UNIVERSIDAD DE GRANADA

Modelos de Decisión y Optimización
Grupo de Investigación - Universidad de Granada

Máster Universitario Oficial en Ciencia de Datos e Ingeniería de Computadores

Técnicas de Soft Computing para Aprendizaje y Optimización

Redes Neuronales y Metaheurísticas, Programación Evolutiva y Bioinspirada

David A. Pelta

Grupo de Trabajo en Modelos de Decisión y Optimización

DECSAI

Datos de Contacto

- E-mail: *dpelta@decsai.ugr.es*
- Página web: *https://wpd.ugr.es/~dpelta/wordpress/*
- Despacho Nº 21, 4ta planta ETSI Informática

Contenidos

1. Conceptos Básicos de Optimización y Búsqueda
2. Heurísticas y Metaheurísticas basadas en trayectorias

Problema de Optimización

Una instancia de un problema de optimización es un par (S, f) y consta de:

- Un conjunto de variables $X = (x_1, x_2, \dots, x_n)$
- Dominios de las variables $(D_1, D_2, \dots, D_n)$
- un conjunto de restricciones entre las variables
- una función objetivo

$$f: D_1 \times D_2 \times \cdots \times D_n \rightarrow \mathcal{R}$$

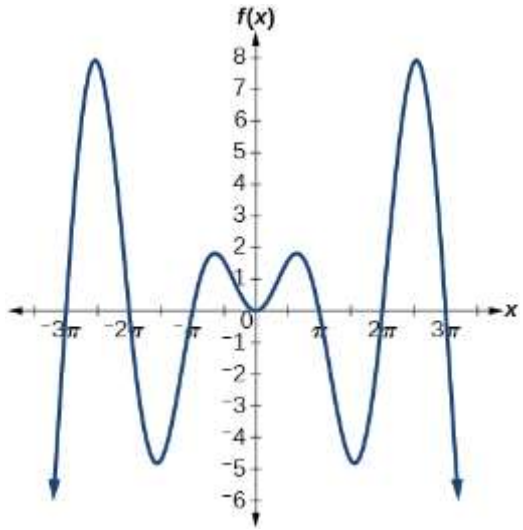
El conjunto de todas las asignaciones factibles es

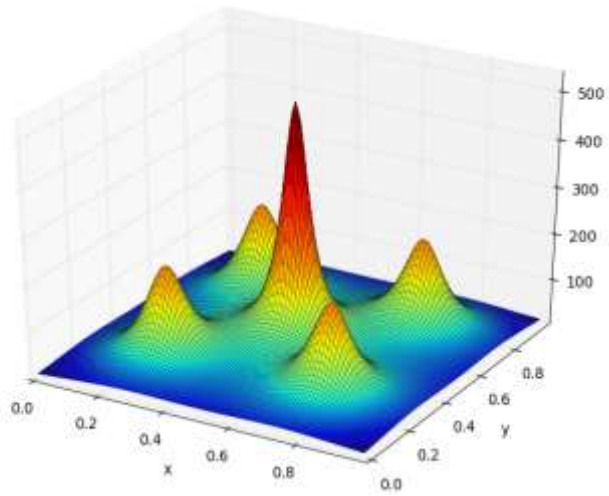
$$S = \{s \mid s = (x_1 = v_1, x_2 = v_2, \dots, x_n = v_n)\}$$

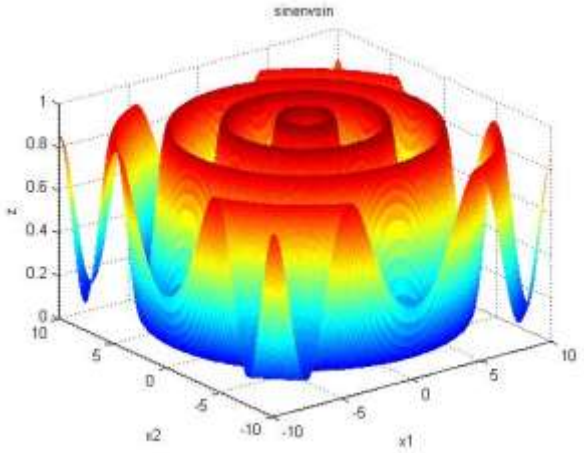
con $v_i \in D_i, i = 1, 2, \dots, n$ y s satisface las restricciones.

Problema de Optimización

- S recibe el nombre de espacio de búsqueda
- Resolver el problema: encontrar una solución (una asignación de valores a variables) que minimice/maximice la función objetivo.
- Si existe $s^* \in S, \mid f(s^*) < f(w)\ \forall\ w \in S$, entonces s^*es el óptimo de (S, f)







¿ Dónde surgen estos problemas ?

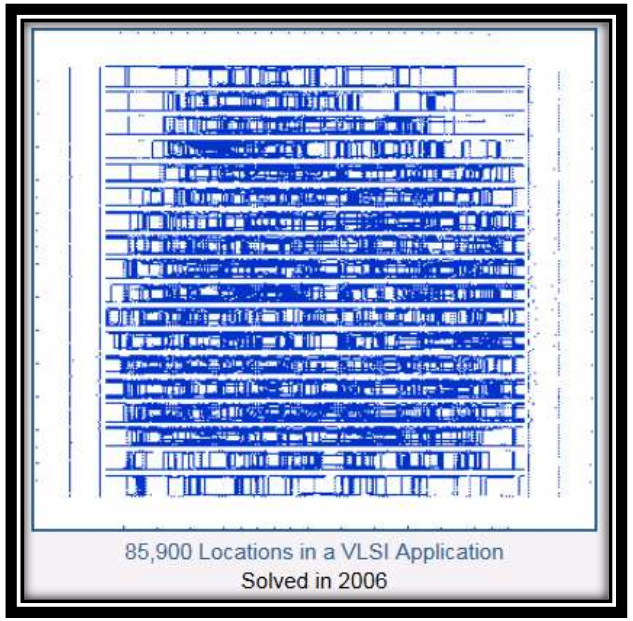
A compendium of NP optimization problems
Pierluigi Crescenzi, and Viggo Kann
http://www.nada.kth.se/~viggo/problemlist/compendium.html

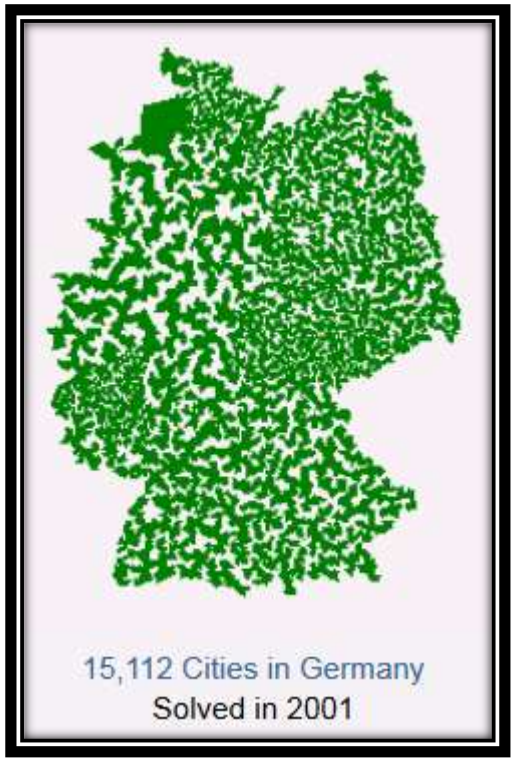
Ejemplos

El problema del viajante de comercio

Resolución de forma óptima

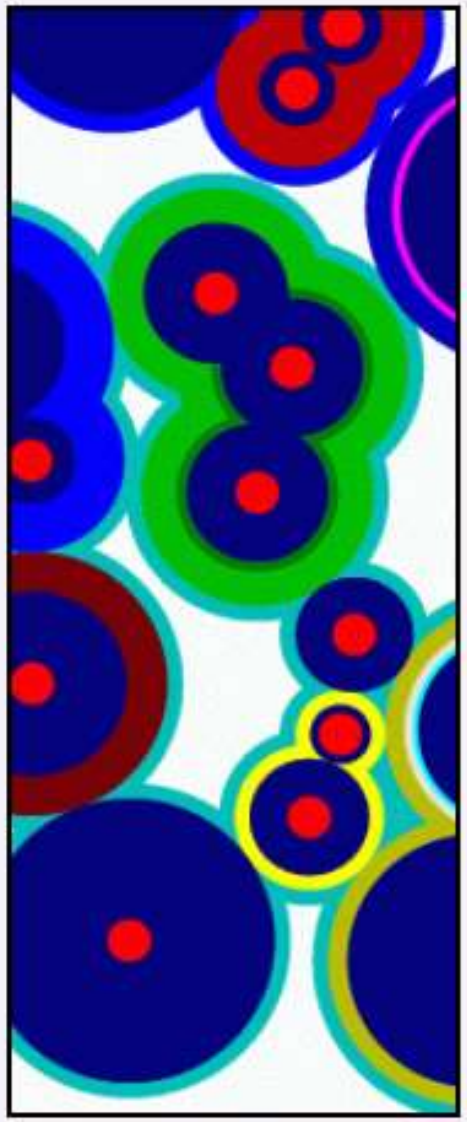
Year	Research Team	Size of Instance	Name
1954	G. Dantzig, R. Fulkerson, and S. Johnson	49 cities	dantzig42
1971	M. Held and R.M. Karp	64 cities	64 random points
1975	P.M. Camerini, L. Fratta, and F. Maffioli	67 cities	67 random points
1977	M. Grötschel	120 cities	gr120
1980	H. Crowder and M.W. Padberg	318 cities	lin318
1987	M. Padberg and G. Rinaldi	532 cities	att532
1987	M. Grötschel and O. Holland	666 cities	gr666
1987	M. Padberg and G. Rinaldi	2,392 cities	pr2392
1994	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	7,397 cities	pla7397
1998	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	13,509 cities	usa13509
2001	D. Applegate, R. Bixby, V. Chvátal, and W. Cook	15,112 cities	d15112
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook, and K. Helsgaun	24,978 cities	sw24798





¿ Cuánto cuesta resolver esto de manera óptima ?

TSP Lower Bounds: Certifying Tour Quality



Solving an instance of the traveling salesman problem (TSP) does not mean finding a good tour or even finding one that is better than any previously known. To solve a TSP we need to find an absolute shortest tour and to know indeed that no better tour is possible.

Given the complexity of the calculations it seems unlikely that we will ever be able to construct a proof that our Sweden tour is optimal that can be checked completely by hand, without the aid of a computer. Indeed, our Sweden TSP computation used 84.8 CPU years to verify that a best-possible tour had been found.

Our verification process included numerous checks to ensure that the computation was accurate and we describe the techniques used in the following pages.

- Geometric lower bounds with control zones
- Subtour elimination
- The cutting-plane method
- Local cuts
- Branch-and-cut
- Accuracy of computations

http://www.math.uwaterloo.ca/tsp/methods/opt/opt.htm

Resolución

Si debemos resolver este tipo de problemas usualmente, entonces es necesario emplear algoritmos aproximados o heurísticos.

Existen además otras razones que justifican su uso:

- Se busca una solución razonablemente buena.
- Ausencia de recursos computacionales/tiempo para calcular el optimo.
- el modelo es impreciso y por tanto, el óptimo de un modelo impreciso no tiene sentido.
- en ocasiones, se puede obtener el óptimo mediante algoritmos aproximados.

Algoritmos de resolución

Exactos

- Garantizan la obtención de la solución óptima.

Aproximados

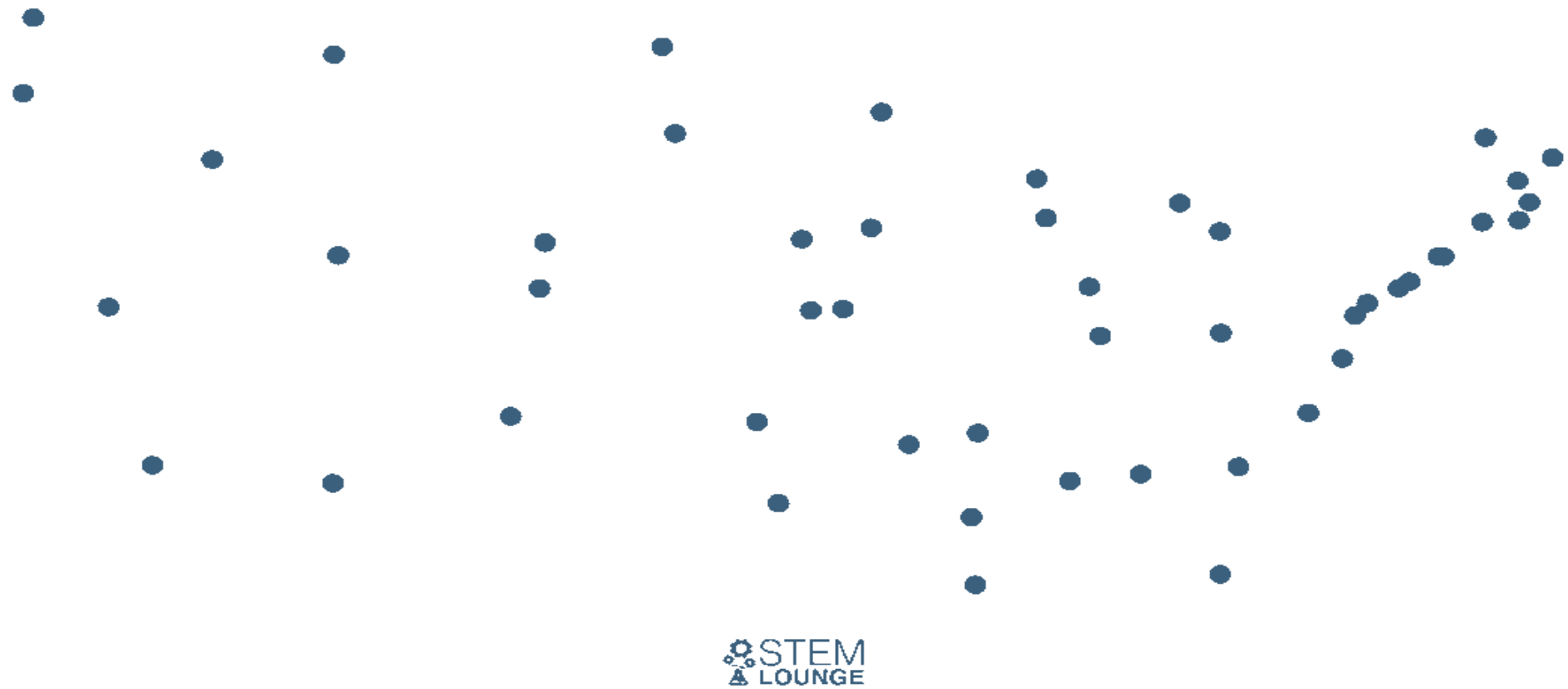
- Garantizan que la solución obtenida es como máximo, un % pero que la óptima.

