PRÁCTICA 3

Pablo Baeyens Antonio Checa Iñaki Madinabeitia José Manuel Muñoz Darío Sierra

Algorítmica

ÍNDICE

CONTENEDORES EN UN BARCO

PROBLEMA

Rellenar un buque con carga limitada (K) con un conjunto de contenedores $c_1, \ldots, c_n$ con pesos $p_1, \ldots, p_n$.

Entrada: Vector de pesos de los contenedores, p y capacidad total K

Salida: Vector con los contenedores elegidos

ESTRUCTURA DE DATOS

```
struct Cont{
  int id;
  peso_t peso;

  Cont(int i, peso_t p){
    id = i;
    peso = p;
  }
};
```

- Basta coger los contenedores de menor peso hasta rellenar el buque
- Emparejamos cada elemento con su posición y ordenamos en función de los pesos.
- Una vez ordenados, rellenamos hasta agotar la capacidad.
- **Eficiencia:** $O(n \log(n))$

```
vector<int> max_num_conts(const vector<peso_t> p,
    peso_t K){
  vector<Cont> conts = a_cont(p);

  sort(conts.begin(), conts.end(), menor);

  vector<int> elegidos;

  for(int i = 0; i < conts.size() && conts[i].peso <=
      K; i++){
    elegidos.push_back(conts[i].id);
    K -= conts[i].peso;
  }

  return elegidos;
}
```

Este criterio siempre halla la solución con un mayor número de contenedores.

Vamos a razonar por reducción al absurdo.

Este criterio siempre halla la solución con un mayor número de contenedores.

Vamos a razonar por reducción al absurdo.

Partimos de nuestra solución $o_1, \ldots, o_k$ del algoritmo anterior. Estos contenedores son los menores entre todos los posibles.

Suponemos otra solución, $s_1, \ldots, s_m$, una solución del problema, con mayor número de contenedores. Es decir, $m > k$.

- Cogemos el contenedor de mayor peso que quepa en el buque.
- El algoritmo es muy similar al del apartado anterior.

```
vector<int> max_peso_greedy(const vector<peso_t> p,
    peso_t capacidad){
  vector<Cont> conts = a_cont(p);
  sort(conts.begin(), conts.end(), mayor);
  vector<int> elegidos;

  for(int i = 0; i < conts.size() && capacidad > 0; i
      ++){
    if(conts[i].peso <= capacidad){
      elegidos.push_back(conts[i].id);
      capacidad -= conts[i].peso;
    }
  }
  return elegidos;
}
```

Hemos querido comparar el algoritmo Greedy anterior (que coge en cada momento el contenedor que más aumenta el peso total) con uno que encuentra la mejor solución siempre.

De esta forma, podremos ver las ventajas que presenta enfocar las soluciones de una forma simple.

A continuación colocamos el algoritmo que encuentra siempre el óptimo:

- Está basado en la fuerza bruta
- Recorre todas las posibilidades y se queda con la mejor.
- **Eficiencia:** $O(n!)$

MAXIMIZAR EL NÚMERO DE TONELADAS CARGADAS

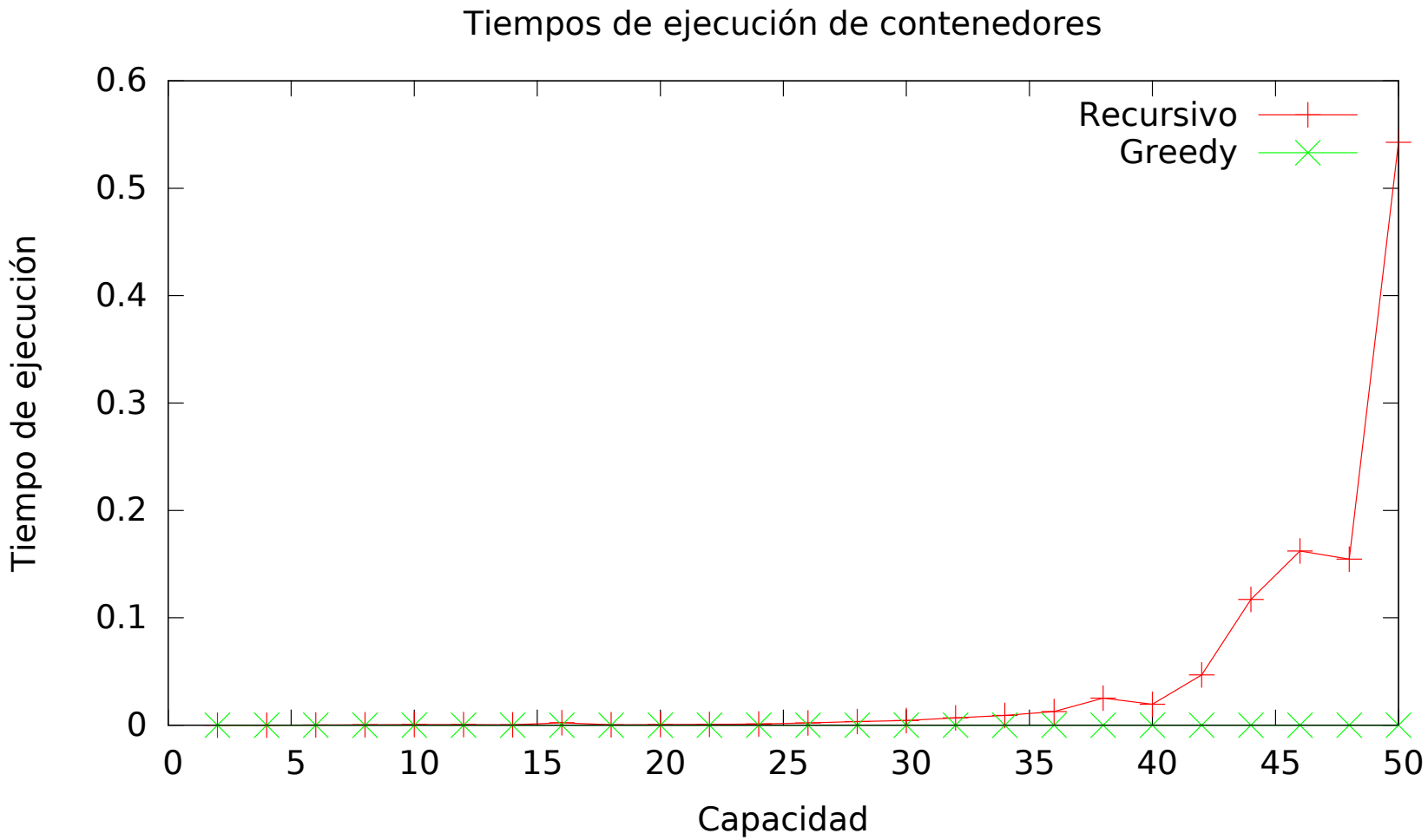
```
vector<Cont> bruto_sub(const vector<Cont> p, peso_t K)
    {
  if(p.empty() || K == 0)
    return vector<Cont>();

  peso_t max = 0;
  vector<Cont> max_v;
  for(vector<Cont>::const_iterator it = p.cbegin(); it
      != p.cend() && max < K; ++it){
    vector<Cont>::const_iterator next(it);
    vector<Cont> orig_copy(++next, p.cend());   //
       Copia todos los posteriores
    vector<Cont> v_candidato;
    if(it->peso > K)
      v_candidato = bruto_sub(orig_copy, K);
    else {
      v_candidato = bruto_sub(orig_copy, K-it->peso);
      v_candidato.push_back(*it);   // Si usamos
         listas, mejor usar push_front
    }
    peso_t suma1 = suma(v_candidato);
    if(suma1 > max && suma1 <= K){
      max = suma1;
      max_v = v_candidato;
    }
  }
  return max_v;
```

