos que no empezaron la recarga, teniendo en cuenta los que están en recarga



Problemas de Satisfacción de Restricciones



- Son como los problemas anteriores, pero no hay función objetivo. Basta con encontrar soluciones factibles

Definition — A constraint satisfaction problem on finite domains (or CSP) is defined by a triplet $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ where:

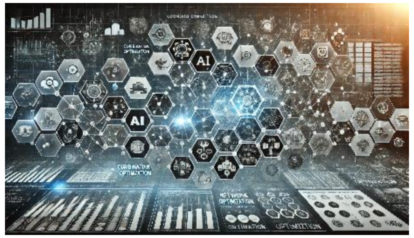
- $\mathcal{X} = \{x_1, \ldots, x_n\}$ is the set of variables of the problem;
- $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ is the set of domains of the variables, i.e., for all $k \in [1; n]$ we have $x_k \in \mathcal{D}_k$;
- $\mathcal{C} = \{C_1, \ldots, C_m\}$ is a set of constraints. A constraint $C_i = (\mathcal{X}_i, \mathcal{R}_i)$ is defined by a set $\mathcal{X}_i = \{x_{i_1}, \ldots, x_{i_k}\}$ of variables and a relation $\mathcal{R}_i \subseteq \mathcal{D}_{i_1} \times \cdots \times \mathcal{D}_{i_k}$ that defines the set of values allowed simultaneously for the variables of $\mathcal{X}_i$.

Definition — An assignment (or model) $\mathcal{A}$ of a CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is defined by the couple $\mathcal{A} = (\mathcal{X}_\mathcal{A}, \mathcal{V}_\mathcal{A})$ where:

- $\mathcal{X}_\mathcal{A} \subseteq \mathcal{X}$ is a subset of variable;
- $\mathcal{V}_\mathcal{A} = \{v_{\mathcal{A}_1}, \ldots, v_{\mathcal{A}_k}\} \in \{\mathcal{D}_{\mathcal{A}_1}, \ldots, \mathcal{D}_{\mathcal{A}_k}\}$ is the tuple of the values taken by the assigned variables.

Definition — A solution of a CSP is a total assignment that satisfies all the constraints of the problem.

Problemas de Satisfacción de Restricciones



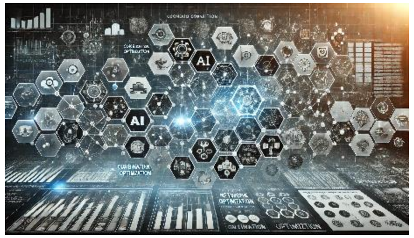
- Ejemplo:

Sudoku as a Constraint Satisfaction Problem (CSP)

	1	2	3	4	5	6	7	8	9
A		6		1		4		5	
B			8	3		5	6		
C	2								1
D	8			4		7			6
E			6				3		
F	7			9		1			4
G	5								2
H			7	2		6	9		
I		4		5		8		7	

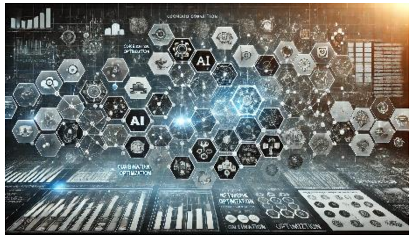
- Variables: 81 variables
  - A1, A2, A3, …, I7, I8, I9
  - Letters index rows, top to bottom
  - Digits index columns, left to right
- Domains: The nine positive digits
  - A1 ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9}
  - Etc.; all domains of all variables are {1,2,3,4,5,6,7,8,9}
- Constraints: 27 *Alldiff* constraints
  - *Alldiff*(A1, A2, A3, A4, A5, A6, A7, A8, A9)
  - Etc.; all rows, columns, and blocks contain all different digits
  - Fixed positions cannot be changed