Heurísticos

- Sirven para obtener soluciones factibles
- En ocasiones muy buenas.
- En ppio, no hay garantías de calidad.

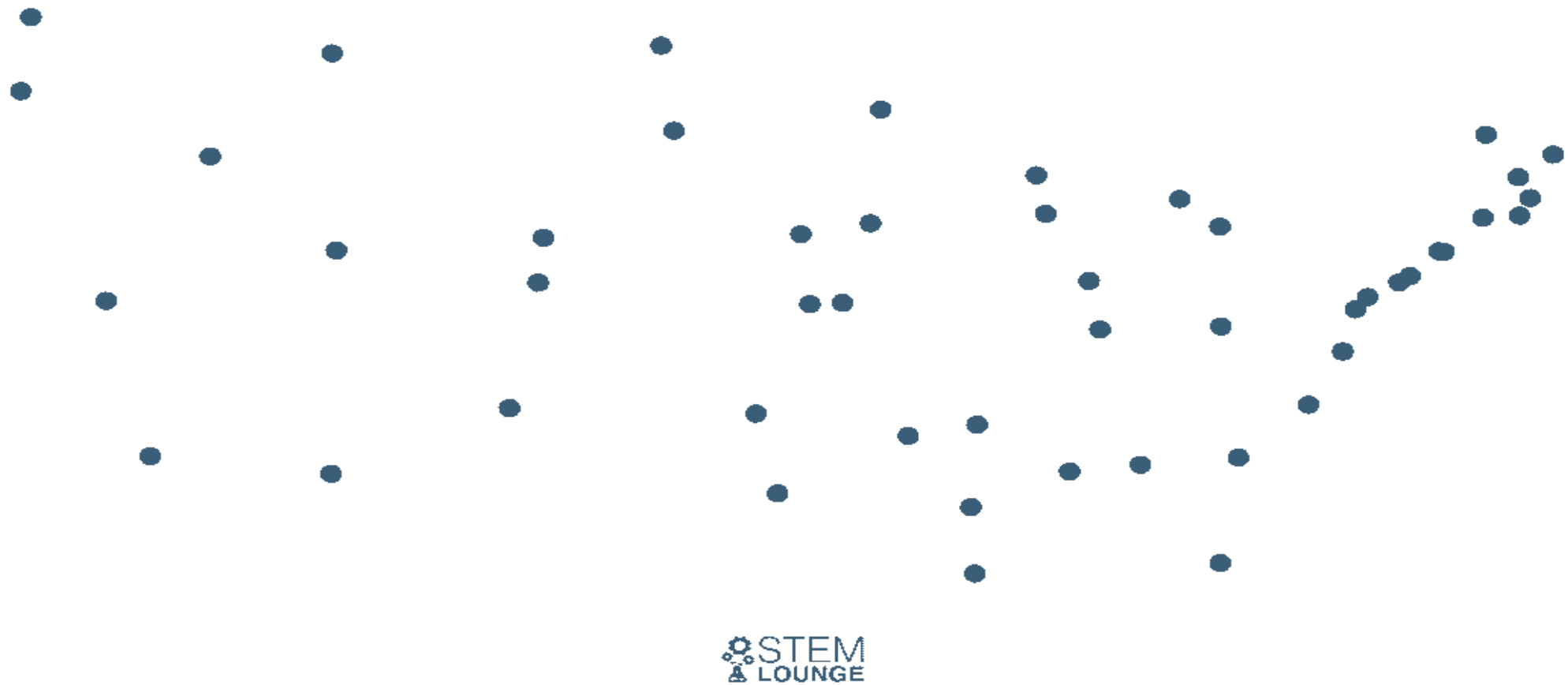
Heurísticas	Metaheurísticas
• Constructivas • De mejoras	• Basadas en trayectoria • Basadas en poblaciones