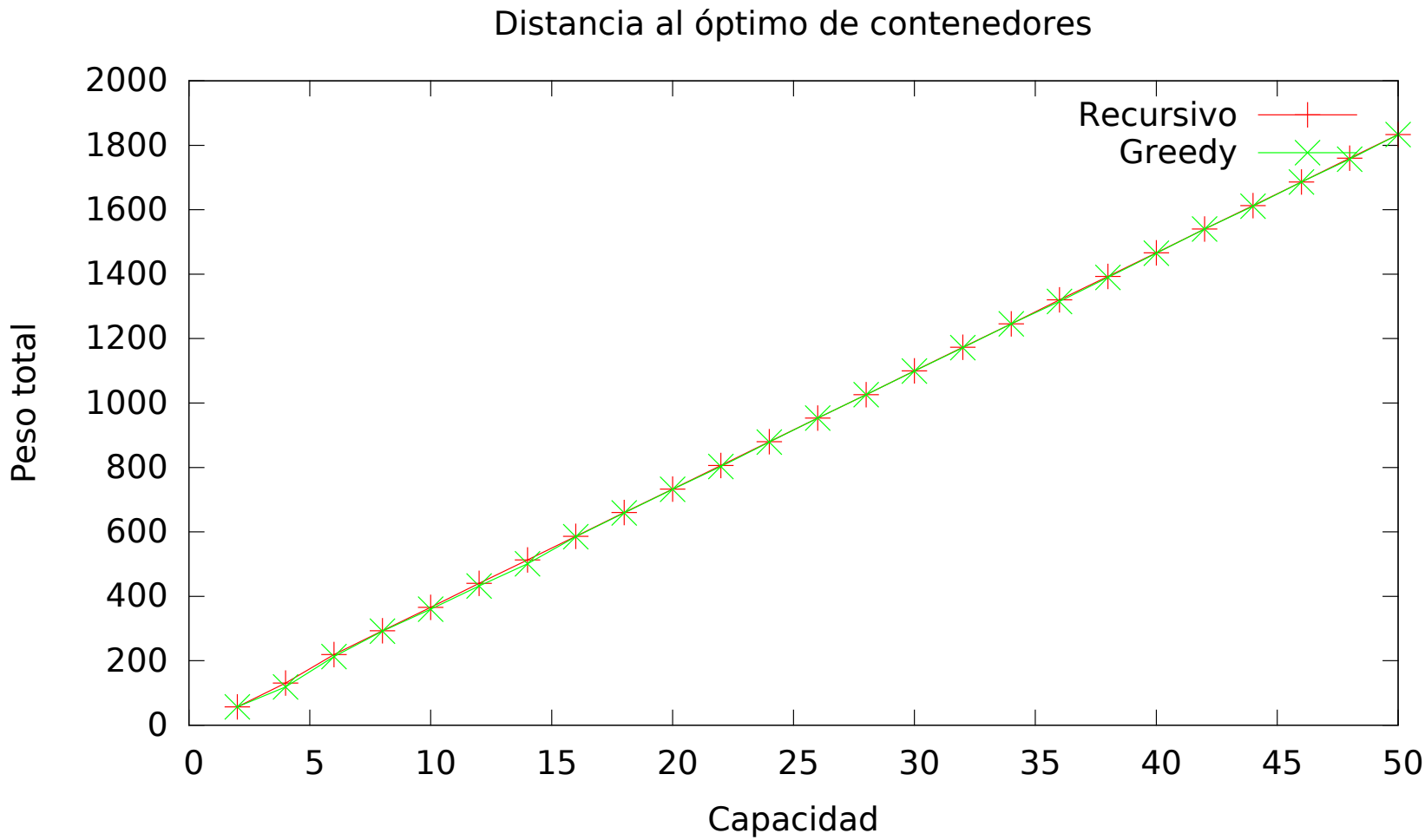
Es fácil ver que el algoritmo greedy es más eficiente que el óptimo, pero el óptimo tiene asegurada la mejor respuesta.

Para poder contrastar mejor, hemos hecho las gráficas de comparación.

MAXIMIZAR EL NÚMERO DE TONELADAS CARGADAS



MAXIMIZAR EL NÚMERO DE TONELADAS CARGADAS

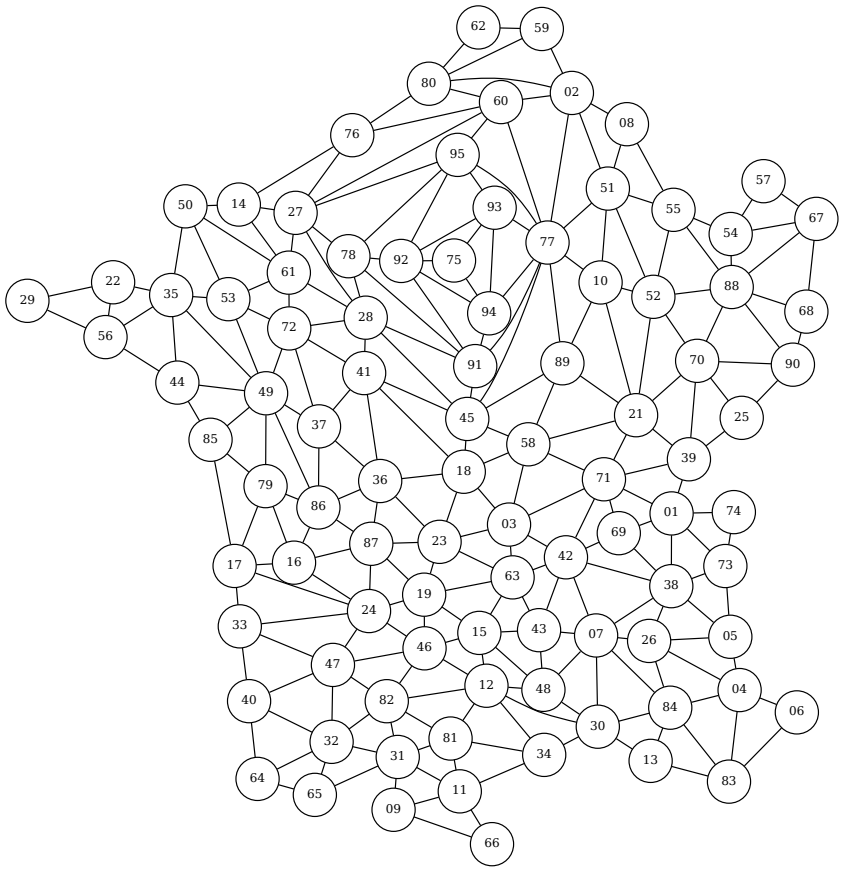


Este es un ejemplo en el que es posible que nos interese trabajar con algoritmos greedy si quisiéramos conseguir una solución rápida a un problema grande.

EL PROBLEMA DEL VIAJANTE DE COMERCIO

PROBLEMA

Hallar el recorrido con distancia mínima en un conjunto de ciudades que pase por todas las ciudades y regrese al punto inicial.



Entrada: Ficheros con ciudades indicadas como puntos en el plano según sus coordenadas.

Salida: `vector<int>` con el orden en el que se recorren las ciudades.

GRAFOS

Un grafo consta de:

- Una **cantidad de nodos**, almacenada en el atributo `nodos`.
- Una **matriz de pesos**, almacenada en el vector `lados`.

```
template<class T>
class Grafo {
   unsigned int nodos;
   T* lados;
```

CONSTRUCCIÓN DEL GRAFO

Construimos el grafo a partir de la distancia euclídea redondeada al entero más próximo.

```
void pesosDesdeCoordenadas(double c[][2]) {
   for (int i = 0; i < nodos-1; i++)
      for (int j = i+1; j < nodos; j++)
         setPeso(i, j, dist(c[i],c[j]));
}
```

Tomamos la ciudad inicial y almacenamos en las ciudades no visitadas:

```
vector<int> trayecto(1, 0);
list<int> disponibles;
for (int i = g.numNodos()-1; i > 0; i--)
  disponibles.push_front(i);
```

Devolveremos `trayecto`.

VECINO MÁS CERCANO

Recorremos la lista buscando aquella ciudad con distancia mínima:

```
while(!disponibles.empty()) {
  list<int>::iterator it = disponibles.begin(),
      cercano = it, fin = disponibles.end();
  int actual = trayecto.back();
  peso_t d_actual = g.peso(actual, *cercano);
  for (++it; it != fin; ++it) {
    peso_t d_candidato = g.peso(actual, *it);
    if (d_candidato < d_actual) {
      cercano = it;
      d_actual = d_candidato;
    }
  }

  trayecto.push_back(*cercano);
  disponibles.erase(cercano);
}
```

INSERCIÓN (ALGORITMO)

- Toma como camino inicial los nodos más al norte, este y oeste
- Mientras haya nodos disponibles:
  - Para cada nodo disponible, halla la posición entre dos nodos del camino donde, de insertarse, se incrementaría menos la longitud del camino
  - Inserta el nodo que aumente menos la longitud en la posición que se calculó para ese nodo

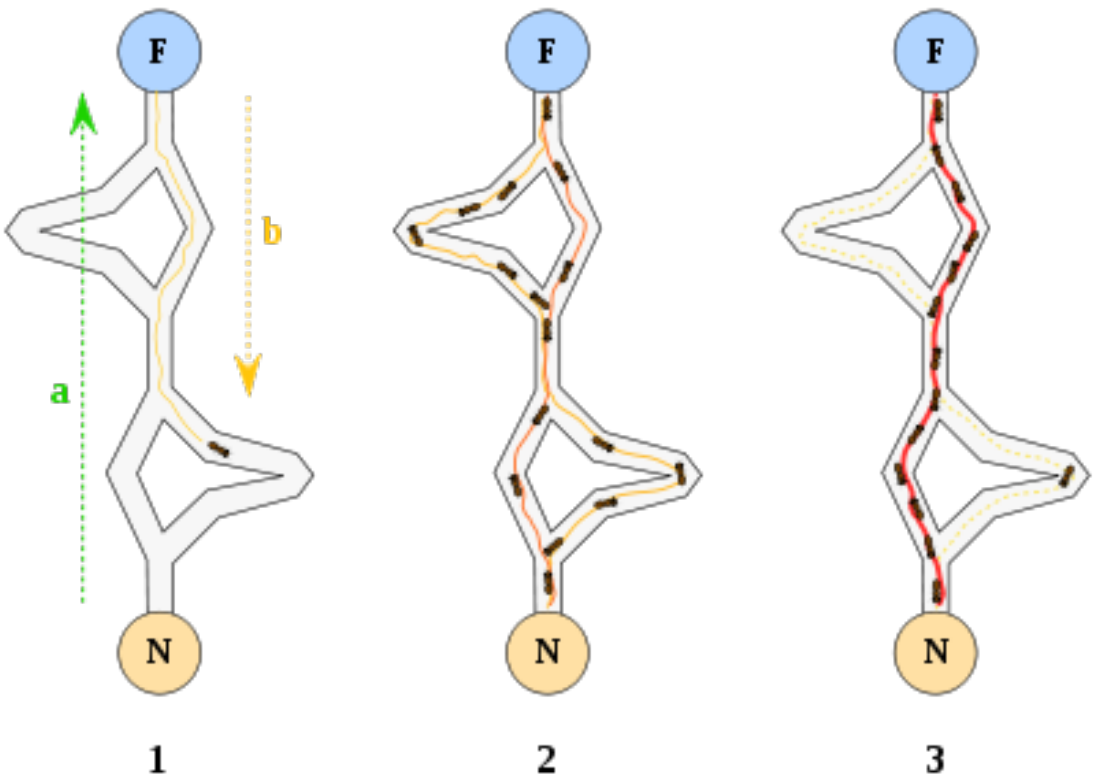
INSERCIÓN (CÓDIGO)

```
while (!disponibles.empty()){
  list<int>::iterator minimo = disponibles.begin();
  pair<int,list<int>::iterator> PesoIndiceMin =
      PesoNuevoCircuito(*minimo,trayecto_l,g);

  for(list<int>::iterator it = ++disponibles.begin()
      ; it != disponibles.end(); it++){
    pair<int,list<int>::iterator> PesoIndicetmp =
        PesoNuevoCircuito(*it,trayecto_l,g);
    if(PesoIndiceMin.first > PesoIndicetmp.first){
      minimo = it;
      PesoIndiceMin=PesoIndicetmp;
    }
  }

  trayecto_l.insert(PesoIndiceMin.second,*minimo);
  disponibles.erase(minimo);
}
```