3.- Algunas técnicas de resolución propias de la IA



- Método heurístico: método de resolución basado en conocimiento del problema
- Técnicas exactas
  - Búsqueda heurística en espacios de estados
  - Programación con Restricciones
  - Programación Lineal (LP, ILP, MILP)
- Técnicas aproximadas
  - Metaheurísticas
    - Algoritmos Evolutivos (AG, PSO, ACO, …)
    - Búsqueda Local (Recocido Simulado, Búsqueda Tabú, Path Relinking, . . .)
  - Aprendizaje Automático
    - Regresión numérica y simbólica
    - Hyper-heurísticos (Programación Genética)
    - Aprendizaje por Refuerzo

Búsqueda heurística en espacios de estados



- Modelado del espacio de soluciones como un grafo dirigido con costes en los arcos
  - Ejemplo: cálculo de rutas en mapas

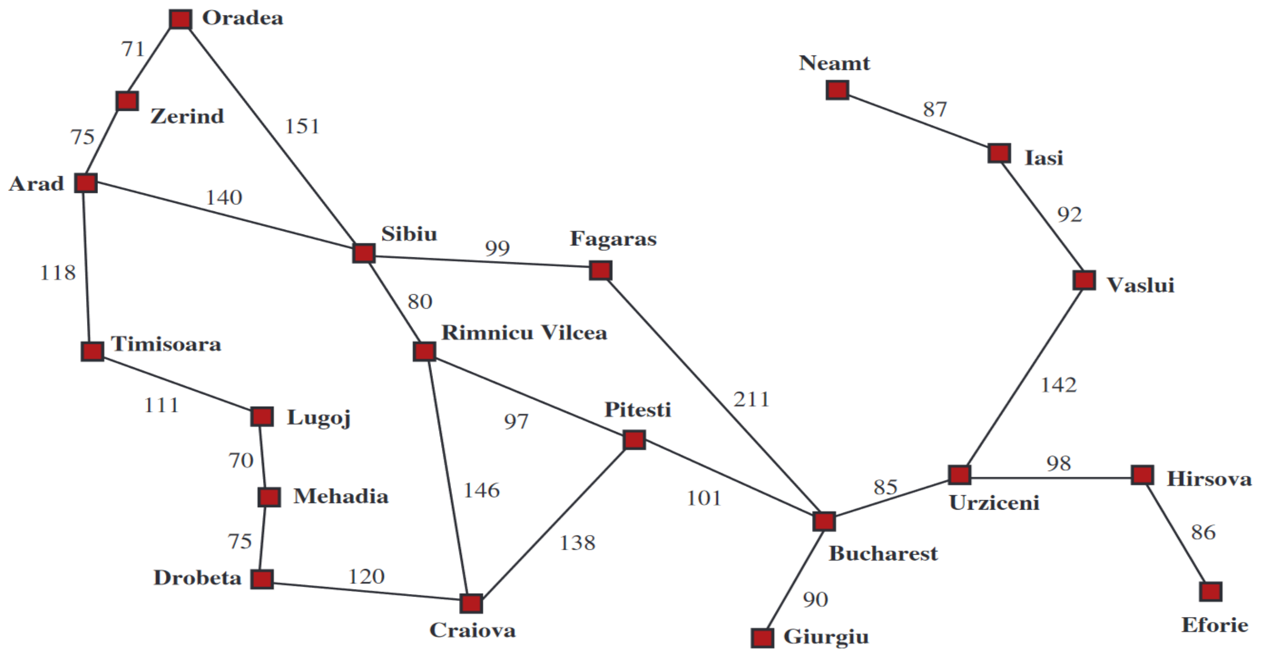


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

Búsqueda heurística en espacios de estados

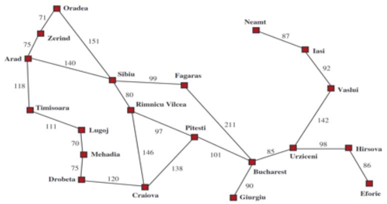


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

- Solución: algoritmo de búsqueda en grafos
  - Ejemplo: algoritmo "Best First Search"
    - Permite introducir conocimiento sobre el dominio el problema a través de la función f (función heurística