TSP como ejemplo



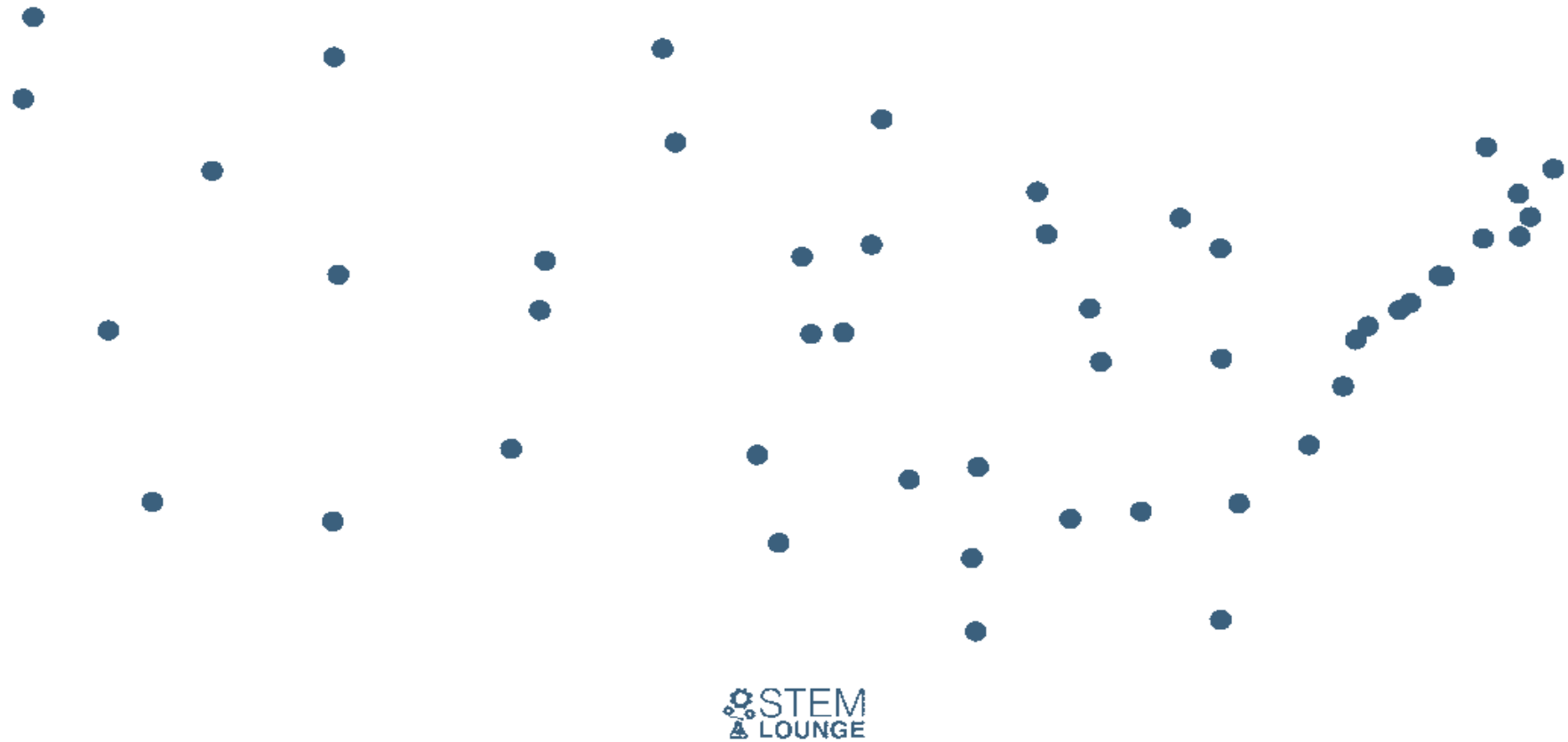
Vecino más cercano

https://stemlounge.com/animated-algorithms-for-the-traveling-salesman-problem/



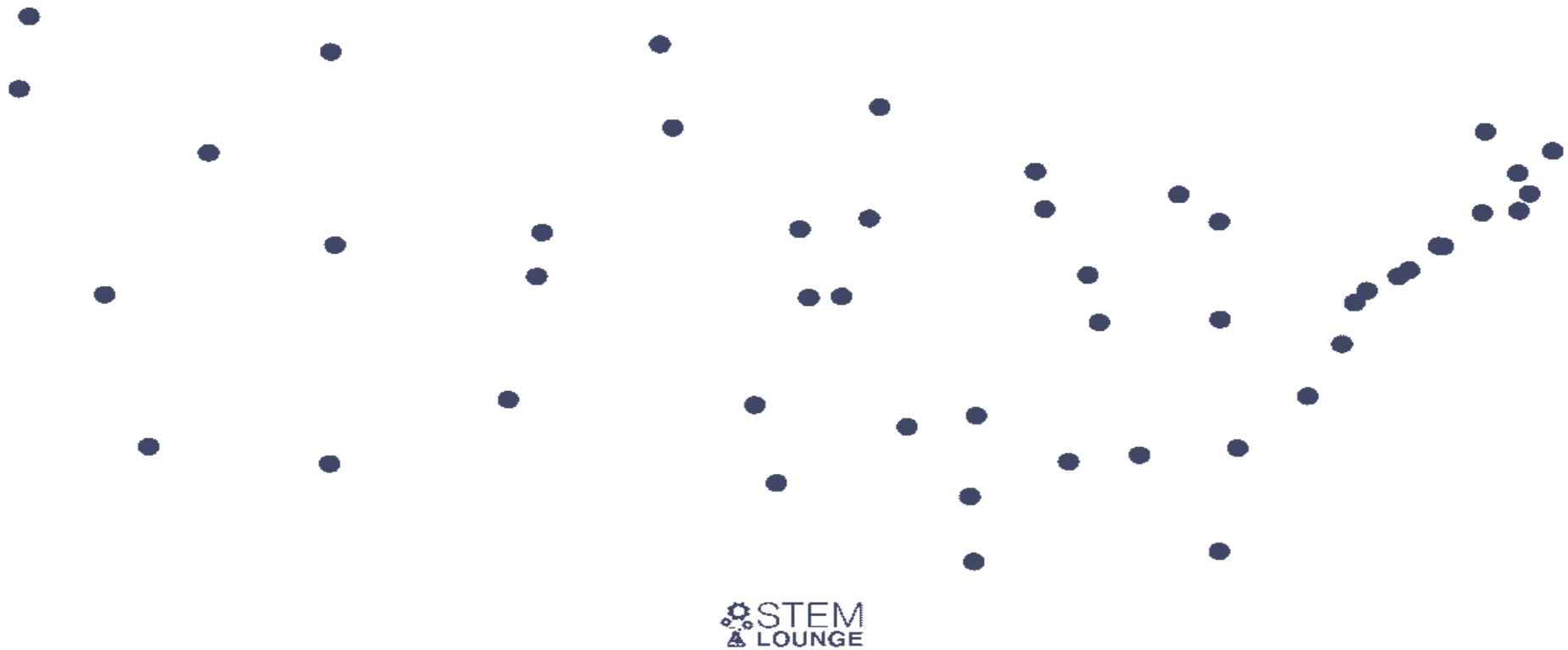
Inserción más cercana

https://stemlounge.com/animated-algorithms-for-the-traveling-salesman-problem/



Inserción más económica

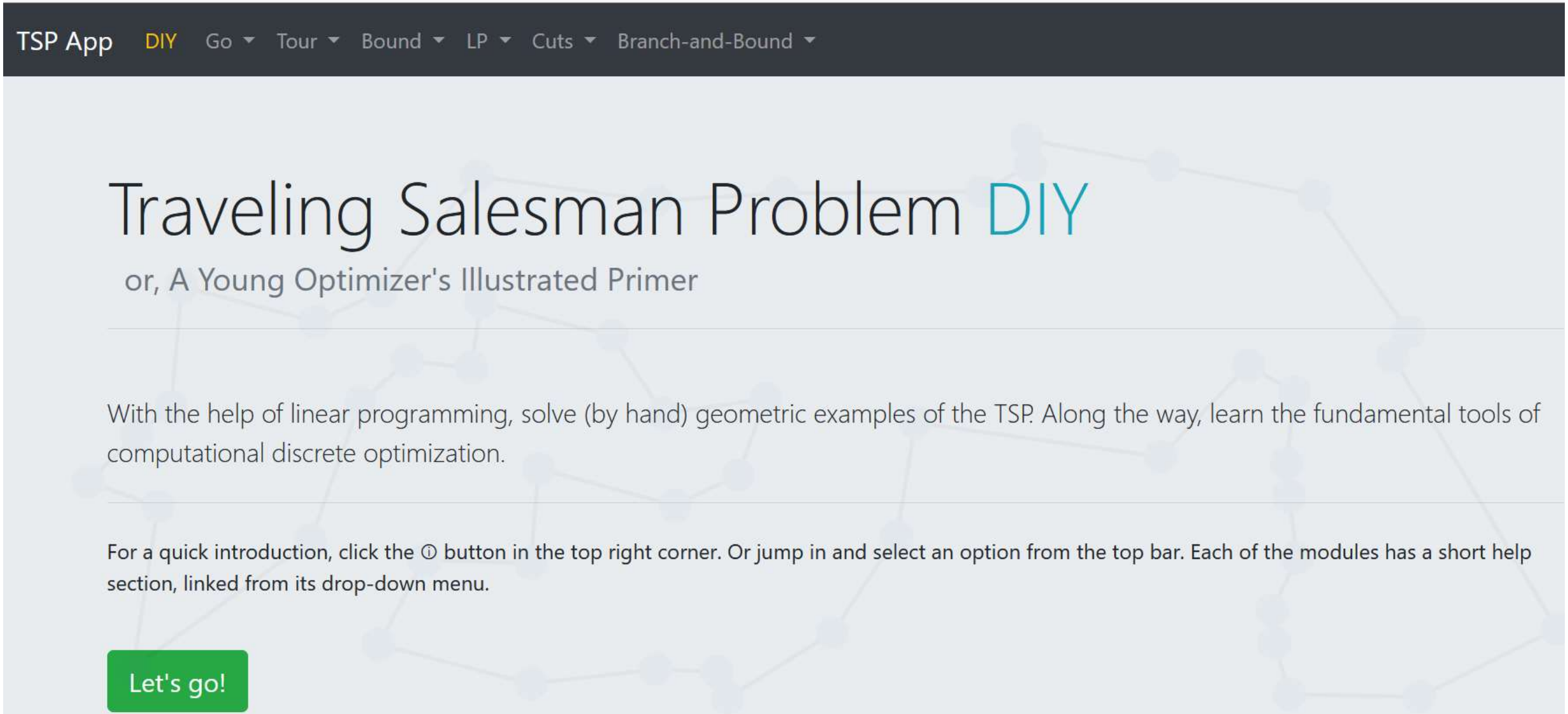
https://stemlounge.com/animated-algorithms-for-the-traveling-salesman-problem/



Lin-Kernighan

https://stemlounge.com/animated-algorithms-for-the-traveling-salesman-problem/

https://www.math.uwaterloo.ca/tsp/app/diy.html#



https://tspvis.com/

TSPVIS

Visualize algorithms for the traveling salesman problem. Use the controls below to plot points, choose an algorithm, and control execution. (Hint: try a construction alogorithm followed by an improvement algorithm)

CURRENT BEST:	21445.85	KM
EVALUATING:		KM
RUNNING FOR:	38	S

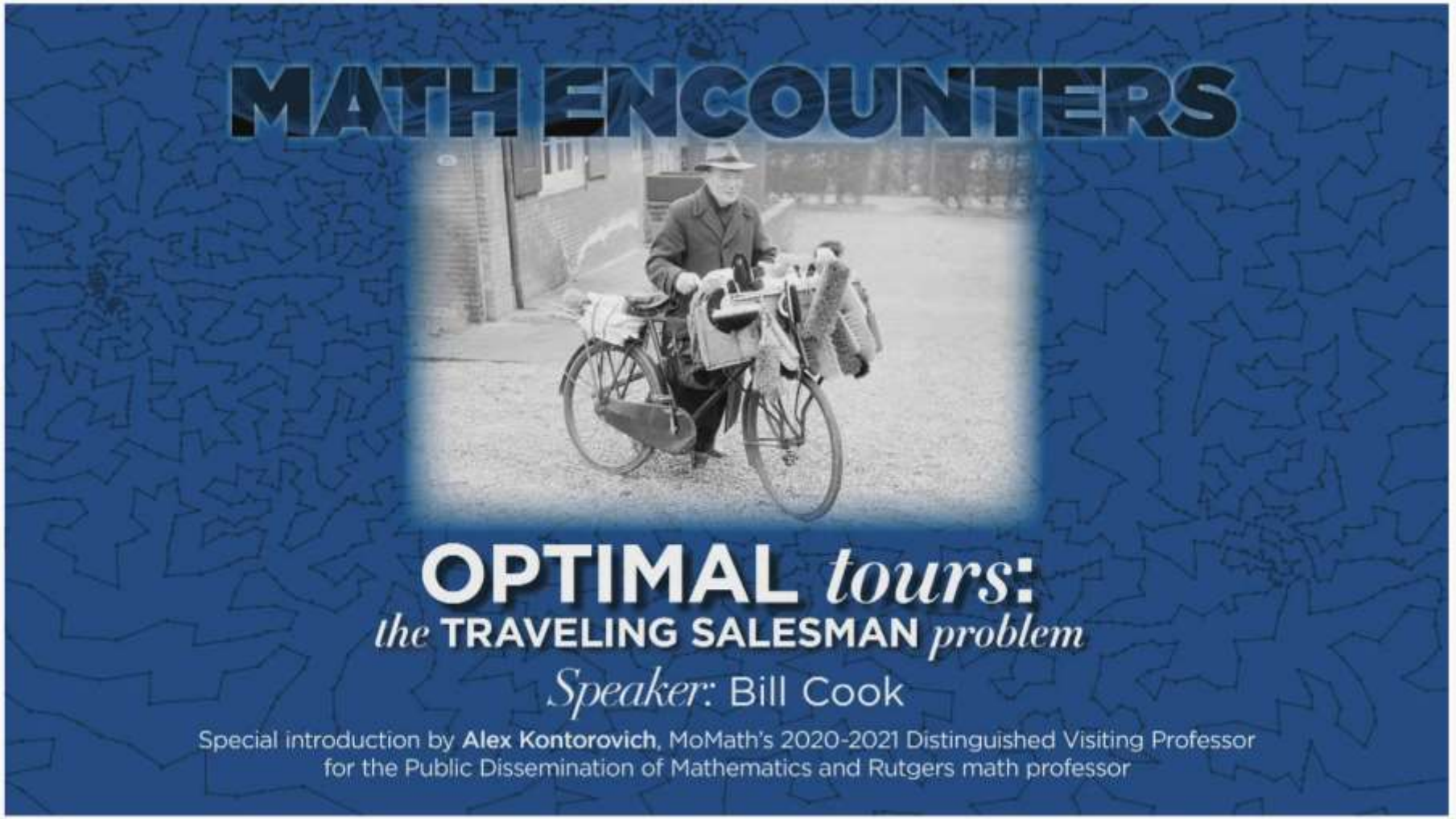
ALGORITHM

Branch and Bound (Cost)

CONTROLS

DELAY

Para saber más del TSP



Math Encounters: "Optimal Tours: The Traveling Salesman Problem" with Bill Cook on August 4, 2021

https://www.youtube.com/watch?v=tChnXG6ulyE

www.math.uwaterloo.ca/tsp/concorde/index.html

TSP

Concorde Home

- Home
- > Concorde Home
- Windows GUI
- Benchmarks
- Documentation
- Downloads
- Contact Info

Concorde TSP Solver

Concorde is a computer code for the symmetric traveling salesman problem (TSP) and some related network optimization problems. The code is written in the ANSI C programming language and it is available for academic research use; for other uses, contact William Cook for licensing options.

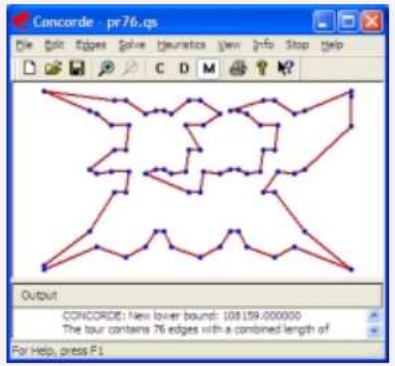
Concorde's TSP solver has been used to obtain the optimal solutions to the full set of 110 TSPLIB instances, the largest having 85,900 cities.

The Concorde callable library includes over 700 functions permitting users to create specialized codes for TSP-like problems. All Concorde functions are thread-safe for programming in shared-memory parallel environments; the main TSP solver includes code for running over networks of UNIX workstations.

Concorde now supports the QSopt linear programming solver. Executable versions of concorde with qsopt for Linux and Solaris are available

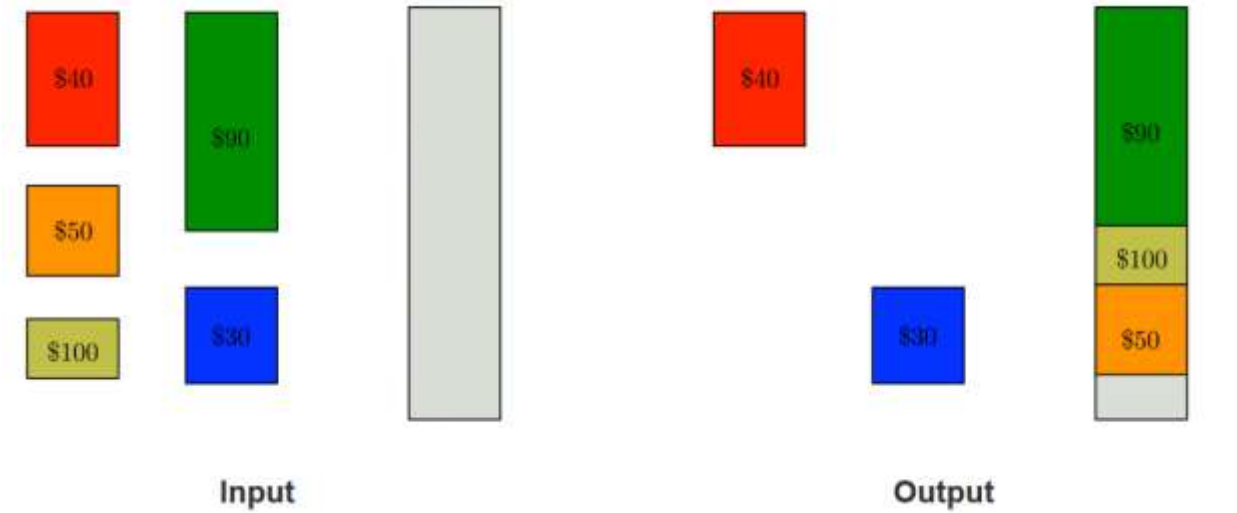
Hans Mittelmann has created a NEOS Server for Concorde, allowing users to solve TSP instances online.

Pavel Striz wrote a nice package for creating LaTeX images from the solution files produced by Concorde.



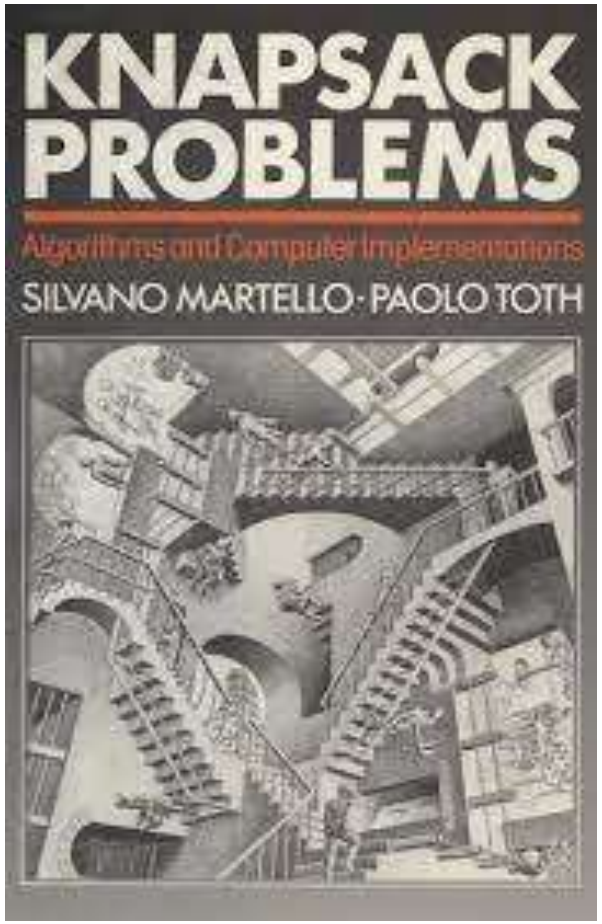
- Concorde for Windows
- Benchmark Results
- User Documentation for Source Code
- Downloading Concorde