Este algoritmo se inspira en la comunicación por feromonas de una colonia de hormigas para encontrar el camino mínimo hacia una fuente de comida.



Una Colonia consta de:

- Grafo de **distancias** entre las ciudades.
- Grafo con las **feromonas** de cada camino, inicialmente arbitrarias.
- Constantes $\alpha, \beta, \rho, \xi, C, P, I$.

α Peso que tienen las feromonas.

β Peso que tienen la distancias.

ρ Coeficiente de evaporación de las feromonas.

ξ Coeficiente de debilitamiento de las feromonas.

C Cuántas feromonas se añaden a un camino.

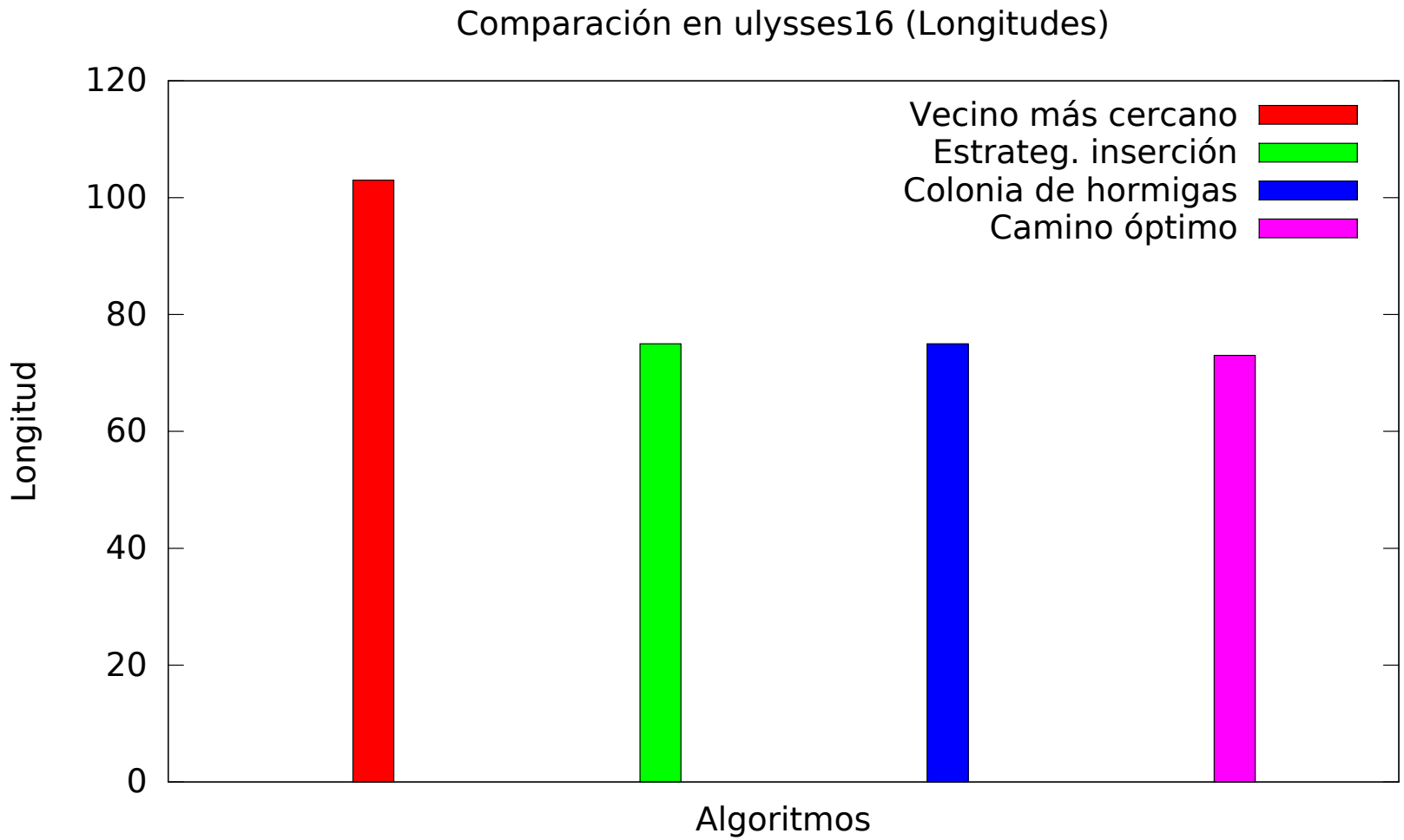
P Probabilidad de tomar el camino más atractivo.

I Cantidad de feromona inicial entre cada par de nodos.

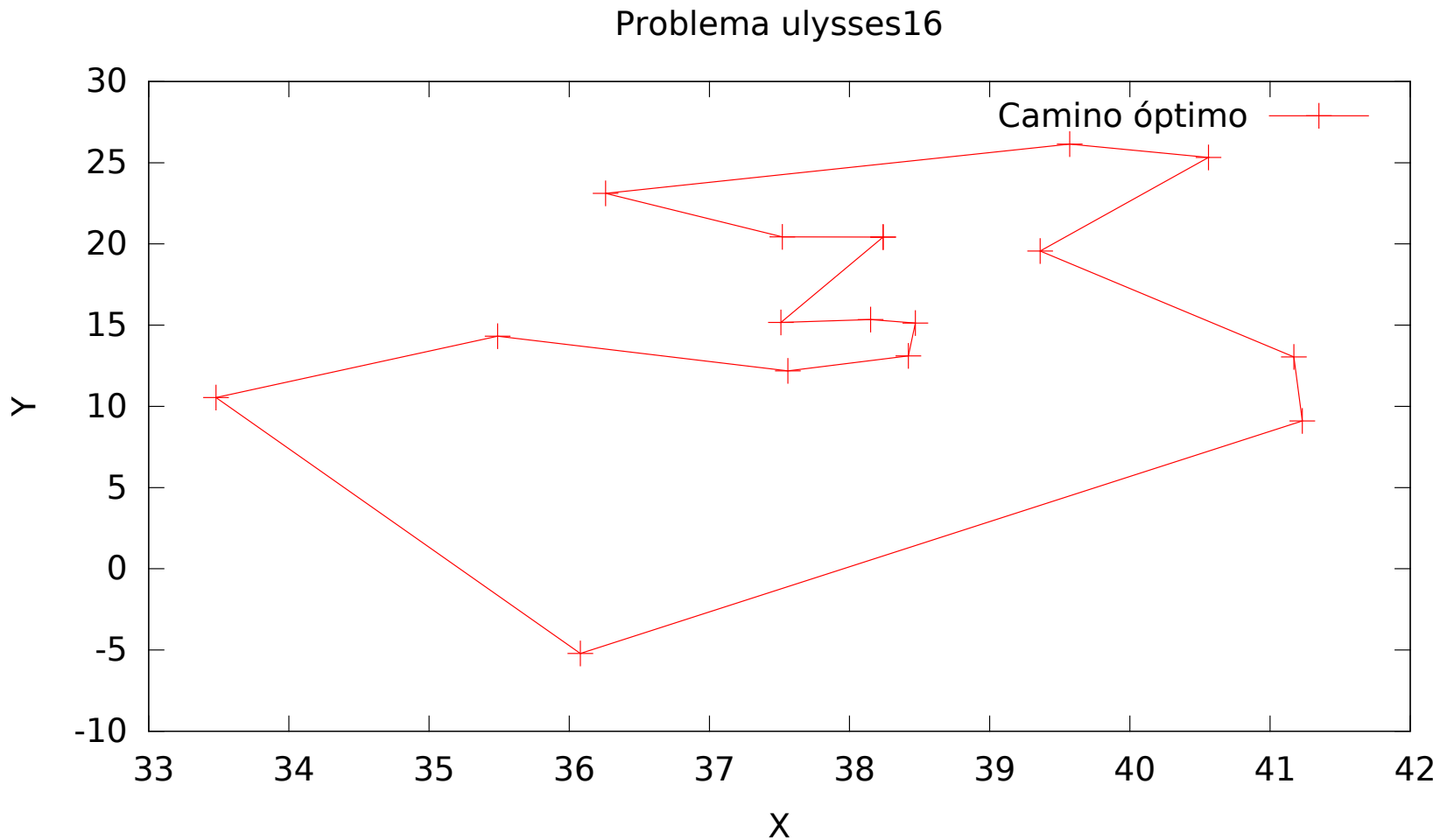
A continuación ponemos tres ejemplos de ejecución de los algoritmos, para poder verlos y compararlos de una forma más eficaz.