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```

Búsqueda heurística en espacios de estados

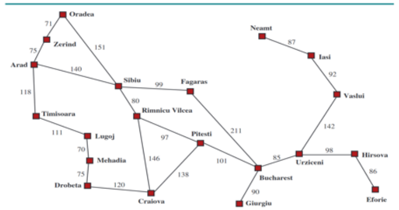
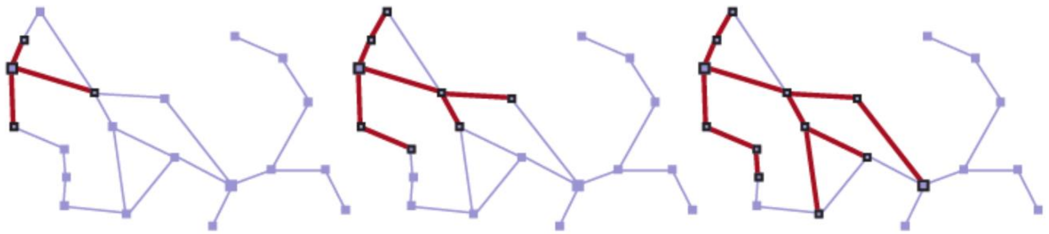


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

- Solución: algoritmo de búsqueda en grafos
  - Ejemplo: algoritmo "Best First Search"
    - Permite introducir conocimiento sobre el dominio el problema a través de la función f (función heurística)
    - Desarrolla un árbol de búsqueda hasta que a través de una rama se encuentra una solución

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```



Búsqueda heurística en espacios de estados

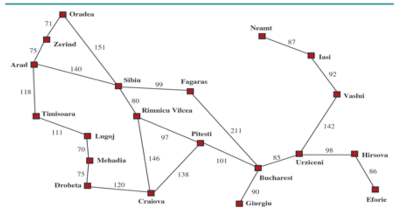
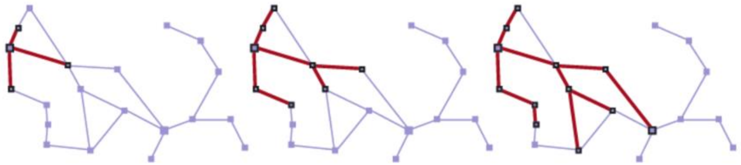


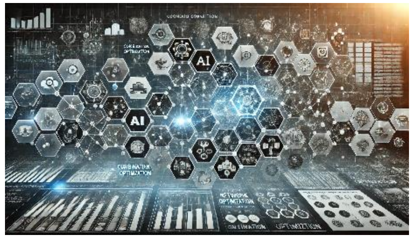
Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

- Solución: algoritmo de búsqueda en grafos
  - Ejemplo: algoritmo "Best First Search"
    - Permite introducir conocimiento sobre el dominio el problema a través de la función f (función heurística)
    - Desarrolla un árbol de búsqueda hasta que a través de una rama se encuentra una solución
    - La claves son:
      - La definición del espacio de búsqueda (tamaño, factor de ramificación, …)
      - La definición de la función f (dependiendo de sus características de la función f, el algoritmo puede ser exacto o aproximado, o ser más o menos eficiente)

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```

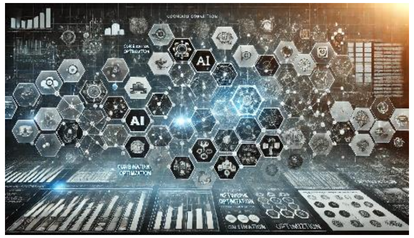


Metaheurísticas



- Una metaheurística es una estrategia que guía la aplicación de un heurístico
  - Basadas en poblaciones
    - Evolucionan una población de soluciones potenciales, normalmente inspirándose en la naturaleza
    - Ejemplos: Algoritmos Evolutivos (Genetic Algorithms (GA), Enjambres de partículas, Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), . . .)
  - Basadas en trayectorias
    - Modifican una solución a través de una serie de pasos
    - Ejemplos: Local Search (LS), Simulated Annealing (SA), Taboo Search (TS), Path Relinking (PR)