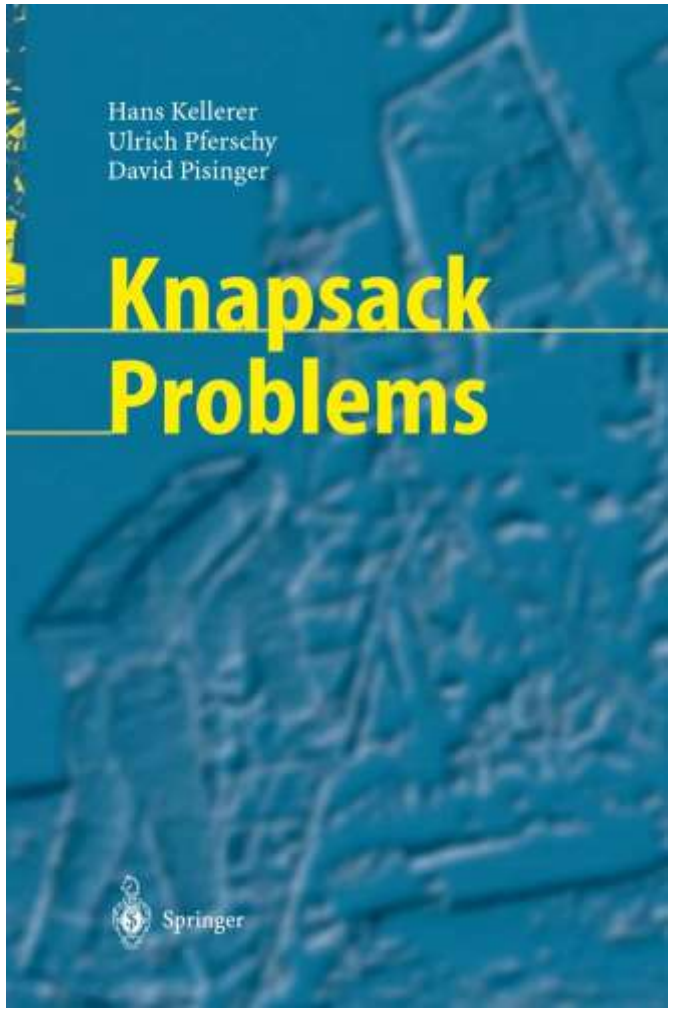
Heurísticas para otros problemas



Seleccionar un conjunto de ítems para agregar a una mochila de cierta capacidad C. Cada ítem *j* tiene asociado un peso w_j y un beneficio p_j. Se desea maximizar el beneficio sin sobrepasar la capacidad.

$$\max \sum x_j \times p_j, x_j \in \{0,1\}, j = 1 \dots n$$
$$\text{tal que } \sum w_j \times x_j \leq C, j = 1 \dots n$$



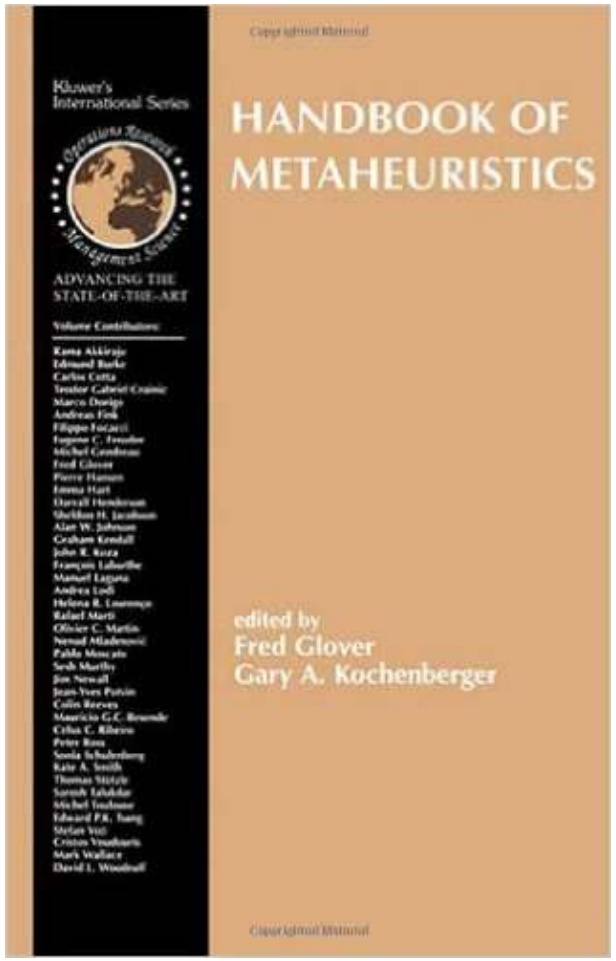


https://link.springer.com/book/10.1007/978-3-540-24777-7

Metaheurística

- Proceso maestro iterativo que guía y modifica las operaciones de heurísticas subordinadas para producir, de forma eficiente, soluciones de alta calidad.
- En cada iteración, puede manipular una solución (completa o incompleta) o un conjunto de soluciones.
- Las heurísticas subordinadas pueden ser procedimientos de alto o bajo nivel, o simplemente una búsqueda local o método constructivo.

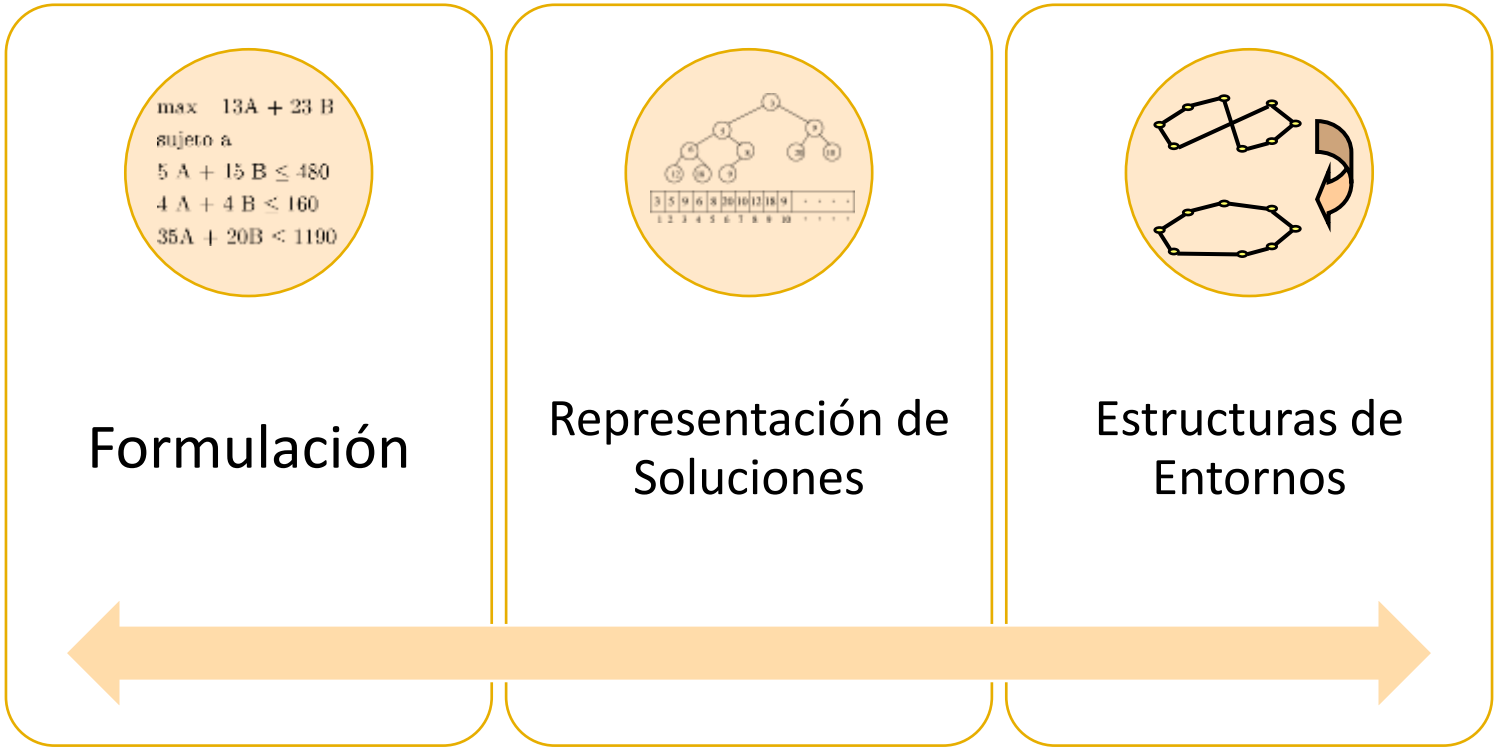
Metaheurística



". . . these methods have over time also come to include any procedure for problem solving that employs a strategy for overcoming the trap off local optimality in complex solution spaces, especially those procedures that utilize one or more neighborhood structures as a means of defining admissible moves to transition from one solution to another, or to build or destroy solutions in constructive and destructive processes."

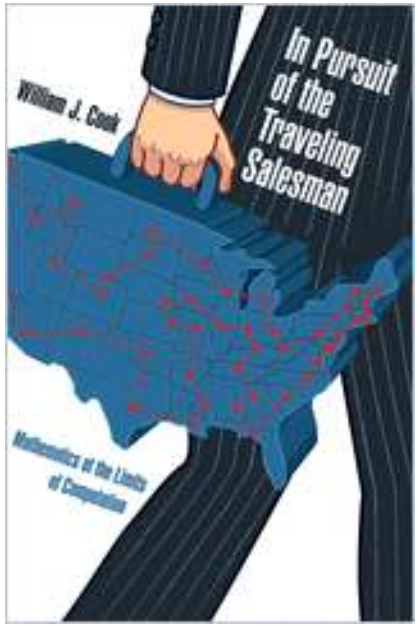
Pero antes….

Es necesario identificar algunos aspectos básicos antes de abordar la implementación de una heurística para un problema particular.



Problema del Viajante de Comercio

Dado un conjunto de puntos (ciudades) $\{1,2,3,\ldots,n\}$ y una matriz C, donde $C_{\{ij\}}$ indica la distancia entre la ciudades i,j se pretende encontrar el circuito de longitud mínima que pase por todos los puntos una sola vez.



El Problema de la Mochila

Seleccionar un conjunto de items para agregar a una mochila de cierta capacidad C. Cada item *j* tiene asociado un peso w_j y un beneficio p_j.

Se desea maximizar el beneficio sin sobrepasar la capacidad.

$$\max \sum x_j \times p_j, x_j \in \{0,1\}, j = 1 \dots n$$

$$\text{tal que } \sum w_j \times x_j \leq C, j = 1 \dots n$$

Problemas de Localización

En un Problema de Localización se pretende determinar la ubicación de ciertos servicios de forma que, según unos determinados criterios y cumpliendo unas determinadas restricciones, éstos satisfagan de forma óptima las necesidades demandadas por los usuarios.

Servicio debe entenderse en un sentido amplio, ya que por él se entiende desde un punto hasta una estructura (camino, circuito, árbol, …)

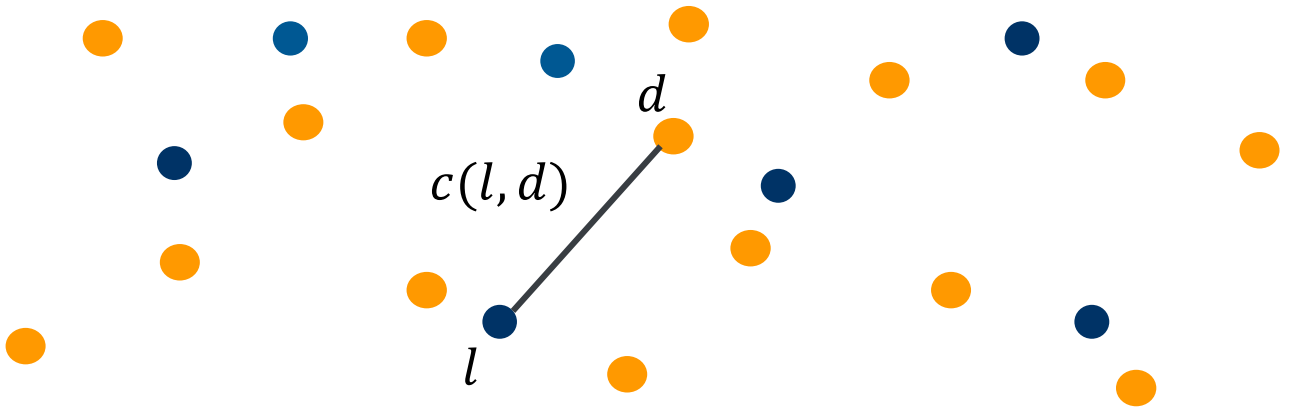
Algunos problemas de localización

D, conjunto de puntos de demanda (puntos naranjas)

L, conjunto de posibles localizaciones (puntos azules)

$c(l,d), l \in L, d \in D$, coste en que se incurre si se atiende el punto de demanda d desde la localización l