El tiempo de ejecución del algoritmo de colonia de hormigas es arbitrario y puede ajustarse a voluntad. A mayor tiempo de ejecución, mejor resultado ofrece. En general necesita más tiempo que los otros dos algoritmos para dar un resultado igual o mejor.

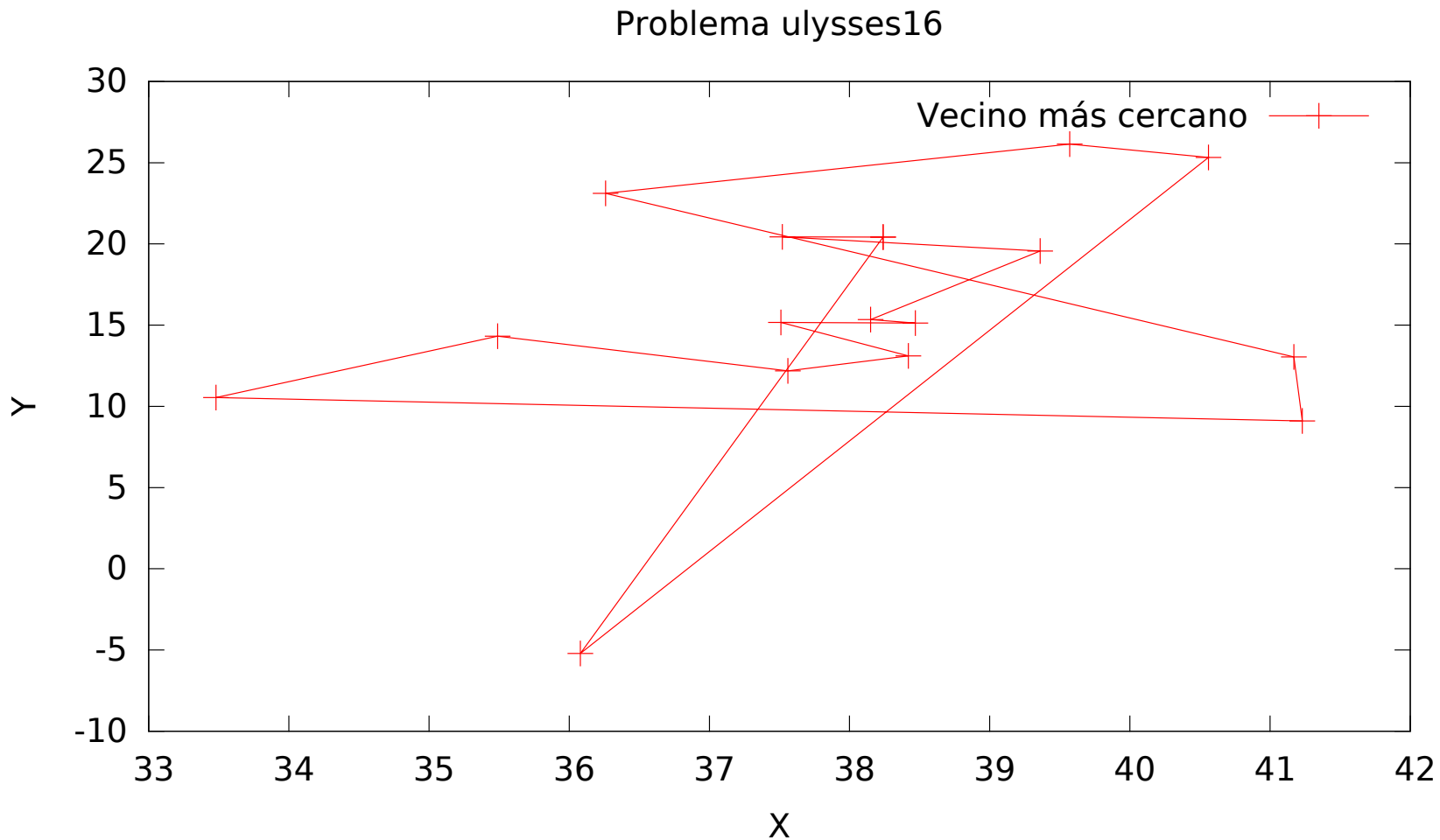
COMPARATIVA DE LONGITUDES



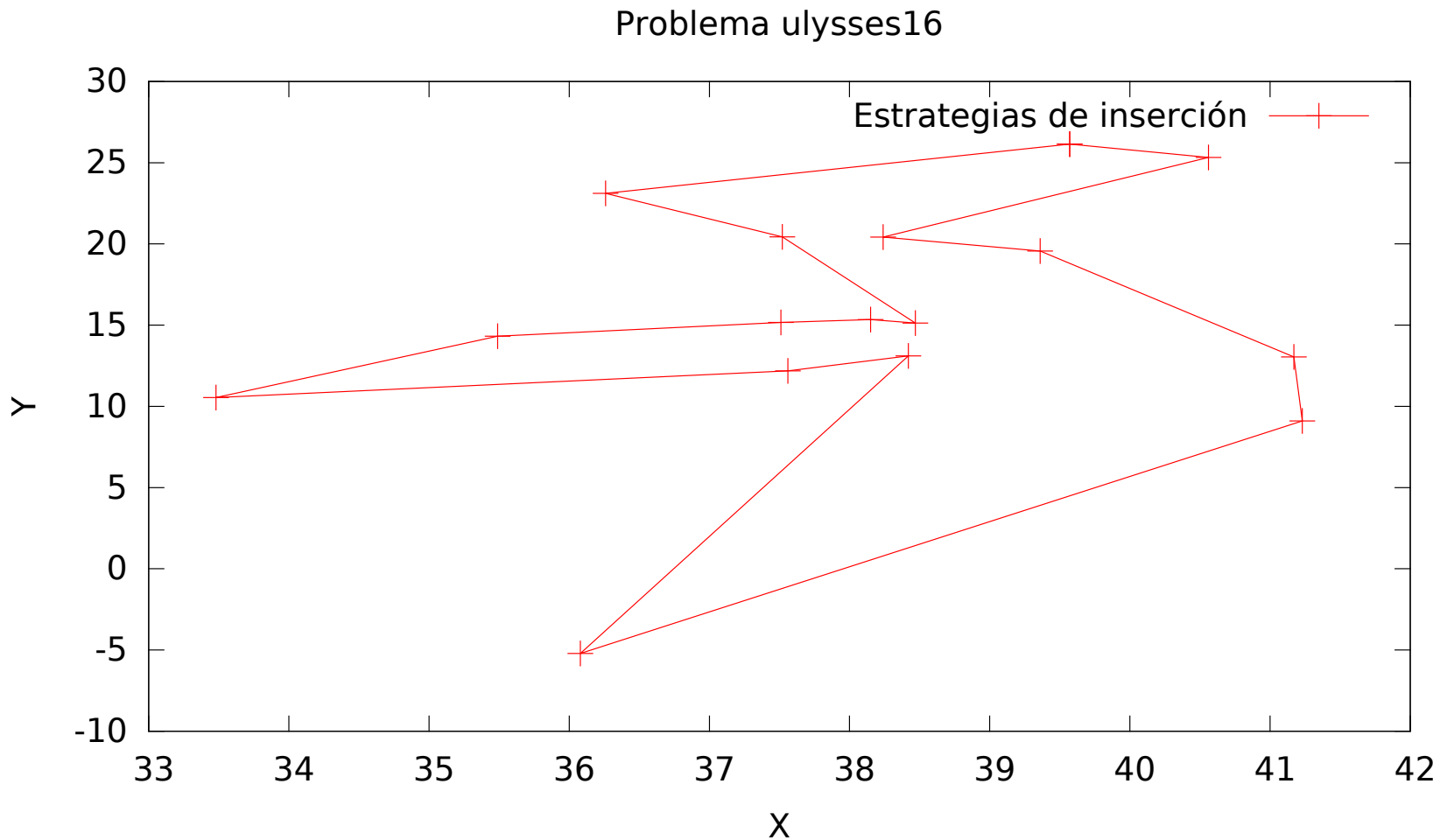
SOLUCIÓN ÓPTIMA



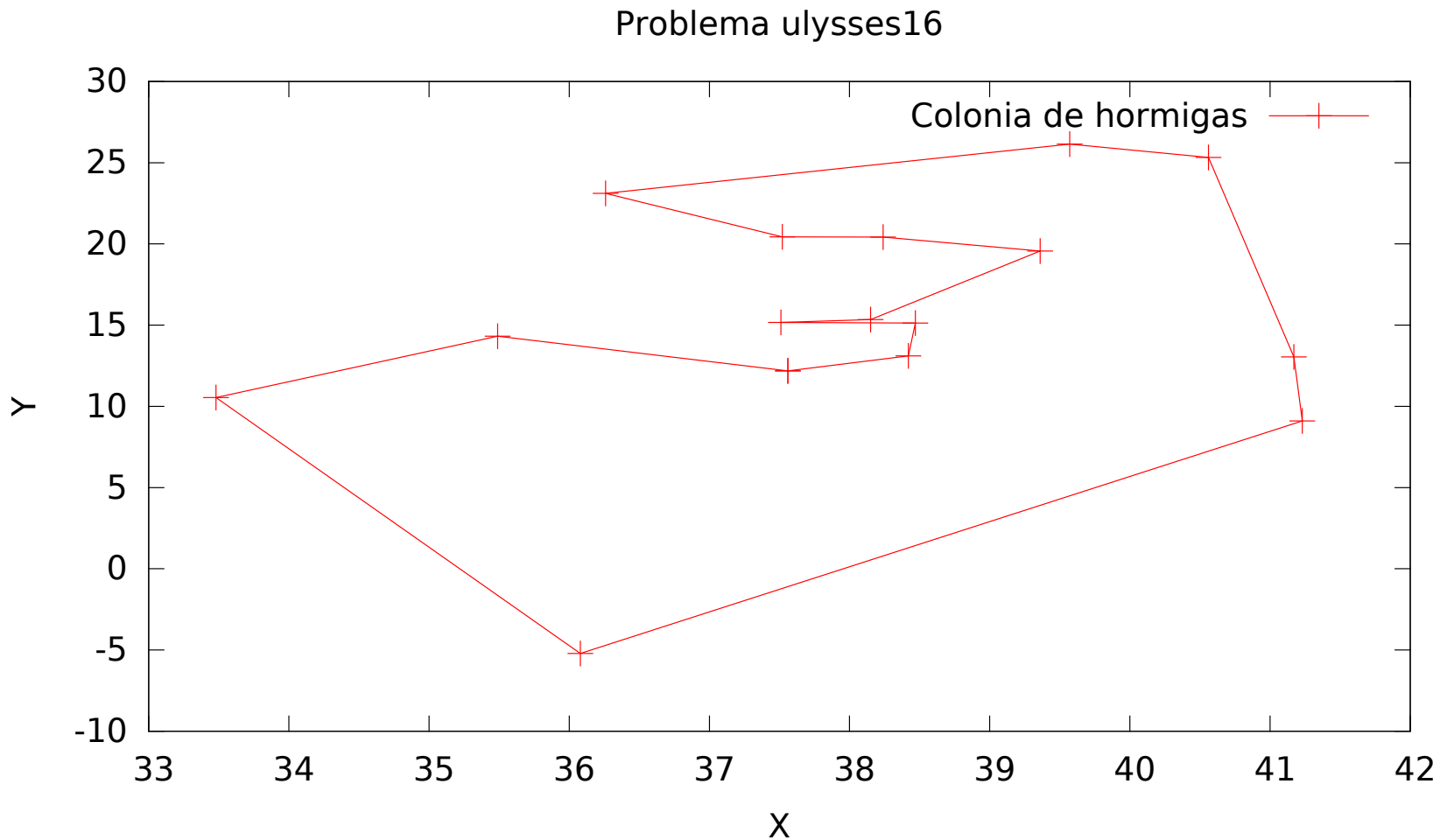
SOLUCIÓN DE VECINO MÁS CERCANO



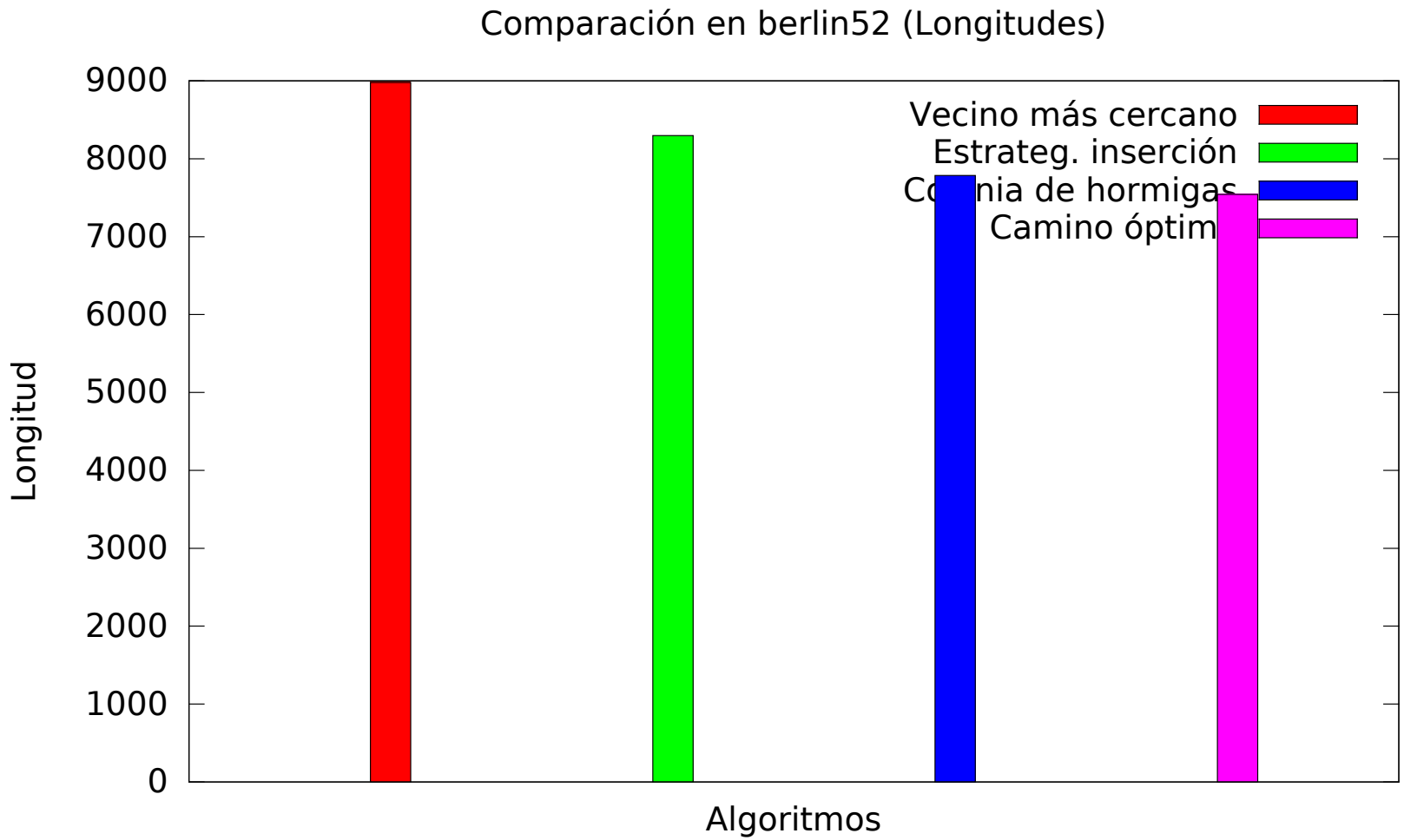