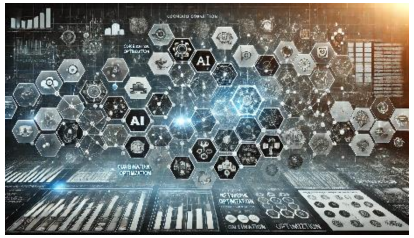
Algoritmos Genéticos



- Evoluciona una población de soluciones potenciales, inicialmente aleatoria, mediante la aplicación de operadores de Evaluación, Selección, Cruce, Mutación y Reemplazamiento

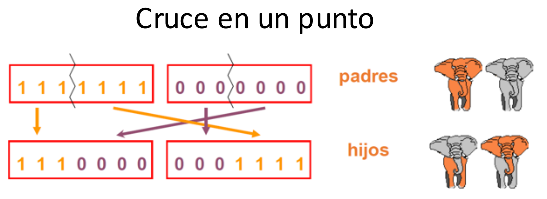
```
Algoritmo Genético
    Parámetros de entrada (ProbCruce, ProbMutacion, maxGen, PobSize, … );
    numGen ← 0;
    Inicializar(Pob(0));               // Población inicial
    Evaluar(Pob(0));                   // Función de fitness
   while ( numGen < maxGen )  {        // Condición de parada
        numGen ← numGen+1;
        Pob’(numGen) = Selección(Pob(numGen-1));              // Selección
        Pob’’(numGen) = Cruce(Pob’(numGen));                  // Cruce
        Pob’’’(numGen) = Mutación(Pob’’(numGen));             // Mutación
        Evaluar(Pob’’’(numGen));                              // Función de fitness
        Pob(numGen) = Reemplazo(Pob’(numGen), Pob’’’(numGen));   // Reemplazo
   }
    return el mejor individuo en Pob(maxGen);
end
```

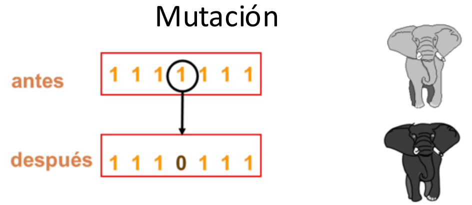
Algoritmos Genéticos



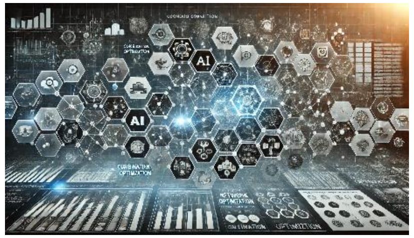
- Evoluciona una población de soluciones potenciales, inicialmente aleatoria, mediante la aplicación de operadores de Evaluación, Selección, Cruce, Mutación y Reemplazamiento
  - Las soluciones potenciales se codifican, normalmente, con cadenas de símbolos, por ejemplo, 0 y 1 (codificación binaria

```
Algoritmo Genético
    Parámetros de entrada (ProbCruce, ProbMutacion, maxGen, PobSize, … );
    numGen ← 0;
    Inicializar(Pob(0));          // Población inicial
    Evaluar(Pob(0));              // Función de fitness
    while ( numGen < maxGen )  {          // Condición de parada
        numGen ← numGen+1;
        Pob'(numGen) = Selección(Pob(numGen-1));      // Selección
        Pob''(numGen) = Cruce(Pob'(numGen));          // Cruce
        Pob'''(numGen) = Mutación(Pob''(numGen));     // Mutación
        Evaluar(Pob'''(numGen));                      // Función de fitness
        Pob(numGen) = Reemplazo(Pob'(numGen), Pob'''(numGen));   // Reemplazo
    }
    return el mejor individuo en Pob(maxGen);
end
```