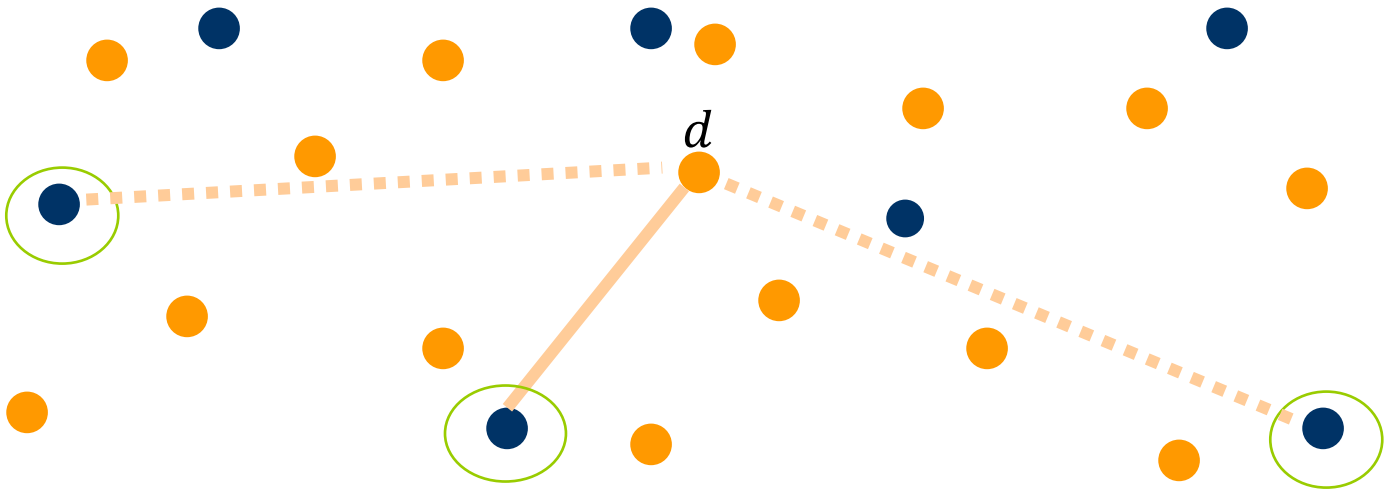
$$S = \{X \subset L \mid |X| = p\}$$



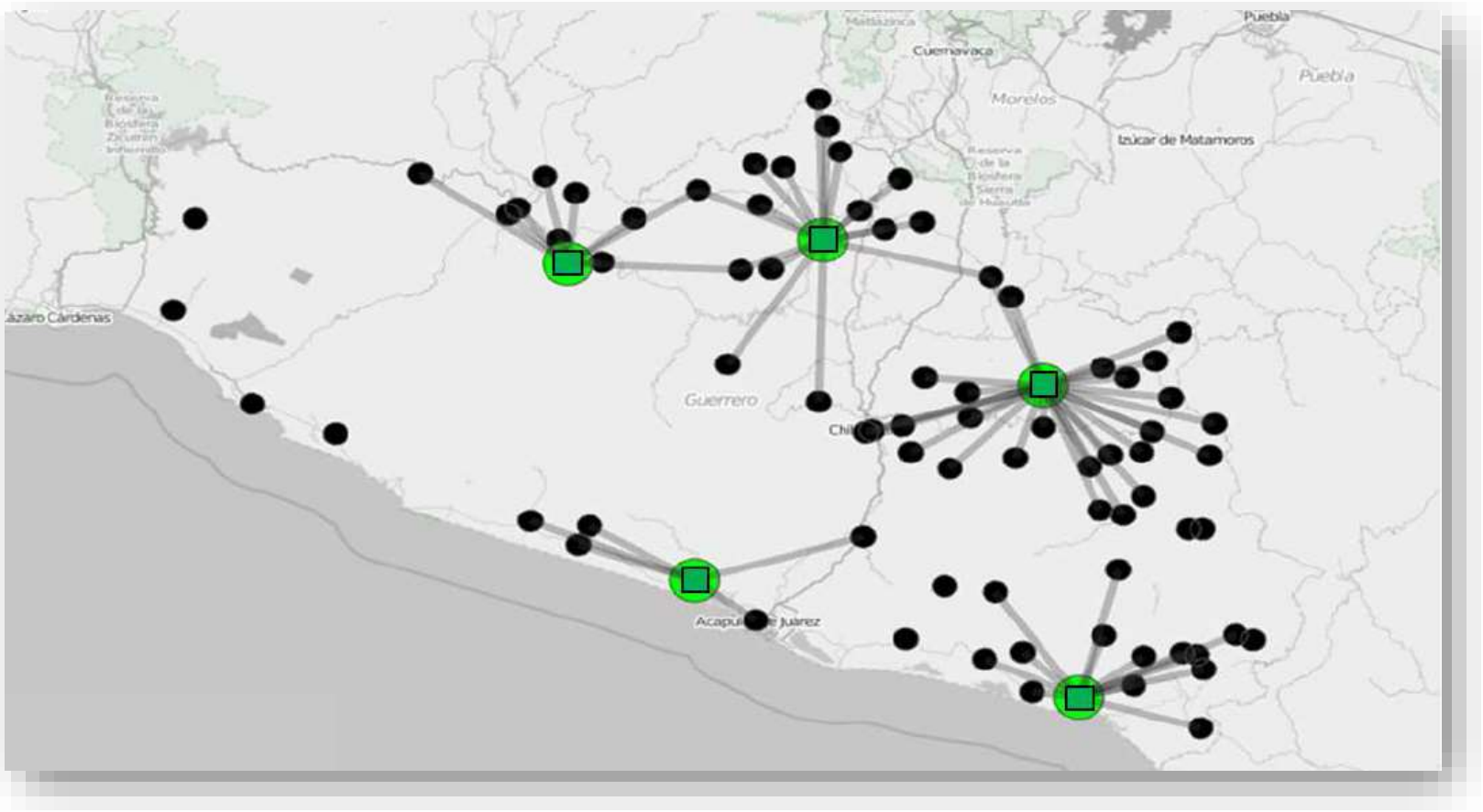
Algunos problemas de localización

Cada punto de demanda d se atiende desde su localización más cercana

$$c(X,d) = \min_{\{l \in X\}} c(l,d)$$



Algunos problemas de localización



Problemas “Continuos”

$$F_8(\mathbf{x}) = -20\exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} z_i^2}) - \exp(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi z_i)) + 20$$

$$F_9(\mathbf{x}) = \sum_{i=1}^{D}(z_i^{\,2} - 10\cos(2\pi z_i) + 10)$$

$$F_{11}(\mathbf{x}) = \sum_{i=1}^{D}(\sum_{k=0}^{k\max}[a^k \cos(2\pi b^k (z_i + 0.5))]) - D\sum_{k=0}^{k\max}[a^k \cos(2\pi b^k \cdot 0.5)]$$

Parámetros de un controlador, de un conjunto de ecuaciones diferenciales, series temporales, problemas inversos, etc.

Formulación

Un problema consta de:

- el conjunto de alternativas o soluciones al problema,
- posibles restricciones,
- *la función objetivo o de costo*, y
- el criterio que, por medio de la función objetivo, permite establecer una comparación entre las diferentes alternativas.

Representación

- Las soluciones se pueden codificar o representar de diferentes formas.
- La representación de las soluciones condiciona la forma en que se calcula/implementa la función objetivo.
- Debe resaltarse que la representación usada determina el espacio de soluciones y su tamaño.
- Además, la representación de las soluciones tiene un impacto directo en la eficiencia de una heurística.

Representación de soluciones

Problema del Viajante

Representación directa o de permutación: una solución se codifica por medio de un vector de tamaño n. Si en la i-ésima posición aparece el valor j, esto significa que la i-esima ciudad visitada es la ciudad j.

Representación: (2 4 8 3 9 7 1 5 6)

Solución: 2 - 4 - 8 - 3 - 9 - 7 - 1 - 5 - 6

Representación de soluciones

Problema del Viajante

Representación de adyacencia: una solución se codifica por medio de un vector de tamaño n. Si en la i-ésima posición aparece el valor j, esto significa que desde la ciudad i se visita la ciudad j.

Representación: (2 4 8 3 9 7 1 5 6)

Solución: 1 - 2 - 4 - 3 - 8 - 5 - 9 - 6 - 7

Representación de soluciones

Problema del Viajante

Representación ordinal: Se supone que las ciudades se etiquetan con los números del 1 al n en algún orden. Esto suministra una lista

$$L = \{l_1, l_2, \dots, l_n\}.$$

Cada circuito se representa por medio de un vector de tamaño n. La i-ésima posición de este vector toma un valor entre 1 y $n - i + 1$ e indica la posición que ocupa la ciudad l_i en el circuito, tras eliminar $l_1, l_2, \dots, l_{\{i-1\}}$, de L.

L = {1 2 3 4 5 6 7 8 9}

Solución: 1 - 2 - 4 - 3 - 8 - 5 - 9 - 6 - 7

Representación: (1 1 2 1 4 1 3 1 1)

Representación de soluciones

Problema de la mochila

Representación binaria: una solución se codifica por medio de un vector de tamaño n. Si en la i-ésima posición aparece el valor 1, esto significa que el item i está seleccionado (0 en caso contrario)

Representación: (0 0 1 0 1 1 0 0 0 0)

Solución: 3 - 5 - 6

Representación de soluciones

Problema de Localización

Codificación binaria: n-upla binaria con exactamente p unos; un 1 en la i-ésima posición indica que la i-ésima localización está presente en la solución.

(1 0 0 1 0 0 1 0 1 0 0 0 0 1)

Codificación indexal: p-upla (ordenada o no) de los índices de las localizaciones presentes en la solución

(1 4 7 9 14)

Representación de soluciones

Propiedades deseables de una representación:

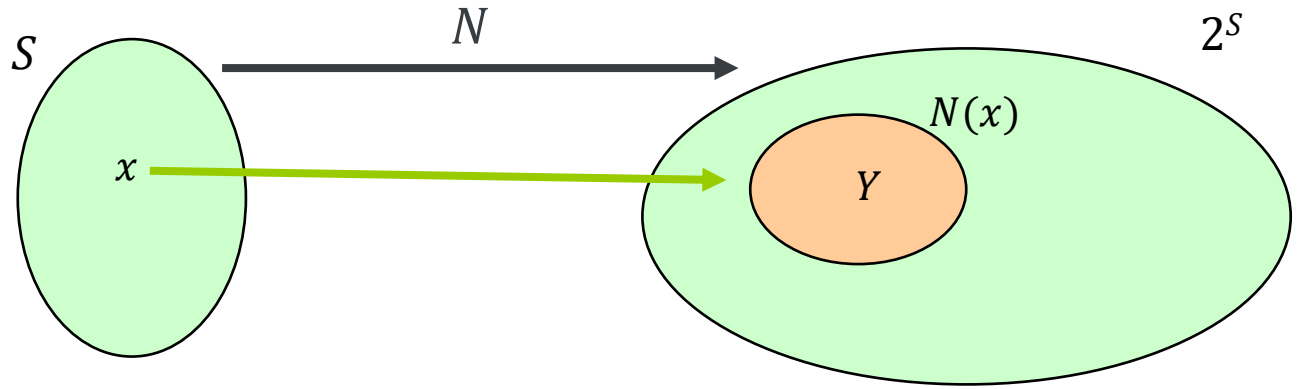
- Una solución $\Rightarrow$ una única representación.
- Todas las soluciones deben admitir, al menos, una representación. En particular, la solución óptima.
- Las anteriores propiedades no siempre se cumplen en una representación particular.

Función Objetivo

- Una vez establecida la representación / codificación de las soluciones, es necesario determinar la forma en que se evalúa la calidad de las mismas.
- En general, la calidad de una solución se mide a través de una o varias funciones objetivo.
- Podrían usarse otras medidas: probabilidad de obtener una solución mejor, grado de cumplimiento de las restricciones, etc.
- ¿Cómo evaluar aspectos como: confortabilidad, suavidad, jugabilidad, “espectacularidad”, etc. ?

Estructura de Entorno

- Dado el problema (S, f), una estructura de entorno es una función $N: S \rightarrow 2^S$ que asocia a cada solución x $\in S$ un conjunto $N(x) \subseteq S$ *de soluciones cercanas a x en algún sentido*.
- El conjunto $N(x)$ se llama entorno de la solución x y cada y $\in N(x)$, solución vecina de x.



Estructura de Entorno

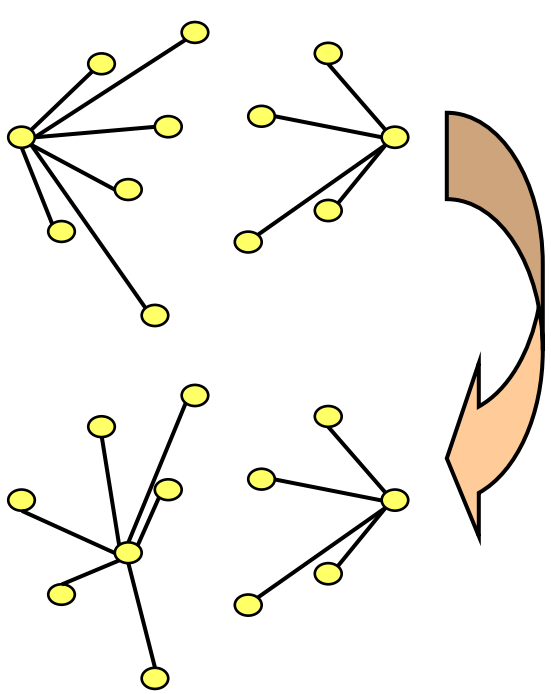
Generalmente, las estructuras de entorno se definen a través de movimientos que, aplicados sobre una solución, suministran otras.