SOLUCIÓN DE INSERCIÓN



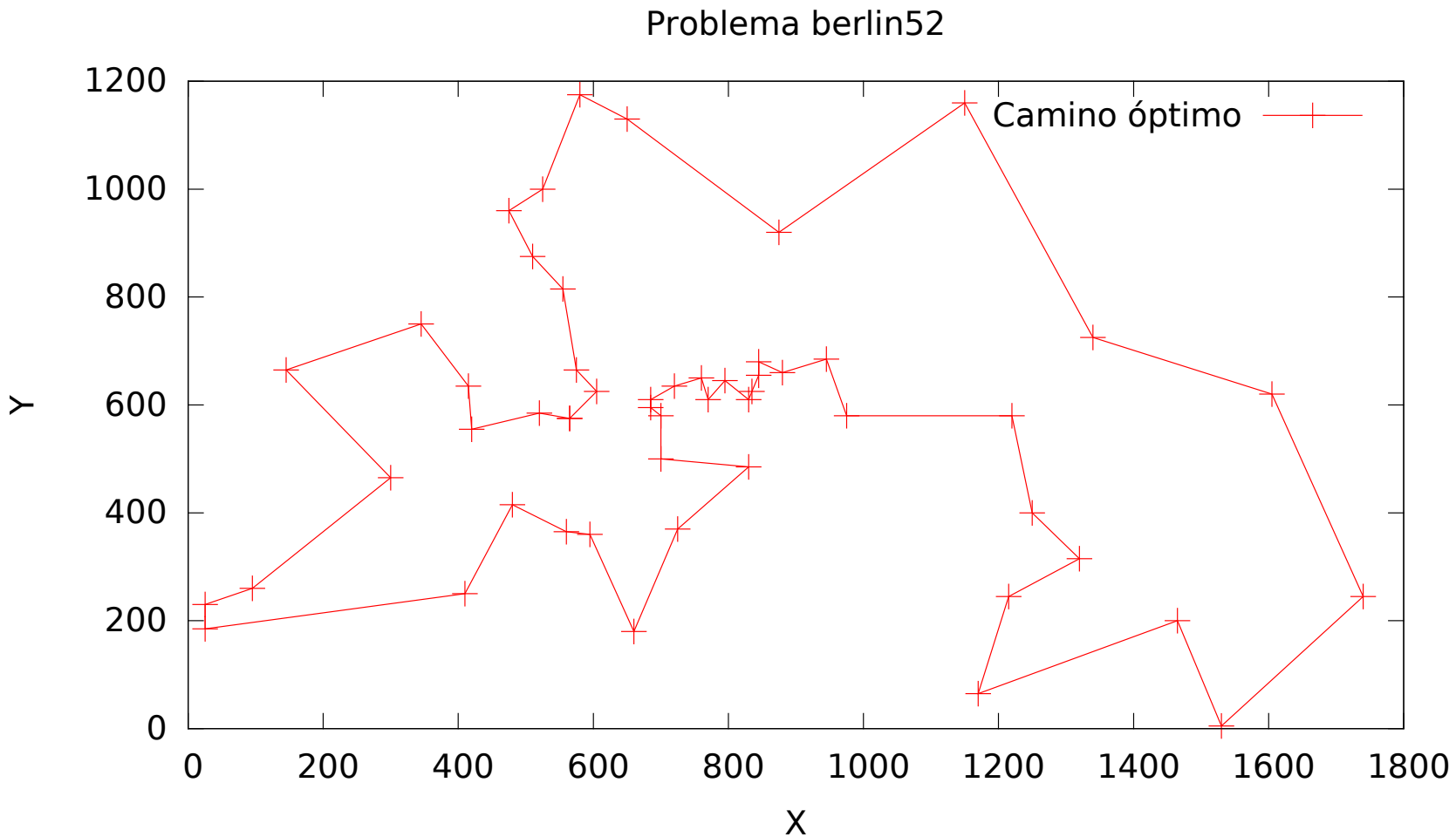
SOLUCIÓN DE LA COLONIA DE HORMIGAS



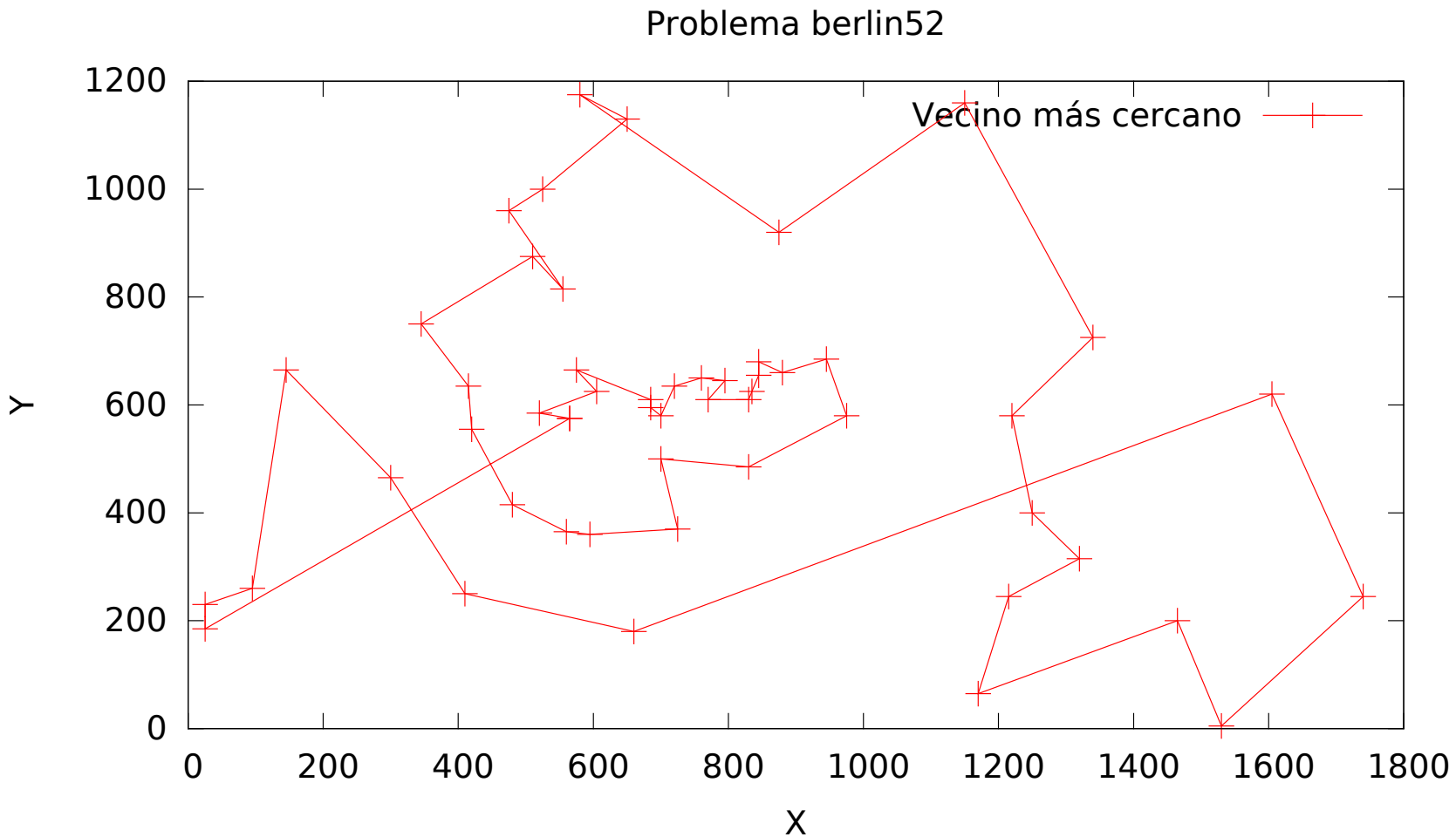
COMPARATIVA DE LONGITUDES



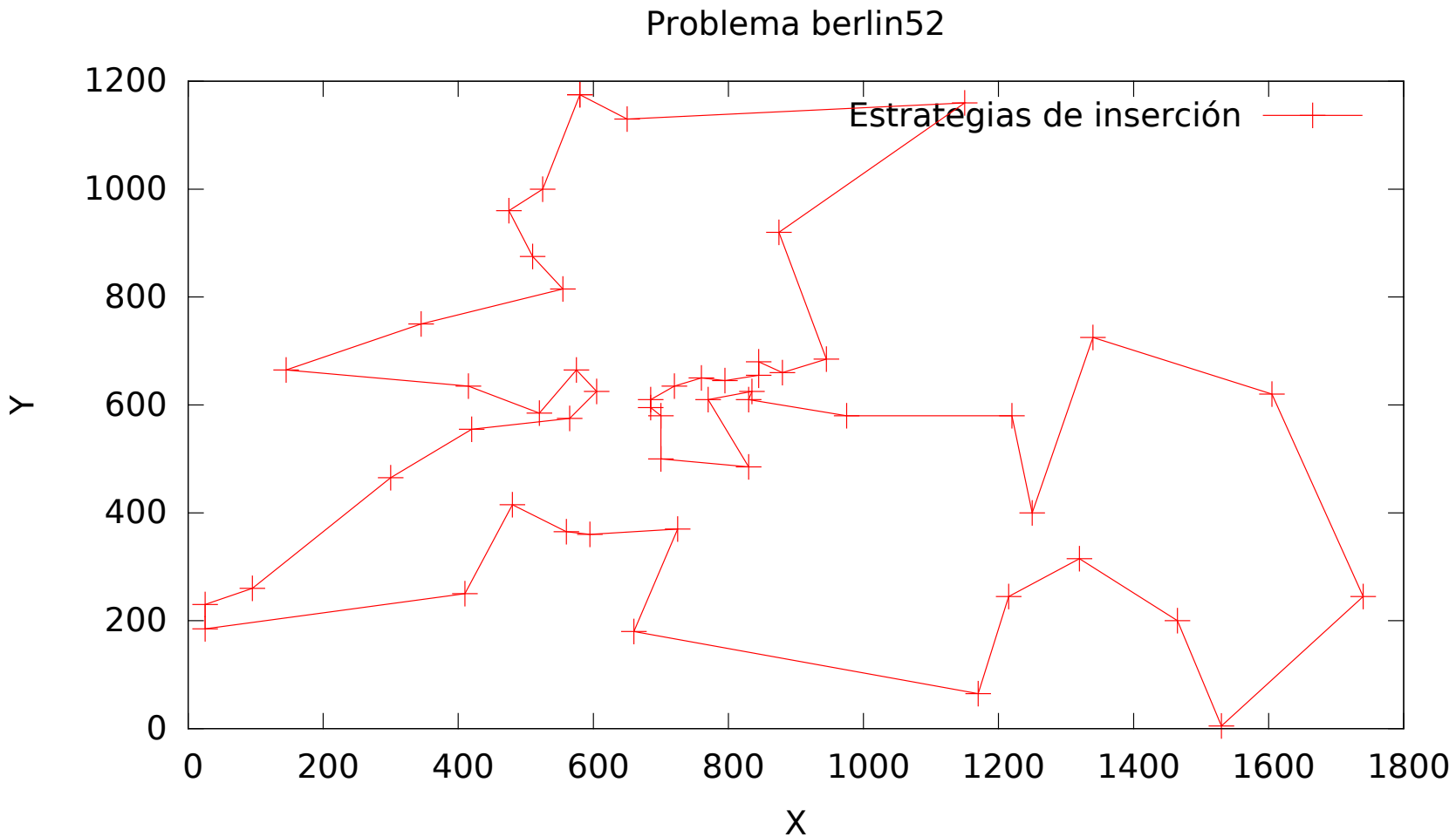
SOLUCIÓN ÓPTIMA