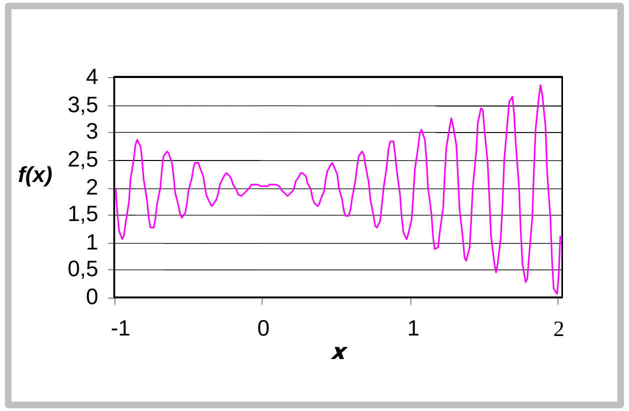


  - Las claves son:
    - Un esquema de codificación adecuado (tamaño del espacio de búsqueda, calidad media de las soluciones, . . .)
    - Algoritmo de evaluación eficiente
    - Operadores de cruce y mutación adecuados (herencia de propiedades, cambios no muy disruptivos, . . .)



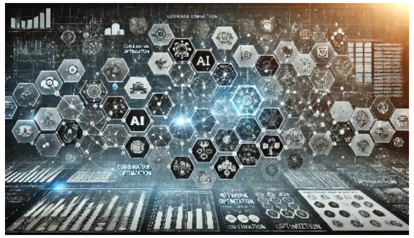
Algoritmos Genéticos

- Ejemplo: optimización numérica
  - Calcular el máximo de una función en un intervalo
    - $f(x) = x*\sin(10\pi x)+2.0$
    - Intervalo [-1,2]

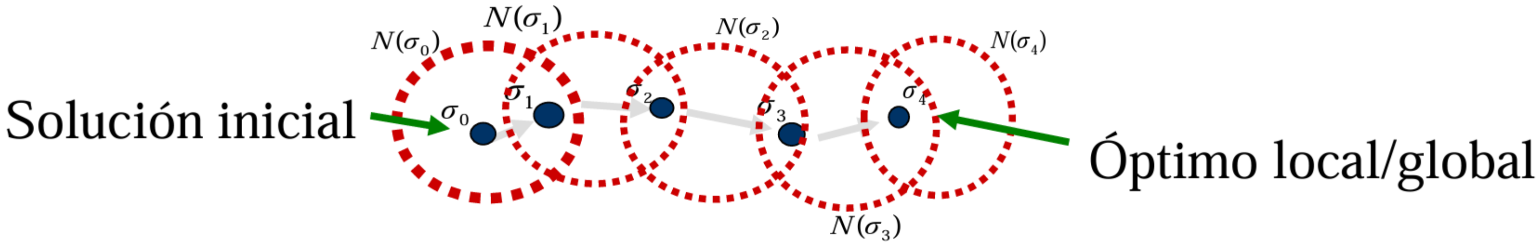


  - Solución con un AG
    - Soluciones candidatas: $x \in [-1,2]$
    - Codificación: s cadena binaria
    - Decodificación: $s \rightarrow x$; paso de binario a decimal, con $(0\ 0\ ...\ 0) \rightarrow -1$; $(1\ 1\ ...\ 1) \rightarrow 2$
      - La longitud de s depende de la precision y determina el número total de cromosomas diferentes
      - Si la precisión es 10^6 la longitud debe ser 32, ya que $2^{31} \leq 10^6 \leq 2^{32}$
    - Fitness(s) = f(Decodifica(x))

Búsqueda Local

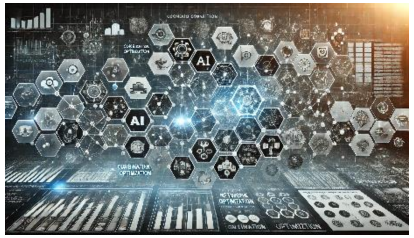


- Proceso iterativo
  - Calcula un conjunto de soluciones vecinas
  - Selecciona una de las soluciones vecinas con algún criterio
  - El proceso se repite hasta que se cumple un criterio de terminación