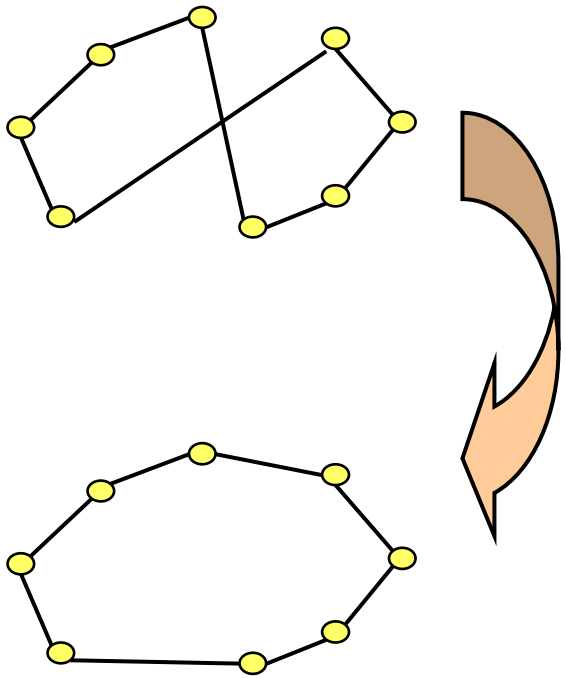
Por ejemplo

- para *problemas de localización* pueden considerarse los movimientos de apertura, cierre e intercambio de servicios, y los de reasignación y traslado de puntos de demanda.
- Para el *problema del viajante*: intercambiar el orden de dos ciudades.
- Para la *mochila*: agregar/quitar elementos.

Estructura de Entorno

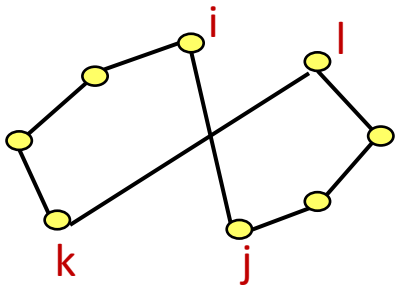
Estructura del k-intercambio: soluciones vecinas de una dada son aquellas que se obtienen al sustituir k elementos (vértices, aristas,…) presentes en la solución por k elementos no presentes en la misma.



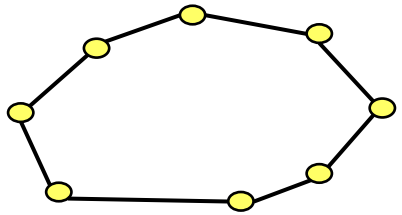


Estructura de Entorno

- La aplicación de uno de los movimientos disponibles a una solución actual suministra una nueva solución que debe ser evaluada.
- En ocasiones, la evaluación de una solución es un proceso computacionalmente costoso, por lo que la eficiencia de una heurística se incrementa si dicha evaluación se realiza eficientemente.
- Concepto de evaluación reducida



Solución actual



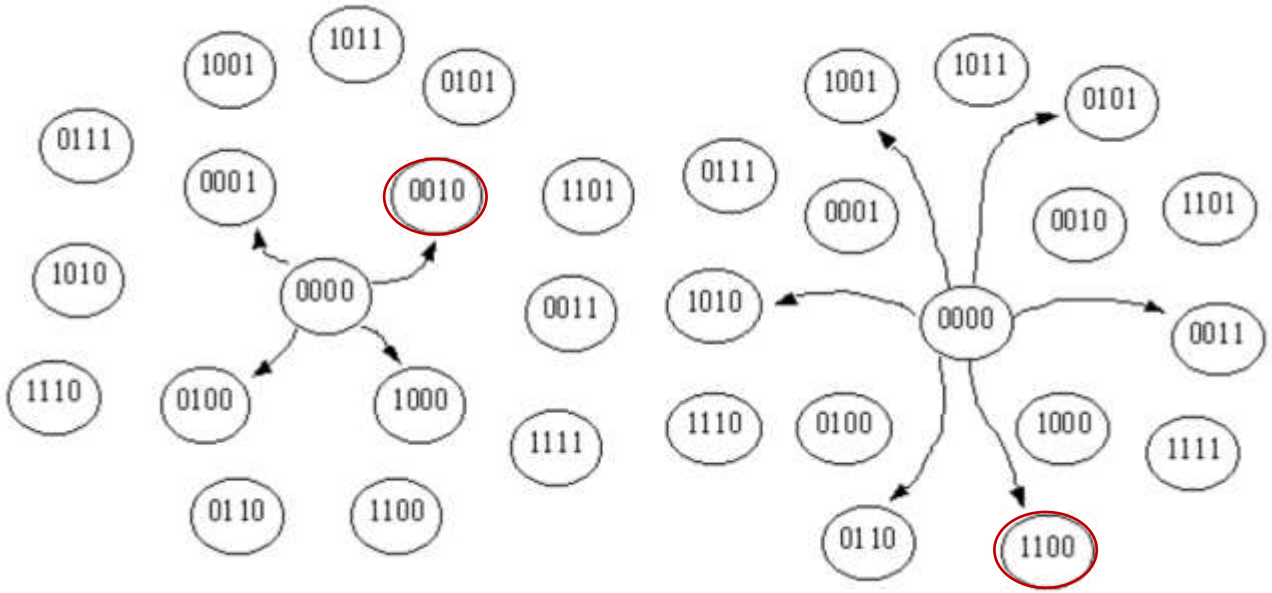
Nueva solución

```
F(Nueva solución) = F(Solución actual)
                    - D(i,j) - D(k,l)
                    + D(i,l) + D(k,j)
```

Concepto de mínimo local

Dada una estructura de entorno N, una solución $z \in N(x)$, es un mínimo local si para toda $y \in N(x)$, $f(z) < f(y)$

Un mínimo local en una estructura de entorno no tiene por qué ser mínimo local en otra estructura de entorno.



Sobre estas ideas básicas, podemos empezar a construir heurísticas

Esquemas Básicos

```
Proc. Busqueda Local:
Begin
  s = solucion-inicial();
  While ( mejorar(s) = SI) ) Do
    s = mejorar(s);
  endDo
  retornar(s);
End.
```

mejorar(x) retorna (si es posible), una solución $y \in N(x)$, t.q. $f(y) < f(x)$.

Dos estrategias:

1. **First:** mejorar retorna la primera solución que satisface la condición
2. **Best:** mejorar explora todo el vecindario y toma la mejor solución

Ejemplos

Encontrar la palabra clave de *n* símbolos.

- Representación
- Función objetivo
- Vecindario

https://bit.ly/3f7VPmm

Esquemas Básicos

Proc. Busqueda Local:

```
Begin
  s = solucion-inicial();
  While ( mejorar(s) = SI) ) Do
    s = mejorar(s);
  endDo
  retornar(s);
End.
```

Proc. Busqueda Local MultiStart:

```
Begin
  s = solucion-inicial();
  restart = 0;
  While ( restart<MAX ) Do
    While ( mejorar(s) = SI) ) Do
      s = mejorar(s);
    endDo
    If (s<best) Then
      best = s;
    endIf
    s = solucion-inicial();
    restart++;
  endDo
  retornar(best);
End.
```

Esquemas basados en Umbral

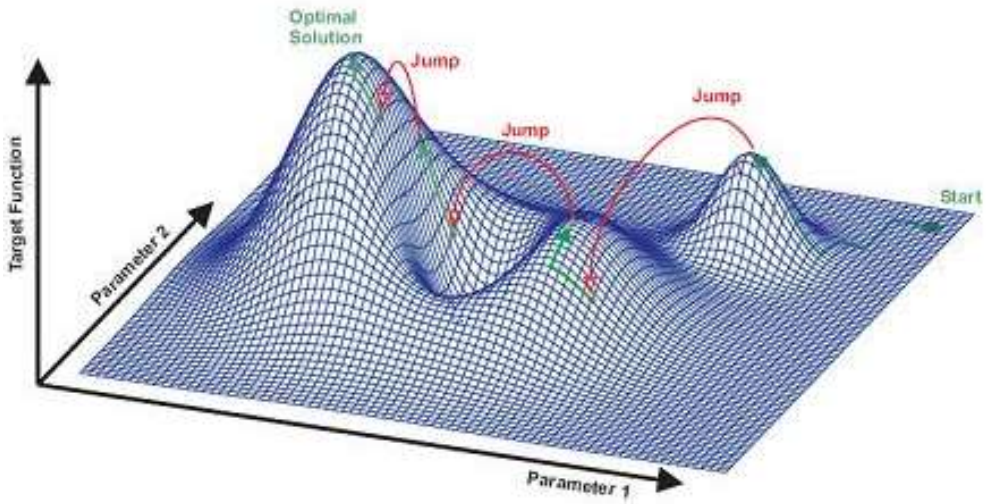
Algoritmos tipo “Simulated Annealing”

Procedure Algoritmo de Umbral:
Begin
  s_a = solucion-inicial();
  While (no-finalizacion) **Do**
    s_v = GenerarVecino(s_a);
    If $(f(s_v) - f(s_a)) < t_k$ **Then**
      $s_a = s_v$;
    endIf
    k=k+1;
  endDo
End.

$t_k$es el umbral. Varía según un “esquema de enfriamiento”.

Permite moverse a soluciones peores.

$$rand(0,1) < exp^{(f(x_{i+1})-f(x_i))/T}$$



Experimentando con Simulated Annealing

https://csvisualized.eschirtz.com/#/demos/ep01

https://www.netlogoweb.org

- Code Examples, HillClimbing
- Sample Models, Comp. Science, Unverified, Simulated Annealing