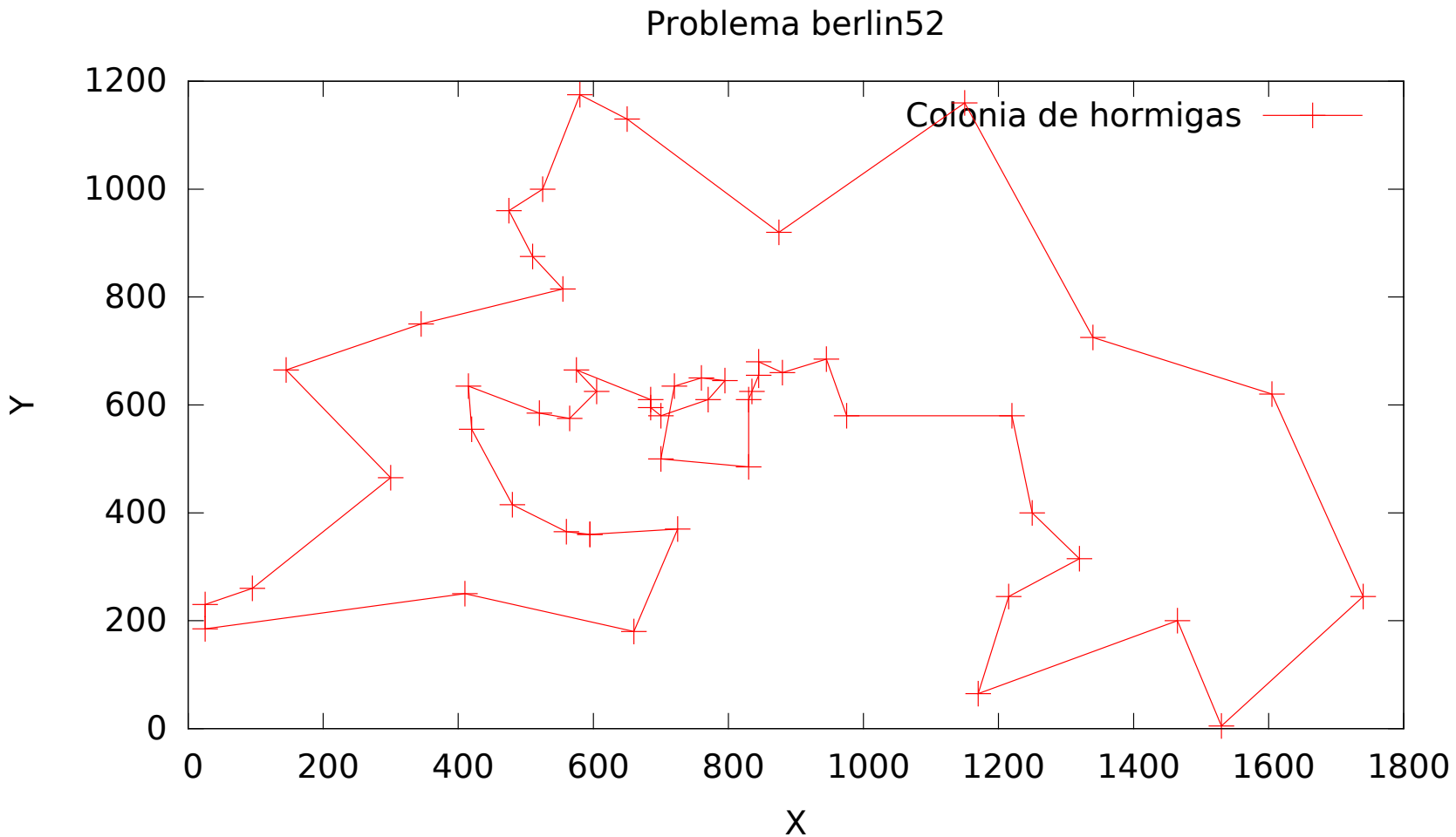
SOLUCIÓN DE VECINO MÁS CERCANO



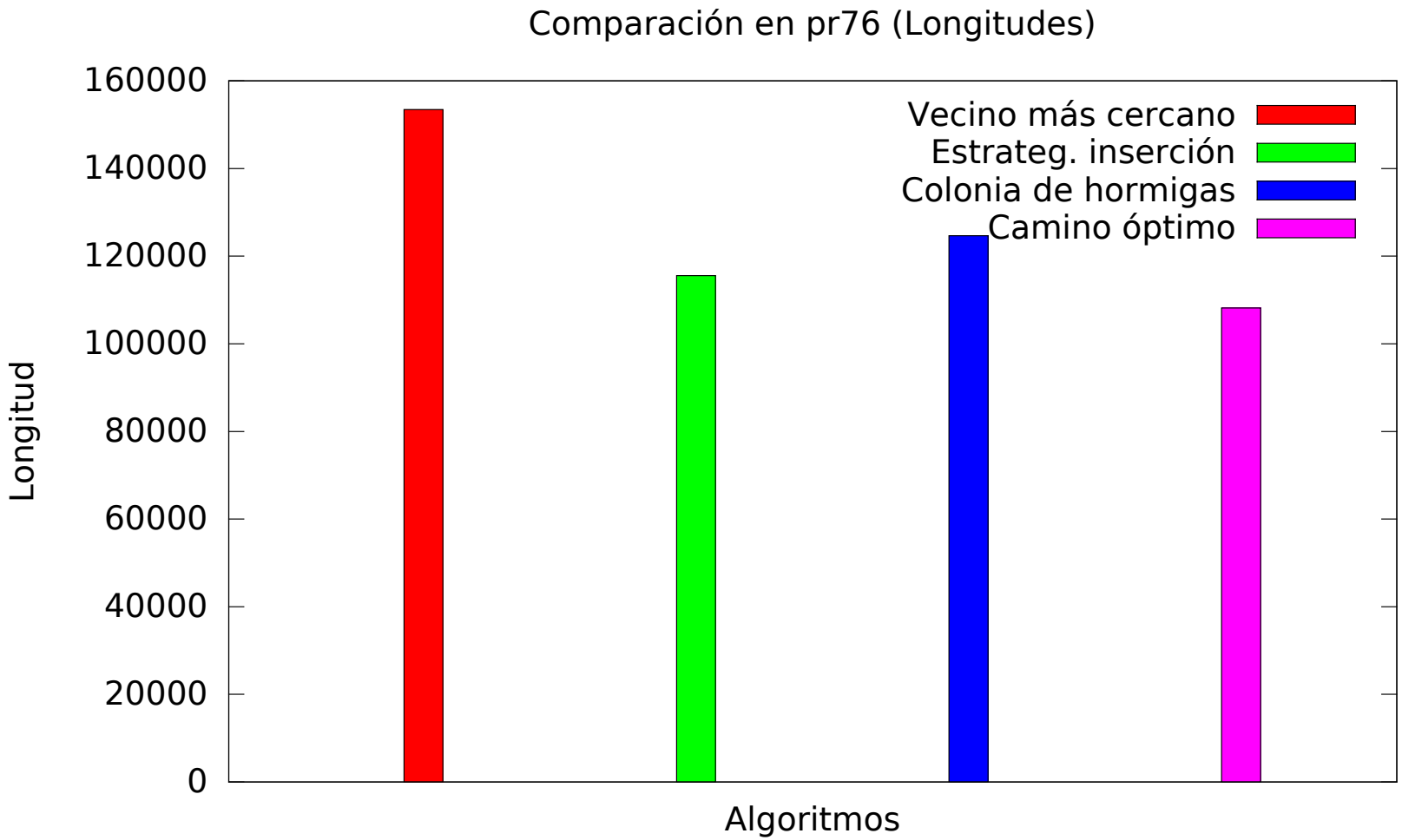
SOLUCIÓN DE INSERCIÓN



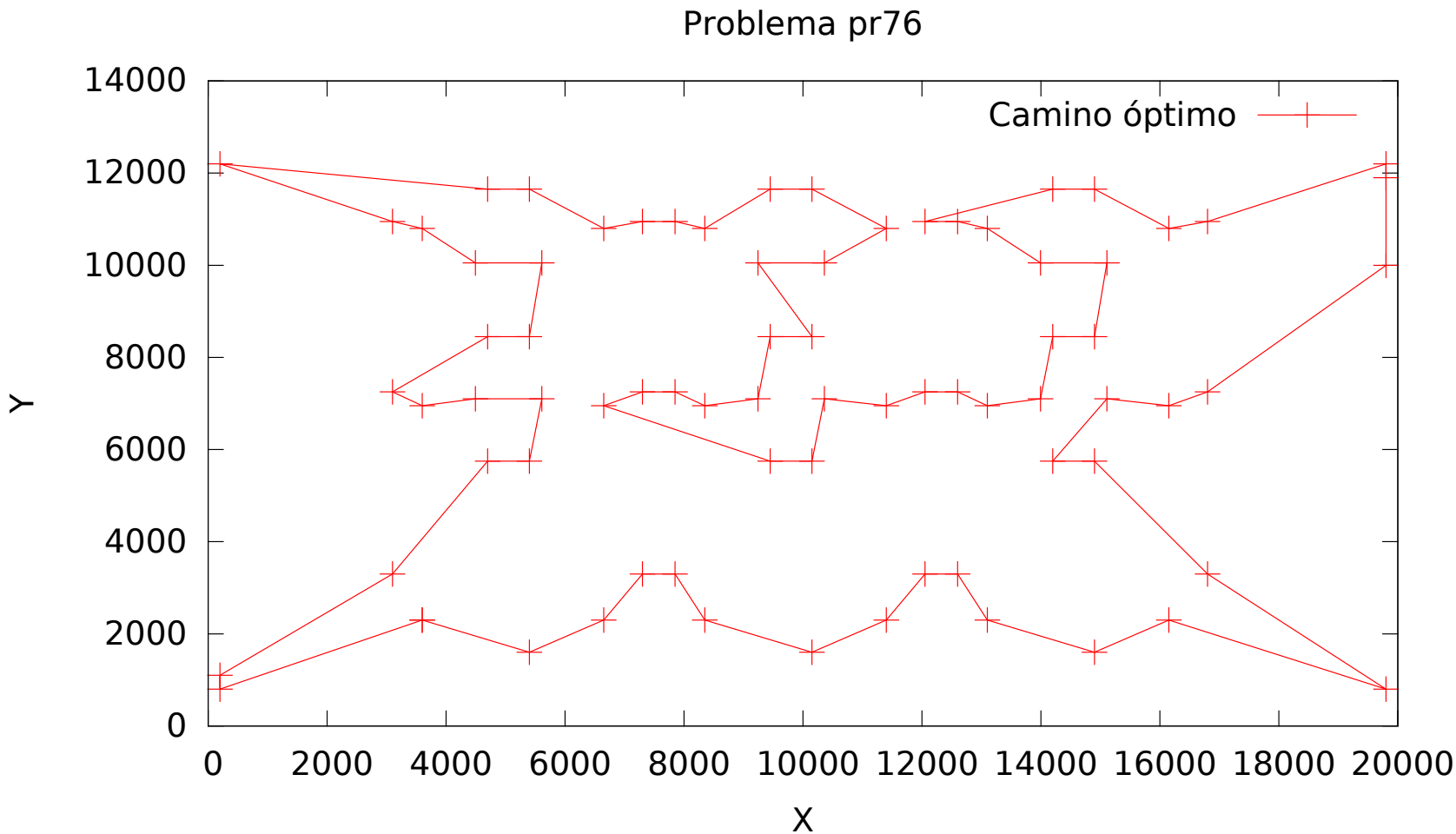
SOLUCIÓN DE LA COLONIA DE HORMIGAS



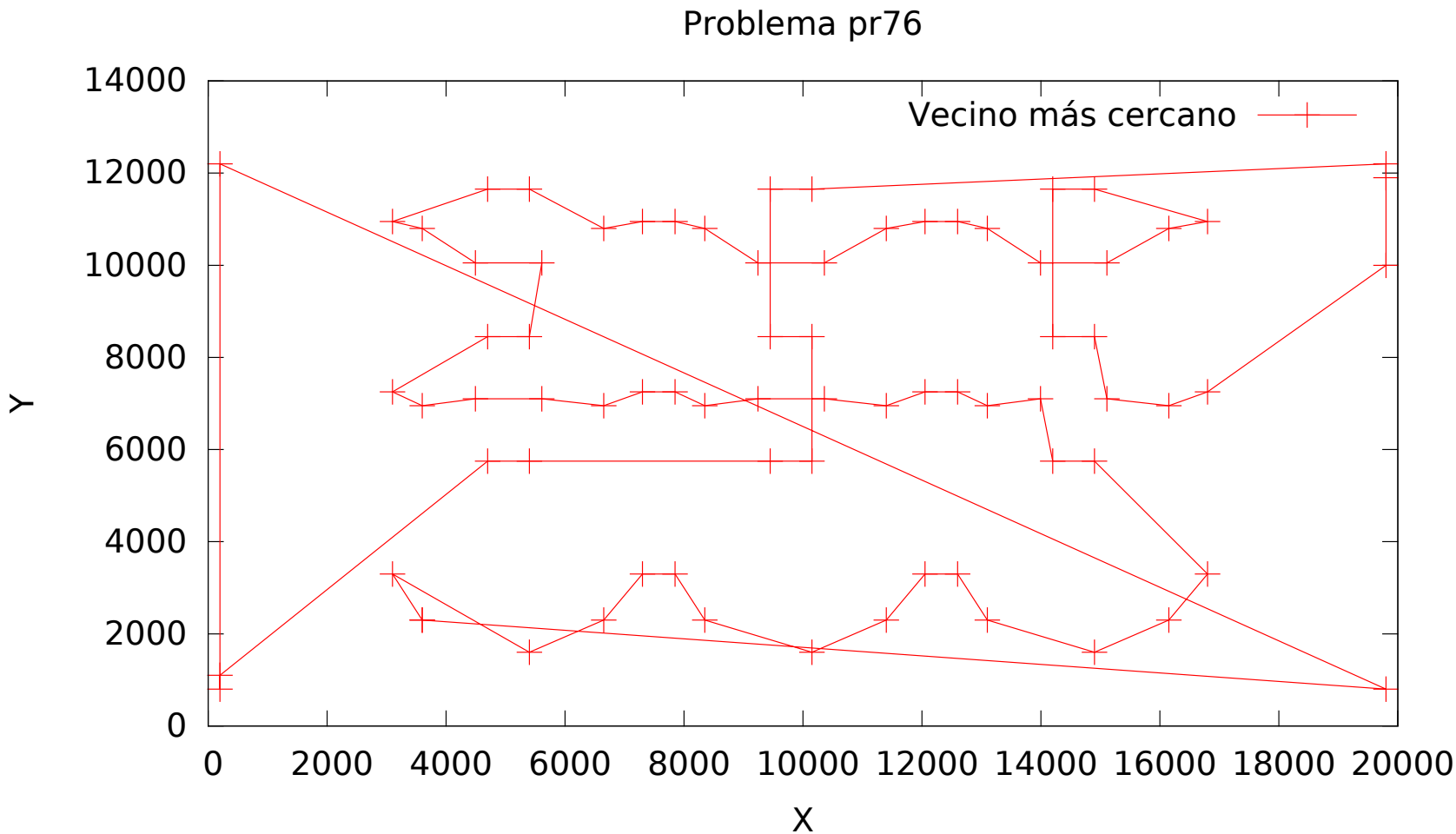