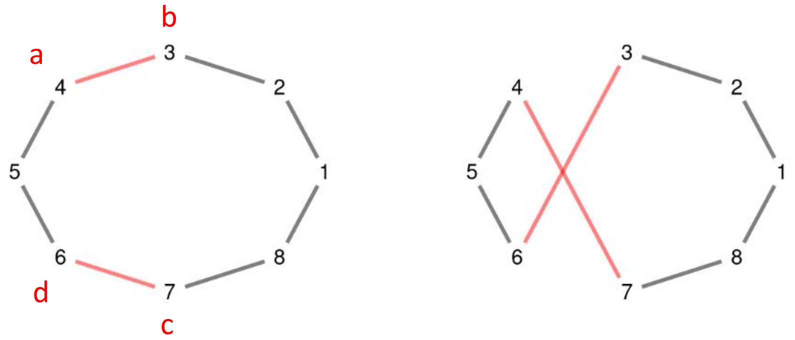


- Las claves son
  - Cómo definir una estrategia de vecindad razonable (tamaño, posibilidades de mejora)
  - Cómo establecer un criterio de selección de vecinos (solo mejoras, ...)
  - Cómo evaluar las soluciones vecinas (estimaciones eficientes, evaluaciones completas, . . .)
  - Cuándo parar (no hay mejoras, después de un número de pasos, . . .)

Búsqueda Local

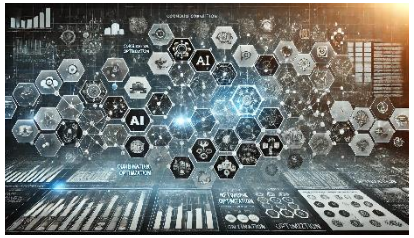


- Ejemplo: el viajante de comercio (TSP)
  - Una estructura de vecindad clásica es “2-opt move”: se eligen dos arcos no consecutivos (a,b) y (c,d) y se reconectan las ciudades con los arcos (a,c) y (b,d)

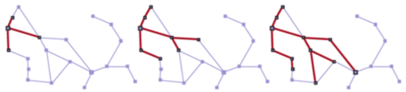


  - La solución vecina es factible y fácil de evaluar (no hay que reevaluar toda la solución)
  - Pero la cantidad de soluciones vecinas es muy grande (tantas como pares de arcos no consecutivos en la solución)

Algoritmos greedy



- Son un caso particular de algoritmos de búsqueda en los que en cada expansión se elige una opción y se descartan el resto, con lo que generan árboles de búsqueda con una sola rama



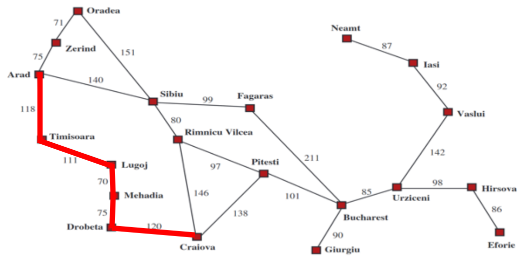
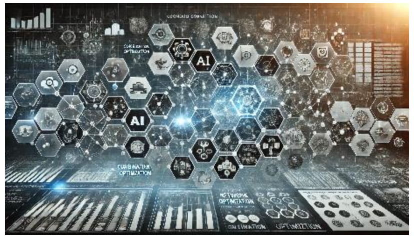