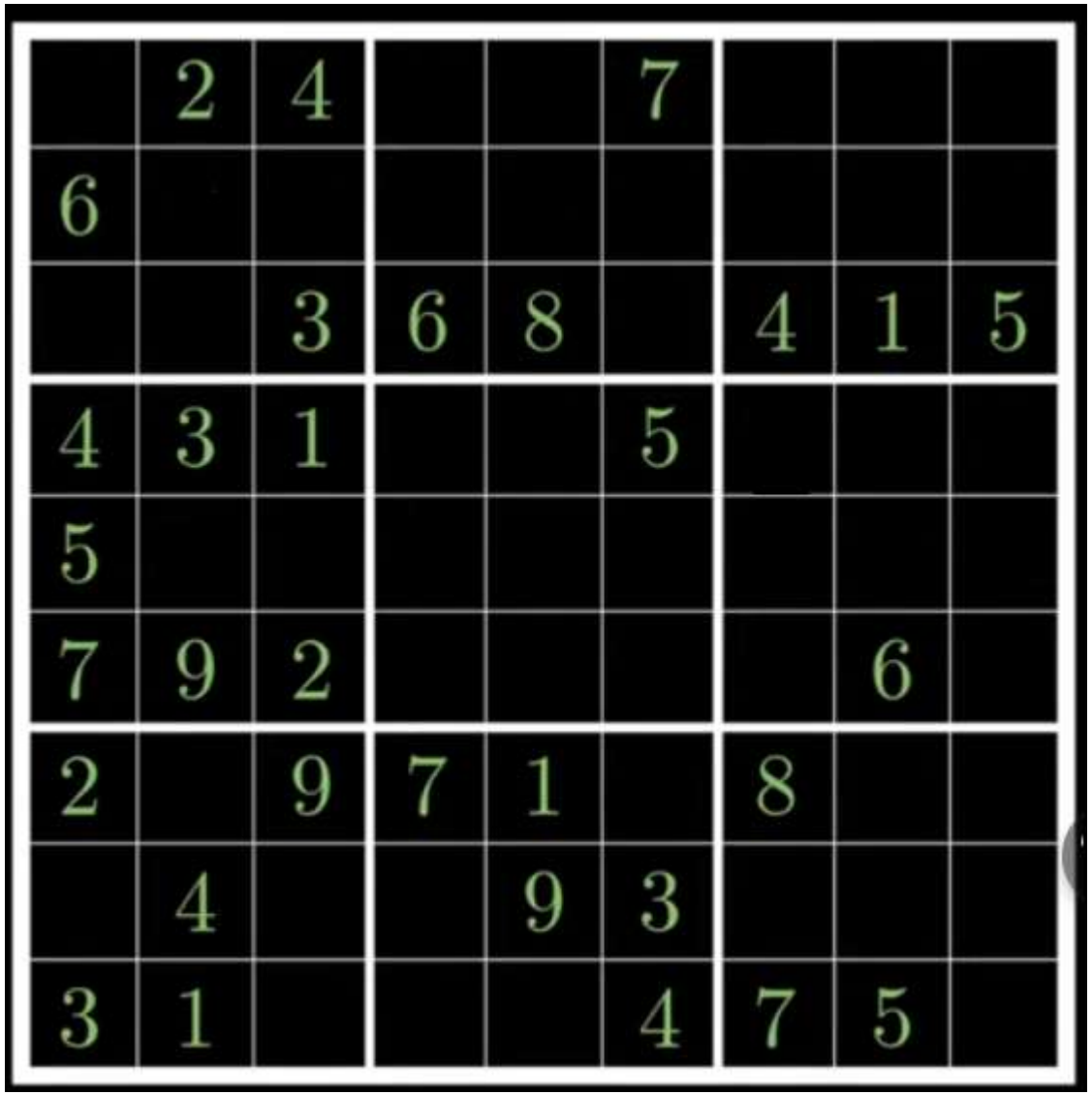
Ejemplos

Resolver SUDOKU

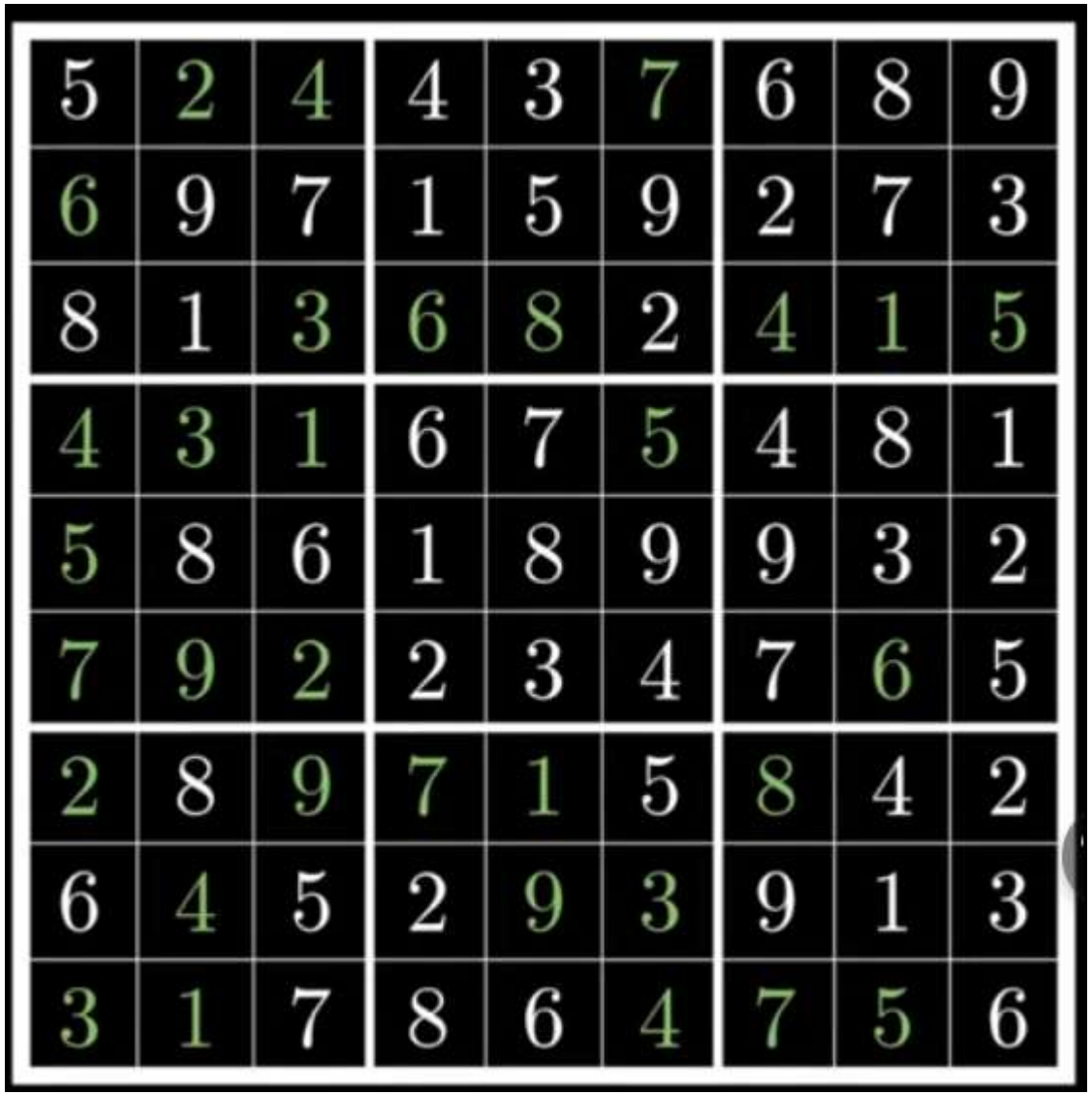
- Representación
- Función objetivo
- Vecindario

https://www.youtube.com/watch?v=FyyVbuLZav8

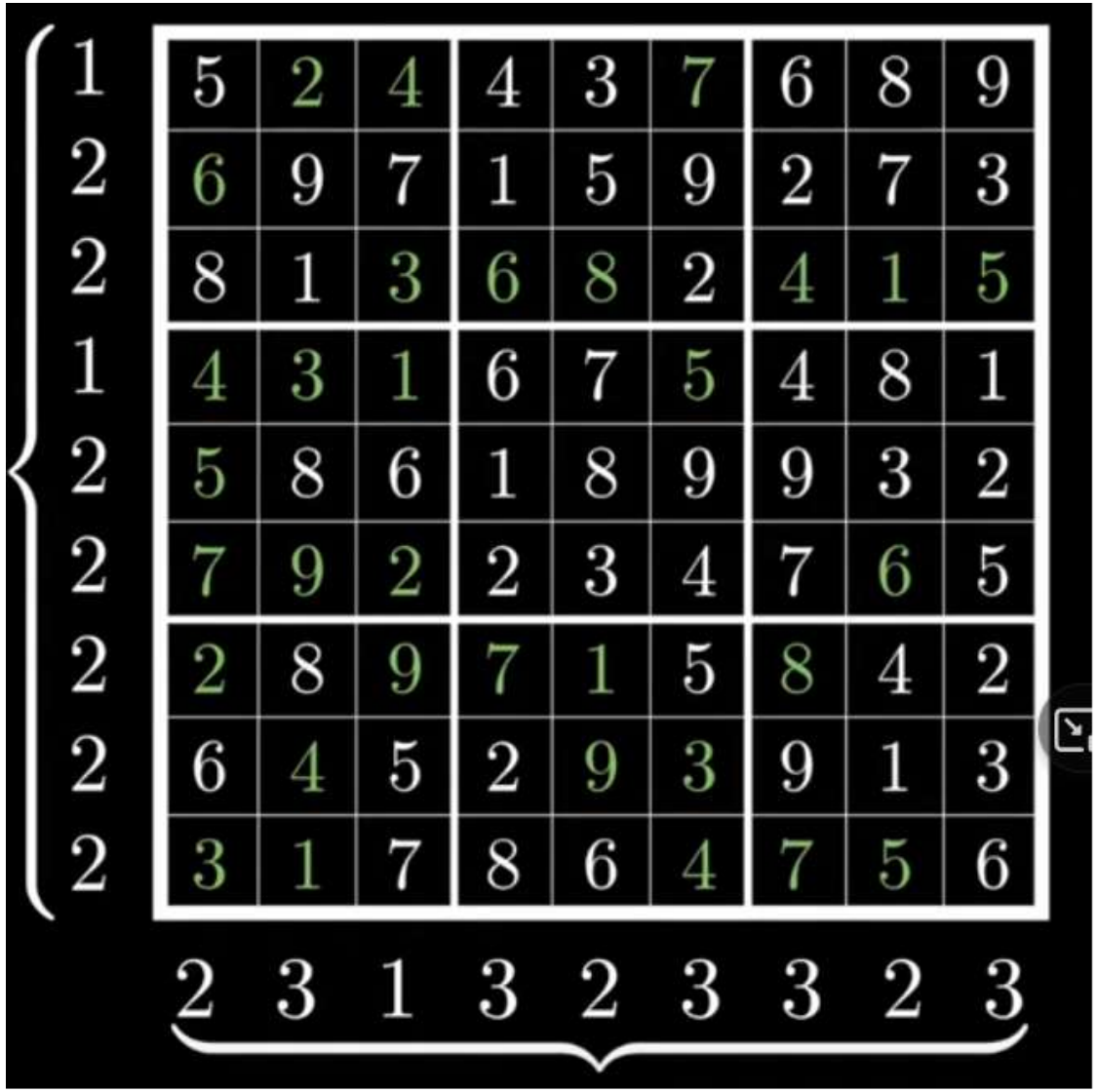
http://tpcg.io/AE1JX7



	2	4			7			
6								
		3	6	8		4	1	5
4	3	1			5			
5								
7	9	2					6	
2		9	7	1		8		
	4			9	3			
3	1				4	7	5	



5	2	4	4	3	7	6	8	9
6	9	7	1	5	9	2	7	3
8	1	3	6	8	2	4	1	5
4	3	1	6	7	5	4	8	1
5	8	6	1	8	9	9	3	2
7	9	2	2	3	4	7	6	5
2	8	9	7	1	5	8	4	2
6	4	5	2	9	3	9	1	3
3	1	7	8	6	4	7	5	6


1 5 2 4 4 3 7 6 8 9
2 6 9 7 1 5 9 2 7 3
2 8 1 3 6 8 2 4 1 5
1 4 3 1 6 7 5 4 8 1
2 5 8 6 1 8 9 9 3 2
2 7 9 2 2 3 4 7 6 5
2 2 8 9 7 1 5 8 4 2
2 6 4 5 2 9 3 9 1 3
2 3 1 7 8 6 4 7 5 6
2 3 1 3 2 3 3 2 3

Búsqueda Tabú (BT)

- Parte desde una solución inicial s_a, construye un entorno $N(s_a)$ y selecciona la mejor solución $s_b \in N(s_a)$.
- Así como un movimiento m permitió transformar $s_a \rightarrow s_b$, existirá el movimiento inverso m^{-1} para transformar $s_b \rightarrow s_a$
- Para evitar ciclos, el movimiento m^{-1} se agrega a la lista tabú.
- El procedimiento continúa desde s_b hasta que cierta condición de finalización se verifique.

Procedure Busqueda Tabu:
Begin
  s_a = solucion-inicial();
  $ListaTabu = \{\}$;
  While (no-finalizacion) **Do**
    s_b = Mejorar($s_a, \mathcal{N}(s_a), ListaTabu$);
    $s_a = s_b$;
    Actualizar($ListaTabu$);
  endDo
  retornar(s);
End.

Búsqueda Local Iterativa

- **Idea básica**: realizar una trayectoria que pase unicamente a través del conjunto de óptimos locales S^* *en lugar de utilizar todo el espacio* S.
- Naturalmente no es posible construir una estructura de vecindario en S^*.
- Sin embargo, la trayectoria $s_1^*, s_2^*, s_3^*, \ldots, s_t^*$ puede obtenerse implícitamente.

Proc. Busqueda Local Iterativa:

```
Begin
  /* s_a:  solucion actual */
  /* s_p:  solucion intermedia o perturbada */
  /* s_ol:  solucion localmente optima */
  /* H: historia o memoria */
  s_1 = solucion-inicial();
  s_a = BusquedaLocal(s_1);
  Repeat Until ( finalizacion ) Do
    s_p = Perturbar(s_a,H);
    s_ol = BusquedaLocal(s_p);
    s_a = Aceptar?(s_a,s_ol,H);
  endDo
End.
```

Búsqueda por entornos variables

Proc. Busqueda por Entornos Variable:

Begin

```
/* N_k, k = 1,...,k_max, estructuras de vecindarios */
/* s_a: solucion actual */
/* s_p: solucion vecina de s_a */
/* s_ol: solucion localmente optima */
Repeat Until ( finalizacion ) Do
  k=1;
  Repeat Until ( k = k_max ) Do
    /* generar vecino s_p del k-esimo vecindario de s_a (s_p ∈ N_k(s_a)) */
    s_p = ObtenerVecino(s_a, N_k);
    s_ol = BusquedaLocal(s_p);
    If (s_ol es mejor que s_a) Then
      s_a = s_ol;
    Else
      k=k+1;
    endIf
  endDo
endDo
End.
```

Si no consigo mejoras en el entorno actual, cambio de entorno

Esquemas Basados en Poblaciones

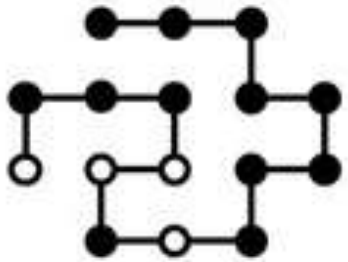
- Algoritmos Evolutivos
- Búsqueda Dispersa
- Algoritmos Meméticos
- Enjambres
- *y un largo etc.*

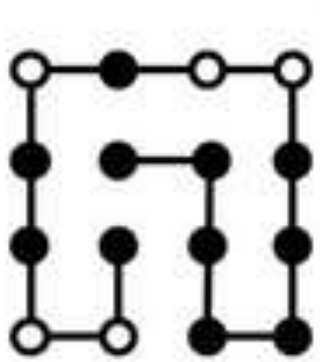
Procedure Algoritmo Genetico:

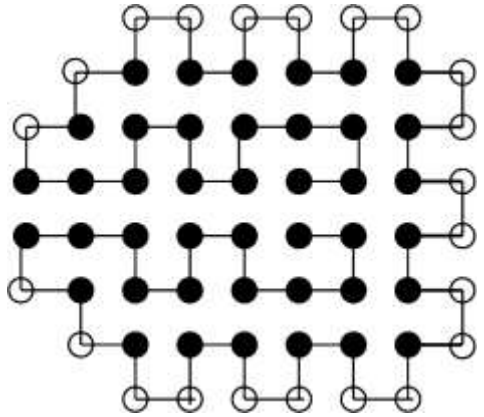
```
Begin
  t = 0;
  inicializar(P(t));
  evaluar(P(t));
  While ( no-finalizacion ) Do
    t = t + 1;
    Seleccionar P(t) desde P(t-1);
    Hacer-Cruzamientos(P(t));
    Aplicar-Mutaciones(P(t));
  endDo
  retornar mejor solucion;
End.
```