COMPARATIVA DE LONGITUDES

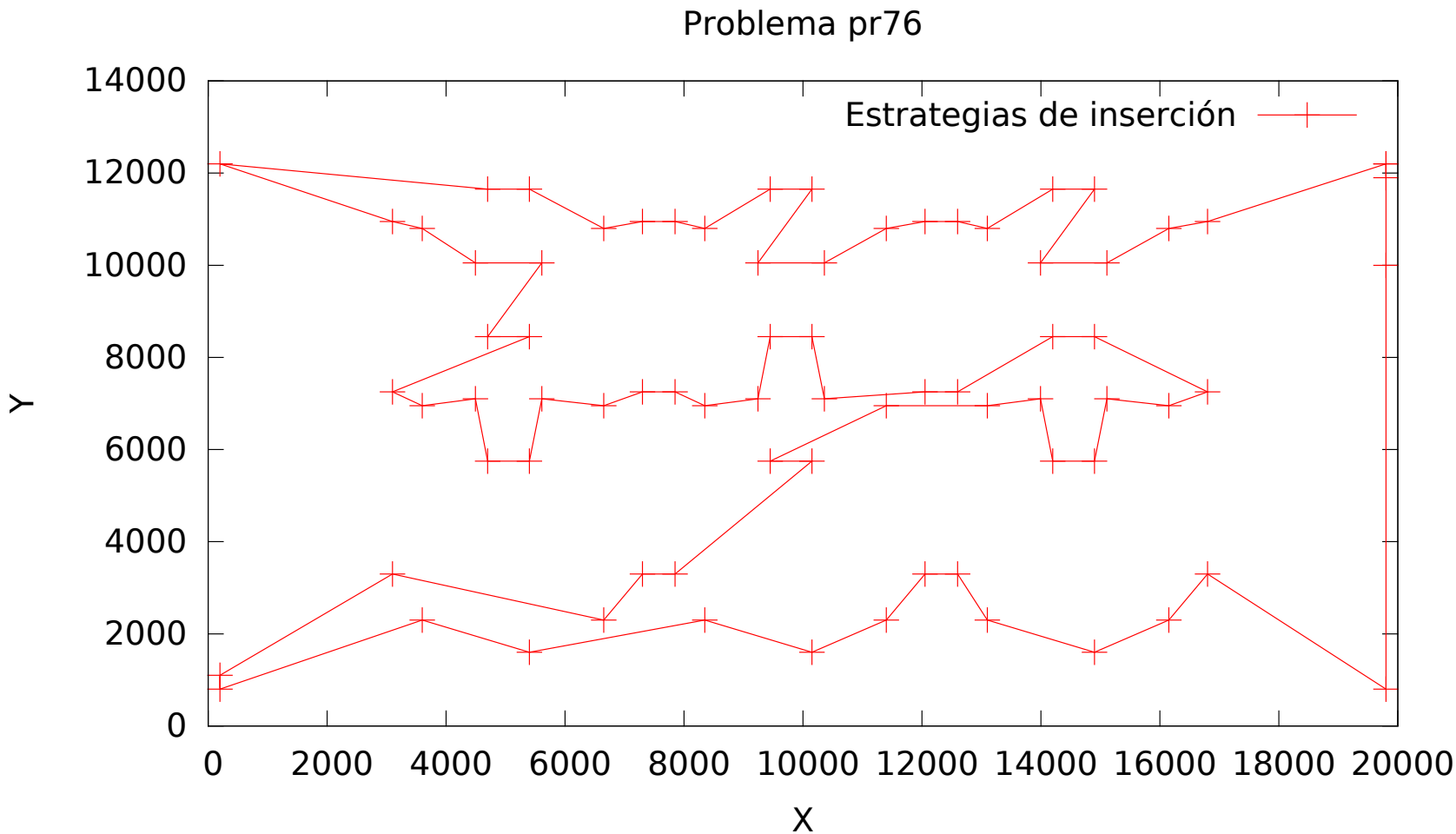


SOLUCIÓN ÓPTIMA



SOLUCIÓN DE VECINO MÁS CERCANO




Problema pr76
Estrategias de inserción
Y
X
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000

SOLUCIÓN DE LA COLONIA DE HORMIGAS