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

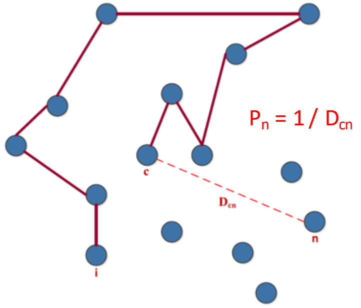
- También se llaman "generadores de soluciones" (en scheduling se denominan "Schedule builders" o "Schedule generation schemes")
- La clave de su eficiencia está en la "regla de prioridad" o "heurístico" que permite tomar la siguiente decisión
  - Mediante conocimiento de los expertos
  - Mediante Aprendizaje Automático (Programación Genética, Aprendizaje por Refuerzo, Aprendizaje Supervisado)

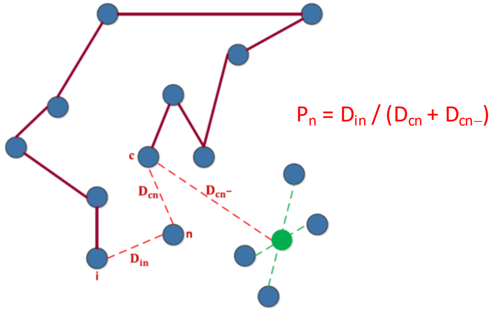
Algoritmos greedy



- Ejemplo: En el problema del viajante de comercio
  - Un algoritmo clásico

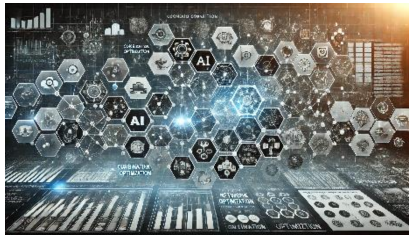
Data: A TSP instance.
Result: A feasible route R.
$R \leftarrow$ starting city;
$UVC \leftarrow$ all unvisited cities;
while $UVC \neq \emptyset$ **do**
  A city $u \in UVC$ is selected heuristically;
  Add u to the route R;
  Remove u from UVC;
end
return *The route* R;





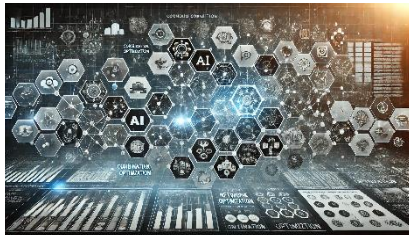
  - Una regla clásica es NN (Nearest Neighbour) que asigna a cada ciudad candidata una prioridad en relación inversa con la distancia a la actual
  - Pero se pueden considerar otros atributos, como la distancia a la inicial *Din* o la distancia al centroide *Dcn-*, para calcular otras reglas de prioridad más elaboradas

Herramientas de Optimización



- Lenguajes de programación de propósito general
  - C++, Java, Python, . . .
- Solvers para Programación Matemática (LP, ILP, MILP)
  - IBM ILOG CPLEX Optimizer, Gurobi
- Solvers para Programación con Restricciones (CSP)
  - IBM ILOG CP Otimizer, MiniZinc
- Frameworks de metaheurísticas
  - JCLEC, jMetal, Open Opt4j, ParadisEO/EO, . . .

Contenido, calendario y profesores del curso



17/03	Ramiro Varela	(Introducción)	ramiro@uniovi.es
18,19/03	María R. Sierra	(Problemas de scheduling)	sierramaria@uniovi.es
20/03	Ramiro Varela	(Búsqueda heurística)	ramiro@uniovi.es
21/03	Miguel A. González	(Algoritmos evolutivos)	mig@uniovi.es
24/03	Jorge Puente	(Algoritmos evolutivos)	puente@uniovi.es
25/03	Carlos Mencía	(Programación con restricciones)	menciacarlos@uniovi.es
26/03	Raúl Mencía	(Programación con restricciones)	menciaraul@uniovi.es
27/03	Pablo Barredo	(Problema: scheduling en cloud computing)	UO237136@uniovi.es
	Jesús Quesada	(Problema: corte de piezas, prog. genética)	quesadajesus@uniovi.es
31/03	Sezin Afsar	(Problema: gestión de hidrógeno)	afsarsezin@uniovi.es
	Jorge Puente	(Problema: control de drones)	puente@uniovi.es