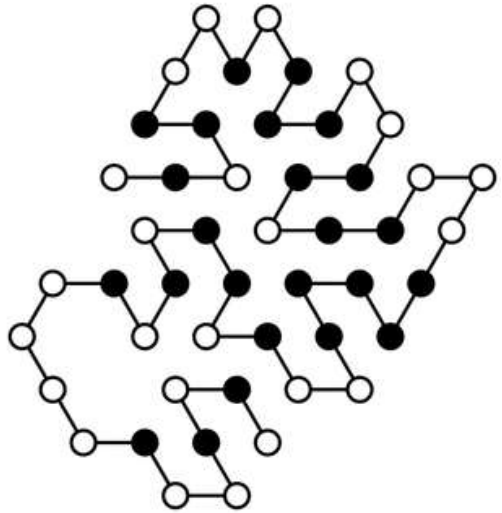
Ejercicio

Dado este problema

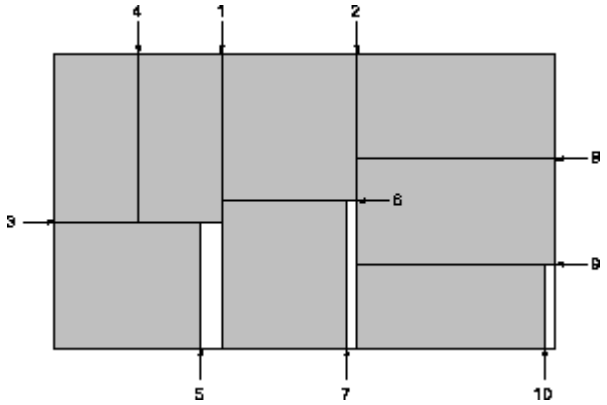


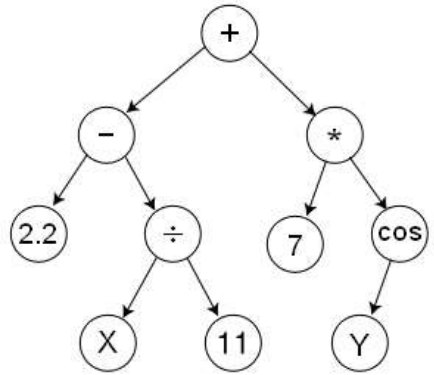




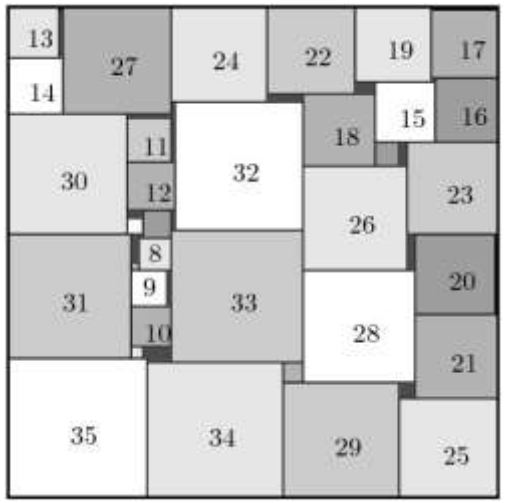


1. Instancia del problema
2. ¿Como representaría la solución?
3. ¿Posibles vecindarios?



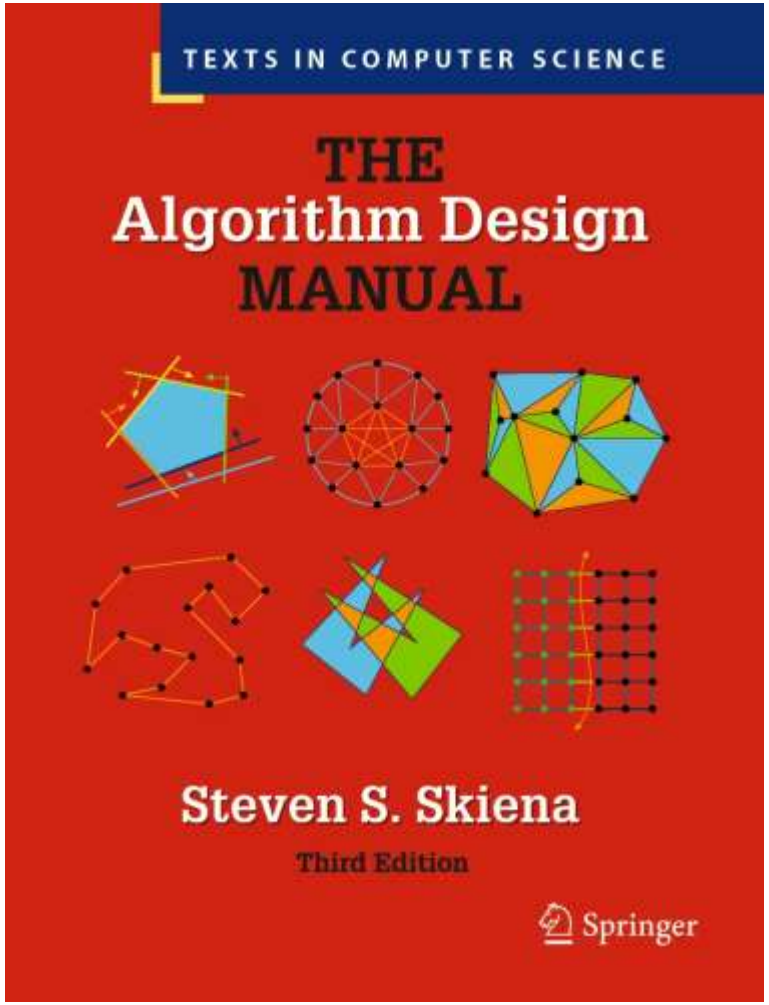


$$\left(2.2 - \left(\frac{X}{11}\right)\right) + \left(7 * \cos(Y)\right)$$



- Y los parámetros
- Y el manejo de las restricciones
- Y que pasa con la calidad de las soluciones

Guía para el diseño de algoritmos



https://link.springer.com/book/10.1007/978-3-030-54256-6

¿Entiendo realmente el problema? (I)

- ¿En qué consiste exactamente la entrada?
- ¿Cuáles son exactamente los resultados o la salida deseada?
- ¿Puedo construir un ejemplo de entrada lo suficientemente pequeño como para resolverlo a mano? ¿Qué ocurre cuando intento resolverlo?
- ¿Qué importancia tiene para mi aplicación que siempre encuentre la respuesta óptima?
- ¿Puedo conformarme con algo cercano a la respuesta óptima?

¿Entiendo realmente el problema? (II)

- ¿Qué tamaño tiene un caso típico de mi problema? ¿Voy a trabajar con 10, 1000 o 1.000.000 de elementos?
- ¿Qué importancia tiene la velocidad en mi aplicación? ¿Debo resolver el problema en un segundo? ¿En un minuto? ¿En una hora? ¿Un día?
- ¿Cuánto tiempo y esfuerzo puedo invertir en la aplicación? ¿Estaré limitado a algoritmos sencillos que se pueden codificar en un día, o tengo libertad para experimentar con un par de enfoques y ver cuál es el mejor?
- ¿Intento resolver un problema numérico? ¿Un problema de algoritmos gráficos? ¿Un problema geométrico? ¿Un problema de cadenas? ¿Un problema de conjuntos? ¿Qué formulación parece más fácil?

¿Puedo encontrar un algoritmo sencillo para mi problema?

(a) ¿Resolverá la fuerza bruta mi problema correctamente buscando entre todos los subconjuntos o arreglos y eligiendo el mejor?

- Si es así, ¿por qué estoy seguro de que este algoritmo siempre da la respuesta correcta?
- ¿Cómo puedo medir la calidad de una solución una vez que la construya?
- ¿Esta solución simple y lenta se ejecuta en tiempo polinómico o exponencial? ¿Es mi problema lo suficientemente pequeño como para que esta solución de fuerza bruta sea suficiente?
- ¿Estoy seguro de que mi problema está lo suficientemente bien definido como para tener una solución correcta?

¿Puedo encontrar un algoritmo sencillo para mi problema?

(b) ¿Puedo resolver mi problema probando repetidamente alguna regla sencilla, como elegir primero el elemento más grande? ¿El elemento más pequeño primero? ¿Un elemento al azar primero?

- En caso afirmativo, ¿con qué tipos de entradas funciona bien esta heurística? ¿Corresponden estos corresponden a los datos que podrían surgir en mi aplicación?
- ¿Con qué tipos de datos funciona mal esta heurística? Si no se encuentran ejemplos, ¿puedo demostrar que siempre funciona bien?
- ¿Con qué rapidez llega mi heurística a una respuesta? ¿Tiene una
- una implementación sencilla?

¿Está mi problema en algún catálogo?

- ¿Qué se sabe sobre el problema? ¿Existe una implementación disponible que pueda utilizar?
- ¿He buscado mi problema en el lugar correcto?
- ¿He mirado todas las imágenes? ¿He buscado en el índice todas las palabras clave posibles?

¿Existen casos especiales del problema que sepa resolver?

- ¿Puedo resolver el problema de forma eficiente si ignoro algunos de los parámetros de entrada?
- ¿El problema es más fácil de resolver cuando pongo algunos de los parámetros de entrada de entrada a valores triviales, como 0 o 1?
- ¿Puedo simplificar el problema hasta el punto de poder resolverlo eficientemente?
- ¿Por qué no se puede generalizar este algoritmo de caso especial a una clase más amplia de entradas?
- ¿Es mi problema un caso especial de un problema más general del catálogo?

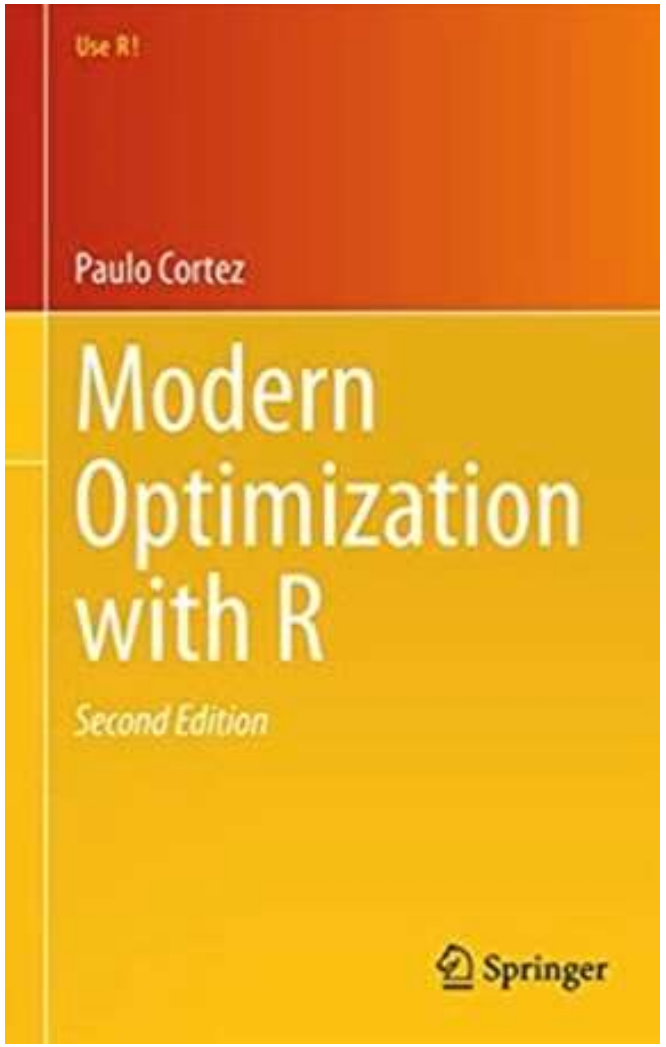
¿Cuál de los paradigmas de diseño de algoritmos estándar es más relevante para mi problema

- ¿Existe un conjunto de elementos que pueda ser ordenado por tamaño o por alguna clave? ¿Este orden facilita la búsqueda de la respuesta?
- ¿Existe una forma de dividir el problema en dos problemas más pequeños, quizás haciendo una búsqueda binaria? ¿Qué tal dividir los elementos en grandes y pequeños, o en izquierda y derecha? ¿Sugiere esto un algoritmo de "divide y vencerás"?
- ¿Los objetos de entrada o la solución deseada tienen un orden natural de izquierda a derecha, como los caracteres de una cadena, los elementos de una permutación, etc.?

¿Cuál de los paradigmas de diseño de algoritmos estándar es más relevante para mi problema

- ¿Hay ciertas operaciones que se realizan repetidamente, como la búsqueda, o encontrar el elemento mayor/menor? ¿Puedo utilizar una estructura de datos para acelerar acelerar estas consultas? ¿Qué tal un diccionario/tabla hash o un montón/cola de prioridades?
- ¿Puedo utilizar el muestreo aleatorio para seleccionar el siguiente objeto? ¿Se pueden construir muchas configuraciones aleatorias y elegir la mejor? ¿Puedo utilizar algún tipo de aleatoriedad dirigida como el recocido simulado para acercarme a la mejor solución?
- ¿Puedo formular mi problema como un programa lineal? ¿Qué tal un programa entero?

Para seguir investigando



Disponible a través de SpringerLink
https://link.springer.com/book/10.1007/978-3-030-72819-9

Página del autor:
https://pcortez.dsi.uminho.pt/mor-book